All too seldom system evolution is taken into consideration when software systems are developed, resulting in a need to redevelop systems when the system’s environment changes or new requirements to functionality are found. It is important that development processes support modifiability in order to provide cost-effective software evolution. The main aim is to propose a draft development process that put focus on architectural design for modifiability. This will be done through the development of a prototype for an existing security-safety critical system, which was designed without considering modifiability. We will impose requirements to modifiability, redesign the system, and analyse the two prototypes in terms of their ability to meet requirements to modifiability. This will be done in order to show that modifiability can be introduced without compromising security or safety, and to show that modifiability is not a quality attribute you get for free, but must be integrated as part of the development.
ABSTRACT

It is not unusual that software systems are redeveloped when the systems environment changes or new requirements to functionality are found. The reason for this is that system evolution is usually not considered during the initial development of a system.

In this report we propose a development process focusing on modifiability in security-safety critical systems. The process is based on the standard IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems and the CORAS framework for model-based risk assessment. We focus on integrating modifiability as a non-functionality and extend the development process with an architecture and design phase. Modifiability is addressed using Architecture Tradeoff Analysis Method (ATAM) and Attribute Driven Design (ADD). An evaluation of this process is given by redeveloping a prototype for an already existing security-critical system using the approaches given by the process. The main result from the evaluation showed that we where able to obtain a higher level of modifiability than the original prototype, without compromising the functionality and safety level required.
PREFACE

This report documents the work of my Master Thesis, spring 2004, for the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Ph.D. student Siv Hilde Houmb for her guidance, patience and encouraging words throughout the project. I would also like to thank my subject teacher professor Maria Letizia Jaccheri for helpful advice and for sharing her knowledge. Furthermore, I would like to thank my fellow students at my work place for many recreational discussions, my good friends Odd Inge Hillestad and Lars Torstensen for inspiration and social support, and especially thanks to Kristian Abrahamsen, Andreas Knudsen, and Kristin Roald Skau for the hours we spent playing chess.

Trondheim, June 18, 2004

John Inge Hervik
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT ...........................................................................................................</td>
</tr>
<tr>
<td>PREFACE .............................................................................................................</td>
</tr>
<tr>
<td>CONTENTS ...........................................................................................................</td>
</tr>
<tr>
<td>INTRODUCTION .....................................................................................................</td>
</tr>
<tr>
<td>1.1 Motivation and background .................................................................</td>
</tr>
<tr>
<td>1.2 Problem Description ...............................................................................</td>
</tr>
<tr>
<td>1.3 Related work .............................................................................................</td>
</tr>
<tr>
<td>1.4 Structure of the report ............................................................................</td>
</tr>
<tr>
<td>DEVELOPMENT METHODS ..................................................................................</td>
</tr>
<tr>
<td>1.5 Introduction ..............................................................................................</td>
</tr>
<tr>
<td>1.6 Rational Unified Process .........................................................................</td>
</tr>
<tr>
<td>1.7 IEC 61508 ..................................................................................................</td>
</tr>
<tr>
<td>1.8 Risk Management ......................................................................................</td>
</tr>
<tr>
<td>1.8.1 CORAS .....................................................................................................</td>
</tr>
<tr>
<td>1.9 Aspect-oriented Software Development (AOSD) .......................................</td>
</tr>
<tr>
<td>SOFTWARE ARCHITECTURE .................................................................................</td>
</tr>
<tr>
<td>1.10 Introduction ............................................................................................</td>
</tr>
<tr>
<td>1.11 Software architecture ............................................................................</td>
</tr>
<tr>
<td>1.12 Views and structures ..............................................................................</td>
</tr>
<tr>
<td>1.13 Modifiability ............................................................................................</td>
</tr>
<tr>
<td>1.14 Architectural pattern ...............................................................................</td>
</tr>
<tr>
<td>1.15 Attribute-driven design ..........................................................................</td>
</tr>
<tr>
<td>1.16 Software Architecture Analysis ............................................................</td>
</tr>
<tr>
<td>1.16.1 ATAM – Architectural Tradeoff Analysis Method ..................................</td>
</tr>
<tr>
<td>SECURITY AND SAFETY ....................................................................................</td>
</tr>
<tr>
<td>1.17 Introduction ............................................................................................</td>
</tr>
<tr>
<td>1.18 Safety ........................................................................................................</td>
</tr>
<tr>
<td>1.19 Security .....................................................................................................</td>
</tr>
<tr>
<td>THE ORIGINAL PROTOTYPE ...............................................................................</td>
</tr>
<tr>
<td>1.20 Description of the original prototype system .......................................</td>
</tr>
<tr>
<td>1.20.1 The prototype .......................................................................................</td>
</tr>
<tr>
<td>1.20.2 LEGO Mindstorms .................................................................................</td>
</tr>
<tr>
<td>1.20.3 Sony AIBO ............................................................................................</td>
</tr>
<tr>
<td>1.21 Requirements ...........................................................................................</td>
</tr>
<tr>
<td>1.22 Architecture and design of the original prototype ..................................</td>
</tr>
<tr>
<td>1.22.1 Design of AIBO .....................................................................................</td>
</tr>
</tbody>
</table>
1.22.2 Design of PC Controller ................................................................. 44
1.22.3 Design of LEGO system ................................................................. 45

DEVELOPMENT ......................................................................................... 46
1.23 Concept and Scope ............................................................................. 47
1.24 Preliminary Hazard Analysis (PHA) ..................................................... 50
1.25 Requirements ...................................................................................... 52
  1.25.1 Constraints .................................................................................... 53
  1.25.2 Functional requirements ............................................................... 53
  1.25.3 Quality requirements ................................................................. 54
  1.25.4 Use cases ..................................................................................... 56
  1.25.5 Legal states .................................................................................. 59
1.26 Risk Management ............................................................................... 60
  1.26.1 Identify Security threats ............................................................... 61
  1.26.2 Analyze Risk ............................................................................... 64
  1.26.3 Evaluate Risk .............................................................................. 65
  1.26.4 Treat Risk .................................................................................... 65
1.27 System Architecture ........................................................................... 66
  1.27.1 Utility tree scenarios ..................................................................... 67
  1.27.2 Decomposition view .................................................................... 69
  1.27.3 Safety view .................................................................................. 72
  1.27.4 Deployment view ......................................................................... 74
  1.27.5 Security view .............................................................................. 74
1.28 Design of the system ......................................................................... 76
  1.28.1 Authorization mechanism ............................................................ 76
  1.28.2 Control Application ..................................................................... 79
  1.28.3 Factory subsystem ....................................................................... 81

EVALUATION ............................................................................................... 83
1.29 Evaluation strategy ............................................................................ 83
1.30 Evaluation ........................................................................................... 85
  1.30.1 Comparing the functionality ......................................................... 85
  1.30.2 Comparing the safety level ........................................................... 89
  1.30.3 Evaluating modifiability ............................................................... 89
  1.30.4 Summary of evaluation ............................................................... 96

DISCUSSION AND CONCLUSION .............................................................. 97
1.31 Development process .......................................................................... 97
1.32 Aspect Oriented Software Development ............................................ 99
1.33 Evaluation .......................................................................................... 100
1.34 Modifiability in critical systems ................................................................. 101

FURTHER WORK ........................................................................................................... 102

A SECURITY-HAZOP TABLES ................................................................................. 103
B RISK ANALYSIS TABLES ................................................................................... 108
C REFERENCES ........................................................................................................ 112
INTRODUCTION

1.1 Motivation and background

Unfortunately it is far too common to hear horror stories of software project failures that involve vast amounts of money. One study showed that 31% of software projects where never completed, that 53% cost almost twice as much as originally planned, and estimated that American companies and government agencies spent 81 billion dollars for cancelled software projects in 1995 [8].

Over the years, methods and life cycles processes have been developed to assist the development of systems. Rational Unified Process (RUP) [49] is a well known process that guides the developers through the phases of system development, and the IEC standard IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems) [25] is a lifecycle process focusing on safety concerns when developing systems. These processes propose methods to organize the activities related to creation, delivery, and maintenance of software systems. Lately there has been an increased attention on software architecture activities, with the main focus on ensuring that the most important quality requirements to the system are met.

The London Ambulance Service Computer Aided Dispatch is an example of a system development project which ended in failure because the quality requirements were not met [9]. This system was responsible for receiving calls and dispatching ambulances based on the type of calls and available recourses. Immediately after the system became operational, it was evident that it could not keep track of the location of units. Consequently, ambulances where dispatched non-optimal and even multiple ambulances where dispatched to the same location. It seems that the heart of the failure was breakdowns in specification and design, which is common for many software development projects [15]. An inquiry report, conducted by International Workshop on Software Specification and Design (IWSSD) of the Computer Aided Dispatch [9] found the following key problems. The software was incomplete and effectively untested, the implementation approach was high risk, inappropriate, and unjustified assumptions where made during the specification process, there was a lack of consultation with the users and clients in the development process with serious consequences and poor fit of the system with organisational structure of ambulance service. Furthermore, the user interface was poorly designed, the system had
poor performance and robustness, and included ordinary and common bugs and errors [15]. In their report IWSSD also give a strong opinion about the attempt to change working practices through the specification, design, and implementation of a computer system.

After a software system is developed and deployed it usually needs to evolve in order to meet new user requirements and to adapt to environmental changes. This system evolution represents a large cost in software systems. Research has shown that up to 70% of the total lifecycle cost for software systems are spent on evolving systems [52]. This suggests that focusing on modifiability already from the initial stages in development of a system should pay off. All too often requirements to modifiability are added as an afterthought, along with being imprecise. Phrases like “the system shall be easy to modify” do not specify in what regard the system should be able to evolve. This may result in a system suitable for modification in parts of the system that never changes, while the parts of the system exposed to changes are not designed for modifiability. It is hard to anticipate changes that will be made to the systems requirements or the systems environment. Nevertheless it is important to put effort in identifying the most likely changes and designing systems in order to address these concerns. The basis for constructing a system for modifiability is a task for the architecture and design phases, but it is crucial that the implementation follows the constraint laid in the early phases.

The work in this master thesis will be based on a literature study in order to assess possible approaches and methods for developing security-safety critical systems that support achievement of a high level of modifiability. Based on our findings in the literature study we will propose a development process, which will be evaluated by developing a security-safety critical system supporting system evolution. For validating the development process we will use assertion, meaning that the developer will play the role of both experimenter and subject of study. We use this evaluation method since it is inexpensive and because this validation is conducted as a preliminary test before we perform formal validations.
1.2 Problem Description

All too seldom system evolution is taken into consideration when software systems are developed, resulting in a need to redevelop systems when the system’s environment changes or new requirements to functionality are found. It is important that development processes support modifiability in order to provide cost-effective software evolution. The main aim is to propose a draft development process that put focus on architectural design for modifiability. This will be done through the development of a prototype for an existing security-safety critical system, which was designed without considering modifiability. We will impose requirements to modifiability, redesign the system, and analyse the two prototypes in terms of their ability to meet requirements to modifiability. This will be done in order to show that modifiability can be introduced without compromising security or safety, and to show that modifiability is not a quality attribute you get for free, but must be integrated as part of the development.

1.3 Related work

Traditionally, in software development projects the focus has mainly been to fulfil the functionality requirements from the end users [34], which has lead to neglect of the quality attributes of the system such as modifiability and security. One reason for this neglect is that support for quality attributes are not an integrated part of the development process. Various development processes and lifecycle models have been developed to promote different quality attributes; the IEC 61508 standard [25] is a lifecycle model that focuses on the safety attribute in systems, the Australian/New Zealand standard AS/NZS 4360:1999 Risk Management [1] is a general standard to help in risk management in system development. The IST-project CORAS [10] is a framework for system development based on the concept of model based risk assessment. It includes an integrated system development and risk management process aiming at security-critical systems. This process is based on AS/NZS 4360, Rational Unified Process [49], and the Reference Model for Open Distributed Processes (RM-ODP) [33]. The focus of CORAS is on handling security issues all the way through the system development process.

There have been several attempts to show that it is possible to gain a high level of modifiability in software in addition to other quality requirements. Parnas et al. [50] tried to demonstrate that using object-oriented techniques to provide modifiability could be used
to construct real-time high-performance software. The system was constructed in 1977, and was a software system to an A-7e Corsair II attack aircraft used by the U.S. Navy, used in the 1960s, 1970s and 1980s. Although the system was never taken in use, the project is considered a research success [41]. The fact that revolutionary principles such as encapsulation and information hiding used in the project gained a general acceptance is an indicator of the success of the project.

In recent years the new software development paradigm Aspect Oriented Software Development (AOSD) has emerged as an alternative for developing software systems. This paradigm seeks to separate crosscutting concerns, such as security, from the main functionality of the system [23]. Aspect Oriented Modelling (AOM) techniques help system architects to design the systems primary system modularization focused on end users functionality requirements, and to design the additional system concerns (such as quality attributes) separate from the primary modularization [20]. Separating concerns in this way can aid in the evolution of the system, since a developer can focus on realization of a single concern without having to worry about its interactions with other concerns or the primary functionality.

1.4 Structure of the report

This report consists of the following chapters. The chapters should be read in sequential order if the reader is not already familiar with the background material.

Introduction
Describes the motivation for our work, the problem definition, and gives an account of related work.

Development Methods
Gives an introduction to different software development methods

Software Architecture
Gives an introduction to software architecture and how to design software architecture to meet the quality requirements to software systems
Security and Safety
Describes the quality attributes safety and security, and the relationship between them.

The original prototype
Gives a short description of the original prototype

Development
Describes the development process we propose.
Describes the development of the new prototype.

Evaluation
Gives a short description of evaluation methods
Compares the functionality, safety level, and modifiability level for the new and the original prototype

Discussion
Debates our work and our findings

Further work
Suggestions for further work
DEVELOPMENT METHODS

1.5 Introduction

This chapter gives an introduction to some methods for developing software systems. Several development processes and standards exist to help organize the activities necessary to build software systems. Rational Unified Process, described in section 1.6, is a widely acknowledged development process focusing on building high quality software systems. In section 1.7 we give a brief introduction to the IEC standard IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems), which is a life cycle process standard aimed at safety critical systems. Risk Management which is described in section 1.8, has a primary goal of eliminating or reducing risk in software systems. An example of risk management used in a development process is the CORAS framework (described in section 1.8.1). This model-based risk assessment methodology provides an integrated risk management and system development process aimed at security critical systems. Aspect oriented software development (AOSD), introduced in section 1.9, has become a new paradigm in software development. The principle of AOSD is to separate each crosscutting concern, such as security and performance, from the primary system modularization. This makes system evolution a lot easier, since the developer can focus on a single concern at one time and not worry about main functionality or other system concern.

In our work we have developed our own development process, which is described in section 0. This development process is aimed at security-safety critical systems, and includes hazard analysis and risk management. The development process is designed to promote software evolution by integrating modifiability into the system primary through software architecture and design.
1.6 Rational Unified Process

A software development process is a method to organize the activities related to creation, delivery, and maintenance of software systems [11]. Rational Unified Process (RUP) is an example of such a process. Its goal is to enable the production of high quality software that meets end user needs within predictable schedules and budgets [21]. RUP is a widely acknowledged development process that captures some of the best practices of current software development in a form that is adaptable for a wide range of projects. The process is an iterative process, which gives several advantages compared to linear development process. These advantages include increasing understanding through successive refinements, flexibility to accommodate new requirements or tactical changes in business objects and identifying and resolving risks sooner rather than later [53]. As shown in Figure 0-1, there are four phases in the RUP development process; inception phase, elaboration phase, construction phase, and transition phase. Booch et al. describe these in [21]:

![Figure 0-1: Rational unified process](image)
Inception. In the inception phase one establishes the business case and delimits the project's scope. The business case includes criteria for success, risk assessment, estimates for the resources needed, and a phase plan providing a schedule of major milestones. The phase is often concluded by producing a prototype that serves as a proof of concept.

Elaboration. This phase focuses on analyzing the problem domain, establishing a sound architectural foundation, developing the project plan and eliminating the highest risk elements of the project. At this point it is important to make architectural decisions with an understanding of the whole system and to describe system requirements. To verify the architecture, a system restricted to demonstrating the architectural choices and executing significant use cases is implemented.

Construction. The focus of this phase is to iteratively and incrementally develop a complete product that is ready for transition to its user community. This implies describing the remaining requirements and acceptance criteria, providing the design and completing the implementation and test of the software. At the end of the construction phase one decides whether software, sites and users are ready to go operational.

Transition. This phase deals with deploying the software to the user community. Once the system is in the hands of the users, issues often arise requiring additional development in order to adjust the system, correct undetected problems or finish some feature that has been postponed. The phase typically starts with a beta release of the system which is later replaced with the production system. At the end of this cycle it is decided whether the life cycle objectives of the project have been met and determine if a new development cycle should be started.
1.7 IEC 61508

The IEC standard IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems [26] is a lifecycle process that focuses on safety. It covers important aspects that need to be addressed when electrical, electronic, and programmable devices are used in safety functions. The strategy of the standard is to derive safety requirements from a hazard and risk analysis and to design the system to meet those safety requirements, taking all possible causes of failure into account. The essence is that all activities relating to functional safety are managed in a planned and methodical way, with each phase having defined inputs and outputs. The standard considers all aspects in a safety lifecycle from initial concept and requirement elicitation, through functional allocation, implementation, operation, and maintenance to decommissioning.
The process consists of the following phases:

1. **Concept**: An understanding of the system and its environment is developed.
2. **Overall scope definition**: The boundaries of the system and its environment are determined, and the scope of the hazard and risk analysis is specified.
3. **Hazard and risk analysis**: Hazards and hazardous events of the system, the event sequences leading to the hazardous events, and the risks associated with the hazardous events are determined.
4. **Overall safety requirements**: The specification for the overall safety requirements is developed in order to achieve the required functional safety.
5. **Safety requirements allocation**: The safety functions contained in the overall safety requirements specification are allocated to the safety-related system, and a safety integrity level is allocated to each safety function.
6. Overall operation and maintenance planning: A plan is developed for operating and maintaining the system, and the required functional safety is ensured to be maintained during operation and maintenance.

7. Overall safety validation planning: A plan for the overall safety validation of the system is developed.

8. Overall installation and commissioning planning: Plans, ensuring that the required functional safety are achieved, are developed for the installation and commissioning of the system


10. Safety-related systems: other technology: Other technology safety-related systems are created to meet the requirements specified for such systems (outside scope of the standard).

11. External risk reduction facilities: External risk reduction facilities are created to meet the requirements specified for such facilities (outside scope of the standard).

12. Overall installation and commissioning: The E/E/PES safety-related system is installed and commissioned.

13. Overall safety validation: The E/E/PES safety-related system is validated to meet the overall safety requirements specification.

14. Overall operation, maintenance and repair: The system is operated, maintained and repaired in order to ensure that the required functional safety is maintained.

15. Overall modification and retrofit: The functional safety of the system is ensured to be appropriate both during and after modification and retrofit.

16. Decommissioning or disposal: The functional safety of the system is ensured to be appropriate during and after decommissioning or disposing of the system.

### 1.8 Risk Management

Risk management is concerned with establishing the context, identifying, analyzing, evaluating, treating, monitoring, and communicating risk [10]. AS/NZS 4360:1999 Risk Management [1] describes a general applicable approach to risk management with the primary goal to eliminate or reduce risk in order to minimize losses. This structural iterative process starts with an identification of the context for the assessment. This includes organizational and risk management context, as well as criteria and structure of the assessment. When the context is established, potential risks are first identified and then analyzed. In the analyzing phase both the likelihood and the consequences of risks are estimated. Predefined acceptance criteria are used when evaluating if the risks are acceptable or not. The unacceptable risks are treated in the final phase of the process. In parallel to this process one have the communicate and consult sub-process and the monitor and review sub-process, the management part of the process. These ensure that communication and consultation with the stakeholders in order to establish common understanding of risk issues, and that change does not alter risk priorities, respectively.
1.8.1 CORAS

CORAS is a model based risk assessment methodology, developed under the Information Society Technologies (IST) Programme. The main objective of the CORAS project was to [10]:

- Develop a practical framework, exploiting methods for risk analysis developed within the safety domain, semiformal description methods, and computerised tools, for a precise, unambiguous, and efficient risk analysis of security critical systems.
- Apply the framework in security critical application domains.
- Assess the applicability, usability, and efficiency of the framework.
- Promote the exploitation potential of the CORAS framework.

As shown in Figure 0-3 the CORAS framework consist of a model-based risk assessment methodology funded on four pillars, namely risk documentation (RM-ODP), risk management (AS/NZS 4360), integrated risk management and system development (UP), and a platform for tool inclusion based on data-integration (XML). The risk management process and the integrated risk management and system development are of most interest for this thesis, and we give a short described of these below. The reader is referred to ([14]) for a fuller description of CORAS.

![Figure 0-3: The CORAS framework ([14])](image-url)
1.9 Aspect-oriented Software Development (AOSD)

In the recent years problems known as code-tangling and code crosscutting has received increasing attention. As a result of this attention, aspect oriented software development (AOSD) has emerged as an alternative paradigm for developing software systems. AOSD seeks to modularize concerns in software that crosscut over functionality or system elements [23]. Aspect oriented modelling (AOM) is one research area which focus on this point at the design level. In [24], Nusiebeh defines an aspect as a crosscutting concern. Furthermore he defines a concern as a property of interest to a stakeholder, where a crosscutting concern is an interest to a stakeholder that is intertwined with several functional requirements or system elements.

In software system development it is not unusual to decompose problems and their solutions to manage their complexity. This decomposition partitions competing concerns in the problem and solution space. Some of the most important concerns determine the primary structure(s) of the system. Other concerns that are not used to determine the primary structure are often realized through added functionality that crosscut the primary modularization [20]. Aspect oriented modelling techniques help system architects to design the primary system modularization and to design additional system concerns not part of the primary system modularization.

AOM techniques allow system developers to address concerns like security, fault-tolerance, safety, and availability separately from core functional requirements at design level [19]. This separation of concerns implies that developers must be able to analyze the integration of these additional concerns with the core functionality of the system in order to compare realizations that may be based on conflicting design goals. The principle of separating concerns is not new, but in AOSD we separate concerns as aspects that have no knowledge of each other. This can aid in the system evolution since a developer can focus on the realization of a single concern without having to worry about its interactions with other concerns or primary functionality. The separating of concerns promotes modifiability, in that high cohesion of system elements (i.e. a communication module does not have to concern itself with encryption, since encryption is modelled as an aspect) and low coupling (i.e. the communication module is oblivious of the encryption aspect). Compared to object-oriented development and implicit invocation, aspect oriented development takes cohesion one step further. In Object-oriented development an object
calls another object to get something done, and in implicit invocation an object sends a message at certain points to inform subscribers of an event. In aspect oriented development it is the aspect, or more specific the pointcut to the aspect that is responsible for handling the control flow. The original object will have no knowledge of the weaving at execution time.

The system design in AOSD, consist of a primary system modularization, a set of aspects representing pervasive concerns that impacts the primary modularization, and a weaving mechanism that connects the aspects with the primary modularization. The weaving mechanism is a set of rules for when an aspect should be used. These rules are called pointcuts and identify specific points in a model that are affected by a crosscutting concern [35]. The key advantage of this approach is that an entirely different set of constraints, represented by an aspect, can be weaved into the model by making a change at one single point. This solves a serious scalability and modifiability issue. Another advantage is the formal notation used to describe the pointcuts and the aspects, which promote traceability from design to implementation [35].
SOFTWARE ARCHITECTURE

1.10 Introduction

In a development process, the first phase to focus on how the system will behave to meet its requirements is the software architecture phase. The architecture is the foundation on which the system will be designed, implemented, and later evolved. The design of the architecture will influence the quality attributes of the system. Choices in the architecture design phase will promote some quality attributes and prohibit others. An architecture designed for modifiability will make it easier to evolve the system after it has been deployed.

We start this chapter with explaining what we mean by software architecture and why software architecture is an important phase in developing software systems in section 1.11. We then describe architectural views and structures and how these to concepts relate in section 1.12. In section 1.13 we give a brief description of the quality attribute modifiability. To ensure that they design an architecture that provide the desired quality attributes, developers use architectural patterns, which are described in section 1.14. Attribute Driven Design (ADD) is a method that further helps the developer in designing an architecture to meet the required quality attributes. In ADD, described in section 1.15, the developer use architectural patterns to fulfil a set of scenarios that are based on the quality requirements for the system. We conclude this chapter with a description of software architecture analysis in section 1.16, where we use the Architectural Tradeoff Analysis Method (ATAM) as an example.

1.11 Software architecture

Since the emerging of software architecture as a separate research discipline in the early 90’s, researchers have come up with several definitions of architecture in software intensive systems. Shaw & Garlan [13] suggest that software architecture “defines the system in terms of computational components and interaction among those components.” Another definition given by Bass et al. in [40] states: “The software architecture of a program or computing system is the structure or structures of the system, which comprise software
elements, the externally visible properties of those elements, and the relationships among them.”

In IEEE STD 1471-2000 [27] software architecture is defined as “The fundamental organization of the system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution” There are several important aspects of this definition. First, the architecture is a description of how the system is decomposed, meaning which components the system consist of? Second, the architecture tells us which components interact, and in general terms how this interaction will take place (e.g. one-way interaction or two-way interaction). Third, the architecture also describe how the system will interact with the environment, which components are responsible for receiving requests and which components are responsible for responding. Finally, the architecture specify guidelines and constraints for the design and implementation of the system, and just as important what constraints the system engineers will have to follow when changes or additions to the system are made. In [47] Nico Lassing illustrates this by considering an architecture consisting of three layers, a presentation layer, a processing layer, and a data layer. The constraints of this architecture are that all data are stored in the data layer, while the business logic is handled in the processing layer, and the interaction with the users is handled in the presentation layer. If the system is to be extended after deployment the architecture prescribes, for instance, a new dialog box has to be realized in the presentation layer.

Bass et al. provides three reasons why architecture is important [40]:

1. Communication among stakeholders. Software architecture represents a common high-level abstraction of a system that most, if not all of the system’s stakeholders can use as a basis for creating mutual understanding forming consensus, and communicating with each other.

2. Early design decisions. Software architecture represents the manifestation of the earliest design decisions about a system and these early bindings carry weight far out of proportion to their individual gravity with respect to the system’s remaining development, its deployment, and its maintenance life. It is also the earliest point at which the system to be built can be analyzed.

3. Transferable abstraction of a system. Software architecture constitutes a relatively small, intellectual graspable model for how a system is structured an how its components work together; this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements and can promote large-scale reuse.

If architectural description of a system is to work as communication language between different stakeholders, we need an adequate description. It is not unusual that the
stakeholders have different backgrounds and different interests in the system. A common method to incorporate the stakeholders’ different interests is to describe the architecture through different views. Views are further described in section 1.12. The early design decisions represented in the architecture have extensive effects, they are the most difficult to get correct and are hardest to change at a later time [40]. It is therefore imperative to design the architecture with the most important quality requirements in mind.

The architecture is the foundation of the system to be built and it is important that decisions concerning the tradeoff between different quality attributes are considered as part of this phase in the development process. The architecture chosen will influence this by inhibiting some quality attributes and enabling others [40]. From this we can see that it is imperative for the success of the system that the architecture is designed to meet the most important quality requirements. It is possible to predict if a system will exhibit the required quality attributes before the system is developed and deployed. This is further elaborated in section 1.12. The functionality of the system is also embodied in the architecture. The different components described in the architecture all have responsibility for a part of the systems functionality and cooperate to provide services required by the users. Finally, the early design decisions, made when making architecture decisions, have impact on the organization. Often the work is divided according to the decomposition of the system. This imposes constraints on changing the responsibilities of the different components, since it can prove difficult to move the responsibility from one team to another.

1.12 Views and structures

Modern software systems are increasingly complex and difficult to grasp. To simplify understanding of a system, we put focus on one aspect of the systems architecture at the time. The different aspects we look at are called structures, where a structure is a way to organize the elements in the system. This implies that a system can consist of several structures as described in the definition of system architecture given by Bass et al in [41]. To describe the structures of a system we use views. A view is a representation of a structure, and it describes the elements and their relationships in the structure. Bass et al argue in [41] that architectural structures can be divided into three groups according to the type of elements and relations.
• Module structures. Describes elements as modules, which are units of implementation. Modules represent a code based way of considering the system. They are assigned areas of functional responsibility. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization relationships? Decomposition structure and layered structure are examples of module based structures.

• Component-and-connector structures. Here the elements are runtime components which are principal units of computation, and connectors which are the communication vehicles among the components. Components-and-connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system’s structure change as it executes?

• Allocation structures. Allocation structures show the relationship between the software elements and the elements in one or more external environments in which the software is created and executed. They answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing and system building? What is the assignment of software elements to development teams?

There exist many sets of structures, but only a small set of structures are typically used in one system. There have been proposed several different sets of views to describe these structures. In Kruchten’s 4+1 view model [48] he includes a logical viewpoint, a process viewpoint, a development viewpoint, physical viewpoint, and a set of scenarios (the +1 viewpoint) to describe the architecture of a system. The focus of the logical view is to describe the functionality of the system, by showing what services the system offers to its users. The process viewpoint shows the different processes in the system and how they interact. In development viewpoint we are concerned with how the system can be divided into units or modules that can be developed independently of each other. As the name suggests, the physical viewpoint describes how the system is deployed in terms of its hardware resources. The role of the scenarios in the 4+1 view model is to describe how the elements in the four viewpoints work together. Clements et al. argues in [42] that a healthier approach than a fixed model is that architects should choose views based on the needed engineering leverage that each view provides and reflect the stakeholder interests. Thus the views chosen to describe an architecture should be chosen by considering the main quality goals of the system. Safety critical systems should have a view describing how
safety concerns are addressed. Systems designed for modifiability should include a view describing how requirements to modifiability are addressed.

1.13 Modifiability

The modifiability of a software system is the ease with which it can be modified to meet changes in the environment, requirements or functional specification [52], while maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in environment, and in requirements and functional specification [31]. These to definitions demonstrate the main difference between modifiability and maintainability. Maintainability is concerned with the correction of bugs in software, while modifiability is not.

Maintenance and system evolution generally presents the major cost factor of the lifecycle of software systems. Consequently, stakeholders will favour systems designed in order to support future changes and thus reducing the cost incurred by these changes.

Module-based architectural structures are the foundation for modifiability in a software system. In particular, the decomposition of a system into subsystems and components and the assigning of functional responsibilities influence the level of system modifiability. The reason for this is that the decomposition structure can ensure that changes fall within a few small modules [41].

1.14 Architectural pattern

When experts work on a particular problem they seldom solve it by inventing a completely new solution. They usually make use of experience from a similar problem and apply a somewhat similar solution to the new problem. When we abstract the general from specific problem-solution pairs we get patterns. Christopher Alexander describes a pattern as a thee-part rule that expresses the relation between a certain context, a problem, and a solution [7].
A software architectural pattern is a description of software element and relation types together with a set of constraints on how they may be used [41]. Architectural patterns provide a solution to well known problems concerned with particular quality attributes. These solutions can be seen as structure templates. That is, an architectural pattern offers a well known structural solution to achieve a specific quality goal. Some patterns promote performance while others may promote modifiability. In this way patterns help to specify the fundamental structures of the system to ensure the most important quality requirements. A pattern imposes constraints to an architecture in order to fulfil requirements to various quality attributes. As mentioned, different patterns are suitable to meet some quality attributes, and unsuitable to meet others. Therefore an architecture usually consist of more than one pattern. The fact that patterns exhibit known quality attributes is on of the most useful aspects of architectural patterns. Examples of an architectural pattern are the Model-View-Controller (MVC) and the Presentation-Abstraction-Control patterns. Both are frequently used in interactive applications, but while the MVC pattern are usually more effective (performance), the PAC pattern supports multitasking (concurrency) and task-specific user interfaces (usability) better [16].

1.15 Attribute-driven design

Attribute-driven design (ADD) is a method for designing an architecture to meet quality and functional requirements [41]. ADD is an approach that bases the decomposition on the quality goals of the system. At each decomposition-level, tactics and architectural patterns are chosen to satisfy a set of quality scenarios and then functionality is allocated to the modules in this decomposition. The ADD method is used for the appropriate views in the architecture, providing the first several levels of these views. The resulting architecture and design from ADD needs to be further detailed before the system can be implemented, but the ADD gives a good starting point for this detailed design.

The ADD method is composed of the following steps [41].

1. Choose the module to be decomposed. The module to start with is usually the whole system. All required inputs for this module should be available (constraints, functional requirements, quality requirements).
2. Refine the module decomposition according to these steps:
a. Choose the architectural drives from the set of concrete quality scenarios and functional requirements. This step determines what is important for the decomposition.

b. Choose an architectural pattern that satisfies the architectural drivers. Create (or select) the pattern based on tactics that can be used to achieve the drivers. Identify child modules required to implement tactics.

c. Instantiate modules and allocate functionality from the use cases and represent using multiple views.

d. Define interfaces of the child modules. The decomposition provides modules and constraints on the types of module interactions. Document this information in the interface document for each module.

e. Verify and refine use cases and quality scenarios and make them constraints for the child modules. This step verifies that nothing important was forgotten and prepares the child modules for further decomposition.

3. Repeat the steps above for every module that needs further decomposition.

1.16 Software Architecture Analysis

Software architecture is a result of early design decisions based on requirements. Before you put all your project money and resources into the development of a system based on these decisions, you would like to investigate the reasonability of these decisions and that they support the quality requirements posed upon the system. One way to explore the consequences of your decisions before you start building the system is to evaluate your architecture using an architecture analysis method. Clements et al. [57] has identified several benefits of performing an architecture evaluation.

- Uncovers problems that would be orders of magnitude more expensive to correct at later time. This is the most obvious and main benefit from architecture evaluation. A common problem of this kind is that the architecture doesn’t support the most important quality attributes.

- Put stakeholders in the same room. An architecture evaluation session may be the first opportunity stakeholders get to meet. By getting stakeholders together there can emerge an understanding that everybody really wants the same; a successful system. New channels of communication can be created, since participants often exchange e-mail addresses and phone numbers.

- Forces an articulation of specific quality goals. The role of the stakeholders is to articulate the quality goals that the architecture should meet. These goals are often poorly documented. Scenarios provide explicit quality benchmarks.

- Results in the prioritization of conflicting goals. Conflicts that might arise among the goals expressed by different stakeholders will be aired. If the architect cannot satisfy all the conflicting goals, he will receive clear and explicit guidance about which ones are most important.

- Forces a clear explication of the architecture. The architect is compelled to make a group of people not privy to the architecture’s creation understand it, in
detail, in an unambiguous way. Among other things, this will serve as a dress rehearsal for explaining it to the other designers, component developers, and testers. The project benefits by forcing this explication early.

- Improves the quality of architectural documentation. Often, an evaluation will call for documentation that has not yet been prepared. If the evaluation requires it, then it’s an odds-on bet that somebody on the project team will need it also. Again, the project benefits because it enters development better prepared.

- Uncovers opportunities for cross-project reuse. Stakeholders and evaluation team are often external to the development project, but do often work on or are familiar with other projects within the same parent organization. As such, they are in a good position to identify components that can be reused later or to have knowledge of components that already exist and perhaps could be imported into the current project.

- Results in improved architecture practices. Organizations that practice architecture evaluation as a standard part of their development process report an improvement in the quality of architectures that are evaluated. As development organizations learn to anticipate the kinds of questions that will be asked, the kinds of issues that will be raised, and the kinds of documentation that will be required for evaluations, they naturally preposition themselves to maximize their performance on evaluations. Over time, an organization develops a culture that promotes good architectural design.

There are also costs associated to an architecture analysis. First, the actual analysis requires some effort by the evaluation team. This cause extra personnel cost to the development project. Although there usually isn’t much calendar time added to the project, there will be some. Second, there are some indirect costs relating to training of the evaluating team. Finally, it is common that the best people are assigned to evaluation teams resulting in reduced expertise in the development team [17].

When should we evaluate the architecture? Generally it is thought to be good practice to hold an evaluation when development teams start to make decisions that depend on the architecture and when the cost of undoing those decisions would outweigh the cost of holding an evaluation [57]. However, an architecture analysis could also be conducted in order to look into alternative architectures or as a way to apprehend an existing system.

The purpose of an architecture analysis is to evaluate whether the architecture will satisfy the functional and quality goals. The architecture is strongly related to meeting the quality goals, thus it is extremely important that the architecture is suitable for the systems needs. Clements et al. argues in [57] that an architecture has to meet two criteria to be suitable:
1. The system that results from it will meet its quality goals. That is, the system will run predictably and fast enough to meet its performance (timing) requirements. It will be modifiable in planned ways. It will meet its security constraints. It will provide the required behavioural function. Not every quality property of a system is direct result of its architecture, but many are, and for those that are, the architecture is suitable if it provides the blueprint for building a system that achieves those properties.

2. The system can be built using the resources at hand: the staff, the budget, the legacy software (if any), and the time allotted before delivery. That is, the architecture is buildable.

It is important to recognize that an architecture that is suitable in one system not necessarily is suitable in another system. The quality goals are the key here. If the system requires a high level of security and performance, an architecture designed for modifiability may be unsuitable. Furthermore, it is important to be aware that the result of an architecture analysis is qualitative. That is, it will not tell you how good your architecture is on a scale, but rather identify which quality goals your architecture are likely to satisfy, and which quality goals are at risk.

Although the architecture has a great impact on the quality of the system, it is important to remember that design and implementation also has an affect on the quality of the system. A system can still be a failure even though your architecture fulfils all your requirements.

1.16.1 ATAM – Architectural Tradeoff Analysis Method

ATAM is a method for analyzing architecture in a software intensive system, developed by Kazman et al. at Software Engineering Institute [56]. When using this method the software architecture evaluator team seek to reveal whether the architecture satisfies the quality goals of the system or not. However, ATAM doesn’t provide a straight forward “good” or “bad” answer, but will rather tell you where the risk and strengths lies in your architecture. Obviously, the quality goals and the architecture have to be clearly defined before an analysis can be performed. ATAM takes this into consideration. Two major goals of ATAM are [56]

- To elicit and refine a precise statement of the architecture’s driving quality attribute requirements
- To elicit and refine a precise statement of the architectural design decisions
There are four phases in ATAM, phases 0-3. Phase 0 is concerned with preparation of the evaluation. In this phase the evaluating team is created and a partnership with the client has to be established. The last phase is a follow-up phase in which the results from the evaluation are collected and documented. Phases 1 and 2 are the evaluating phases. The evaluation is divided into two phases, where phase 1 is architecture-centric and phase 2 is concerned with the stakeholders’ points of view and the verification of the results of phase 1.

Together phase 1 and phase 2 consist of the nine steps that define the architecture evaluation (described below). The evaluation starts with a presentation of ATAM, of the business goals that are motivating the developing effort and of the architecture, in steps 1, 2 and 3 respectively. The architect presents the architecture for the evaluation team polling for probing questions to arise. It is likely that the architect will prepare for the anticipated questions, resulting in an architecture description of better quality.

In step 4 the evaluation team try to identify and understand the architectural approaches chosen. In this step the architectural approaches are identified and explained, but not analyzed. The task of step 5 is to generate a utility tree. A utility tree describes the quality attributes of the system and decomposes these into scenarios. In ATAM a scenario is defined as a short statement describing an interaction of one stakeholder with the system [57]. Scenarios should be concrete and unambiguous. When the utility tree is built, the scenarios are prioritized by the level of difficulty and how important they are to the success of the system. The resulting prioritized lists of scenarios serve as a plan for the rest of the evaluation. Only the scenarios of some importance and difficulty are analyzed.

At this point we have a precise statement of the architectural design decisions (step 5) and clearly defined quality goals (step 4) according to the two major goals of ATAM. Step 6 tries to determine whether the architectural approaches are suitable for the quality goals identified. This is done in a scenario walkthrough. For each quality attribute scenario the architectural approach used for meeting the attribute-specific requirement (captured in the scenario) is described. In the walkthrough the sensitivity points and tradeoff points in the architecture are identified. A sensitivity point is defined as a property of one or more components (and/or component relationships) that is critical for achieving a particular quality attribute response [56]. For example availability of a web-site may be sensitive to the
number of replicated web-servers. A tradeoff point is a sensitivity point that affect more
than one quality attribute. Sensitivity and tradeoff points are either a risk or a non-risk.
Before this step is finished all sensitivity and tradeoff points should be characterized in
terms of risk and non-risk and documented together with the attribute associated with it.
At the end of this step the evaluation team should have a clear picture of the most
important aspect of the entire architecture, the rationale for key design decisions that have
been made, and a list of sensitivity points, tradeoff points, risks and non-risks.

Step 7 is the first step in the stakeholder-centric phase 2 of the analysis. In this step the
stakeholders are gathered and asked to participate in a brainstorming session. In the
brainstorming session the focus is on finding scenarios and then to prioritize them. The
stakeholders are asked to find three kinds of scenarios; Use case scenarios, Growth
scenarios, and Exploratory scenarios. Use case scenarios describe an interaction with the
system to execute a function. Growth scenarios are descriptions of how the architecture is
expected to accommodate changes and growth. While Exploratory scenarios describes
dramatic changes or extreme growth to the system. The latter can stimulate to find more
sensitivity points and tradeoff points in the architecture. The scenarios that are found
during the brainstorm are prioritized using a voting scheme. The scenarios that are deemed
most important (e.g. top five) are added to the utility tree. If a scenario doesn’t fit to a
branch in the utility tree, this is an indication that the architect has failed to consider an
important quality attribute. This should be investigated further, and a risk could be the
result and should be documented. In step 8 we again analyze the architectural approaches
as in step 6, but now we use the new utility tree from step 7. The final step of the ATAM,
step 9, consists of collecting information from the evaluation and then summarizing and
presenting it to the stakeholders.
SECURITY AND SAFETY

1.17 Introduction

This chapter gives a short introduction to the quality attributes safety (section 1.18) and security (section 1.19), and how security relates to safety in a security-safety critical system, such as the prototype system described in section 0. In a security-safety critical system we focus our security concerns towards security aspects that may affect the safety of the system. In [2], Avizieni, Laprie, and Randell identify security to comprise of availability, integrity, and confidentiality. These three attributes are security attributes if being broken may have consequences for safety. If availability is compromised in such a way that unauthorized users gain access to the security-safety critical system, they may maliciously alter the systems behaviour and lead to damage to personnel. Compromised integrity in the system may lead to system states where the system is not safe. Confidentiality is indirectly related to safety in respect to prevention of disclosure of passwords, encrypted keys, or information of system elements that are vulnerable to attacks, which may compromise availability.

1.18 Safety

In systems with physical properties there may appear situations where the system may harm something in its environment. If a system by failing can cause harm to life, property, or the environment, the system is deemed as safety-critical [36]. In [46] Nancy Leveson defines safety as “freedom from accident or losses”. She describes an accident as an undesired or unplanned event that results in a specified level of loss, where loss is any negative consequence. It is an important aspect of the definition of accident that while it is an event that is undesired or unplanned, it may well be an event that can be foreseen. For example, we expect there to be automobile accidents, but they are highly undesired and they are not planned [46].

System safety is concerned with preventing foreseeable accidents and to prevent unforeseeable ones, by the use of theory and systems engineering approaches. Safety is an emergent property of a system, not a component property. That is, the whole system must deal with safety, not only some of the components or subsystems. It is important that...
safety considerations must be part of the initial stage of developing a system. Research has shown that up to 90% of design decisions that affect safety properties are made in the initial stages of development [61].

A well known standard to aid in the development of safety critical systems is the IEC 61508 standard made by The International Electrotechnical Commission (IEC). This standard covers all safety-related systems of electrotechnical nature, including electromechanical systems, solid-state electronic systems and computer-based systems. It defines a development lifecycle with focus on safety, that can either be used as a standalone development lifecycle or as a part of a development process. IEC 61508 is further described in section 0.

1.19 Security

Security is concerned with protecting important business assets such as software, physical assets (e.g. hardware, communication facilities), information and data (e.g. documents, databases), the ability to produce some product or provide a service, intangibles (e.g. goodwill, image), and people [29]. IT security relates to an IT system, while information security is concerned with security in an information system, where an IT system is the computerized part of an information system. In ISO 17799:2000 [30] information security is characterized as the preservation of confidentiality, integrity, and availability. Bass et al. adds the attributes non-repudiation, assurance, and auditing to a system that provides security. They describe these six security attributes in [41] as:

1. Non-repudiation is the property that a transaction (access to or modification of data or services) cannot be denied by any of the parties to it. This means you cannot deny that you ordered a particular item over the Internet if, in fact, you did.
2. Confidentiality is the property that data or services are protected from unauthorized access. This means that a hacker cannot access your income tax returns on a government computer.
3. Integrity is the property that data or services are being delivered as intended. This means that your grade has not been changed since your instructor assigned it.
4. Assurance is the property that the parties involved in a transaction are who they claim to be. This means that, when a customer sends credit card number to an Internet merchant, the merchant is who the customer thinks they are.
5. Availability is the property that the system will be available for legitimate use. This means that a denial-of-service attack won’t prevent your ordering a book on the Internet.

6. Auditing is the property that the system track activities within it at levels sufficient to reconstruct them. This means that, if you transfer money out of one account to another account, the system will maintain a record of the transfer.

Furthermore, ISO\IEC 13335 include reliability to the list of security attributes, and define IT security as all aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability [29]. ISO\IEC 13335 defines accountability similar to Bass et al.’s auditing property, and defines reliability as the property of consistent intended behaviour and results.

To achieve information security we need to implement a suitable set of controls that may include security policy, practices, procedures, organizational structures, and software functions [30]. However, most often the situation for security oriented requirement is that it is too often considered as an afterthought rather than an integrated part of the development [54]. The EU funded CORAS project provides an integrated risk management and system development process for security critical systems [10]. CORAS uses a model-based risk assessment, and its foundation is based on four pillars. The first is Risk documentation framework based on RM-ODP. The second is a Risk management process based on AS/NZS 4360. The third is an integrated risk management and system development process based on Rational Unified Process (RUP). And the fourth is a platform for tool inclusion on data-integration based on XML. CORAS is further described in section 1.8.1.

A security attack is an action that exploits a vulnerability of the system to violate one or more security attributes. These attacks can either be intentional or unintentional, performed by either internal or external users. Research indicates that approximately 80 per cent of all security breaches are caused by a company’s own staff [52]. Furthermore, attacks can be of a passive or active nature [4]. In a passive attack the intruder observes the information passing through the observing point without interfering with the information flow or its contents. Interception of information is an example of a passive attack, where an unauthorized user gains access to confidential information. An active attack on the other hand, is an attack where the intruder seeks to disrupt the information channel in
some way. It may be by inserting fabricated messages, by modifying messages in a communication channel, or by prevent or inhibit normal use of the system (e.g. by overloading communication channels).
**THE ORIGINAL PROTOTYPE**

In our work we will show that modifiability doesn't come for free, rather it has to be built into a system from early in the development process. To accomplish this we developed a new prototype for an already existing security-safety system. The original prototype for this security-critical system was developed without focusing on modifiability. This prototype developed by Karine Sorby in [37] is described in this chapter. We start the chapter with a short description of the prototype and the technology used, in 1.20. The functional and quality requirements are given in section 1.21. Section 1.22 gives a description of the architecture and the design of the original prototype. The requirements, architecture, and design will be used in an evaluation of the two prototypes in section 0.

**1.20 Description of the original prototype system**

In this section we give a brief introduction of the original prototype system developed in [37]. We start with a short description of the functionality of the prototype in 1.20.1, and then shortly describe two technologies used in the prototype; LEGO Mindstorms in 1.20.2 and Sony AIBO in 1.20.3.

**1.20.1 The prototype**

The prototype used in this master thesis is a security-safety critical system. In [37] Karine Sorby describes a security-safety critical system as a system where a security critical system monitors and controls a safety critical system.

The prototype was developed in Karine Sorby's master thesis in spring 2003 [37]. The prototype is comprised of a cutting robot, a control system with an authorization mechanism (see Figure 0-1). The cutting robot is placed inside a safety zone, which is a restricted area sealed off by a gate. Only authorised personnel are allowed to enter the safety zone. The authorization mechanism, which is an advanced security camera, is placed outside the safety zone and monitors the area in front of the gate. When authorized personnel enter this area, the security camera informs the control system, which controls both the locking mechanism on the gate and the cutting robot. The gate is unlocked for authorized personnel only, and if he or she opens the gate to enter the safety zone the control system shuts down the cutting robot. Outside the safety zone there are placed one
emergency stop button and one button to start the cutting robot. Anyone can stop the cutting robot with the emergency button, but the start button will be activated only when an authorized person is identified in front of the gate.

![Diagram of the security-safety critical prototype](image)

Figure 0-1: The security-safety critical prototype [38].

Both the cutting robot and the gate to the safety zone are run by a LEGO Mindstorms robot. The LEGO Mindstorms robot and a control application run on a PC that represents the control system. A Sony AIBO robot is used for the security camera and comprises the monitoring system along with the control system. The control application and the Sony AIBO communicate through a wireless LAN, while infrared signals are used to send messages from the control application to the LEGO Mindstorms robot.

### 1.20.2 LEGO Mindstorms

Lego Mindstorms are a new generation of Lego Robots. It is a robotics kit developed by LEGO in collaboration with Massachusetts Institute of Technology (MIT). It consists of a
programmable brick called Robotics Commanding Explorer (RCX), motors, sensors, and the traditional LEGO bricks. Out of the box the user can build and program robots using the included diagrams and example programs. But the user may also make custom made robots, by building the robot from imagination and by programming the robot using the included visual programming tool [3], [12],[18].

Furthermore, the RCX can be programmed using other programming environments, like the brickOS used for the prototype described in this thesis. brickOS is an operating system designed for the RCX brick. It provides a C/C++ programming environment to develop and compile new programs, and the necessary utilities to download the compiled programs to the RCX [5]. The functions provided in the brickOS library can be divided into three categories, output, input, and program control. As indicated by the name the output functions interact with the outside environment through controlling motors and to the user through a LCD display on the RCX. The input functions collect and interpret information from the outside world through sensors and the buttons placed on the RCX. The program control functions provide control flow like sleep functions and threading.

1.20.3 Sony AIBO

AIBO is an autonomous entertainment robot developed and produced by Sony. It was developed to encourage human and robot interaction. AIBO can hear, see, feel, and walk, and has the ability to learn from new experiences [22]. It has emotions and depending on its mood it will respond differently to stimuli. You can affect AIBO’s mood by speaking to it, stroke it or play with it. Over time, AIBO’s personality will change depending on the relationship it has with you.

The software controlling AIBO resides on a memory stick, which can be replaced and reprogrammed. Sony recommends using the OPEN-R SDK [59] to program software to AIBO. OPEN-R SDK is a development environment based on gcc (C++), and includes libraries and tools to develop applications for AIBO.
1.21 Requirements

These requirements are the requirements for the prototype in [37]. The list given below is the resulting revised requirements after performing preliminary HazOp and Security HazOp. The functional requirements are prefixed with an ‘F’ for functional requirement, while the non-functional requirements are prefixed with an ‘N’ for non-functional requirement.

F1. The LEGO cutting robot shall
   F1.1. stop operating if
         i. the Off button in the entrance area is pressed.
         ii. the “stop cutting” button in the control window on the PC controller is pressed.
         iii. the connection between the AIBO and the PC controller is lost.
         iv. the connection between the PC controller and the LEGO system is lost.
         v. the gate is opened.
   F1.2. start operating if
         i. the On button in the entrance area is pressed by an authorized person
         ii. the “Start cutting” button in the control window on the PC controller is pressed.

F2. The RCX of the LEGO system shall
   F2.1. communicate with the PC controller through IR signals.
   F2.2. control
         i. the lights
         ii. the cutting robot
         iii. the gate
         iv. the gate lock
   F2.3. interpret data from
         i. the On and Off buttons
         ii. the sensors on the gate (touch sensors which are used to determine when the gate is opened or closed)
   F2.4. inform the PC controller about state changes of
         i. the cutting robot (started/stopped)
         ii. the gate (opened/closed)

F3. The AIBO shall
   F3.1. communicate with the PC controller through wireless LAN
   F3.2. search for green man and pink ball.
   F3.3. notify the PC controller if
         i. it detects an authorized person (green LEGO man) or an intruder (pink ball) entering the entrance area.
         ii. it detects an authorized person or an intruder leaving the entrance area.

F4. The PC controller shall
   F4.1. communicate with the LEGO system through IR signals.
   F4.2. communicate with AIBO through wireless LAN.
   F4.3. send a start signal to the LEGO system when the start button in the control window is pressed.
   F4.4. send a stop signal to the LEGO system when
         i. the stop button in the control window is pressed.
         ii. the connection to the AIBO is lost.
iii the AIBO becomes unavailable.

F4.5. send a "close gate" signal to the LEGO system when
i   the connection to the AIBO is lost.

F4.6. display status information when
i   communication to AIBO is lost.
ii  communication to LEGO system is lost.
iii the cutting robot has started.
iv  the cutting robot has stopped.
v   intruder is detected in the entrance area.
vi  authorized person is detected in the entrance area.
vii the cutting robot can not be started.

F5. The gate shall
F5.1. unlock if an authorized person (green LEGO man) enters the entrance area.
F5.2. open if the gate wheel in the entrance area is turned.
F5.3. close if the authorized person leaves the entrance area without closing the gate.
F5.4. lock if the authorized person leaves the entrance area.
F5.5. close if it is open and the On button in the entrance area is pressed.
F5.6. close and lock if the connection between the AIBO and the PC controller is lost.
F5.7. close and lock if the connection between the PC controller and the LEGO system is lost.

N1. Safety requirements
N1.1. The Cutting Robot shall not
i   be operating if a person is inside the safety zone.
ii  be operating if the control system is not activated and available.
iii be operating if one or several of the assets of the control system stop working properly.

N1.2. The Gate shall not
i   be open if the industry robot is operating.
ii  be unlocked when an intruder is in the entrance area.
iii be unlocked when authorized staff is not in the entrance area or is not inside the safety zone.

N1.3. The Control System shall not
i   be initiated when people are inside the safety zone (if so, the system will not know that the zone is not empty, and the industry robot may be started).

N2. Security requirements
N2.1. Confidentiality
i   Only authorized personnel shall have access to the control window on the PC controller, and only administrators should have administrative access to the PC controller. Administrative access includes access to the source code and to the control parameters of the system.

N2.2. Integrity
i   The data exchange between the AIBO and the PC controller shall not be corrupted or altered.
ii  The data exchange between the PC controller and the LEGO system shall not be corrupted or altered.
iii The AIBO shall not be corrupted or altered.
iv  The LEGO system shall not be corrupted or altered.
v  The PC controller shall not be corrupted or altered.
N2.3. Availability

i  The control system shall be available 100% of the time the system is in running mode. However, we allow maintenance as long as the cutting robot is stopped while the control system is unavailable.

ii  Data exchanged between AIBO and PC controller shall not be lost.

iii Data exchanged between PC controller and LEGO system shall not be lost.

N2.4. Authenticity

i  Authorized users must log on to the system using user name and password (weak authentication).

1.22 Architecture and design of the original prototype

This section describes the architecture and design of the prototype developed by Sorby in [37]. The architecture of the prototype is not well documented and the description given here represents the architecture as far as we comprehend it from the documentation given. Furthermore, we will in the following description focus mainly on architecture and design choices that affect modifiability.

The prototype is decomposed into three sub systems; the AIBO, the PC controller and the LEGO system, as shown in Figure 0-2.

![Subsystems of the original prototype](uml.png)

Figure 0-2: Subsystems of the original prototype. Key: UML

The main functionality for the AIBO is to search for authorized and unauthorized personnel and report its findings to the PC Controller subsystem. The PC controller represents the communication link between the AIBO and the LEGO system, in addition to providing a user interface for an administrator. The LEGO system is responsible for controlling the physical units in the factory, such as the cutting robot, the gate and the lock on the gate.
1.22.1 Design of AIBO

The AIBO subsystem consists of multiple objects, each of which is implemented by a separate class. The objects are shown in Figure 0-3 and have the following functionality:

- PowerMonitor monitors the battery power level, the Pause button, and related events, reports the status to the console, and shuts down AIBO when appropriate.
- MovingHead2 Moves the head of the surveillance robot that contains the camera in order to monitor the whole entrance area permanently.
- MovingLegs2 Initializes AIBO into a standard monitoring pose by stretching all and its rear legs after the start-up.
- LostFoundSound notifies the user by playing an alert sound each time the authorized staff or intruder is detected in the visible frame.
- BallTrackingHead, is the main client object that gives commands to the server objects named above and continuously processes the vision image from AIBO's camera. Whenever the amount of the specified colour (either green - for the staff, or pink for the intruder) in the camera view exceeds the threshold, an event is printed on the telnet console and thus becomes available for PC controller. In addition, the robot moves its head so that the detected area remains in the centre of the cameras focus.
- OVirtualRobot is not part of the BallTrackingHead application, but serves as an interface between the application and the OPEN-R.
Objects communicate exclusively by sending messages according to a predefined interface scheme. The following interfaces are defined (shown in Figure 0-3):

- **MovingHead Interface** between BallTrackingHead and MovingHead2 for sending control commands for moving AIBO's head.
- **MovingLegs Interface** between BallTrackingHead and MovingLegs2 for sending control commands for moving AIBO's legs.
- **LostFoundSound Interface** between BallTrackingHead and LostFoundSound for sending control commands for playing sounds on the occurrence of events.
- **RbkImageSensor Interface** between BallTrackingHead and OVirtualRobot for retrieving the image from AIBO's camera view.
- **Sensor Interface** between BallTrackingHead and OVirtualRobot for retrieving information from AIBO's sensors (for computing the head-movement commands for centring the focus).
- **Joint Interface** between BallTrackingHead and OVirtualRobot for controlling the position of head when focusing on the intruder or staff.
- **Move Interface** between MovingLegs2 and OVirtualRobot for controlling the legs.
- Play Interface between LostFoundSound and OVirtualRobot for playing the Wave files.
- Move Interface between MovingHead2 and OVirtualRobot for controlling the position of swinging head (Note: the OPEN-R allows to define two different services with the same name).

### 1.22.2 Design of PC Controller

PC Controller is comprised of three classes, Talking CBall3App, and Ball3Dlg as shown in Figure 0-4. The only responsibility of the CBall3App is to initiate three threads that run simultaneously on the PC. The three threads are the thread for communicating with the AIBO, the thread for communicating with the LEGO system, and the thread for the user interface. The two communication threads are implemented in the Talking class, while the thread for the user interface is implemented in the Ball3Dlg.

The Talking class is responsible for communicating with the two other subsystems in the prototype. Telnet is used to communicate with the AIBO. The AIBO sends messages to the Telnet application and the Talking class loads the Telnet log to detect messages sent from the AIBO, such as found or lost intruder or authorized personnel. These messages are sent to the Ball3Dlg, which prints them in the user interface dialog. Furthermore, the messages are forwarded to the LEGO system. To communicate with the LEGO system the Talking class uses an IR application and a separate thread is responsible for sending and receiving messages.

![Figure 0-4: Design of the PC Controller Key: UML.](image-url)
1.22.3 Design of LEGO system

The LEGO system is implemented in C, and there are therefore no classes only modules. The two modules that comprise the LEGO system is the Factory controller and the PostOffice communication module. The PostOffice module is responsible for sending and receiving messages from the PC controller subsystem, through an IR link. As the name indicates the Factory controller is responsible for controlling the factory units. It is implemented as a finite state automation. The state transitions between the possible states: wait_start, stopped, stopped_staff, stopped_staff_in, running, running_staff, disconnected trigger various actions, such as operating the actuators (gate, gate lock, motor to cutting robot), or sending/receiving messages to/from PC controller.

![Diagram of LEGO system](image-url)

Figure 0.5: Design of the LEGO system: Key UML
DEVELOPMENT

The prototype described in this report was developed using a development process based on the IEC 61508 [26] standard and the CORAS integrated risk management and system development process for security critical systems [10]. The development process was developed by Karine Sorby in [37]. The process is a stepwise iterative and incremental development process for developing security-safety critical systems. In our work we focus on architecture and design in developing the prototype. Therefore we have added an architecture and design phase, and an architecture and design analysis phase. The development process is shown in Figure 0-1.

![Diagram of the development process]

Figure 0-1: The development process used in this report

The process starts with a concept definition and an overall scope definition of the system. The output of these steps is the system description and a general idea of the functional and quality requirements of system. The documentation from the first steps is
used when performing the preliminary hazard analysis (PHA) in step 3. The benefit of performing a PHA at an early stage is twofold. First, potential hazards can be identified early, reducing cost and the difficulty of handling them. Second, the PHA aids in finding safety requirements for the system [37]. Step 4 focuses on risk management. This step is comprised of seven sub steps. Safety and security requirements are elicited in step 4.1 and 4.2. Security threats are identified in step 4.3, while risks concerning safety are analysed and evaluated in step 4.4 and 4.5. After evaluating the risks, a choice is made to accept or not to accept the risks. If the risks are unacceptable they have to be treated, which is step 4.6, and then the system description and requirements are updated before a new iteration in step 4 begins. When only acceptable risks remain, we start with architecture and design of the system in step 5. In step 6 we analyse the architecture and design in order to verify that the most important quality attributes are met. The design documentation will be input to the implementation in step 7, and finally the system is tested and validated in step 8. In our work we will not go through phases 7, Implementation or 8, Testing and validation. The main reason for this is that we will focus on modifiability concerns, which will be evaluated through expert judgement of the architecture and design. The expert judgement, which only analyse the system for modifiability in this context will be conducted in step 6, Architecture and design analysis and described in section 0. The development process is described in more detail in the following sections.

1.23 Concept and Scope

This phase will give an overall description of what the system shall do and how the concepts related to the system are related to each other. It also defines the system scope (see Figure 0.2) and how the environment relates to the system concepts (see Figure 0.3). Furthermore, a proposal of the functional requirements and quality requirements are made in this phase. The requirements are listed in section 1.25.

The system constructed in our work is a prototype of a factory system. The production unit of the factory is a cutting robot. The main task for the prototype we are making is to ensure that no one gets injured or killed by the cutting robot. The prototype is a security-safety critical system that is comprised of a security critical subsystem and a safety critical subsystem. The safety critical subsystem is comprised of a cutting robot placed within a safety zone sealed off by a gate. While the cutting robot is operational it imposes a threat to
persons inside the safety zone, and the access to the safety zone needs to be controlled and restricted to authorized personnel. A security critical control system is responsible for preventing unauthorized access to the safety zone and for stopping the cutting robot when authorized personnel enters the safety zone. Security attacks in the control system with the purpose of altering its behaviour or technical failures represent a safety threat. It is the responsibility of the control system to ensure that the safety is not compromised.

As shown in Figure 0.2, the control system is divided into three subsystems according to three responsibility areas, a protection system, an authorization mechanism, and a control application. The protection system is responsible for starting and stopping the motor of the cutting robot, opening and closing the gate, and locking and unlocking the gate. It is important that the cutting robot is stopped whenever the gate is open. Outside the safety zone an emergency stop button (Off button) and a start button to the cutting robot (On button) is placed. Anyone can stop the cutting robot with the emergency button, but the start button is enabled only for authorized personnel. The protection system will report its different states (i.e. cutting robot running, gate closed and locked) to a
control application (see below). A LEGO Mindstorms robot (RCX) is used to coordinate and execute the aforementioned responsibilities through sensors and actuators, and will communicate using infrared signals. See Figure 0-3 for the relationships between the concepts in the Security-Safety critical system.

![Diagram](image-url)

Figure 0-3: The relationships between the elements in the system, and between the system and the environment.

The authorization mechanism is represented by a Sony AIBO robot. Its task is to monitor the entrance area to the safety zone. The Sony AIBO will identify persons within the entrance area as authorized or unauthorized personnel and report to a control application (see below) when these persons enters or leaves the entrance area. The communication between the Sony AIBO and the control application is conducted through a wireless LAN.

The main part of the control system is the control application which is responsible for the overall state of the system. That is, the control application will have information of the
state of the protection system and information of any authorized or unauthorized personnel in the entrance area. Based on this information the control application informs the protection system to allow access to the safety zone.

1.24 Preliminary Hazard Analysis (PHA)

The PHA is performed as the third step in the development process. At this stage there has not been made any explicit choices of how the system should perform its tasks, but rather what the system should be able to do (the functional requirements) together with some technology requirements (hardware and software that already has been chosen by e.g. the organization). Based on these requirements and the system description the PHA uncovers possible hazards in the system. The resulting list of hazards is used in the safety requirements elicitation and in the software architecture phase.

Our PHA is based on the one performed by Karine Sorby in [38]. We will use the same identified hazards, consequences, and causes in order to show that the new prototype maintain at least the same safety level. The countermeasures we propose are listed in Table 0-1. Some of the identified causes to hazards are already taken into consideration through the requirements. In the proposed countermeasures for these causes we will refer to the appropriate requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Hazard</th>
<th>Consequences</th>
<th>Causes</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cutting robot does not stop when person enters the safety zone</td>
<td>Possible death or injury of person</td>
<td>AIBO fails to detect person</td>
<td>The gate is only unlocked if AIBO successfully detects an authorized person. (F4.1 and F4.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>An alert from the AIBO of persons entering safety zone does not reach the LEGO system</td>
<td>The gate is only unlocked if AIBO successfully detects an authorized person. (F4.1 and F4.4) In addition the cutting robot will stop if the communication between the AIBO and LEGO system becomes unavailable. (F1.1.iii, F1.1.iv, F4.6, and F4.7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Natural disaster or sabotage makes the AIBO or the Control Application unavailable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hardware failure in the LEGO system</td>
<td>Check battery status and actuator status.</td>
</tr>
</tbody>
</table>

Table 0-1: Preliminary Hazard Analysis (PHA) results
<table>
<thead>
<tr>
<th></th>
<th>Scenario</th>
<th>Risk</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Cutting robot starts cutting when a person is inside the safety zone</td>
<td>Possible death or injury of person</td>
<td>Prevent the cutting robot to be started from the control application</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Check battery status and actuator status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code reviews</td>
</tr>
<tr>
<td>3</td>
<td>Cutting robot is not stopped when Off button is pressed</td>
<td>If the system also fails to stop the cutting robot when someone enters the safety zone: Possible death and injury</td>
<td>Check battery status and actuator status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code reviews</td>
</tr>
<tr>
<td>4</td>
<td>Gate is not closed when intruder enters the entrance area</td>
<td>Unauthorized person is allowed to enter the safety zone and can be injured if he is not trained to operate in such environments</td>
<td>The gate is only unlocked if AIBO successfully detects an authorized person. (F4.1 and F4.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The gate is only unlocked if AIBO successfully detects an authorized person. (F4.1 and F4.4) In addition the cutting robot will stop if the communication between the AIBO and LEGO system becomes unavailable. (F1.1.iii, F1.1.iv, F4.6, and F4.7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code reviews</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code reviews</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>A person gets injured by the gate</td>
<td>Injury to person</td>
<td>The gate closes to late and hits person entering the safety zone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exposed internal parts of the gate. Avoid unnecessary exposure of internal parts in the gate design</td>
</tr>
<tr>
<td>6</td>
<td>Person bumps into the cutting robot</td>
<td>Injury to person</td>
<td>Incautiousness of person working in the safety zone</td>
</tr>
<tr>
<td>7</td>
<td>Cutting blade falls off the cutting arm</td>
<td>Person is hit by blade: Possible death or injury</td>
<td>Defective blade or cutting arm and cutting arm stopped in upright position</td>
</tr>
</tbody>
</table>

1.25 Requirements

This section lists the requirements for the system. The requirements describe the functionality, in the functional requirements, and the quality attributes, in the quality requirements, provided by the system. These requirements are based on the final requirements for the prototype developed by Karine Sorby [37]. The requirements from [37] are listed in 1.21. We have rewritten these in order to separate design choices from system requirements. The main difference between the requirements is that we have extracted constraints to hardware put it a separate section. We have done this since our focus is on modifiability and this will allow us to use other hardware components without changing the system requirements if the limitations to resources change. Our requirements provide the same requirements to functionality and quality with two exceptions; the requirements to modifiability are new and the option to start the cutting robot from the control application is removed for safety reasons identified in the preliminary hazard analysis.
1.25.1 Constraints

This section describes the constraints to the prototype system. Although these constraints are part of the requirements specification, they are not considered to be necessary for the success of the system. They are rather limitations set by the limited amount of resources and the hardware available. The constraints are listed below and prefixed with a C for constraint.

C1. The authorization mechanism shall be a Sony AIBO
C2. The cutting robot shall be built in LEGO
C3. The gate to the safety zone shall be built in LEGO
C4. The safety zone surrounding the cutting robot shall be restricted by a wall built in LEGO and a gate
C5. The start and stop buttons placed at the entrance shall be built in LEGO
C6. A LEGO Mindstorm RCX shall control the cutting robot (start/stop), the gate (open/close and lock/unlock), and interpret the signals from the on and off buttons.
C7. The communication between the control application and the RCX shall be through infrared signals.
C8. The communication between the control application and the AIBO shall be through a wireless LAN.

1.25.2 Functional requirements

This section lists the functional requirements for the prototype. These requirements are prefixed with an F for functional requirements.

F1. The cutting robot shall
   F1.1. stop operating if
      i   the Off button in the entrance area is pressed
      ii  the control application sends a “stop cutting” signal
      iii the communication between the authorization mechanism and the controller application is lost (replicated in F3.1.ii)
      iv  the communication between the control application and the cutting robot is lost
      v   the gate is opened
   F1.2. start operating if
      i   the On button in the entrance area is pressed by an authorized person
F2. The authorization mechanism shall
   F2.1. monitor the entrance area for personnel (search for green man and pink ball)
   F2.2. search for green LEGO men (authorized personnel) or pink ball (unauthorized)
   F2.3. notify the control application if
      i   it detects an authorized person (green LEGO man) or an unauthorized person (pink ball) entering the entrance area
      ii  it detects an authorized person or intruder leaving the entrance area
F3. The control application shall
   F3.1. send a stop signal to the cutting robot
      i   on user request
ii the communication between the authorization mechanism and the controller application is lost (replicated in F1.1 iii)
iii the authorization mechanism becomes unavailable

F3.2. send a “closed and lock gate ” signal to the gate when the communication with the authorization mechanism lost

F3.3. display status information when
i communication with authorization mechanism is lost
ii communication to factory system is lost
iii The cutting robot has started
iv The cutting robot has stopped
v Intruder is detected in the entrance area.
vi Authorized person is detected in entrance area
vii The cutting robot can not be started
viii The gate is opened
ix The gate is closed

F4. The gate shall
F4.1. unlock if an authorized person (green LEGO man) enters the entrance area
F4.2. open if the gate wheel in the entrance area is turned
F4.3. close if the authorized person leaves the entrance area without closing the gate
F4.4. lock if it authorized person leaves the entrance area
F4.5. close if it is open and the On button in the entrance area is pressed
F4.6. close and lock if the communication between the authorization mechanism and the control application is lost
F4.7. close and lock if the communication between the control application and the gate is lost

1.25.3 Quality requirements

This section lists the quality requirements for the system. These requirements are based on the final requirements for the prototype developed in [37]. The quality requirements are prefixed with an N for non-functional requirements.

N1. Safety requirements
N1.1. The cutting robot shall not
i be operating if a person is inside the safety zone
ii be operating if the control application is not activated and available
iii be operating if the authorization mechanism is unavailable
iv be operating if the gate is open

N1.2. The gate shall not
i be unlocked when an intruder is in the entrance area
ii be unlocked when authorized staff is not in the entrance area

N1.3. The control application shall not
i start the cutting robot

N2. Security requirements
N2.1. Confidentiality
i only administrators shall be able to operate the control application
ii source code and control parameters shall only be available to system administrators
N2.2. Integrity
   i  the data exchange between the authentication mechanism and the control
       application shall not be corrupted or altered
   ii the data exchange between the control application and the entrance gate shall
       not be corrupted or altered
   iii the data exchange between the control application and the cutting robot shall
       not be corrupted or altered
   iv the authentication mechanism shall not be corrupted or altered
   v mechanisms that control the entrance gate and the cutting robot shall not be
       corrupted or altered
   vi the control application shall not be corrupted or altered

N2.3. Availability
   i  the control application shall be available 100% of the time the cutting robot is
       running
   ii data exchanged between the authentication mechanism and control application
       shall not be lost
   iii the data exchange between the control application and the entrance gate shall
       not be lost
   iv the data exchange between the control system and the cutting robot shall not
       be lost

N2.4. Authenticity
   i to operate the control application, personnel has to identify themselves as
       authorized users

N3. Modifiability
N3.1. It shall be possible to change or exchange the user interface to the control
   application, without changing any other elements in the system
N3.2. It shall be possible to change authorization mechanisms at the entrance gate,
   without changing any other elements in the system. Possible changes are:
   i  Add more monitoring units (AIBO)
   ii Add communication between monitoring units (AIBO)
   iii Add functionality for the monitoring units (AIBO) to follow authorized
       personnel or intruders in a defined area (entrance area or safety zone)
   iv Replace the monitoring unit with a different authorizing mechanism, e.g. key
      card at the gate.
N3.3. It shall be possible to change the communication interface between the
   authorization mechanism and the control application from telnet to CORBA.
   There shall be no need to change system elements responsible for the
   functionality of the system. It should be possible to implement this change in 1
   person-week
N3.4. It shall be possible to change the communication interface between the
   mechanisms controlling the gate and the cutting robot, and the control
   application from IR to Bluetooth. There shall be no need to change system
   elements responsible for the functionality of the system. It should be possible
   to integrate this change in 1 person-week.
N3.5. Adding, removing or changing a functional requirement shall not impose
   changes to elements of the system implementing other functional
   requirements. (Since the prototype is used in a research area it is most likely
   that the prototype can be altered so that the functionality is changed totally. In
   addition it should be possible to use the same architecture in a real world
   version of the system, e.g. with a real cutting robot)
1.25.4 Use cases

<table>
<thead>
<tr>
<th>Actor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>This actor is the system administrator. It is his responsibility that the system is operational. He is authorized to operate the control application.</td>
</tr>
<tr>
<td>Authorized personnel</td>
<td>This actor is authorized to enter the safety zone.</td>
</tr>
<tr>
<td>Unauthorized personnel</td>
<td>This actor is personnel not allowed into the safety zone.</td>
</tr>
</tbody>
</table>

![UML Diagram](image)

Figure 0.4 Key UML.

<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Start cutting robot (F1.2.i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Authorized personnel</td>
</tr>
<tr>
<td>Summary:</td>
<td>The actor starts the cutting robot from the On button placed in the entrance area</td>
</tr>
<tr>
<td>Precondition:</td>
<td>• There are not any persons in the safety zone. (N1.1.i)</td>
</tr>
<tr>
<td></td>
<td>• The control application is activated and available (N1.1.ii and F1.1.iv)</td>
</tr>
<tr>
<td></td>
<td>• The authorization mechanism is activated and available (N1.1.iii, and F1.1.iii, and F3.1.iii)</td>
</tr>
<tr>
<td></td>
<td>• The gate is closed (F1.1.v and N1.1.iv)</td>
</tr>
<tr>
<td>Basic Course of Events:</td>
<td>1. An Authorized person enters the entrance area</td>
</tr>
<tr>
<td></td>
<td>2. The system identifies the person as Authorized personnel (F2.1, F2.2, and F2.3.i)</td>
</tr>
<tr>
<td></td>
<td>3. The Authorized person presses the On button</td>
</tr>
<tr>
<td></td>
<td>4. The system starts the cutting robot</td>
</tr>
<tr>
<td>Alternative Paths:</td>
<td>None</td>
</tr>
<tr>
<td>Exception Paths:</td>
<td>None</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>Post condition:</td>
<td>The cutting robot is operating</td>
</tr>
<tr>
<td>Author:</td>
<td>John Inge Hervik</td>
</tr>
<tr>
<td>Date:</td>
<td>25.03.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Stop cutting robot (F1.1.i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Authorized personnel, Unauthorized personnel</td>
</tr>
<tr>
<td>Summary:</td>
<td>The actor stops the cutting robot from the Off button placed in the entrance area</td>
</tr>
<tr>
<td>Precondition:</td>
<td>• The cutting robot is operating</td>
</tr>
<tr>
<td>Basic Course of Events:</td>
<td>1. The actor presses the Off button in the entrance area 2. The system stops the cutting robot</td>
</tr>
<tr>
<td>Alternative Paths:</td>
<td>None</td>
</tr>
<tr>
<td>Exception Paths:</td>
<td>None</td>
</tr>
<tr>
<td>Post condition:</td>
<td>The cutting robot is not operating</td>
</tr>
<tr>
<td>Author:</td>
<td>John Inge Hervik</td>
</tr>
<tr>
<td>Date:</td>
<td>25.03.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Remote stop cutting robot (F1.1.ii and F3.1.i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Summary:</td>
<td>The actor stops the cutting robot from the control application</td>
</tr>
</tbody>
</table>
| Precondition:     | • The cutting robot is operating  
                  | • The administrator is identified as an authorized user to the control application (N2.1.i and N2.4.i) |
| Basic Course of Events: | 1. The administrator submits a request to stop the cutting robot to the control application  
                  | 2. The system stops the cutting robot |
| Alternative Paths: | None |
| Exception Paths:  | None |
| Post condition:   | The cutting robot is not operating |
| Author:           | John Inge Hervik |
| Date:             | 25.03.04 |

<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Open gate (F4.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Authorized personnel</td>
</tr>
<tr>
<td>Summary:</td>
<td>An authorized person opens the gate to the safety zone</td>
</tr>
</tbody>
</table>
| Precondition:     | • The control application is activated and available (N1.1.ii and F1.1.iv)  
                  | • The authorization mechanism is activated and available (N1.1.iii, and F1.1.iii, and F3.1.iii)  
                  | • The gate is closed (F1.1.v and N1.1.iv) |
| Basic Course of Events: | 1. An Authorized person enters the entrance area  
<pre><code>              | 2. The system identifies the person as Authorized personnel |
</code></pre>
<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Close gate (F4.3 and F4.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Authorized personnel</td>
</tr>
<tr>
<td>Summary:</td>
<td>An authorized person closes the gate to the safety zone</td>
</tr>
</tbody>
</table>
| Precondition: | • The gate is open  
|             | • An Authorized person is in the entrance area or the safety zone |
| Basic Course of Events: | 1. The Authorized person starts closing the gate form the entrance area.  
|            | 2. The system closes the gate (F4.5)  
|            | 3. The Authorized person leaves the entrance area  
|            | 4. The system locks the gate (F2.3.ii, F4.4, and N1.2.ii) |
| Alternative Paths: | In step 1 the authorized person does not start losing the gate. The system closes the gate when the Authorized person leaves the entrance area (F4.3) |
| Exception Paths: | None |
| Post condition: | • The gate is closed  
|                | • The gate is locked |
| Author: | John Inge Hervik |
| Date: | 25.03.04 |
1.25.5 Legal states

This section describes the legal states of the system. The set of legal states define the system states that will ensure safety for its environment. Figure 0-5 shows an overview of legal states in the system and the actions that impose state changes. When the system is in System Off state, all the elements of the system are shut down. As indicated by the note attached to this state, the cutting robot must be turned off and the gate must be locked in this state. After starting all the system elements the system is in the state “System Available”. This state is the operational state and represents the state the system will be in under normal conditions. The state “System Available” is decomposed into five operational states as shown in Figure 0-6. If one or more elements in the system stop, the system goes into the state “Unavailable”. As with the state “System Off”, the cutting robot must be shut off and the gate must be locked.

Since the system is a safety critical system it has a fail safe state. When the cutting robot is turned off and the gate is locked, the system is said to be in a fail safe state. This means that the system is unable to cause any safety risk.
1.26 Risk Management

This phase of the development is comprised of seven sub steps as shown in Figure 0-7. Safety and security requirements are elicited in step 4.1 and 4.2 (listed in section 1.25.3). Security threats are identified in step 4.3, while risks concerning safety are analysed and evaluated in step 4.4 and 4.5. After evaluating the risks, a decision to accept or not to accept the various risks is made. If a risk is unacceptable it needs to be treated, which is done in step 4.6. After this the system description and requirements are updated before the phase iterates. When all remaining risks are considered acceptable the phase completes.
1.26.1 Identify Security Threats

In step 4.3 of the development process, the security threats to the system are identified using a Security-HazOp [10]. The Security-HazOp is based on traditional HazOp but adjusted to handle security concerns in a safety context [55]. It is performed as a traditional HazOp, using guidewords and attributes in a brainstorm activity. However, the guidewords are adapted to the security domain and divided into pre-guidewords and post-guidewords. Thus, the structure of an expression in a Security-HazOp is

“Pre-guideword Attribute of Component due to Post-Guideword.”

[55] proposes basic guidewords and attributes to identify security-threats, shown in Table 0-2. The proposed attributes in the security-HazOp are confidentiality, integrity, and availability. As shown in Table 0-2, the proposed attributes are disclosure, manipulation, and denial, which is the negation of the security attributes confidentiality, integrity, and availability.
In [37] a security-HazOp on the prototype was conducted. [37] identified the three subsystems the AIBO, the RCX, the control application, and the communication links (WLAN and IR) as the most security critical components of the system. In our work we seek to maintain at least the same level of security, and we therefore use the set of guidewords, attributes, and components as [37] (see Appendix A) We found that most of the security threats and their countermeasures are relevant for the new prototype as well. However, some security threats are already dealt with in the requirements, and a few others are dealt with by different design choices. The differences are shown in Table 0-4.

<table>
<thead>
<tr>
<th>Pre-guideword</th>
<th>Attribute</th>
<th>Post-guideword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliberate</td>
<td>Disclosure of COMPONENT due to Insider</td>
<td></td>
</tr>
<tr>
<td>Unintentional</td>
<td>Manipulation Outsider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denial Technical failure</td>
<td></td>
</tr>
</tbody>
</table>

In Table 0-3 the attributes, the guidewords, and the components used in the security-HazOp are listed. The pre-guidewords are used to cover the two possibilities that an attack can be deliberate or unintentional, while the post-guidewords describe the various agents who can be responsible for an attack on the system. The list of attributes is an extended list of the proposed list from [55]. The added attributes fabrication and masquerade are negations of the security attribute authenticity, while the delay attribute is a negation of the security attribute availability.

The results from the security-HazOp identified possible breaches to the four security attributes of concern [37]. First, breaches to confidentiality included that persons may get...
unauthorized access to messages transmitted through the WLAN and the IR, to the data and settings of the AIBO, the LEGO system, and the control application. Second, threats to integrity included modifications in transmitted messages and manipulation of AIBO, the LEGO system, and the control application in such a way that these ignore important messages or fail to send important messages. Third, the availability is threatened if messages between AIBO and the control application or between the RCX and the control application are lost or delayed. Finally, if false messages are introduced into the system or an agent masquerades as the AIBO or the control application the authenticity attribute is compromised. The results from security-HazOp in [37] with consequences, causes, and countermeasures can be found in appendix A. We found that these results largely also apply to the new prototype. The differences found are given in Table 0-4.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Security threat</th>
<th>Consequences</th>
<th>Causes</th>
<th>New Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliberate delay of alerts/messages from AIBO (From table C.4)</td>
<td>The Control Application receives the alert messages from AIBO too late</td>
<td>The gate may close too late, and the cutting robot may stop too late if someone enters the safety zone. Possible death or injury of person</td>
<td>Denial of service attack</td>
<td>The gate is always closed unless the AIBO notifies the LEGO system of an authorized person in the entrance area. See requirement N1.2.ii.</td>
</tr>
<tr>
<td>Unintentional delay of alerts/messages from AIBO (From table C.4)</td>
<td></td>
<td></td>
<td>Technical failure</td>
<td></td>
</tr>
<tr>
<td>Deliberate denial/delay of messages in the WLAN communication channel (From table C.5)</td>
<td>An “Intruder alert” from AIBO to the Control Application is lost or delayed</td>
<td>The gate is not closed or closes to late when a person enters the entrance area. Possible injury to person.</td>
<td>Interruption of signals/Interference of radio signals by other senders/Transmitted data destroyed by other electrical device. Errors in configuration settings of network settings, WLAN card removed from PC or AIBO</td>
<td>The gate is always closed unless the AIBO notifies the LEGO system of an authorized person in the entrance area. See requirement N1.2.ii.</td>
</tr>
<tr>
<td></td>
<td>An “entered safety zone” signal from AIBO to the Control Application is lost or delayed</td>
<td>The cutting robot is not stopped or stops too late when a person enters the safety zone. Possible death or injury of person.</td>
<td></td>
<td>The LEGO system will stop the cutting robot if the gate opens. See requirement N1.1.iv.</td>
</tr>
<tr>
<td></td>
<td>The communication link becomes unavailable</td>
<td>The gate is not closed or closes to late when a person enters the</td>
<td></td>
<td>The gate is always closed unless the AIBO notifies the LEGO system of an</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>system of an</td>
</tr>
</tbody>
</table>
### Deliberate fabrication/masquerade of AIBO (From table C.6)

<table>
<thead>
<tr>
<th>Unauthorized user/component acts as AIBO and fabricates messages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerade AIBO sends messages that staff or intruder is not in entrance area or safety zone any longer when they still are present. Possible death or injury to person.</td>
</tr>
<tr>
<td>Masquerade AIBO has taken the role as the original AIBO</td>
</tr>
<tr>
<td>The administrator connects the Control Application to the AIBO from the Control Application. Thus, the administrator has control (and the responsibility) over the connection. See section 1.28.2.</td>
</tr>
</tbody>
</table>

Deliberate fabrication/masquerade of Control Application (From table C.6)

<table>
<thead>
<tr>
<th>Unauthorized user/component acts as Control Application and fabricates messages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerade Control Application sends start cutting robot (or open gate) message to LEGO system. Possible death or injury to person.</td>
</tr>
<tr>
<td>Masquerade Control Application has taken the role as the original Control Application</td>
</tr>
<tr>
<td>The Control Application, the LEGO system, and the Communication between them are physical shielded. See the Security view section 1.27.5</td>
</tr>
</tbody>
</table>

### 1.26.2 Analyze Risk

The fourth step in the risk management phase is step 4.4 Analyse risk. It consists of identifying safety consequences (4.4.1), identifying likelihood of a risk (4.4.2), and estimating the level of risk (4.4.3). In step 4.4.1 the safety consequences are identified for each security threat found in the security-HazOp (which can be found in the security-HazOp tables in appendix A and in section 1.26.1). For each safety consequence a likelihood value is estimated in step 4.4.2. This likelihood value is a representation of the likelihood that a security threat will lead to a safety consequence. For the identification of safety consequences and likelihood, value tables adopted from the UK MoD standard 00-56 [45] where used. The results from step 4.4.1 and 4.4.2 are used to estimate the level of risk in step 4.4.3. The classification used to estimate the level of risk is defined in ISO 61508 [25]. [25] defines four classes of risk and describes the relationship between these
classes and the severity and frequency of the hazards. The security-HazOp tables with consequence value, likelihood, and classification level are found in Appendix B.

1.26.3 Evaluate Risk

After assigning the security threats and their safety consequences to appropriate risk classes, the risks are evaluated to decide whether the risks are acceptable or not. The risks are evaluated against acceptance criteria. In this prototype system risks assigned to risk classes I, II, and III are considered unacceptable and need to be handled in step 4.6 Treat risk. While the risks assigned to class IV are considered acceptable and will not be treated.

We have identified countermeasures for all risks with classification I, II, III. We have found alternative countermeasures for the security threats that differentiate from Sorbys security-HazOp, and we therefore argue that we have maintained the same security level as Sorby in [37].

1.26.4 Treat Risk

Risks not considered acceptable (risk classes I, II, and III) need to be treated. Four techniques can be applied to reduce risks in order to meet the acceptance criteria [46]. The first is to eliminate the hazard, which is the least expensive and most effective way to reduce risks. In the prototype hazard elimination can be achieved by removing the cutting robot from the system. Unfortunately, this will also remove the main purpose of the system as a cutting factory. Another technique is hazard reduction. The system can be designed for controllability, barriers can be included in the system, or reducing the failure rates of components. An example of hazard reduction in our prototype was to put up a gate to seal of the safety critical zone around the cutting robot. A third technique is to control the hazards in the system. This can be achieved by limiting the exposure time of the hazard by isolating the hazardous elements, or to build a protection system that takes the system to a safe state if a hazardous event occurs. Our prototype has such a protection system to reduce risk. Finally, risk can be reduced by minimizing the damage a hazard can afflict. Examples of damage minimization are to prepare emergency plans and escape routes or devises for limiting damage to equipment or people.
Most of the security threats identified in the security-Hazop are already dealt with in the requirements to the new prototype. The main reason for this is that [37] revised the requirements after the security-HazOp. The new prototype is based on these requirements, which result in eliminating several threats at an early stage. Two security threats needed a new countermeasure at design level, the deliberate fabrication/masquerade of AIBO and of control application, as shown in Table 0-4. For these two security threats the design choices described as countermeasures will more or less eliminate the possibility of these threats.

1.27 System Architecture

In our development process step 5, we build the architecture and design (design is described in section 1.28) of the system. This section describes the architecture of the system. The architecture is described using four views, a decomposition view, a safety view, a security view, and a deployment view, in addition to a set of scenarios organized in a utility tree [41]. The scenarios in the utility tree reflect the most important architectural drives of the system; security, safety, and modifiability. When creating the views for the system we use the Attribute Driven Design (ADD) method described in section 1.15, with the scenarios from the utility tree as input.

The decomposition view shows how the system is decomposed to accomplish its functional tasks. Thus, this view identifies the large-scale pieces of the system (such as subsystems and their components) and how they depend on each other [6]. The focus in this view is to design the architecture in order to meet the requirements to modifiability. The safety view is concerned with how the system obtains the required safety level defined by the safety requirements. This view describes the system components that provide the system safety and how their logically related. Another important aspect in this view is how the system deals with system failure that can affect the safety of the system [6]. The security view describes the system components that are concerned with security issues, and what measures are taken to ensure that the requirements to security are met. The deployment view focus on how the software architecture maps onto the physical devices of the system. The deployment view is often used to describe performance, availability, and security concerns [41]. In the new prototype performance is not an important concern and we are only interested in availability from a security perspective (which is included in the security
view). Consequently, we only use the deployment view to show how the subsystems are
mapped onto processing units and the communication mediums that are used between
them. The decomposition view is the primary view in the architectural description, and the
other views are based on it. The safety view shows how the modules from the
decomposition view concerned with safety interact in order to describe how safety is
provided. The same apply for security in the security view. The deployment view shows
how the subsystems from the decomposition view are allocated onto the hardware
resources.

In section 1.27.1 the scenarios from the utility tree are listed. All the scenarios refer to
one or more of the quality requirements in order to show the tracing from requirements to
architectural design. The following sections, section 1.27.2 through 1.27.5 describe the
structures of the system in the four different views. These views will refer to the scenarios
in the utility tree where choices are made in order to fulfil a scenario. By using these
references the architectural choices can be traced back to the quality requirements they are
meant to address.

1.27.1 Utility tree scenarios

A utility tree tries to capture the most important quality attributes of the system and
decomposes these down to the level of scenarios. Although a utility tree is usually used to
help in analyzing software architecture, as in ATAM (see section 1.16.1), we use it for
helping design the architecture too meet the quality requirements and to help trace design
choices to the requirements. A utility tree scenario is defined as a short statement
describing an interaction of one stakeholder with the system [57]. Scenarios should be
concrete and unambiguous. The scenarios are given a qualitative rating of their importance
related to the success of the system, and a qualitative rating describing the difficulty of
fulfilling them. This rating is set in parenthesis before the scenario description. A rating of
(H, M) will signify a scenario that is of high importance for the success of the system and
of medium difficulty. Below we have listed the scenarios in the utility tree. The root node
of the three is the Utility of the system. It has three child nodes, security, safety, and
modifiability, which are the most important quality attributes to the system. The leaf nodes
of the tree are the scenarios describing the quality requirements.

Security
Availability

U1. (H, M) the control application shall be available 100% of the time the cutting robot is running (N2.3.i)

U2. (H, L) messages sent from the authentication mechanism to the control application shall be confirmed at arrival (N2.3.ii)

U3. (H, L) messages sent between the mechanism controlling the entrance gate and the control application shall be confirmed at arrival (N2.3.iii)

U4. (H, L) messages sent between the mechanism controlling the cutting and the control application shall be confirmed at arrival (N2.3.iv)

Integrity

U5. (H, H) the data sent between the authentication mechanism and the control application arrives intact and unaltered 100% of the time the cutting robot is active (N2.2.i)

U6. (H, H) the data sent between the entrance gate and the control application arrives intact and unaltered 100% of the time the cutting robot is active (N2.2.ii)

U7. (H, H) the data sent between the cutting robot and the control application arrives intact and unaltered 100% of the time the cutting robot is active (N2.2.iii)

U8. (H, I) the software of the authentication mechanism is unavailable to change 100% of the time the cutting robot is running (N2.3.iv)

U9. (H, I) the software of the mechanisms that control the entrance gate and the cutting robot is unavailable to change 100% of the time the cutting robot is running (N2.2.v)

U10. (H, I) the control application shall run on a computer system available only to system administrator (N2.2.vi)

Confidentiality

U11. (M, M) only administrators shall be able to operate the control application 100% of the time (N2.1.i)

U12. (L, L) source code and control parameters shall only be available to system administrators 100% of the time (N2.1.ii)

Authenticity

U13. (H, L) personnel shall identify themselves as authorized users before operating the control application (N2.4.i)

Safety

The cutting robot shall not be able to harm persons

U14. (H, L) an authorized person is inside the safety zone and the cutting robot does not operate (N1.1.i)

U15. (H, L) the control application is unavailable and the cutting robot is inactive and can not be started (N1.1.ii)

U16. (H, L) the authorization mechanism is unavailable and the cutting robot is inactive and can not be started (N1.1.iii)

U17. (H, L) the gate is open and the cutting robot is inactive and can not be started (N1.1.iv)
U18. (H, L) the system is for some reason unsure if there can be persons in the safety zone and the control application can not start the cutting robot (N1.3.i)

The gate shall restrict entrance to the safety zone
U19. (H, M) an intruder is in the entrance area and the gate is locked (N1.2.i)
U20. (H, L) an authorized person is not in the entrance area and the gate is locked (N1.2.ii)

**Modifiability**

Add functionality
U21. (M, M) more AIBO to monitor the entrance gate are added and enabled to communicate with each other. Changes to existing elements in the AIBO are completed in less than 1 person-week. (N3.2.i and N3.2.ii)
U22. (M, M) add functionality for the AIBO to follow intruders or authorized personnel in the entrance area and the safety zone. Changes to existing elements in the AIBO are completed in less than 1 person-week (N3.2.ii)

Improve usability
U23. (H, L) a new user interface to the control application replaces the old one no elements other than the user interface is affected (N3.1)

Replace elements
U24. (H, H) the communication interface between the control application and AIBO is changed from telnet to CORBA and elements responsible for functionality are not affected (N3.3)
U25. (H, M) replace the AIBO with a key card mechanism for authorization of personnel (N3.2.iv)
U26. (H, H) the communication interface between the control application and mechanism controlling the gate and the cutting robot is changed from IR to Bluetooth and elements responsible for functionality are not affected (N3.4)

**1.27.2 Decomposition view**

The primary architectural drive in this view is modifiability. The system is decomposed into three subsystems, the Authorization Mechanism, the Control Application, and the Factory System. The motivation of this decomposition is to separate the functionality, ensuring that one subsystem can be replaced as long as it provides the same interface (U25). The subsystems also encapsulate its internal behaviour so that internal changes in one subsystem do not require changes in other subsystems (U21, U22, and U23). As shown in Figure 0-8 the Control Application is dependent on the Authorization Mechanism and the Factory System in order to reason about the overall system state and to initiate the necessary state changes (see section 1.25.5). The Factory System is dependent on the Control Application, but only in respect to receive requests for state changes.
The responsibility of the Authorization Mechanism is to identify personnel in the entrance area as authorized or unauthorized and report its findings to the Control Application. The Authorization Mechanism will only identify personnel in terms of categories (i.e. green Lego man, pink ball), not make a decision of whether personnel are authorized or unauthorized to enter the safety zone. Since the authorization mechanism doesn’t need to reason about which personnel are allowed to enter the safety zone, rules for who are authorized and who are not can be modified in a single module; the System State controller in the Control Application. By separating the authorization rules from the functionality of identifying personnel we need to modify both the Find/Search/Identify module and the Control Application’s System State controller to make use of new categories of personnel.

The Authorization Mechanism will have no direct knowledge of the Control Application, but will provide an interface that respond to requests, called Authorization Interface. As shown in Figure 0-9 the Authorization Interface communicates with the Control Application while the Find/Search/Identify module is responsible for the functionality. Separating the responsibility for communication from the functionality of the Authorization Mechanism will ease the work related to changing the Authorization Interface.
The Factory system is responsible for controlling the cutting robot, the gate to the safety zone, and the on and off buttons placed at the entrance gate. Thus the Factory system is responsible for ensuring that cutting robot is only operating when it is supposed to, and that the gate is not opened when not allowed. This is ensured by having a factory system state controller equipped with rules for when the cutting robot is allowed to operate and through stimulus from the Control Application and the on and off buttons. If the system state rules are changed, the Factory State Controller will need to be modified. The state of the factory is reported to the Control Application at every state change, even if the Control Application initiated the state change. The Factory system will have no direct knowledge of the Control Application, but will provide an interface, the Factory Interface that responds to requests. The Factory interface is separated from the functionality in the Factory System, ensuring that the interface can be replaced without affecting modules responsible for functionality. As shown in Figure 0-10 there are separate modules to control the cutting robot, the gate, and the button sensors. This ensures that one unit can be replaced without affecting the other.

![Diagram of Factory System](image)

Figure 0-10: The Factory System. Key UML.

The Control Application is responsible for the overall system state. That is, it has the responsibility for ensuring that authorized personnel are given access to the safety zone and that the cutting robot is not operating when they are inside the safety zone. Furthermore, it is responsible for not granting access to unauthorized personnel. The Control Application depends on information from the Authorization Mechanism to grant access to authorized personnel, and informs the Factory System to unlock the gate if authorized personnel are in the entrance area. It has a system state controller with rules for what different system states are legal and for which category of personnel is authorized or
authorized to enter the safety zone. A user interface is provided to system administrator, where system state information is shown (i.e. Authorization Mechanism available, cutting robot running, gate locked). The user interface will also give the administrator the opportunity to remote stop the cutting robot.

1.27.3 Safety view

Safety is the primary architecture drive in this view. The systems shall provide safety for personnel in the factory, and all the three subsystems described in the decomposition view participate to provide this safety. The system is considered a success if only authorized personnel gain access to the cutting robot and if the cutting robot is not running when authorized personnel are in the safety zone. When the cutting robot is off and the gate is locked, the system is defined to be in a fail safe state. It is important to ensure that the system gets into this fail safe state in case of a system failure, such as loss of communication between the subsystems.

Under normal conditions (all three subsystems are operational and available) there are defined a set of legal states (see section 1.25.5). These rules say that it is only in the “system available state” the cutting robot is allowed to operate and the gate can be unlocked.

Two modules are responsible for ensuring that these rules are followed, the System State Controller and the Factory State Controller. The System State Controller in the Control Application will ensure that the overall system at all times is in a legal state (U14, U16, U18, U19, and U20). As shown in Figure 0-12 the System State controller uses the Find\Search\Identify personnel module, in the Authorization Mechanism, to get information of personnel at the entrance gate. It is the System State Controller module’s
responsibility to decide if the identified personnel are authorized or not. The System State Controller will also receive information of state changes (and thus have state status) in the cutting robot and the entrance gate from the Factory State Controller. Based on this information and the set of rules given, the System State Controller can initiate state changes in the Factory State Controller as necessary, such as unlock gate for authorized personnel or stop cutting robot if the Authorization Mechanism is no longer available. The System State Controller does not communicate directly with the Find\Search\Identify personnel or the Factory State Controller modules as indicated in the figure, but rather through the communication interfaces Authorisation Interface and Factory Interface as shown in the decomposition view.

![Diagram of system state controller and factory state controller](image)

Figure 0-12: Safety view. The System State controller doesn’t interact directly with the Factory State Controller or the Find\Search\Identify modules, but uses interfaces described in the decomposition view. The interface modules are left out to simplify the model. Key: UML

The Factory State Controller, in the Factory System, is the second module responsible for upholding the state rules. More precisely it is responsible for ensuring that the Factory System and its physical units comply with the state rules (U14, U15, and U17), such as not allowing the cutting robot to operate when the gate is open and to react correctly to stimuli from the start and stop buttons at the entrance. In addition the Factory State Controller receives requests to move from one state to another from the System State Controller (i.e. to unlock the gate or stop the cutting robot) and is responsible for carrying out the state change.

The Gate Controller and the Cutting Robot Controller are the modules controlling the gate and the cutting robot respectively. These two modules and the Button Sensor module function as drivers for the gate, the cutting robot, and the start and stop buttons.
1.27.4 Deployment view

The three sub systems described in the decomposition view are mapped onto three different devices. As shown in Figure 0-13 the Authorization Mechanism module is run on a Sony AIBO, the Control Application is run on a PC, and the Factory system is run on a LEGO Mindstorm RCX robot unit.

The three processing units, the AIBO, the PC, and the RCX are processing in parallel. This means that the three subsystems described in the decomposition view can run independently and processing their own data and communicate with each other when necessary.

![Deployment View Diagram](image_url)

Figure 0-13: Deployment view. Key: UML.

1.27.5 Security view

The primary architectural drive in this view is security. The security issues of importance in this system are those that can affect safety. Safety consequences of security breaches and security threats are further discussed in section 1.26 Risk Management. The requirements to security are divided into availability, integrity, confidentiality, and authenticity.

By availability we mean the property that ensures that authorized users have access to information and associated assets when required [30]. As defined in security requirement N2.3.i the Control Application shall be available when the cutting robot is running (U1).
This is not accomplished through redundancy or fault tolerance, but rather by shutting down the cutting robot when the control application is not available (see also U15). There are two reasons for not adding redundancy in this system. First, by introducing redundancy we also increase the security attack points (both the communication between redundant units and the redundant units themselves). Second, we do not have the recourses for redundancy and the loss of productivity of this prototype is not prioritized in the new prototype.

Messages sent from one module to another shall be confirmed to ensure that they arrive (U2, U3, and U4). As shown in Figure 0-14, the Control Application will have a specialized sub module of the communication module for each communication interface. This will ease the exchange or change of one communication interface without affecting the other (U23 and U26), and it will also open for taking extra security measures for some communication interface where needed. One such measure is to add measures to keep the integrity of messages between the Control Application and the Authorization Mechanism (U5).

Figure 0-14: Security view. Key: UML.

Data sent between the Control Application one side and the cutting robot and the entrance gate on the other side, is sent through an IR communication channel (see Figure 0-13). To prevent anyone to access and alter these messages or prevent messages from arriving intact the IR communication link is placed in a room only accessed by administrators (U6 and U7). This will also prevent the case where another RCX tries to pass for the original RCX and send false messages. As described in the deployment view the software for controlling the cutting robot and the entrance gate reside on the LEGO Mindstorms RCX unit, a programmed Sony AIBO represent the Authorization Mechanism, and the Control Application is run on a PC. To prevent anyone to alter the software on these units (U8, U9, U10, and U12), the RCX, the AIBO and the PC will be
placed such as to only allow administrators access to the software (i.e. RCX placed in a locked room). Furthermore, the PC where the Control Application resides will have access control too ensure that only personnel identified as authorized can operate the Control Application (U11 and U13).

1.28 Design of the system

System design is preformed together with architectural design in step 5 in the development process. We describe the design in a separate section to promote readability of this report. In contrast to the architecture, which is concerned with the systems structures and elements that are visible to the environment, the design is concerned with all system elements and their interaction. The system design builds on the choices made in the architectural design. It describes the components the architectural elements consist of and the internal interaction between these components.

This section describes the design choices in the new prototype. The software classes and their responsibilities and their interaction are described. Design choices that are made in order to fulfil quality requirements are described with reference to scenarios the design choices address. The reader is also referred to section 1.27 for architectural choices.

1.28.1 Authorization mechanism

![Diagram of Authorization Mechanism](image)

Figure 0-15: Class diagram for Authorisation Mechanism subsystem. Key: UML.
A Sony AIBO is used as the Authorization mechanism in this system (see 1.27.4). This subsystem will be implemented using the OPEN-R SDK development environment. The classes described in Figure 0-15 will use the OPEN-R libraries to accomplish their tasks. For simplicity the OPEN-R classes are not described here, but the reader is referred to [22] for further information on OPEN-R SDK.

As described in the decomposition view, in section 1.27.2, the Authorization Mechanism (from now AIBO) is responsible for identifying personnel at the entrance area. This section will describe how the software elements of the AIBO will solve this task.

- The Startup class is responsible for initialising the AIBO. It initialises an object implementing a MovingLegs interface, and requests that the legs are set in a start position. There are no requirements for the AIBO to move, thus the legs will be set so that the AIBO is either standing or sitting. The MovingLegsImpl class is the class responsible for setting the legs in a start position. After the AIBO is initialized, the Startup class gives the control to the SearchforObject class.

- SearchForObject is designed as a task controller for the search and identify personnel task. Which mean that the SearchForObject is the class responsible for assigning subtasks to other classes in order to identify any personnel that might come into the entrance area. It will use a class implementing the MovingHead interface to move the AIBO's head when searching for objects (personnel). The input from the camera in AIBO’s head is sent to IdentifyObject class to match with known objects. If an object is recognised the SearchForObject class sends the identification of the object to the NotifyLostFound class. If an identified object leaves the entrance area (and the view of the AIBO), SearchForObject informs the NotifyLostFound of this.

- The NotifyLostFound class is responsible for informing users or other systems when the AIBO finds an identified object and when it loses an identified object from sight. In this version of the system two things happen when an identified object (i.e. authorized/unauthorized personnel) is found or lost. First, a sound is played by the PlayLostFoundSound class on request from NotifyLostFound.
Second, the AuthorizationInterface is used to communicate of the lost or found event.

- The AuthorizationInterface is the class responsible for communicating with other subsystems. In this version of the system the AIBO only communicates with the Control Application subsystem, although communication with other systems (such as another AIBO) will in the same way (U21). The AuthorizationInterface class provides an opportunity for the Control Application (and other systems) to register their “contact information”, and the AIBO will notify the registered systems when objects (such as personnel) are either lost or found. The communication protocol implemented will confirm all messages received by the AIBO, and it will resend messages if no confirmation is received on messages the AIBO sends (U2). Furthermore, the protocol will ensure a secure transmission by providing a checksum with all messages. This will ensure that the receiver of a message knows if the message is intact and unaltered (U5). If the communication protocol is changed or the communication channel is replaced, the AuthorizationInterface is the only class in the AIBO which is affected (U24). The AuthorizationInterface will use classes from the OPEN-R SDK in order to send and receive messages.

- This subsystem is designed for high cohesion. That is, the classes in this subsystem have only one limited task. If new functionality is added such as getting the AIBO to follow an identified object (U22), a new class is added for controlling this task. This new class will use the already existing classes, such as IdentifyObject and MovingLegs, in order to accomplish its task. The only change to existing classes will be to pass control over to the class which will implement the new task. In the case of adding the functionality for following an identified object, the NotifyLostFound class passes control to the new “follow” class when an identified object has been found.
1.28.2 Control Application

![Diagram of SystemStateController, Communication, UserInterface, AIBO communicate, Lego Communicate]

Figure 0-16: Design of the control application. Key: UMI.

The main responsibility of the Control Application is to ensure that authorized personnel are given access to the safety zone and that the cutting robot is not operating when they are inside the safety zone. It is also responsible for not granting access to unauthorized personnel. The software classes of the Control Application subsystem are described below:

- SystemStateController is responsible for ensuring that the system at all times is in a legal state according to section 1.25.5. The SystemStateController gets information of the system state through the communication interfaces AIBO communicate and LEGO communicate, and determines the correct measures to move the system into a legal state. The SystemStateController sends messages to the factory subsystem in order to obtain a legal system state.

- The Control Application provides a simple user interface to the administrator of the system. By using the UserInterface the administrator is able to connect the ControlApplication to the AIBO (Authorization Mechanism). It will be hard to for an attacker to masquerade as the original AIBO, since an administrator will be able to control when to connect to the AIBO. The UserInterface will show the system state information to the administrator, and it will provide the
administrator with an opportunity to stop the cutting robot. If the administrator chooses to stop the cutting robot the interface creates a “stop cutting robot” event, which will be picked up by the SystemStateController. The UserInterface will subscribe to events concerning state changes in created by the SystemStateController in order to update system state information shown to the administrator.

• As indicated above the communication between the SystemStateController and the UserInterface by creating events and using implicit invocation mechanism. In this way the SystemStateController and the UserInterface does not need to have direct knowledge of the other, which will make it easier to replace the UserInterface (U23).

• To communicate with the AIBO and the LEGO system (Factory Subsystem), the Control Application uses a communication interface. This interface generalized in the Communication class, and has a specialized communication class for the AIBO and the LEGO system (described below). By sub classing the general Communication class the specific communication interfaces are easily replaced (U26 and U23). The communication protocol used will be implemented in the Communicate class, while communication concerns related to the particular subsystem will be implemented in the subclasses. All messages sent to and from the Control Application will be confirmed. If messages are not confirmed the messages will be resent until a given number of times, before the unit not responding will be assumed unavailable (U2, U3 and U4). When a message arrives the Communication creates a message received event, which the SystemStateController will subscribe on.

• AIBO Communicate is a subclass of the Communicate class. It provides a communication interface to the AIBO. All messages sent between the AIBO and the Control Application will provide a checksum along with the message. This will ensure that the receiver of a message knows if the message is intact and unaltered (U5). The Control Application will send idle messages (ping) to the AIBO at regular intervals in order to check if the AIBO is available. Telnet over a wireless LAN will be used to communicate with the AIBO.
- LEGO communicate is a subclass of the Communicate class, and provides a communication interface to the LEGO system. The LEGO communicate will send and receive messages through an IR link. As for the AIBO Communicate the LEGO Communicate will provide a checksum along with messages that are sent (U6 and U7). Furthermore, the IR link will be physical shielded to avoid deliberate interference with the communication.

### 1.28.3 Factory subsystem

![Diagram of Factory subsystem](image)

Figure 0-17: Design of the Factory subsystem. Key: UML.

The Factory subsystem (from now on LEGO system) is built in LEGO, and the physical LEGO units in the system are controlled by software run on a LEGO Mindstorms robot (RCX). The software classes of the LEGO system and their relations to each other are shown in Figure 0-17, and are described below.

- The FactoryInterface is the communication interface used to communicate with the LEGO system. All messages sent to and from the LEGO system will be confirmed at arrival (U3 and U4). If messages are not confirmed they are resent a given number of times, before the system the message is sent to is assumed unavailable. The communication medium used by the FactoryInterface is an IR link. To ensure that messages arrive intact and unaltered, the FactoryInterface will provide a checksum with all messages that are sent (U6 and U7). The IR link will also be physical shielded to avoid deliberate interference with the communication.
• The FactoryStateController is responsible for keeping the LEGO system in a legal state. That is, it is responsible for ensuring that the cutting robot only is operating when it is supposed to (see section 1.25.5) and that the gate is not opened when not allowed. It receives information from the Control Application through the FactoryInterface for when the gate is allowed to open and for when the cutting robot is not allowed to operate. Furthermore, the FactoryStateController will receive stimuli from the on and off buttons at the entrance gate, through the ButtonSensor class. The FactoryStateController will use the CuttingRobotController and the GateController in order to start/stop the cutting robot and to open/close or lock/unlock the gate.

• There will be sent idle messages (similar to ping) between the FactoryInterface and the Control application whenever there are no other messages to be sent. The reason for this is that when the LEGO system does no longer receive messages from the Control Application the LEGO system assumes that the Control Application is unavailable (U1 and U15). The FactoryInterface will notify the FactoryStateController and the cutting robot will be shut down, until the Control Application is available.

• The ButtonSensor class, the CuttingRobotController class, and the GateController class are used as driver classes to the buttons, cutting robot and the gate respectively.
EVALUATION

1.29 Evaluation strategy

According to [60] there are four main research approaches:

- **Scientific method.** Scientists develop a theory to explain a phenomenon in order to do so propose a hypothesis and then test alternative variations of the hypothesis. As they collect data to verify or refuse the claims of the hypothesis.
- **Engineering method.** Engineers develop and test a solution to a hypothesis. Based upon the results of the test, they improve the solution until it requires no further improvement.
- **Empirical method.** A statistical method is proposed as a means to validate a given hypothesis. Unlike scientific method, there may not be a formal model or theory describing the hypothesis. Data is collected to verify the hypothesis.
- **Analytical method.** A formal theory is developed, and results derived from that theory can be compared with empirical observations.

These approaches apply to science in general, but in software engineering we need methods that take the specific characteristics of software into consideration. In [43] Zellkowitz et al. have looked experimental approaches that are used in the field of computer science. They have developed a taxonomy consisting of 12 different experimental approaches grouped into 3 model categories; observational, historical, and controlled. In observational methods there is relatively little control over the development process besides using the technology studied. This method collects data throughout the development process. In historical methods the data already exists in form of documentation of previous projects, and it is only necessary to analyse the data already collected. The controlled methods provide for statistical validity of the result by providing for multiple instances of an observation.

Below we give a short description of the 12 experimental methods from [43].

Observational methods

- **Project monitoring** collects data available data during a development process without trying to influence it or redirect the process or the methods used. Strengths: Provides a baseline for future work, and it is inexpensive. Weakness: No specific goals.
- **In Case study** a particular attribute is monitored over time in a project. Researchers collect data for the case study derived from a specific project goal, and measure the data related to the particular attribute.
Strengths: Can constrain one factor at low cost.
Weakness: Poor controls for later replication.

- **Assertion** is a method where the developers are both experimenters and subject of the study. This method can be used as a preliminary test before formal validation of a technology’s effectiveness.
  Strengths: Serves as a basis for future experiments.
  Weakness: Insufficient validation
- **A Field study** may collect data from several projects simultaneously and examine them. In this method it is difficult to collect all relevant data since the primary goal is often not to influence the subjects.
  Strengths: Inexpensive form for replication.
  Weakness: Treatments differ across projects.

**Historical methods**

- In a **Literature search** the investigator analyse result from publicly available papers and documents. This method can be used to confirm existing hypothesis or to enhance data collected on one project with data that has been collected by a previously project.
  Strengths: Large available database, inexpensive
  Weakness: Selection bias, treatments differ.
- **Legacy data.** Here existing files are examined to determine trends and to understand previously completed projects in order to use this information in a new project.
  Strengths: Combines multiple studies, inexpensive
  Weakness: Cannot constrain factors, data is limited
- In **Lessons learned studies** are studies of lessons learned documents researchers have produced after completing large industrial projects. The goal of these studies is to find qualitative aspects that can improve future developments.
  Strengths: Determines trends, inexpensive
  Weakness: No quantitative data, cannot constrain factors
- **Static analysis** focus on studying completed products. To determine the products character, researchers analyse its structure.
  Strengths: Can be automated, applies to tools
  Weakness: Not related to development method

**Controlled methods**

- **Replicated experiment** is a method where several projects are staffed to perform a task in multiple ways, and the factors of interested are controlled for these approaches. Data is collected and compared for the different replications.
  Strengths: Can control factors for all treatments
  Weakness: Very expensive, Hawthorne effect
- **Synthetic environment experiments** are replication experiments performed in smaller and controlled environments. A relative small objective is identified and all
variables except the control method being modified are fixed.
Strengths: Can control individual factors, moderate cost
Weakness: Difficult to scale up, interaction among multiple factors

- Dynamic analysis is a method that analyse the product itself. Instruments can test
  the product by adding debugging or testing code in such a way that the
  products features can be demonstrated and evaluated while it executes.
Strengths: Can be automated, applies to tools
Weakness: Not related to development method

- In Simulation methods researchers evaluate technology by executing the
  product. They make assumptions on how the real environment will interact
  with the product and make a model of the real environment in which the
  product is executed.
Strengths: Can be automated, applies to tools, evaluation in safe environment
Weakness: Data may not represent reality, not related to development method

1.30 Evaluation

In this section we will present the results of and discuss the evaluation between the new
and the original prototype. The goal of this evaluation is to validate that the development
process used to develop the new prototype supports design for modifiability in a security-
safety critical system. We do this by evaluating the new prototype according to three
success criteria. First, the new prototype must provide the functionality described in the
requirements. The requirements for the new prototype are based on the requirements from
[37], and we will therefore show that the functional requirements for the new prototype
provide at least the same functionality. Second, we require that the new prototype provide
at same safety level as the original prototype, since the prototype is a safety critical system.
Third, we require that the new prototype provide a higher level of modifiability, which will
 favour system evolution.

Since the developer of the new prototype system is both the experimenter and the
subject of study, we have used assertion as research method. This is a weak validation
method, which may be biased. However, the main goal of the evaluation is to provide a
qualitative and preliminary evaluation as a pre-study. By using assertion as research method
we keep the costs of this pre-study at a minimum.

1.30.1 Comparing the functionality

In this section we will show that the functional requirements and the constraints to the
new prototype cover the functional requirements as described in [37]. In Table 0-1 we have
listed the functional requirements for the original prototype in the left column with the equivalent functional requirement or constraint for the new prototype in the right column.

The requirements F1.2.ii and F4.3 for the original prototype state that it shall be possible to start the cutting robot from the user interface in the control application. In the Preliminary Hazard Analysis (see section 1.24) this is identified as a safety risk. This is also identified in the Preliminary Hazard Analysis for the original prototype in [37]. Although the requirements for the original prototype where not modified accordingly, it was accounted for in the implementation. The rest of the functional requirements are covered by the functional requirements and constraints for the new prototype.

Table 0-1: Functional requirements of the two prototypes. The prefix F stands for functional requirement and the prefix C stands for constraint.

<table>
<thead>
<tr>
<th>The original prototype</th>
<th>The new prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1. The LEGO cutting robot shall</td>
<td>F1. The cutting robot shall</td>
</tr>
<tr>
<td></td>
<td>C2. The cutting robot shall be built in LEGO</td>
</tr>
<tr>
<td>F1.1. stop operating if</td>
<td>F1.1. stop operating if</td>
</tr>
<tr>
<td>i. the Off button in the entrance area is pressed.</td>
<td>i. the Off button in the entrance area is pressed</td>
</tr>
<tr>
<td>ii. the &quot;stop cutting&quot; button in the control window on the PC controller is pressed.</td>
<td>ii. the control application sends a &quot;stop cutting&quot; signal</td>
</tr>
<tr>
<td>iii. the connection between the AIBO and the PC controller is lost.</td>
<td>iii. the communication between the authorization mechanism and the controller application is lost (replicated in F3.1.ii)</td>
</tr>
<tr>
<td>iv. the connection between the PC controller and the LEGO system is lost.</td>
<td>iv. the communication between the control application and the cutting robot is lost</td>
</tr>
<tr>
<td>v. the gate is opened.</td>
<td>v. the gate is opened</td>
</tr>
<tr>
<td>F1.2. start operating if</td>
<td>F1.2. start operating if</td>
</tr>
<tr>
<td>i. the On button in the entrance area is pressed by an authorized person</td>
<td>i. the On button in the entrance area is pressed by an authorized person</td>
</tr>
<tr>
<td>ii. the &quot;Start cutting&quot; button in the control window on the PC controller is pressed.</td>
<td>This requirement is removed for safety reasons</td>
</tr>
<tr>
<td>F2. The RCX of the LEGO system shall</td>
<td></td>
</tr>
<tr>
<td>F2.1. communicate with the PC controller through IR signals.</td>
<td>C7. The communication between the control application and the RCX shall be through infrared signals.</td>
</tr>
<tr>
<td>F2.2. control</td>
<td>C6. A LEGO Mindstorm RCX shall control the cutting robot (start/stop), the gate (open/close and lock/unlock), and interpret the signals from the on and off buttons.</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>i the lights</td>
<td></td>
</tr>
<tr>
<td>ii the cutting robot</td>
<td></td>
</tr>
<tr>
<td>iii the gate</td>
<td></td>
</tr>
<tr>
<td>iv the gate lock</td>
<td></td>
</tr>
<tr>
<td>F2.3. interpret data from</td>
<td></td>
</tr>
<tr>
<td>i the On and Off buttons</td>
<td></td>
</tr>
<tr>
<td>ii the sensors on the gate (touch sensors which are used to determine when the gate is opened or closed)</td>
<td></td>
</tr>
<tr>
<td>F2.4. inform the PC controller about state changes of</td>
<td>F3.3. The control application shall display status information when</td>
</tr>
<tr>
<td>i the cutting robot (started/stopped)</td>
<td>iii The cutting robot has started</td>
</tr>
<tr>
<td>ii the gate (opened/closed)</td>
<td>iv The cutting robot has stopped</td>
</tr>
<tr>
<td></td>
<td>ix The gate is closed</td>
</tr>
<tr>
<td>F3. The AIBO shall</td>
<td>F2. The authorization mechanism shall</td>
</tr>
<tr>
<td>F3.1. communicate with the PC controller through wireless LAN</td>
<td>C8. The communication between the control application and the AIBO shall be through a wireless LAN.</td>
</tr>
<tr>
<td>F3.2. search for green man and pink ball.</td>
<td>F2.1. monitor the entrance area for personnel (search for green man and pink ball)</td>
</tr>
<tr>
<td>F3.3. notify the PC controller if</td>
<td>F2.3. notify the control application if</td>
</tr>
<tr>
<td>i it detects an authorized person (green LEGO man) or an intruder (pink ball) entering the entrance area.</td>
<td>i it detects an authorized person (green LEGO man) or an unauthorized person (pink ball) entering the entrance area</td>
</tr>
<tr>
<td>ii it detects an authorized person or an intruder leaving the entrance area.</td>
<td>ii it detects an authorized person or intruder leaving the entrance area</td>
</tr>
<tr>
<td>F4. The PC controller shall</td>
<td>F3. The control application shall</td>
</tr>
<tr>
<td>F4.1. communicate with the LEGO system through IR signals.</td>
<td>F7. The communication between the control application and the RCX shall be through infrared signals.</td>
</tr>
<tr>
<td>F4.2. communicate with AIBO through wireless LAN.</td>
<td>C8. The communication between the control application and the AIBO shall be through a wireless LAN.</td>
</tr>
<tr>
<td>F4.3. send a start signal to the LEGO system when the start button in the control window is pressed.</td>
<td>This requirement is removed for safety reasons</td>
</tr>
<tr>
<td>F4.4. send a stop signal to the LEGO system when</td>
<td>F3.1. send a stop signal to the cutting robot</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>i  the stop button in the control window is pressed.</td>
<td>i  on user request</td>
</tr>
<tr>
<td>ii the connection to the AIBO is lost.</td>
<td>ii  the communication between the authorization mechanism and the controller application is lost</td>
</tr>
<tr>
<td>iii the AIBO becomes unavailable.</td>
<td>iii  the authorization mechanism becomes unavailable</td>
</tr>
<tr>
<td>F4.5. send a “close gate” signal to the LEGO system when</td>
<td>F3.2. send a “closed and lock gate” signal to the gate when the communication with the authorization mechanism is lost</td>
</tr>
<tr>
<td>i  the connection to the AIBO is lost.</td>
<td></td>
</tr>
<tr>
<td>F4.6. display status information when</td>
<td>F3.3. display status information when</td>
</tr>
<tr>
<td>i  communication to AIBO is lost.</td>
<td>i  communication with authorization mechanism is lost</td>
</tr>
<tr>
<td>ii  communication to LEGO system is lost.</td>
<td>ii  communication to factory system is lost</td>
</tr>
<tr>
<td>iii the cutting robot has started.</td>
<td>iii  The cutting robot has started</td>
</tr>
<tr>
<td>iv the cutting robot has stopped.</td>
<td>iv  The cutting robot has stopped</td>
</tr>
<tr>
<td>v intruder is detected in the entrance area.</td>
<td>v  Intruder is detected in the entrance area.</td>
</tr>
<tr>
<td>vi authorized person is detected in the entrance area.</td>
<td>vi  Authorized person is detected in entrance area</td>
</tr>
<tr>
<td>vii the cutting robot is detected in the entrance area.</td>
<td>vii  The cutting robot can not be started</td>
</tr>
<tr>
<td>F5. The gate shall</td>
<td>F4. The gate shall</td>
</tr>
<tr>
<td>F5.1. unlock if an authorized person (green LEGO man) enters the entrance area.</td>
<td>F4.1. unlock if an authorized person (green LEGO man) enters the entrance area</td>
</tr>
<tr>
<td>F5.2. open if the gate wheel in the entrance area is turned.</td>
<td>F4.2. open if the gate wheel in the entrance area is turned</td>
</tr>
<tr>
<td>F5.3. close if the authorized person leaves the entrance area without closing the gate.</td>
<td>F4.3. close if the authorized person leaves the entrance area without closing the gate</td>
</tr>
<tr>
<td>F5.4. lock if the authorized person leaves the entrance area.</td>
<td>F4.4. lock if it authorized person leaves the entrance area</td>
</tr>
<tr>
<td>F5.5. close if it is open and the On button in the entrance area is pressed.</td>
<td>F4.5. close if it is open and the On button in the entrance area is pressed</td>
</tr>
<tr>
<td>F5.6. close and lock if the connection between the AIBO and the PC controller is lost.</td>
<td>F4.6. close and lock if the communication between the authorization mechanism and the control application is lost</td>
</tr>
<tr>
<td>F5.7. close and lock if the connection between the PC controller and the LEGO system is lost.</td>
<td>F4.7. close and lock if the communication between the control application and the gate is lost</td>
</tr>
</tbody>
</table>
1.30.2 Comparing the safety level

To show that the new prototype provides the same safety level as the original prototype, we performed a Preliminary Hazard Analysis (PHA) and a Security-HazOp for the new prototype based on the PHA and Security-HazOp performed for the original prototype in [37].

In the PHA documented in section 1.24 we used the same hazards, consequences, and causes as was used for the PHA for the original prototype. We found that the results from the PHA in [37] also applied for the new prototype. Some of the hazards where already dealt with in the requirements specification (see Table 0-1). In the Security-HazOp we performed in section 1.26 we used the same expressions, security threats, consequences, and causes as used in the Security-HazOp for the original prototype. In most cases the countermeasures identified in [37] also applied for the new prototype. The results from the Security-HazOp for the original prototype are given in appendix A, while all results that differ are given in Table 0-4. In [37] the acceptance level for risk was IV, thus risks categorized as level I, II, or III where not accepted and needed to be treated. To provide the same safety level for the new prototype we used the same strategy. Thus, the countermeasures identified in appendix A and in Table 0-4 where taken into account in the architecture and design.

1.30.3 Evaluating modifiability

When evaluating the conformance to modifiability we analyzed the original prototype [37] (in this section prototype 1) and the prototype in our work (in this section prototype 2) according to their degree of fulfilment of the requirements to modifiability. Our goal is to show that we have gained an increased level of modifiability in the new prototype. To assess this we will use an evaluation technique based on ATAM (see section 1.16.1). We do not use ATAM in full scale since the prototype has few stakeholders. The scenarios used in the evaluation are the scenarios for modifiability identified and described in section 1.27.1. The scenarios are based on the requirements to modifiability for prototype 2.

For each modifiability scenario we describe what changes needs to be done to fulfill the scenario for both prototypes and which elements that are affected by the change. As part of the evaluation we will present the scenarios, the architecture and design description of
both prototypes (see section 0 and section 0), and the change description given below to experts for expert judgement. The expert will classify the difficulty level and time of imposing changes described in the scenarios for both prototypes. We also present an assumption of which prototype we think will be easiest to change according to the scenario given. These assumption where not given to the experts, but are added in the report to ease the reading.

**Scenario U21**

More AIBO to monitor the entrance gate are added and enabled to communicate with each other. Changes to existing elements in the AIBO are completed in less than 1 person-week (N3.2.i and N3.2.ii).

Prototype 1

In order to get two AIBO to communicate with each other the BallTrackingHead class will need to be modified. Either the BallTrackingHead will need to include a method for communicating with the other AIBO or it needs to use another class (a new class) that implement this functionality. This means that the class responsible for the overall functionality of the AIBO (the BallTracknigHead class) needs to be modified and recompiled.

Prototype 2

The Authorization Interface in the AIBO communicates with external systems by callback functionality. In other words, the Authorization Interface provides a method for other systems to register their contact information. The Authorization Interface then uses this information when sending messages. If two AIBO shall is to communicate the Authorization Interface needs to implement a method that is responsible for registering contact information to the other AIBO. The Authorization Interface class will also have to implement a method for processing the messages sent from the other AIBO.

**Assumption:** It is easier to add the communication between two AIBO’s by adding two methods in a class (the Authorization Interface in prototype 2) that is only concerned with communication than to modify the class that is also responsible for the main functionality
(BallTrackingHead in prototype 1) of the AIBO. The reason for this is when modifying the class responsible for main functionality the developer responsible for making the necessary changes may not be aware of hidden dependencies inside the class.

**Expert judgement**

<table>
<thead>
<tr>
<th>Scenario U21</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>hard</td>
<td>medium</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Medium</td>
<td>medium</td>
</tr>
</tbody>
</table>

**Scenario U22**

Add functionality for the AIBO to follow intruders or authorized personnel in the entrance area and the safety zone. Changes to existing elements in the AIBO are completed in less than 1 person-week (N3.2.ii)

Prototype 1

In order to add the functionality of getting the AIBO to follow an intruder or an authorized person, the BallTrackingHead class will have to be modified, so that instead of just searching for an object it will also follow it with its head and move after it. We would not introduce a new class to fulfil this new functionality, since this would require the new class to replicate much of the functionality of the BallTrackingHead (such as the identification of an object/person).

Prototype 2

In order to make the AIBO follow intruders or authorized personnel there will have to be added a class that implement the overall functionality for following the identified object. This class will receive control flow from the NotifyLostFound class when an object has been identified, and it will pass control back to the SearchForObject class through NotifyLostFound when the AIBO looses the object out of sight. The new follow class will use MovingHead and MovingLegs in order to follow the object, and it will use the
IdentifyObject class to confirm that it still sees the object, in the same way as the SearchForObject uses this class. A class diagram is given in Figure 0-1 to show the added class and its relations.

![Class diagram after adding the FollowObject class: Key UML.](image)

*Assumption:* The two approaches will be of the same difficulty to achieve. But complexity of the modules is held at a minimum by introducing a new module. This will ease further modifications at a later time, and ease the understandability/readability.

*Expert judgement*

<table>
<thead>
<tr>
<th>Scenario U22</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>Easy</td>
<td>Short</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Easy</td>
<td>Short</td>
</tr>
</tbody>
</table>

**Scenario U23**

A new user interface to the control application replaces the old one, no elements other than the user interface is affected (N3.1)
Prototype 1

As long as the new user interface is given the same name as the old one and it provides at least the same interface to the Talking class and the CBall3App, no classes will be affected by the replacement.

Prototype 2

As long as the new UserInterface is given the same name as the old one and it provides at least the same interface to the SystemStateController, no classes will be affected by the replacement.

Assumption: This change will be of the same difficulty for both systems.

Expert judgement

<table>
<thead>
<tr>
<th>Scenario U23</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>Easy</td>
<td>Short</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Easy</td>
<td>Short</td>
</tr>
</tbody>
</table>

Scenario U24

The communication interface between the control application and AIBO is changed from telnet to CORBA and elements responsible for functionality are not affected (N3.3)

Prototype 1

In order to replace the communication interface from telnet to CORBA both the BallTrackingHead class in the AIBO and the Talking class in the PC controller needs to be modified. The BallTrackingHead class is responsible for finding and identifying personnel in addition to communicating with the PC controller, and the Talking class is responsible for communicating with the LEGO system and that the user interface (Ball3Dlg class) gets information of system events.
Prototype 2

To replace the communication interface between the Control Application and the AIBO the AuthorizationInterface in the AIBO and the AIBOCommunicate in the Control Application. These two classes are only responsible for communication between the Control Application and the AIBO.

Assumption: It is easier to only modify the AuthorizationInterface and the AIBOCommunicate classes since these are only concerned with communication, than modifying the BallTrackingHead and the Talking classes.

Expert judgement

<table>
<thead>
<tr>
<th>Scenario U24</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, long)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>Hard</td>
<td>Medium</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Easy</td>
<td>Short</td>
</tr>
</tbody>
</table>

Scenario U25

Replace the AIBO with a key card mechanism for authorization of personnel (N3.2.iv)

Prototype 1

The Talking class of the PC controller will have to be modified in order to replace the AIBO with a key card mechanism. This class is also responsible for communicating with the LEGO system and that the user interface (Ball3Dlg class) gets information of system events. Although no changes to the communication with the LEGO system or the functionality that informs the user interface about system events, the whole class must be recompiled.

Prototype 2
In order to replace the AIBO with a key card mechanism to identify personnel as authorized or unauthorized a new subclass of the Communication class in the Control Application will have to be added. No other changes are necessary.

*Assumption:* It is easier to add a subclass of the Communicate class in the Control Application, than to modify the Talking class in order to replace the AIBO with a key card mechanism.

*Expert judgement*

<table>
<thead>
<tr>
<th>Scenario U25</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Easy</td>
<td>Short</td>
</tr>
</tbody>
</table>

**Scenario U26**

The communication interface between the control application and mechanism controlling the gate and the cutting robot is changed from IR to Bluetooth and elements responsible for functionality are not affected (N3.4)

Prototype 1

In order to replace the communication interface from IR to Bluetooth both the PostOffice module in the LEGO system and the Talking class in the PC controller needs to be modified. The Talking class is responsible for communicating with the AIBO and that the user interface (Ball3Dlg class) gets information of system events, while the PostOffice module is only responsible of communicating with the PC controller.

Prototype 2

To replace the communication interface between the Control Application and the LEGO system the FactoryInterface in the LEGO system and the LEGOCommunicate in
the Control Application. These two classes are only responsible for communication between the Control Application and the LEGO system.

Assumption: It is easier to modify the FactoryInterface and the LEGOCommunicate classes since these are only concerned with communication, than modifying the PostOffice module and the Talking class, since the Talking class also is responsible for communication with the user interface and the AIBO.

Expert judgement

<table>
<thead>
<tr>
<th>Scenario U26</th>
<th>Ease of imposing change (easy, medium, hard)</th>
<th>Time of imposing change (short, medium, high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype 1</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Prototype 2</td>
<td>Easy</td>
<td>Short</td>
</tr>
</tbody>
</table>

1.30.4 Summary of evaluation

The evaluation has shown that the new prototype has both the same functionality and safety level as the original prototype. As for the evaluation of modifiability the expert judgment shows that the new prototype has a higher level of modifiability given the requirements that we have imposed to the original prototype.
**DISCUSSION AND CONCLUSION**

Today most software systems need to evolve to meet new user requirements and to adapt to changing environment. Research has shown that up to 70% of the total lifecycle cost for software systems are spent on evolvement of the system [52]. It would therefore be advisable to design systems in order to lower the difficulty and time needed for modifying the systems. As with other quality attributes one can not add modifiability as an afterthought, it has to be built into the system from early in the development process.

1.31 Development process

In our work we have tried to show that modifiability needs to be an integrated part of the development process in order to support system evolution. The main contribution of our work is a draft version of a development process that focuses on system evolution in security-safety critical systems. We have performed a literature study to asses and identify appropriate methods to integrate modifiability as a part of a development process for security-safety critical systems. As a basis for our process we have used a development process for security-safety critical systems developed by Sorby in her master thesis [37]. In her work she focused mainly on the relationship between security and safety, and particularly on safety consequences of security threats. This development process was partly based on the IEC 61508 standard [25], which describes a life cycle process for developing safe systems. The lifecycle model of IEC 61508 is a very detailed and strict process that focuses on safety requirements identification and allocation to ensure the conformance safety demands in the modules in the system. It has no focus on architectural design or system evolution, which are the main concern of this report. To deal with security threats the development process in [37] used the Security-HazOp method from the CORAS framework [10]

We extended the development process in [37] to include a thorough architectural description. The architectural description consists of a decomposition view, a security view, a safety view, a deployment view, and a set of scenarios based on the quality requirements to the system. The scenarios are organized in a utility tree, where the root node is the overall utility of the system. The child nodes represent the most important architectural drives (quality goals) of the system. The leaf nodes of the tree are scenarios that describe the quality goals of the system in detail. We have taken the idea of using a utility tree to
describe the quality goals from the Architecture Tradeoff Analysis Method (ATAM) [42]. In ATAM the utility tree is used to evaluate the utility of the architecture given the most important architectural drives. In our work we have adopted the utility tree when constructing the architecture and design in order to be suitable according to our quality goals; security, safety, and modifiability. Furthermore, the scenarios are used to trace quality requirements, and are referred to in the architectural views where choices are made that address them.

To design the architecture of the system we use the Attribute-Driven Design (ADD) method [41]. ADD decompose the system by applying tactics and architectural patterns to fulfill the scenarios that describe the architectural drives. The first architectural drive we considered was modifiability, which is described in the decomposition view. This view shows allocation of the functionality into modules and subsystems and describes how requirements to modifiability can be met. Since the development process is aimed at security-safety critical systems it is natural to add a security view and a safety view to show how the system provides the required level of security and safety. By extracting security and safety issues into separate views we make it tractable to determine the security and safety level, such as Evaluation Assurance Level (EAL) in the Common Criteria [28] for security and Safety integrity levels (SIL) in IEC 61508 [25] for safety.

In the safety view we use the modules related to safety from the decomposition view to describe how safety is ensured. Equivalently, in the security view we use the modules related to security from the decomposition view to describe how security is ensured. The deployment view show how the subsystems, from the decomposition view, are mapped onto the processing units and how these interact. Since the security, safety, and deployment view uses modules from the decomposition view they are also based on the decomposition view as shown in Figure 0-1. Thus, modifiability becomes the primary architectural drive for the system. This makes sense since the decomposition view focus on separating concerns and allocating functionality. The rational behind this is that when we decomposed according to the modifiability requirements we ensure the system modifiability. Furthermore, we assign responsibilities related to safety to the appropriate modules and describe their interaction to ensure safety in the safety view and security in the security view. The deployment view of the architecture is added to show how the mapping of the subsystems onto the processing units, and to document communication media used.
1.32 Aspect Oriented Software Development

In the decomposition of the prototype we have focused on separating concerns to promote modifiability of the system. Although the concerns have been separated into different modules, the modules that encapsulate still interact, thus relating the concerns indirectly. In Aspect Oriented Software Development (AOSD) the different concerns, or aspect as they are called, are not related. Using Aspect Oriented Modelling (AOM) the system is decomposed according to the functionality, which is called the primary system modularization. Then the different aspects are weaved into the primary system modularization through pointcuts. These pointcuts are a set of rules describing where in the design a particular aspect will interact. Thus, neither the primary system modularization nor other aspects have any knowledge about the particular aspect. This will aid in system evolution since the developer can focus on realizing a single concern at the time without worrying about the interaction with other concerns.

The reason why we have not used this approach in our work is twofold. First, AOSD is an emerging discipline with few proved methods and little support for developing systems in AOM. Second, the framework we have available for developing software for the AIBO and the LEGO Mindstorm do not support aspect oriented programming.
1.33 Evaluation

We evaluated the effectiveness of our development process through the development of a new prototype for an already existing security-safety system. The system used was chosen since it is a relatively small size and low complexity system and since we had limited time this seemed tractable. Modifiability concerns where not considered for the original prototype in [37]. We therefore imposed requirements to modifiability to the original prototype and redesigned the system according to this development process.

The evaluation strategy used where assertion, meaning that the developer played the role as both experimenter and subject of the study. The reason for choosing this research method was the time limit and the limited amount of resources. In addition, the work is meant as a preliminary test of the development process. The main drawback of using assertion as research method is the aspect of problematic validity. The fact that the experimenter is also the subject of study has several implications. First, there may be a threat to external validity since we use a student as subject and the population of subjects may therefore not be representative for the population in a real-life setting. Furthermore, the system developed is a toy system and not representative for the complexity of real industry systems. Consequently our ability to generalize the findings is somewhat limited. Second, there may be several causes to why we obtain a higher level of modifiability then using our development process, inflicting internal validity. The subject developing the prototype is also the experimenter, which influences the focus during the development process. Furthermore, the scenarios used when evaluating the level of modifiability where designed based on the new prototype. This may result in an apparent higher level of modifiability, than if we had used a completely different set of scenarios for the evaluation. Third, we have no quantitative measures or statistical tests to base our conclusion on, which is a threat to validity.

The results from the evaluation described in section 0 show that the new prototype has the same functionality and safety level as the original. In addition the new prototype has a higher level of modifiability. This fulfils the success criteria we have set for the evaluation, indicating that our development process proved successful for emphasising modifiability in the development of security-safety systems.
1.34 Modifiability in critical systems

As with other software systems, security critical and safety critical systems need to evolve. New and better security and safety mechanisms will replace outdated mechanisms or components. A system developed for modifiability will not only reduce time and cost of performing the change, it will also reduce the risk of imposing flaws. The reason for this is that modifiable systems are usually more structured than other systems, and they have few or no hidden dependencies between different concerns. However, caution is advised when modifying critical software systems and appropriate validation techniques should be applied when changes are implemented. Even though a system is designed in order to evolve in a particular manner, the design of the system may include assumptions that are illegitimate. In the related field of software component reuse Leveson [46] pinpoint that one of the most common factors in software-related accidents is due to software component reuse. These accidents occur when the assumption of the software’s environment is incorrect.
FURTHER WORK

Software systems play an increasingly larger role in our ever changing society. As the environment to software systems change the need for designing systems to support evolution increase. Better understanding of how support for software system evolution is acquired should be investigated further. The new software development paradigm of Aspect Oriented Software Development looks promising for ensuring system modifiability, providing support for system evolution.

In our work we have proposed a draft for a development process for security-safety systems which focuses on modifiability. As a pre-study we conducted an evaluation of the process using assertion as evaluation method, and suggest that more formal evaluation methods are used in order to validate the effectiveness of the process. Performing an evaluation of the development process through a case study in a real-life setting would be beneficial. In a case study we would be able to measure the time and cost of this process compared to similar projects. In addition system modifiability could be extensively measured using dynamically analysis methods on the complete system or by imposing changes to the system and measure the time and cost for implementing them.
# A Security-HazOp Tables

These tables are taken from the appendix C of Karine Sorbys master thesis [37].

<table>
<thead>
<tr>
<th>Breaches of Confidentiality</th>
<th>Security threat</th>
<th>Safety consequences</th>
<th>Causes</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliberate disclosure of information on AIBO</td>
<td>An unauthorized party gains access to the application code of the AIBO</td>
<td>As long as the program is not altered, there is no consequences for the safety</td>
<td>An unauthorized party gains access to the programmable memory stick in</td>
<td>See to a secure storage of the memory stick when not in use</td>
</tr>
<tr>
<td>Deliberate disclosure of information in PC controller</td>
<td>An unauthorized party gains access to the application code on the PC controller</td>
<td>Unauthorized copying of files and folders, but no affect on the safety as long as data are not altered. However, it opens up for breaches to integrity and authenticity</td>
<td>Missing or failing access control mechanism, or disclosure of password</td>
<td>Access control mechanism, safe/secure password management (generation and storage) (strong authentication, e.g. using smart card in addition to password)</td>
</tr>
<tr>
<td></td>
<td>An unauthorized party gains access to the control window of the PC controller</td>
<td>Unauthorized user can control the factory through the control window. No effect on safety until information is altered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliberate disclosure of messages in the WLAN communication channel</td>
<td>An unauthorized party gains access to the messages sent between AIBO and the PC controller</td>
<td>No consequence for the safety</td>
<td>An unauthorized party eavesdrops information on the WLAN communication channel</td>
<td>Encryption mechanism</td>
</tr>
<tr>
<td>Deliberate disclosure of messages in the IR communication channel</td>
<td>An unauthorized party gains access to the IR signals sent between the PC controller and the LEGO system</td>
<td>No consequence for the safety</td>
<td>An unauthorized party eavesdrops on the IR communication channel</td>
<td>Shielding of the IR communication channel</td>
</tr>
<tr>
<td>Deliberate disclosure of information in LEGO system</td>
<td>An unauthorized party gains access to the application code on the LEGO RCX</td>
<td>As long as the program is not altered, there is no consequences for safety</td>
<td>An unauthorized party gains access to the application on the LEGO RCX</td>
<td>Shielding of the RCX</td>
</tr>
</tbody>
</table>
### Table C.2: Results of Security HazOp - Breaches of integrity

<table>
<thead>
<tr>
<th>Expression</th>
<th>Security threat</th>
<th>Consequences</th>
<th>Causes</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliberate manipulation of AIBO</td>
<td>AIBO is manipulated to ignore “intruder” or to not send alerts when detecting “intruders”</td>
<td>The PC controller, and then again the LEGO system, is not notified when an intruder enters the entrance area, thus the gate fails to close and the intruder can enter the safety zone, which can lead to possible injury of a person</td>
<td>An unauthorized party gains access to the programmable memory stick in AIBO and manipulates/modifies the application program</td>
<td>See to a secure storage of the memory stick when not in use. Shielding of AIBOs lower part in order to prevent easy access to the memory stick</td>
</tr>
<tr>
<td></td>
<td>AIBO is manipulated to ignore persons entering the safety zone or to not send alerts when detecting person entering the safety zone</td>
<td>The PC controller, and then again the LEGO system, is not notified when a person enters the safety zone, thus the cutting robot fails to stop. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliberate manipulation of PC controller</td>
<td>PC controller is manipulated to ignore “intruder alert” or to not send “close gate” signal when receiving “intruder alert” from AIBO</td>
<td>The LEGO system is not notified when an intruder enters the entrance area, thus the gate fails to close and the intruder can enter the safety zone, which can lead to possible injury of a person</td>
<td>An unauthorized party gains access to the application code on the PC controller and manipulates/modifies it</td>
<td>Extend access control mechanism (strong authentication) and write-protected files</td>
</tr>
<tr>
<td></td>
<td>PC controller is manipulated to ignore “entered safety zone” signals or to not send stop signal to LEGO system when receiving “entered safety zone” signal from AIBO</td>
<td>The LEGO system is not notified when a person enters the safety zone, thus the cutting robot fails to stop. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliberate manipulation of LEGO system</td>
<td>The LEGO system is manipulated to ignore stop signals from the PC controller</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person</td>
<td>An unauthorized party gains access to the application of the LEGO RCX and manipulates it</td>
<td>??</td>
</tr>
</tbody>
</table>

### Table C.3: Results of Security HazOp - Breaches of integrity cont.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Security threat</th>
<th>Consequences</th>
<th>Causes</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliberate manipulation of messages in the WLAN communication channel</td>
<td>An “intruder alert” is modified to a “regular” message</td>
<td>The gate fails to close when an intruder enters the entrance area. The intruder can then enter the safety zone</td>
<td>The receiver on the PC controller is blocked while another receiver receives the transmitted message and modifies its content before unblocking and forwarding the modified message to the PC controller</td>
<td>Encryption of transmitted messages over the WLAN, shielding of the WLAN communication channel, and use of timestamps</td>
</tr>
<tr>
<td></td>
<td>An “entered safety zone” signal is modified to a “regular” message</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>An “entered safety zone” signal is modified to an “intruder alert”</td>
<td>The gate closes instead of robot stopping. Because of time delay, the person may be able to enter before the gate is closed. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliberate manipulation of messages in the IR communication channel</td>
<td>A stop signal is modified to a start signal</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person</td>
<td>The IR receiver on the RCX is blocked while another receiver receives the transmitted message and modifies it before unblocking the RCX and forwarding the modified message</td>
<td>Shielding of the IR communication link</td>
</tr>
<tr>
<td></td>
<td>A “close gate” signal is modified to a stop or start signal</td>
<td>The gate fails to close when an intruder enters the entrance area and the intruder can then enter the safety zone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>Security threat</td>
<td>Consequences</td>
<td>Causes</td>
<td>Countermeasures</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>--------------</td>
<td>--------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Deliberate denial of alerts/messages from AIBO</td>
<td>AIBO is prevented from detecting intruder in the entrance area, or person entering the safety zone. AIBO is prevented from sending messages to the PC controller.</td>
<td>The monitoring part of the system is unavailable. The gate will never close, and the cutting robot will not stop if someone enters the safety zone. Possible death or injury of person.</td>
<td>Sabotage, AIBO is shut down. Denial of Service attack. AIBO is stolen/moved.</td>
<td>Shut down system/stop cutting robot if AIBO is unavailable/connection is lost.</td>
</tr>
<tr>
<td>Unintentional denial of alerts/messages from AIBO</td>
<td>AIBO is prevented from detecting intruder in the entrance area, or person entering the safety zone. AIBO is prevented from sending messages to the PC controller.</td>
<td>The monitoring part of the system is unavailable. The gate will never close, and the cutting robot will not stop if someone enters the safety zone. Possible death or injury of person.</td>
<td>Natural disaster, AIBO has been shut down accidentally because of low battery power.</td>
<td>Shut down system/stop cutting robot if AIBO is unavailable or connection is lost.</td>
</tr>
<tr>
<td>Deliberate delay of alerts/messages from AIBO</td>
<td>PC controller receives the alert messages from AIBO too late.</td>
<td>The gate may close too late, and the cutting robot may stop too late if someone enters the safety zone. Possible death or injury of person.</td>
<td>Denial of Service attack.</td>
<td>Timestamp, alive time, and message number with alert message is “message is out of sequence”.</td>
</tr>
<tr>
<td>Unintentional delay of alerts/messages from AIBO</td>
<td>PC controller receives the alert messages from AIBO too late.</td>
<td>The gate may close too late, and the cutting robot may stop too late if someone enters the safety zone. Possible death or injury of person.</td>
<td>Technical failure.</td>
<td></td>
</tr>
<tr>
<td>Deliberate denial or delay of messages from PC controller</td>
<td>The PC controller is unavailable/prevented from performing its normal operation.</td>
<td>The linking between AIBO and the LEGO system is broken. The gate will never close or close too late, and the cutting robot will not stop or stop too late if someone enters the safety zone. Possible death or injury of person.</td>
<td>Sabotage, power shutdown, PC is shut down. Denial of Service.</td>
<td>Shut down system/stop cutting robot if connection is lost (timeout).</td>
</tr>
<tr>
<td>Unintentional denial or delay of messages from PC controller</td>
<td>The PC controller is unavailable/prevented from performing its normal operation.</td>
<td>The linking between AIBO and the LEGO system is broken. The gate will never close or close too late, and the cutting robot will not stop or stop too late if someone enters the safety zone. Possible death or injury of person.</td>
<td>Natural disaster, technical failure.</td>
<td>Shut down system/stop cutting robot if connection is lost (timeout).</td>
</tr>
<tr>
<td>Expression</td>
<td>Security threat</td>
<td>Consequences</td>
<td>Causes</td>
<td>Countermeasures</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>--------------</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>Deliberate denial or delay of messages in the LEGO system</td>
<td>Actions are not carried out or are carried out too late in the LEGO system</td>
<td>The LEGO system is unavailable. The gate will not close or close too late, and the cutting robot will not stop or stop too late. Possible death or injury of a person</td>
<td>Sabotage, Denial of Service</td>
<td></td>
</tr>
<tr>
<td>Unintentional denial or delay of messages in the LEGO system</td>
<td>Actions are not carried out or are carried out too late in the LEGO system</td>
<td>The LEGO system is unavailable. The gate will not close or close too late, and the cutting robot will not stop or stop too late. Possible death or injury of a person</td>
<td>Low battery level in the LEGO RCX, natural disaster</td>
<td></td>
</tr>
<tr>
<td>Deliberate denial/delay of messages in the WLAN communication channel</td>
<td>An “intruder alert” from AIBO to the PC controller is lost or delayed</td>
<td>The gate is not closed or closes too late when an intruder enters the entrance area. Possible injury of person</td>
<td>Interruption of signals/Interference of radio signals by other senders/Transmitted data destroyed by other electrical device</td>
<td>Shielding of the wireless communication link</td>
</tr>
<tr>
<td></td>
<td>An “entered safety zone” signal from AIBO to the PC controller is lost or delayed</td>
<td>The cutting robot is not stopped or stops too late when a person enters the safety zone. Possible death or injury of person</td>
<td>Errors in configuration settings of network or modified network settings, WLAN card removed from PC or AIBO</td>
<td>Direct communication between AIBO and LEGO system, Frequent control signals from AIBO - control system should shut down system if control signal is lost</td>
</tr>
<tr>
<td></td>
<td>The communication link becomes unavailable</td>
<td>The gate is not closed when an intruder enters the entrance area and the cutting robot is not stopped when a person enters the safety zone. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliberate denial/delay of messages in the IR communication channel</td>
<td>A &quot;close gate&quot; signal from the PC controller is lost or delayed</td>
<td>The gate is not closed or closes too late when an intruder enters the entrance area. The intruder can then enter the safety zone, which can lead to injury.</td>
<td>Physical blocking of IR signals</td>
<td>Shielding of the IR communication link</td>
</tr>
<tr>
<td></td>
<td>A stop signal from the PC controller is lost or delayed</td>
<td>The robot is not stopped or stops too late when a person enters the safety zone. Possible death or injury of person</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>Security threat</td>
<td>Consequences</td>
<td>Countermeasures</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>--------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>Deliberate fabrication/masquerade of AIBO</td>
<td>Unauthorized user/component acts as AIBO and fabricates messages</td>
<td>Masquerade AIBO sends message that staff or intruder is not in entrance area or safety zone any longer when they still are present, which can lead to possible death or injury</td>
<td>Masquerade AIBO (false AIBO) has taken over the role of the real AIBO</td>
<td>Implement authentication mechanism between AIBO and PC controller</td>
</tr>
<tr>
<td>Deliberate fabrication/masquerade of PC controller</td>
<td>Unauthorized user/component acts as PC controller and fabricates messages</td>
<td>PC controller sends start message to LEGO system while someone is in the safety zone, which can lead to possible death or injury</td>
<td>Masquerade PC controller (false PC controller) has taken over the role of the real PC controller</td>
<td>Implement authentication mechanism between PC controller and LEGO system</td>
</tr>
<tr>
<td>Deliberate fabrication/masquerade of messages in the WLAN communication channel</td>
<td>A false “intruder alert” is inserted into the wireless LAN</td>
<td>The PC controller assumes that an intruder has entered the entrance area, and sends a “close gate” signal to the LEGO system. The gate is closed needlessly, but no effect on safety</td>
<td>Not considered because it does not affect the safety of the system</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A false “entered safety zone” signal is inserted into the wireless LAN</td>
<td>The PC controller assumes that a person has entered the safety zone, and sends a stop signal to the LEGO system. The cutting robot is stopped needlessly, but no effect on safety</td>
<td>Not considered because it does not affect the safety of the system</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A false “disconnect” signal for AIBO is inserted into the wireless LAN</td>
<td>AIBO becomes unavailable (for more information see Table C.4 in Appendix C)</td>
<td>Not considered because it does not affect the safety of the system</td>
<td></td>
</tr>
<tr>
<td>Deliberate fabrication/masquerade of messages in the IR communication channel</td>
<td>A false stop signal is inserted into the IR communication channel</td>
<td>The cutting robot is stopped needlessly, but no effect on safety</td>
<td>An unauthorized party masquerades as the PC controller and sends spurious messages to the LEGO system</td>
<td>Shielding of the IR communication channel</td>
</tr>
<tr>
<td></td>
<td>A false “close gate” signal is inserted into the IR communication channel</td>
<td>The gate is closed needlessly, but no effect on safety</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A false start signal is inserted into the IR communication channel</td>
<td>The cutting robot may be started when a person is inside the safety zone. Possible death or injury of person</td>
<td>Prevent start from control window</td>
<td></td>
</tr>
</tbody>
</table>
# B Risk Analysis Tables

These tables are taken from the appendix D of Karine Sorbys master thesis [37].

<table>
<thead>
<tr>
<th>Breaches of Confidentiality</th>
<th>Safety consequences</th>
<th>Consequence value</th>
<th>Likelihood value</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>An unauthorized party gets access to the application code on the AIBO.</td>
<td>As long as the program is not altered, there is no consequences for the safety.</td>
<td>Negligible</td>
<td>Improbable</td>
<td>IV</td>
</tr>
<tr>
<td>An unauthorized party gets access to the application code on the PC controller.</td>
<td>Unauthorized copying of files and folders, but no affect on the safety as long as data are not altered. However, it opens up for breaches to integrity and authenticity.</td>
<td>Negligible</td>
<td>Occasional</td>
<td>IV</td>
</tr>
<tr>
<td>An unauthorized party gains access to the control window of the PC controller.</td>
<td>Unauthorized user can control the factory through the control window. No affect on safety until information is altered.</td>
<td>Negligible</td>
<td>Probable</td>
<td>IV</td>
</tr>
<tr>
<td>An unauthorized party gains access to the messages sent between AIBO and the PC controller.</td>
<td>No consequence for the safety.</td>
<td>Negligible</td>
<td>Occasional</td>
<td>IV</td>
</tr>
<tr>
<td>Unauthorized party gains access to the IR signals sent between the PC controller and the LEGO system.</td>
<td>No consequence for the safety.</td>
<td>Negligible</td>
<td>Occasional</td>
<td>IV</td>
</tr>
<tr>
<td>An unauthorized party gets access to the application code on the LEGO RCX.</td>
<td>As long as the program is not altered there is no consequences for the safety.</td>
<td>Negligible</td>
<td>Remote</td>
<td>IV</td>
</tr>
<tr>
<td>Breaches of Integrity</td>
<td>Security threat</td>
<td>Safety consequences</td>
<td>Consequence value</td>
<td>Likelihood value</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>AIBO is manipulated to ignore &quot;intruders&quot; or to not send alerts when detecting &quot;intruders&quot;.</td>
<td>The PC controller, and then again the LEGO system, is not notified when an intruder enters the entrance area, thus the gate fails to close and the intruder can enter the safety zone, which can lead to possible injury of a person.</td>
<td>Catastrophic</td>
<td>Improbable</td>
</tr>
<tr>
<td></td>
<td>AIBO is manipulated to ignore persons entering the safety zone or to not send alerts when detecting person entering the safety zone.</td>
<td>The PC controller, and then again the LEGO system, is not notified when a person enters the safety zone, thus the cutting robot fails to stop. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Improbable</td>
</tr>
<tr>
<td></td>
<td>PC controller is manipulated to ignore &quot;intruder alert&quot; or to not send &quot;close gate&quot; signal when receiving &quot;intruder alert&quot; from AIBO.</td>
<td>The LEGO system is not notified when an intruder enters the entrance area, thus the gate fails to close and the intruder can then enter the safety zone, which can lead to possible injury of a person.</td>
<td>Critical</td>
<td>Occasional</td>
</tr>
<tr>
<td></td>
<td>PC controller is manipulated to ignore &quot;entered safety zone&quot; signals or to not send stop signal to LEGO system when receiving an &quot;entered safety zone&quot; signal from AIBO.</td>
<td>The LEGO system is not notified when a person enters the safety zone, thus the cutting robot fails to stop. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Occasional</td>
</tr>
<tr>
<td></td>
<td>The LEGO system is manipulated to ignore stop signals from the PC controller.</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Incredible</td>
</tr>
<tr>
<td></td>
<td>The LEGO system is manipulated to never stop the cutting robot.</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of a person</td>
<td>Catastrophic</td>
<td>Incredible</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Breaches of Integrity cont.</th>
<th>Security threat</th>
<th>Safety consequences</th>
<th>Consequence value</th>
<th>Likelihood value</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An &quot;intruder alert&quot; is modified to a &quot;regular&quot; message.</td>
<td>The gate fails to close when an intruder enters the entrance area. The intruder can then enter the safety zone, which can lead to injury.</td>
<td>Critical</td>
<td>Improbable</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>An &quot;entered safety zone&quot; signal is modified to a &quot;regular&quot; message.</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Improbable</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>An &quot;entered safety zone&quot; signal is modified to an &quot;intruder alert&quot;.</td>
<td>The gate closes instead of robot stopping. Because of time delay, the person may be able to enter before the gate is closed. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Improbable</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>A stop signal is modified to a start signal.</td>
<td>The cutting robot fails to stop when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Remote</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>A &quot;close gate&quot; signal is modified to a stop or start signal.</td>
<td>The gate fails to close when an intruder enters the entrance area and the intruder can then enter the safety zone, which can lead to injury.</td>
<td>Critical</td>
<td>Remote</td>
<td>III</td>
</tr>
<tr>
<td>Breaches of Availability</td>
<td>Security threat</td>
<td>Safety consequences</td>
<td>Consequence value</td>
<td>Likelihood value</td>
<td>Classification</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>---------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>AIBO is prevented from detecting intruder in the entrance area, or person entering the safety zone.</td>
<td>The monitoring part of the system is unavailable. The gate will never close, and the cutting robot will not stop if someone enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Probable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AIBO is prevented from sending messages to the PC controller.</td>
<td>The gate may close too late, and the cutting robot may step too late when someone enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Occasional</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>PC controller receives the alert message from AIBO too late.</td>
<td>The linking between AIBO and the LEGO system is broken. The gate will never close or closes too late, and the cutting robot will not stop or stops too late when someone enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Frequent</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>The PC controller is unavailable/prevented from performing its normal operation.</td>
<td>The LEGO system is unavailable. The gate will not close or close too late and the cutting robot will not stop or stop too late. Possible death or injury of a person.</td>
<td>Catastrophic</td>
<td>Probable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Actions are not carried out or are carried out too late in the LEGO system.</td>
<td>The LEGO system is unavailable. The gate will not close or close too late when an intruder enters the entrance area. Possible injury of person.</td>
<td>Critical</td>
<td>Probable</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Breaches of Availability cont.</th>
<th>Security threat</th>
<th>Safety consequences</th>
<th>Consequence value</th>
<th>Likelihood value</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An “intruder alert” from AIBO to the PC controller is lost or delayed.</td>
<td>The gate is not closed or closes too late when an intruder enters the entrance area. Possible injury of person.</td>
<td>Critical</td>
<td>Probable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>An “entered safety zone” signal from AIBO to the PC controller is lost or delayed.</td>
<td>The cutting robot is not stopped or stops too late when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Probable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>The (WLAN) communication link becomes unavailable.</td>
<td>The gate is not closed when an intruder enters the entrance area and the cutting robot is not stopped when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Probable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A “close gate” signal from the PC is lost or delayed.</td>
<td>The gate is not closed or closes too late when an intruder enters the entrance area. Possible death or injury of person.</td>
<td>Critical</td>
<td>Frequent</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A stop signal from the PC controller is lost or delayed.</td>
<td>The robot is not stopped or stops too late when a person enters the safety zone. Possible death or injury of person.</td>
<td>Catastrophic</td>
<td>Frequent</td>
<td>1</td>
</tr>
<tr>
<td>Security threat</td>
<td>Safety consequence</td>
<td>Consequence value</td>
<td>Likelihood value</td>
<td>Classification</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Unauthorized user/component acts as AIBO and fabricates messages.</td>
<td>Masquerade AIBO sends message that staff or intruder is not in entrance area or safety zone any longer when they still are present, which can lead to possible death or injury.</td>
<td>Catastrophic</td>
<td>Remote</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td>Unauthorized user/component acts as PC controller and fabricates messages.</td>
<td>PC controller sends start message to LEGO system while someone is in the safety zone, which can lead to possible death or injury.</td>
<td>Catastrophic</td>
<td>Remote</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td>A false “intruder alert” is inserted into the wireless LAN.</td>
<td>The PC controller assumes that an intruder has entered the entrance area, and sends a “close gate” signal to the LEGO system. The gate is closed needlessly, but no effect on safety.</td>
<td>Negligible</td>
<td>Remote</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>A false “entered safety zone” signal is inserted into the wireless LAN.</td>
<td>The PC controller assumes that a person has entered the safety zone, and sends a stop signal to the LEGO system. The cutting robot is stopped needlessly, but no effect on safety.</td>
<td>Negligible</td>
<td>Remote</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>A false “disconnect” signal for AIBO is inserted into the wireless LAN.</td>
<td>AIBO becomes unavailable (for more information see Table C.4 in Appendix C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A false stop signal is inserted into the IR communication channel.</td>
<td>The cutting robot is stopped needlessly, but no effect on safety.</td>
<td>Negligible</td>
<td>Occasional</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>A false “close gate” signal is inserted into the IR communication channel.</td>
<td>The gate is closed needlessly, but no effect on safety.</td>
<td>Negligible</td>
<td>Occasional</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>A false start signal is inserted into the IR communication channel.</td>
<td>The cutting robot may be started when a person is inside the safety zone. Possible death or injury of person</td>
<td>Catastrophic</td>
<td>Occasional</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>
C REFERENCES


112
[27] IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems
[38] Karine Sorby. Relationship between security and safety in a security-safety critical system: Safety consequences of security threats, NTNU, 2003, Figure 9.3 page 51.
[44] Mary Shaw, David Garlan, Robert Allen, Dan Klein, John Ockerbloom, Curtis Scott, Marco Schumacher, Candidate Model Problems in Software Architecture, Discussion draft 1.3 in circulation for development community consensus. 1994