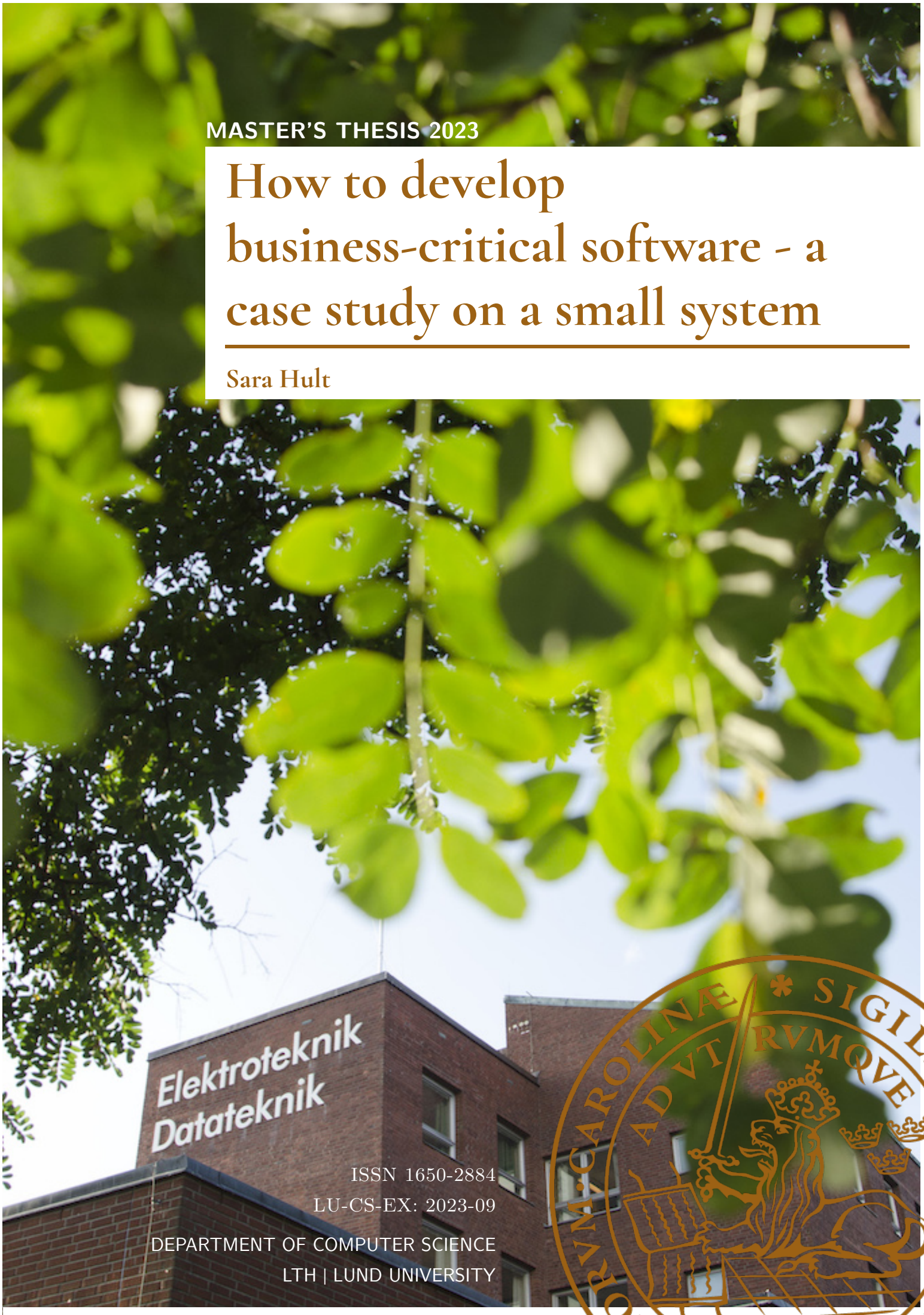

MASTER'S THESIS 2023

How to develop business-critical software - a case study on a small system

Sara Hult

ISSN 1650-2884
LU-CS-EX: 2023-09

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-09

How to develop business-critical software - a case study on a small system

Hur man utvecklar affärskritiska system - en fallstudie av ett mindre system

Sara Hult

How to develop business-critical software - a case study on a small system

Sara Hult
sa4617hu-s@lu.se

April 3, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Martin Höst, martin.host@cs.lth.se

Examiner: Emma Söderberg, emma.soderberg@cs.lth.se

Abstract

The term business-critical defines systems and software whose failure may lead to loss of business or damage to reputation in the market [1]. It can include the majority of systems and software in a business, yet the term is rarely used in literature and at companies. This thesis investigates the scope of the term, how business-critical software is developed, and what experiences can be found by implementing business-critical software techniques and methods on a small system at a company in Sweden.

First, a literature study was conducted followed by a case study where three methods found in the literature study were implemented and examined on a smaller system. Thoughts and experiences were collected through interviews with identified stakeholders and a case study protocol.

The result show that there are several methods, techniques, and approaches for developing business-critical systems and software and that the three methods implemented to varying extent, all contributed to making the system safer and more reliable, however, none solved all the issues.

This implies that none of the methods tried are optimal alone and that the use of several methods might be a good approach for developing business-critical software.

The findings of this thesis provide insight into various methods and techniques that can be used to develop business-critical software and contributes with thoughts, experiences and opinions regarding a few of these methods, which can help others make more informed decisions and develop safer and more reliable systems.

Keywords: Business-critical, critical systems, N-version programming, risk analysis, code analysis

Acknowledgements

I would like to thank my supervisor Martin Höst for all the support and guidance in the process of this thesis. Martin's knowledge and experience has helped tremendously and his feedback has been invaluable.

I would also like to thank my supervisors at the case company for their help and support throughout this thesis. A huge thank you must also be given to the IT-department and other employees at the case company. They have all been of great support, helped with interviews and have always been there to answer any questions that have arisen.

Contents

Chapter 1
Introduction

The following chapter gives an introduction to the background of the thesis where the purpose, research questions, limitations, and outline is presented.

1.1 Purpose

The purpose of this thesis is to contribute to literature regarding the term business-critical and to contribute with experiences of methods that can be used. The aim is to give an overview of the term and examine how well used and widespread it is, and to examine methods and techniques for developing business-critical software and systems.

This is done by a literature study followed by a case study were a few of the methods found, will be implemented and evaluated in a case at a large company in Sweden.

1.2 Research Questions

The two research questions addressed in this report are:

- **RQ1:** How are business-critical software developed today and which techniques exist for developing business-critical software?
- **RQ2:** What experiences can be found by implementing business-critical software methods at a company?

1.3 Limitations

Due to the wide topic of the thesis, several limitations were set. Limitations for the literature study consisted of primarily focusing on business- and safety-critical systems and software

and not on security- and mission-critical systems. Security-critical systems were not examined in depth due to the term not occurring consistently in the literature. Sommerville [2], for example, a well known author in the area of software engineering, only mention business-, safety-, and mission-critical systems. Mission-critical systems were also not investigated thoroughly due to the term considered to be too far away from the scope of business-critical systems, in opposite to safety-critical systems which often seemed to overlap with business-critical systems.

For the case study, the methods chosen were limited due to restrictions, set environments, and tools at the case company. Methods had to function on a small project and only on a personal, project or architectural level since implementing methods on an organisational level was not feasible. The focus of the case study was on planning and implementation, and not initiation and deployment since some of the initiation had already been made and deployment had been too big a project. Only a few specific languages could be used and the application could not be web- or cloud-based.

1.4 Outline

This thesis begins with a literature study found in Chapter 2, in form of a chapter of background and related work regarding the terms business critical and critical systems and software. It is followed by Chapter 3 discussing and explaining the method of the study; both the literature and case study and the result of both studies is found in Chapter 4. Lastly, the discussion is found in Chapter 5 and the conclusion and related work in Chapter 6.

Chapter 2
Background and Related Work

The following chapter introduces the concepts of critical systems, and gives an overview of its methods and approaches. The term business-critical is further described as well as a few methods used for developing, analysing, and validating business-critical and critical software and systems.

Both critical *software* and *systems* are examined in the following chapter and throughout this thesis to gain a wider understanding of the concept of business-critical and other critical software. System is a term used to describe a whole rather than a collection of parts but is used in various different ways which may lead to misunderstandings [3]. Software is also becoming more and more important for systems [4] and both safety- and security critical software systems are becoming increasingly software intensive [5]. This resulted in using both terms when searching for literature and interchangeably throughout the thesis.

2.1 Critical systems

Critical systems can be divided into different types; safety-critical, mission-critical, security-critical, and business-critical [2, 6]. What all critical systems have in common is that their failure may lead to major consequences such as injuries, damage to the environment, spread of secret information, loss of business or damage to the reputation on the market. They all have a high cost attached to a failure and must to a certain degree, be secure and safe for a failure not to happen.

What the failure can *lead to* is the main difference between the systems. A safety-critical system is a system whose failure may result in loss of life, significant property damage or damage to the environment and can be an aircraft- or nuclear plant system. A mission-critical system's failure may instead lead to failure of a goal-directed activity and describes systems such as a navigation system for a spacecraft or a robot [7] while a security-critical system may lead to loss of sensitive and crucial data by theft or accidental loss [6].

For business-critical systems, a few definitions were found in the literature:

> "A business-critical system is a software, or software/hardware, system whose correct operation is crucial to a business or enterprise." [8]

> "Business critical systems (BCS) are those systems whose failure may lead to loss of business or damage to reputation in the market." [1]

> "A system whose failure may result in very high costs for the business using that system. An example of a business-critical system is the customer accounting system in a bank. Business-critical systems may be affected by security-related failures." [2]

The similarity between the descriptions is that they all talk about the criticality of the system operating in a correct manner. However, the two latter descriptions also mention the consequences of the system not working properly and what it can lead to. The second definition was by the author considered to be the most inclusive and distinct defining both loss of business and damage to reputation in the market and it was therefore the definition chosen to be used in this thesis.

The term business-critical is by far the least used out of the four terms which can be viewed in Table 2.1 and Table 2.2. The term with the largest scope of the different critical system terms is safety-critical which has around 69 600 hits on Google Scholar and 10 888 hits on Scopus. Google Scholar is a search tool for scholarly literature and Scopus is an abstract and citation database of peer-reviewed literature. The *-notaion means that the search includes all terms and words that start with the word prior to the symbol, in Table 2.1 both safety-critical *system* and safety-critical *systems* will be included in the search.

Table 2.1: Search results for safety/mission/security/business-critical *system**.

System	Scopus	Google Scholar
Safety-critical system*	10 888	69 600
Mission-critical system*	3 382	15 000
Security-critical system*	885	2 720
Business-critical system*	320	1 600

Table 2.2: Search results for safety/mission/security/business-critical *software*.

Software	Scopus	Google Scholar
Safety-critical software	4 584	17 800
Mission-critical software	1 121	3 660
Security-critical software	465	952
Business-critical software	151	460

2.2 Business-critical

The term business-critical can be applied to various areas such as systems, software, services, operations, and applications [9, 10, 11, 12] and has been around since the 90's, e.g., [13]. A business-critical system can, as the definitions states, be anything from a banking-system to an e-commerce system and the difference from a safety-critical system is that the latter could result in loss of life, while a business-critical system could lead to loss of business or damage the reputation of the business. A very clear and recent example of a business-critical system and the effects of it not being secure enough, is the IT-attack on the Swedish grocery store Coop in 2021. The attack lead to Coop's register system being unusable and the majority of their stores having to close for a few days [14].

2.2.1 Key areas, keywords, and methods

In this thesis, the terms business-critical *systems* and *software* were of primary interest and were also included in the majority of the searches.

Two main searches, and within those several smaller ones, were conducted to find various key areas, keywords, and methods within the scope of critical software and systems. One main search was conducted within the scope of business-critical and one within the scope of safety-critical. The steps taken to conduct the searches can be seen in Section 3.1.1.

From the search process presented, four tables were created stating various keywords and methods and their percentage occurrence within the two main searches. Table 2.3 presents various key areas and keywords in the search for methods and techniques for business-critical software or systems. Each keyword in the table is presented along with the search term used, the amount of article (hits) found for the keyword, and the keyword's percentage occurrence when compared to the larger main search.

Table 2.4 presents the same result, but for various *methods* instead of *keywords*.

Table 2.5 was conducted in the same way as Table 2.3 presenting various keywords, only changing and searching for *safety*-critical instead of *business*-critical.

Searching for safety-critical instead of business-critical was also done to conduct Table 2.6 which contains methods instead of keywords, identical to Table 2.4.

The four tables was an attempt of showing the differences in business- and safety critical systems and software and what terms, keywords, and methods are most commonly used.

Many articles regarding business-critical systems discuss *web*-based systems, e.g., [15] and the term was one of the most recurring ones for business-critical systems. Web-based systems can be different from developing a traditional IT-system since a web-based system is primarily an interaction medium where development of new applications include a great deal of cloning existing components [16]. Since one limitation of this thesis were no web-based interaction application, these articles were merely considered as inspiration for methods and to gain understanding of the subject.

Test being one of the keywords in the scope of business-critical systems and software might not come as a surprise. Developing and keeping a critical system safe tend to include testing to ensure that it meets specified safety criteria.

Various methods used in the scope of business-critical software and systems were not commonly used as keywords, which can be seen in Table 2.4 where all methods had a percentage occurrence of 2.0 % or lower. The method with the most amount of hits were *formal meth-*

ods with an occurrence of 2.0 % which were the most reoccurring method for safety-critical systems as well, see Table 2.6. Formal methods along with other keywords and methods will be discussed further in the sections below.

Table 2.3: Search results for various *keywords* in the larger search for business-critical methods (Scopus).

Keyword	Search term	Hits	%
Test	test*	66	19 %
Web	web*	64	18 %
Reliability	reli*	55	16 %
Risk	risk*	50	14 %
Validation	vali*	40	11 %
Verification	veri*	28	8 %
Legacy	legacy	20	6 %
Agile	agil*	11	3 %

Table 2.4: Search results for various *methods* in the larger search for business-critical methods (Scopus).

Method	Search term	Hits	%
Formal methods	{formal methods}	7	2.0 %
Root-cause analysis/diagnosis	root-cause*	4	1.1 %
SDLC	SDLC	3	0.9 %
UML	UML	3	0.9 %
Modelling framework	{modelling framework}	2	0.6 %
Validation process	{validation process}	1	0.3 %
Model driven	{model driven}	5	0.7

2.2.2 Business vs Safety-critical

The main difference between safety-critical and business-critical systems is, as mentioned, the risk level where the first system can cause consequences on a personal level and the latter on a business level. However, the actual practical differences might be harder to distinguish, Piriou et al. discuss that even though the outcome is not the same for the different systems, they both have a significant impact on assets which make dependability engineering relevant for both systems [17]. The same authors also talk about the lack of standards for developing business-critical systems from a blockchain perspective and borrows concepts and tools from safety-critical systems.

Other resemblances are that both system's software many times must be in compliance with some form of regulation or policy [18, 19], that complex software is used in both systems, and that the systems are seldom subject to heavy changes and may operate over decades with limited changes [20].

When comparing the tables for the searches for business- and safety-critical systems, the differences were relatively small and showed that the keywords *test* and *reliability* were in

the top three terms for both searches. The keyword *web* was significantly more occurring in the search for business-critical keywords (Table 2.3) than in the search for safety-critical keywords (Table 2.5) while the term *risk* was about equally common in both searches.

The methods found for both searches had all an occurrence of around 2 % or lower, see Table 2.4 and 2.6, except for formal methods in the search for safety-critical methods, which shows some of the difficulties in finding and conducting an overview of methods and techniques for the terms.

Table 2.5: Search results for various *keywords* in the larger search for safety-critical methods (Scopus).

Keyword	Search term	Hits	%
Reliability	reli*	2 993	27.2 %
Test	test*	2 749	25.0 %
Verification	veri*	2 202	20.0 %
Validation	vali*	1 840	16.7 %
Risk	risk*	1 364	12.4 %
Agile	agil*	168	1.5 %
Web	web*	143	1.3 %
Legacy	legacy	93	0.8 %

Table 2.6: Search results for various *methods* in the larger search for safety-critical methods (Scopus).

Method	Search term	Hits	%
Formal methods	{formal methods}	961	8.7 %
UML	UML	232	2.1 %
Model Driven	{model driven}	139	1.3 %
Root-cause analysis/diagnosis	root-cause*	65	0.6 %
Validation process	{validation process}	47	0.4 %
SDLC	SDLC	40	0.4 %
Modelling framework	{modelling framework}	10	0.1 %

2.3 Methods, processes, and techniques

Several different techniques, methods, processes, models, and analyses exist in the area of critical systems. This thesis divide and structure the different terms and methods to achieve a better understanding of how critical systems, and primarily business-critical systems, are and can be developed. Hereafter, the term *methods* will be used for various techniques, approaches, methods, and processes.

Different methods in the scope of critical systems can be used and divided by the point in time in the process, by different system layers or by various levels such as organisational-, department-, project- or personal level. One can also divide different methods by its purpose; analysing, developing or evaluating/validating, and several standards and guidelines exists

for critical systems produced by organisations, institutes, and governments. The layers from which the division of methods was based on for this thesis, can be seen in Figure 2.1 and is one layer or point of view one can start from. Other divisions can be done based on people, processes, and tools or by infrastructure, application, and operation.

The methods found can be seen in Figure 2.2. The figures do by no mean cover the entire scope of business- and safety-critical systems but is an attempt of an overview of various ways to plan, develop, maintain, and evaluate these kind of systems. Methods and when, where, and how they are used in critical systems are many and diverse and there is no universal software engineering method or process that fits all software systems [2].

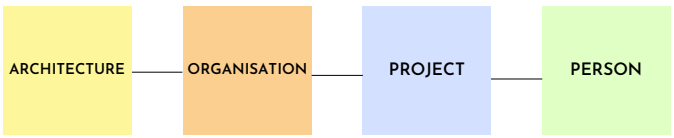

Figure 2.1: The layers from which the division of methods was based.

2.3.1 Formal methods

The development of critical systems are often done with well-tried and trusted techniques [2]. This is due to the risk and cost of failure, and older techniques are often preferred over newer ones due to their uncertainty and risks. The term formal methods is recurrent in the scope of critical systems and have been around since the end of the 1980's [21, 22, 23]. Formal methods are mathematically-based techniques designed to help development of both software and hardware systems [24]. The term originates from that the methods use *formulas* which are texts or diagrams made from symbols that are combined according to specific rules [25]. Formal methods use axiomatic notations, set theory, and first-order logic to describe and specify a system. They can be applied at different times in the development process such as for code verification in the implementation stage [26]. Some formal methods are the B-method, Safety Critical Application Development Environment (SCADE), Z, the Vienna Development Method (VDM), Temporal Logic of Actions (TLA+), and the Protoype Verification System (PVS). The complexity and cost of formal methods are sometimes mentioned as a reason not to use them in software systems while some state that they give a good return on investments [27].

2.3.2 Software Development methods

Some recurrent methods for developing critical systems and software are Software Development Methods [28], also called Software Development Life Cycle (SDLC) methods [29, 2]. Software Development Methods are primarily divided into two classes; plan-driven and agile [28] and are divided into several stages e.g. planning, analysis, design, implementation, and maintenance [30]. What most models have in common are the process activities of specification, development, validation, and evaluation. The difference is how the processes are carried out as well as how and when they are used and introduced. Some SDLC methods can be viewed in Figure 2.2 such as the Waterfall Model, V-Process Model, Iterative/Incremental-Process Model, eXtreme Programming, and Adaptive Software Development (ASD).

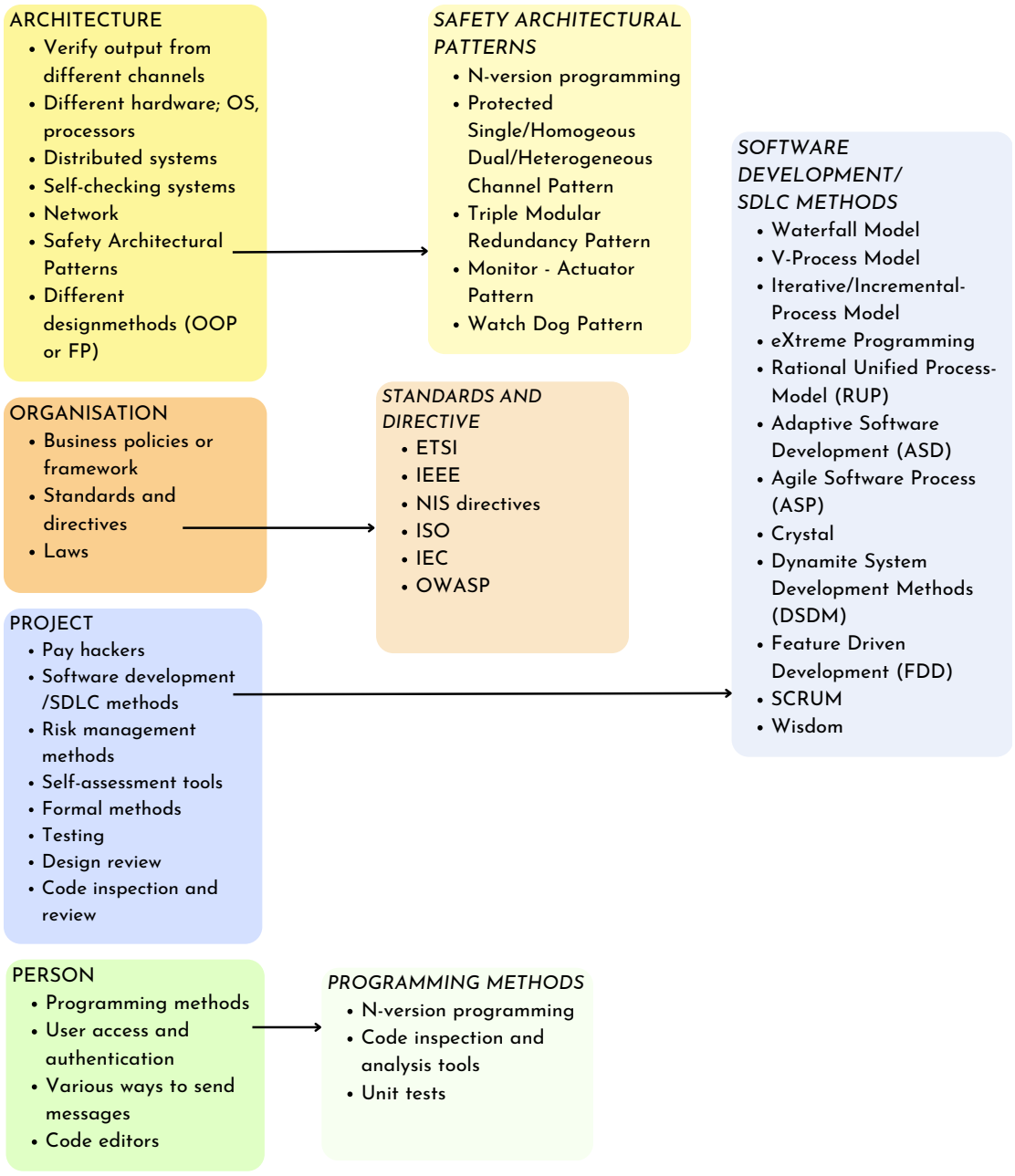

Figure 2.2: An overview of methods divided into architecture, organisation, project, and personal level.

It should be noted that some existing SDLC methods, such as SCRUM, are not primarily used for critical systems, as for example the V-Process Model is. SCRUM can be used in an uncritical system and might not be what one primarily think of for a critical system, which is the opposite to the V-process or Waterfall Model. Safety-critical systems tend to use the Waterfall Model due to the enhanced requirement of analysis and documentation before development [2] as well as the V-model/V-process Model which several standards such as IEC 61508 and IEC 62278 recommend [31, 32].

2.3.3 Programming methods

With the severe effects of failure of a critical system, the quality of the software is of great value [33] and can be improved by various methods used by the programmer, here named programming methods.

Improving the quality of the software can be done by reducing the number of faults, i.e., defects or bugs. However, to what extent minimisation of faults have an effect on the security of a system is debatable where some authors states that it has an affect while others are doubtful. It will nonetheless, decrease the probability of a failure to occur which most likely increases the reliability of the systems and Stefanović et al. mention the quality of the code as a key factor in any software product [34].

Detecting and identifying weaknesses can be done with the help of unit tests, code inspection, code review, or code analysis. The latter can be implemented using static code analysis (SCA) tools, which analyse the source code without executing it [35] to find bugs, duplicates, and various security vulnerabilities [36]. It produces reports that identify and highlights deviation from code standards and is useful for debugging tools and software development frameworks. SCA tools can also include the measurement of the complexity of the code, as for example the cyclomatic complexity which measures the number of linearly independent paths through the code [37]. When compared to manual code reviews done by software developers, the resource requirements can be vastly different [38]. Manually reviewing 1 000 or 1 000 000 lines of code is far more time consuming than letting a SCA tool do the same.

Unit tests are by many thought of as a standard procedure in modern software engineering and is widely adopted [39, 40, 41]. It is an approach for testing where smaller sets, or units, of the code are tested individually and is often measured by its percentage test coverage. Some standards, such as IEC 61508, directly or indirectly mandate unit testing, but it is a demanding technique that is both costly and time consuming [42].

N-version programming

N-version or multiversion programming is a technique where two or more versions of a program is produced from the same specifications to ensure diversity in the design and for fault detection [43]. It can be achieved by diversifying the data, the implementations, the programming languages or the design [44]. Diversifying the data means that the input is given in various forms and passed through a rewriting algorithm to produce a set of equal inputs.

Implementation is the most common method for N-version programming and means that several implementations in the same programming language are made whereas diversifying the programming language is the opposite where one or several programmers write the program in different languages. Different programming languages have different security features and can be prone to different safety issues [45, 46]. Writing in several languages could therefore help detect those issues and guarantee safer and more reliable code. Lastly, diversifying the design means that the specifications or the actual design of the program is made in several editions. Some of the implementations of N-version programming are, and can be made, on both a personal and an architectural level and the method is therefore stated at both these levels in Figure 2.2.

N-version programming is a popular technique in, for example, avionics for controlling

and operating aircraft's [47] but has been criticised by others [48]. The method can have some limitations due to the possible erroneous outputs caused by the versions being different and is in general often criticised for its high cost of implementation since it requires extra cost and hardware for the added redundancy.

2.3.4 Architectural methods

Other ways to improve the security of the code can be done at an architectural level by implementing various techniques for software fault tolerance. These techniques are based on design redundancy and diversity for the same program and can be achieved by writing different versions of the code [49], writing in different programming patterns or by using various design methods such as object-oriented or functional programming.

Software diversity as well as software verification and validation, see more in Section 2.3.6 regarding the latter, can be used to achieve lower failure rates but can not however, be used alone to reach the failure rates required in most critical systems and depending on software solely to ensure the safety of a system should be done with great care [24]. Several safety architectural patterns can be used, such as the N-version programming pattern, the protected single channel pattern, and the watch dog pattern [50].

2.3.5 Risk management

It is almost impossible to discuss critical systems without mentioning risk. A risk is an undesirable outcome which can cause a loss or an injury [2]. The process of risk management can be divided into various phases by different authors and Jaafar et al. [51] have made an evaluation of risk management process models and standards, and concluded that the most common processes in all risk management processes are: risk management plan, risk identification, risk assessment/analysis, risk mitigation/treatment/controlling/resolution, and risk monitoring.

One of the most well known sectioning of the risk process is Boehm's risk management steps which can be seen in Figure 2.3 where the first division is by risk assessment and risk control [52]. Risk assessment and analysis will be described further below while risk control will be excluded due to the focus in this thesis on development and procedure, and not on control and maintenance.

Risk assessment risk analysis

Risk assessment is the process of identifying, evaluating, and analysing possible risks and their effects [53]. There is a great value and importance to risk assessment as Deng, et al. [54] state; *"information security risk assessment is the starting point of and foothold of information security management"* which is agreed by Ying, et al.:s [55] opinion of *"risk assessment being the foundation of all security technology"*. Methods and models for designing a risk assessment process vary [56, 57, 55], and can include Protection Detection Response (PDR) Model, Attack tree Analysis (ATA), and Failure Tree Analysis (FTA).

Risk assessment can, according to Boehm's model, be further divided into three steps of identification, analysis, and prioritisation which can be subdivided into several other

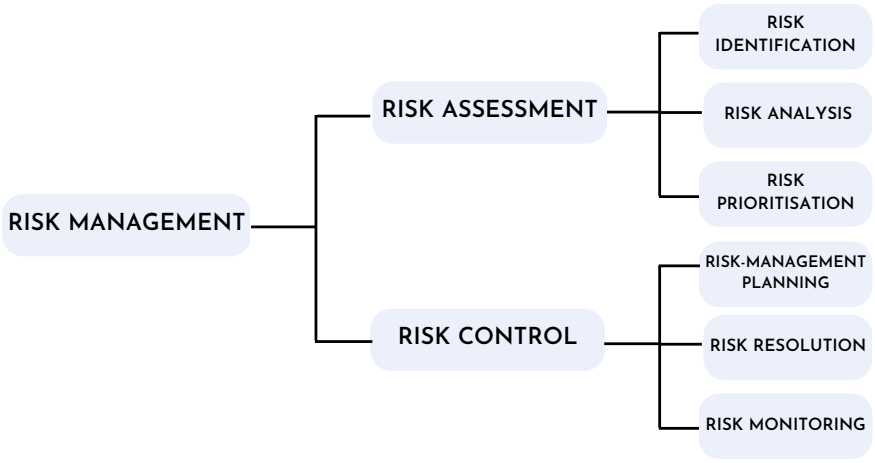

Figure 2.3: Risk process by Boehm [52].

steps [52], while Jaafar et al. [51] conclude that the process of risk analysis and risk assessment have the same functionality. A risk analysis can be performed by various methods such as CRAMM, OCTAVE, VECTOR, Bayesian Belief Network (BBN), Artificial Neural Networks (ANN), Monte Carlo analysis (MC), risk matrices or the bow-tie model [58, 59, 60, 61], and are divided into quantitative or qualitative methods[62].

2.3.6 Verification and Validation

Verification and Validation is commonly abbreviated V&V and is the process of checking and verifying that a system or software meets its requirements and discover errors [63, 64]. It is a process frequently used in critical systems and increase the quality of the software. Validation includes testing specification or software at the end of development while verification evaluates software during the various life-cycle phases to guarantee that the system or software meets the requirement set in the preceding phase. IEEE has defined a set of verification and validation activities in the IEEE Standard for Software Verification and Validation Plans, which can be divided into life-cycle phases, as done by E.A. Addy [63], and include: management of, concept phase, requirements phase, design phase, implementation phase, test phase, installation, and checkout phase, and lastly an operation and maintenance phase.

2.3.7 Software/System Reliability

Software Reliability is one of the main concerns when developing safety-critical software since a total absence of failure is an impossible goal to aim for [65]. It is therefore necessary to set, and work for a reasonable level of reliability or failure rate for the system instead of working for a failure rate of zero. Software reliability is defined as the probability that software will not cause the failure of a system for a specified time under specified conditions. The term is found within the scope of safety-critical systems (SCS) where several software reliability models (SRMs) exist in various categories such as: software reliability growth models, Bayesian Belief Network, test-based methods, input domain models, early prediction models, and correlation methods [66]. Bayesian Belief Networks are previously mentioned in

Section 2.3.5 regarding risk and is a recurring framework to use in security-related systems. There does however, not seem to be a consensus method for estimating software reliability and ways to improve and estimate software reliability have been attempted for several decades [67].

2.3.8 Standards and directives

Several standards and directives have been found in the scope of critical systems. Many researchers, especially in safety-critical systems, mention standards in the aviation industry such as DO-178B [68]. The International Organisation for Standardisation (ISO), and International Electrotechnical Commission (IEC) have several standards related to critical systems and software, such as IEC 61508, ISO 31000, and ISO 27000 [69, 70, 71]. IEC 61508 is a safety standard for electrical, electronic, and programmable electronic safety system while ISO 31000 is a standard intended to help companies and organisations to integrate risk managements [72, 70, 73]. ISO 27000 is a series of standards for information security in a business context and states terms and definitions where one of its standards, ISO 27001, contains the more general requirements [74].

Chapter 3
Method

The following chapter describes the methods taken to answer the research questions. First, background research in form of a literature study and interviews were conducted and is presented in Section 3.1.2 and secondly, a case study was carried out at a large company in Sweden which is presented in Section 3.2 .

3.1 Background research

3.1.1 Literature study

Three different sources were used to find relevant literature online; Scopus (`www.scopus.com`), LUBsearch (`https://lubsearch.lub.lu.se/`) and Google Scholar (`www.scholar.google.com`). Scopus is a database of primarily peer-reviewed literature and acted as the main search source. The book 'Software Engineering' by Ian Somerville [2] was also used and predominantly chapters 9-13 which were thoroughly read.

A systematic literature review was not, and would not have been possible, to conduct in the given time frame. However, inspiration from methods and the structure was taken and implemented from Kitchenham's paper regarding guidelines for a systematic literature review [75]. The study followed the three main phases of; planning the review, conducting the review and reporting the review. The planning consisted of identifying the need of a review and stating the research questions. Conducting the review primarily consisted of creating a search strategy and iteratively updating it. The search strategy was to some extent documented, even though digital libraries were primarily used where searches can be hard to replicate [75].

Searching for literature is usually an iterative process and this process was no exception. Many different search scopes were examined and refined during the course of conducting the study. A large search scope was initially used and followed the procedure of using several AND and OR statements to retrieve relevant literature. As most processes of finding relevant

information, the first few searches were large and broad, from 380 000 to 10 000 articles. The abstract of the most relevant articles were read, and some articles were skimmed to get a better understanding of the scope of safe software development and critical systems. The search was then narrowed down to several smaller searches and different areas such as business/safety-critical, safe/secure/critical business, critical systems and their techniques, SCADA systems and formal methods. Some areas were quickly searched and understood, like what a SCADA system is and if it is relevant to examine for this study, while others demanded a more thorough review. Some of the most used keywords were; *business-critical, safety-critical, software, development, methods, techniques, critical systems*.

As the search narrowed down, the final stage of reporting the review started and both phases were then carried out simultaneously.

Constructing the Tables

Business-critical *systems* and *software* was the main focus when searching for literature in the area of business-critical systems and those keywords were included when investigating which methods, techniques, and approaches exist in the field.

One of the main searches for the literature study can be seen below. It was a search conducted to find key ares, keywords and methods in the scope of business- and safety-critical systems and software.

```
TITLE-ABS-KEY((business-critical) AND (techniq* OR method* OR approach*)
AND (system* OR software) )
```

The first part, TITLE-ABS-KEY, defined which parts of the article would be searched through which was the title (TITLE), the abstract (ABS) and the keywords (KEY). The AND statements divided the search into different sections and each section could include various OR statements to search for closely related words such as methods/techniques/approaches. For example in the search above, both searches including the words *business-critical techniques system* and *business-critical methods software* would be found. The search was conducted on Scopus and included 362 articles by Fall of 2022.

The abstract of the first approximately 100 articles were skimmed through where various key areas, keywords, and methods found were written down. One list for key areas and keywords and one list for methods were conducted. This was done so due to some finds being clear and stated as methods, and some finds being more of a term or an area seeming to be important or having a high occurrence in the scope.

The lists were not conduced in a structured way, and the key areas, keywords, and methods chosen to be written down were chosen based on the author's knowledge and understanding of the abstract. For example, one abstract stated "*Right selection of SDLC-Methodology using a decision support tool can and will help successful completion of business critical software development*" [29], from this the author decided to write down SDLC to the list of methods. If an abstract or potential methods or keywords were not instantly understood, the entire article was skimmed to see if the article, keyword or method had any relevance in the scope of methods for business-critical software or systems.

Once the two lists had been created, each key area, keyword and method were added one by one to the search string. An example of which can be seen below for the word *risk*.

TITLE-ABS-KEY((business-critical) AND (techniq* OR method* OR approach*) AND (system* OR software) AND (risk*))

Table 2.3 shows the different key areas and keywords found and the percentage occurrence of the keyword in the main search. The percentage was calculated by dividing the amount of articles found when *including* the keyword and when not. For example, the search above including the term *risk* found 50 articles, which divided by 362 (the amount of articles in the main search for business-critical) landed on 14 %. Table 2.4 was conducted in the same way using the list of methods instead of the list of key areas and keywords. It presents various methods found and their percentage occurrence in the main search.

The same procedure was then made for the term *safety* instead of *business*. The main search used can be seen below and included 11 016 articles by Fall of 2022.

TITLE-ABS-KEY((safety-critical) AND (techniq* OR method* OR approach*) AND (system* OR software))

Table 2.5 and 2.6 were conducted in the same way as the prior tables by adding, and then removing, each keyword and method one at a time to the main search string to see their occurrence.

3.1.2 Interviews

Three interviews were held to complement the literature search and to gather information regarding which methods are used in the corporate world and how well-known and used the term business-critical is. All interviews were held at the case company with two employees working with IT and development who will be referred to as J1 and J2, and one employee working with risk management who will be referred to as J3. All three had several years of work experience within their fields. The interviews and notes were conducted in Swedish.

All interviews held were of the semi-structured nature where questions are planned but not necessarily asked in a specific order which allows for exploration and improvisation [76]. Semi-structured interviews were chosen to allow for the participants to talk freely and to not have a specific structure of the interview. The following questions and areas were addressed during the interviews:

- How do you handle business-critical software or systems?
- What does your risk management process look like?
- How do you handle safety-critical software or systems?
- Difference between business- and safety-critical systems.
- Which methods or techniques do you use to achieve the security level you want?
- How and when have you reached "safe enough"?
- Do you follow any standards, requirements, regulations, directives, frameworks or laws?

- Which methods or techniques do you use to make sure the software and code is secure?

It is recommended to record one's interviews to make sure no information is missed or misinterpreted, and afterwards transcribing them. The interviews held for this thesis were not transcribed but notes were taken. All interviewees were told and guaranteed that their data would be anonymised and only data not sensitive and critical to the case company or themselves would be used. After the interviews had been held, a revised version of the notes was sent back to each interviewee for approval if desired. The notes were revised due to security reasons where sensitive information had been removed and only the revised notes were used for the study.

3.2 Case study

Software engineering is a multidisciplinary area and is often suitable for a case study research [76]. This case study followed the often used process consisting of the five steps; case study design, preparation for data collection, collecting evidence, analysis of collected data and reporting.

3.2.1 Case company

The case study was carried out in the IT-department of a large company in Sweden. The department works with IT development, infrastructure, support, and administration and all positions are security rated due to the company's vital societal function. This meant that more specific information regarding the implementation of the case study and the case company as well as some information from the interviews conducted, had to be amended and reviewed before they could be included in the thesis.

3.2.2 The case

The case from which the case study was based, was a problem brought to the IT-department by another department at the company that wanted to streamline communication during critical time and events. The software and hardware were therefore business-critical and had to be developed accordingly. Some constraints were set on the project due to the infrastructure at the case company and the security level needed to be reached. Those were that the program should not be cloud- or web-based and that the operating system used should be Windows.

The program consisted of a server and two different clients that communicated via a given protocol. The server and protocol were already given by the case company and the two clients were written in C# and using Windows Forms. The clients were mostly identical but written as two different programs and both used the model-view-presenter pattern (MVP).

3.2.3 Case study plan

A case study can be divided as a positivist, critical or interpretive study [77] where this case study is a positivist case study which relates to the natural science research model in testing

hypotheses and measuring variables [78]. Case studies in the field of software engineering tend to be of the positivist type, especially if they in addition are explanatory [76].

In line with the process mentioned earlier, the case study also followed the plan given by Robson with the stages; *objective*, *the case*, *theory*, *research questions*, *methods* and *selection strategy* [79].

The *objective* of a case study describes what to be achieved and can be divided by it being; exploratory, descriptive, explanatory or improving [76] where this case study was of the exploratory art where the purpose was to collect experiences, as stated in RQ2 in Section 1.2, from the implementation of a few methods used in business-critical systems.

The *case* of the study describes what is studied, which in this case study were three different methods for developing business-critical software.

The *theory* outlines the frame of reference that for this thesis consists of former theory regarding risk analysis and management, methods for code and architectural redundancy, and code analysis.

The *research questions* states what the author wants to examine and investigate in a case study (see RQ2).

The *methods* describes how to collect the data and is further described in Section 3.2.5.

The *selection strategy* concerns where to seek data and were for this case study, at the case company and among the author's own reflections, thoughts, and experiences.

3.2.4 Design and planning

Three different types of methods from the literature study were chosen to be investigated for the case study. Due to the results from the interviews and the recurrence of the term *risk* in the literature study, see Table 2.3 and Table 2.5, a method within the scope of risk assessment/analysis was chosen. A method for architecture redundancy and code analysis was also chosen due to, initially perceived, their ease of implementation. Redundancy was also a recurring term from the interviews hence the choice of a method for architecture redundancy.

Numerous methods were not considered when choosing methods because of the scope and time limit of the thesis and many were considered but decided not to be used. The SDLC methods were such methods due to their wide scope and process reaching from planning to testing and evaluation, and were not selected. Most of the stages of SDLC methods were already planned to be implemented during the thesis, but not all and perhaps not to the extent they are intended to be implemented.

The final choices were thus based on time, scope and ease of implementation given the limitations and frameworks that existed.

An outline of the case study process can be seen in Figure 3.1 and each version is further described in the following sections. Each version took a different amount of time, and the number of weeks given is an approximate breakdown of the amount of time spent on each version and when in time each version was made. The interviews for version 3 were conducted during week 8, during version 4, as there was no opportunity to conduct the interviews immediately after version 3 was finished. Interviews and writing in the case study protocol followed each version to gather information and experiences regarding each method.

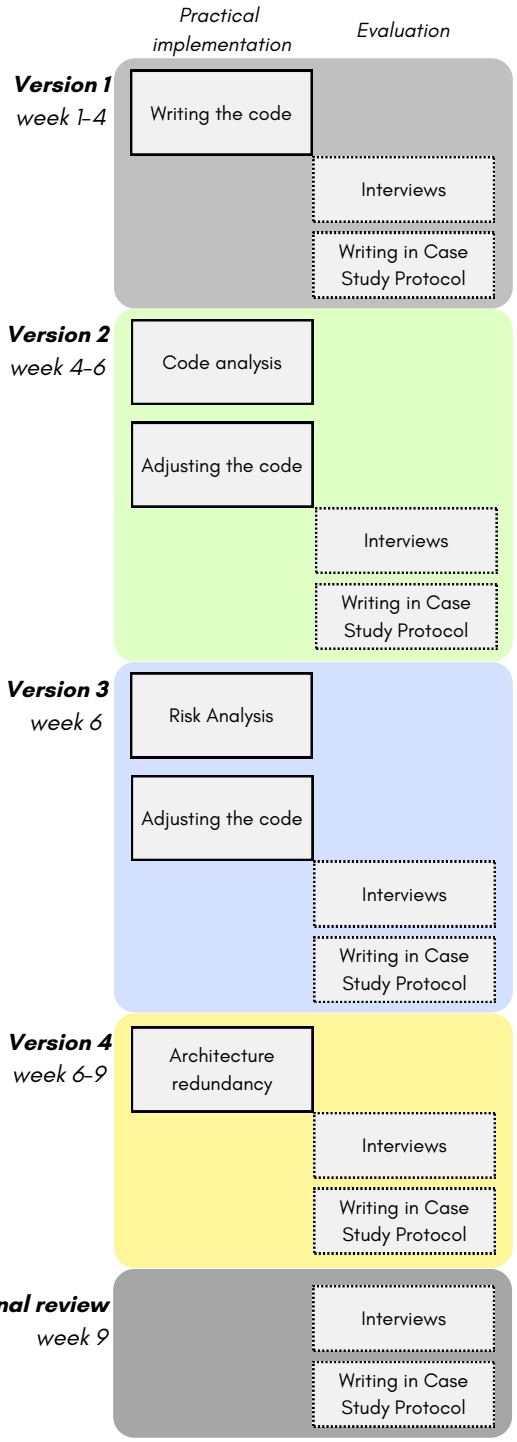

Figure 3.1: An overview of the case study process.

3.2.5 Data collection

Methods for collecting data can be categorised into three methods: direct (e.g. interviews), indirect (e.g. tool instrumentation) and independent (e.g. documentation analysis) [76] and for this study, direct data was chosen as method. Direct data were gathered in the form of interviews with two people from the software development team at the case company to gain

information, opinions, and experiences regarding the methods selected and implemented. One of the interviewees had more than 20 years in the field and will be mentioned as I1 and were also one of the interviewees for the background research. The other had a few years of work experience in the field and had not been interviewed for the background research, and will be mentioned as I2.

The overall data collection and research instruments used in the case study consisted of:

- The written code
- Code analysis
- The risk analysis
- Interviews
- Case study protocol

Even though the written code in all four versions, the code analysis, and the risk analysis were part of the case study as methods to evaluate, they also served as a data collection tools since data, code, and risks were presented and collected during the implementation of the methods. One could also go back and view the results of the various methods to make analysis and reflections.

Data was also collected from the interviews which can be seen in Figure 3.1 and the interviews were, as the previous interviews for the literature study, of the semi-structured type and contained the following questions:

- How reliable do you consider the code/project to be?
- If you had to rate the reliability on a scale of 1-10, what would it be?
- What vulnerabilities do you see in the code/project? For example; data or thread management, dependencies, etc.?
- (Between versions:) Has the code/project become more reliable since the previous version? Why/why not?

Finally, a case study protocol was used which is describe in the following section.

Triangulation is of great importance when conducting a case study and can be done in various ways. It can be done so by triangulating on data, amount of observers, methods, or theory used [76]. For this thesis, triangulation on data was achieved by the amount of observers, where more than one person at the case company was interviewed. It was also achieved by the different methods used to collect data, where four different versions were created and analysed by both the two interviewees and the author whose experiences were collected via the case study protocol.

3.2.6 Case study protocol

A protocol is useful when performing a case study since it helps with data collection to make sure no data that is planned to be collected is missed [76]. It also forces the researcher to concretise ones research which can help all stages of the case study, from designing to data

collection and reporting. It can also serve as a log or a diary and as a valuable tool for receiving feedback and critic from others. It is beneficial to consider which strategies and techniques will be used in the case study a priori, since it forces one to consider what, when, and how the data should be collected and what relevance each type of data has [80].

A case study protocol was made prior to the implementation based on the outline by Pervan and Maimbo [80] with the stages of preamble, general, procedure, research instruments and data analysis guidelines. The main focus of the case study protocol laid on the research instruments where data to be collected was stated which included: the code analysis tool, the code itself, the risk analysis, interviews, and a personal diary. The personal diary acted as tool to include and write down the author's own thoughts, opinions, and experiences during the case study and included three questions to be answered at the end of each week:

- What experiences or lessons have been gained this week?
- How secure and dependable is the code? What is good and what could be improved?
- Compare to last week, has the code improved or become more secure? Why/why not?

3.2.7 Version 1

A first version of the program was made and written in C# along with Windows Forms where firstly a model was written in C# followed by an implementation of the GUI. The GUI however, raised some problems which mainly had to do with the thread management in the GUI. This led to some refactoring of the program as well as the introduction of the model-view-presenter (MVP) pattern. The pattern consist of three parts where the logic is stored in the model, the graphical interface that the user interacts with is stored in the view, and the communication between the two is stored in the presenter [81, 82].

When a working version was finished, it was reviewed by the employees at the case company, I1 and I2, who were further interviewed. The author of the thesis also wrote in the case study protocol and answered the given questions according to the protocol.

3.2.8 Version 2 (Code analysis)

Given the importance and occurrence of testing and validation, a method for testing the code was chosen as a method. Due to time constraint, a static code analysis tool was used to evaluate and analyse the code since static code analyses are faster than conventional testing [34]. Nikolić et al. [36, 34] have analysed and evaluated three different static code analysis tools chosen from their previous research where one was initially intended to be used for this case study. However, due to the complex setup of the tool it was not used but instead another tool for SCA was used where one can choose various providers to upload ones repositories.

The repository was uploaded to the SCA tool which analysed the source code and identified issues. The dashboard of the tool displayed four issue and evaluation values: the repository's complexity given by the amount of complex files determined by cyclomatic complexity, the percentage of issues, the percentage of duplication and the test coverage. The issues percentage described and included code style problems, unused code, security issues and error proneness amongst other metrics and gave the issues a rating between minor, medium, and

major. Once the upload was completed, the listed issues were reviewed and most were resolved according to the suggestions given by the tool.

Once changes had been made accordingly, the repository was reviewed by the two employees, I1 and I2, who were interviewed and the author took notes and answered the questions in the case study protocol.

3.2.9 Version 3 (Risk analysis)

A risk analysis was chosen as a method due to its frequency in the literature and in the corporate world. Risk is one of the key aspects of a critical system since without risk a critical system would not exist and therefore, a risk analysis was constructed and implemented. Heavy and advanced risk analysis methods such as Monte Carlo analysis or Bayesian Belief Network were not considered an option due to time restrictions and for its complexity. Instead, and for convenience, the risk analysis process the case company themselves use was chosen and applied.

The analysis was carried out by the author and one of the managers at the department of risk and security, and will be described as R1, and included three steps: risk analysis, risk assessment, and risk management. All three terms are included in Figure 3.2 as headlines, but where at the case company used as different steps.

The first step, risk analysis, consisted of identifying threats and weaknesses while the risk assessment part included assessing the risks by making a risk matrix and a subsequent consequence description. The rows of the risk matrix defined the probability of the risk to occur, while the columns defined the consequence of the risk. A final risk value could then be gathered and received a value on a scale of 1 to 25, where 25 was the most probable risk with the highest consequence. Lastly, action proposals, ownership distributions, and timetables were conducted for the step of risk management. The process was conducted in person at the case company and all steps and results were written down in a template provided by the risk manager. The notes were later analysed and adjustments to the code made accordingly.

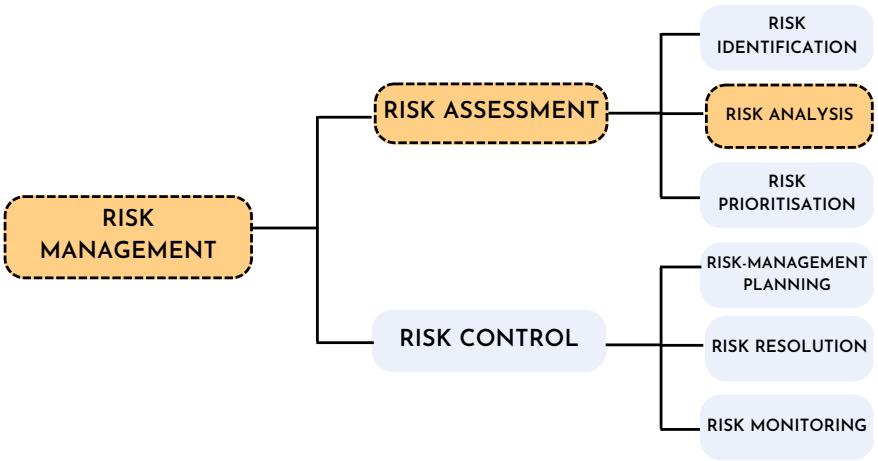

Figure 3.2: Risk process by Boehm [52] with the steps used in the case study highlighted in orange and with a dotted line.

3.2.10 Version 4 (Architecture redundancy)

The third method chosen was a method for architectural redundancy and consisted of an implementation of N-version programming where the programming languages were to be diversified. This was chosen due to its ease of implementation given the framework of the project as well as the importance of secure and well designed code. Writing in several programming languages can help achieve secure and well designed code since, as mentioned in Section 2.3.3, writing in several languages can help detect issues and guarantee safer and more reliable code. The two languages chosen for implementing N-version programming were initially C# and Python but, due to limitations and issues with the framework, neither Python nor a second programming language beyond C# could be used. Java was also tested but could not be implemented with the existing program. The difficulties with a second programming language is further discussed in Section 5.7.

Therefore, a second variant of N-version programming where the *implementation* was diversified was chosen. This was done by having two versions of the model be written by two different persons from the same specification in the same programming language. The outputs from the different versions were verified to be the same before processed by the rest of the program. One version of the model, model 1, was written by the author of the thesis. The other model, model 2, was written by an employee at the case company who had not previously been interviewed or shared information about the research or the thesis and will be mentioned as M1. M1 had over 25 years of experience in the field of IT.

Only the model, and not the presenter nor the view, was duplicated and their outputs verified. This was due to the logic being in the model and that it would not have been possible to test and use two different views.

When the second variant had been integrated into the program, it was reviewed by I1 and I2 as for the prior versions. Interviews were conducted and the case study protocol updated.

A thought of using a chat bot for duplicating the model arose during the course of implementing N-version programming and was tested for the discussion, when all methods had been implemented and final interviews had been held.

3.2.11 Final review

Lastly, a final review of the methods and the case study was made by conducting interviews with I1 and I2, and the author's final thoughts were written in the case study protocol. The interviews were carried out in the same way as before and were based on the following questions:

- Which of the methods do you think contributed the most to the project?
- Which of the methods do you think contributed the least to the project?
- What are you taking with you? Is there any method which you believe you will implement more or less and have you gained a different view on any of the methods?
- When do you think the program is *safe enough*, when would you give the program grade 10?

- Do you have any other comments or opinions on the methods or the concepts of business-critical and critical systems?
- What is your final grade?

3.2.12 Ethics

Ethics are important to consider when conducting interviews [83] and this case study took inspiration from the four ethical principles of: ethical considerations, scientific value, confidentiality, and beneficence, documented by Singer and Vinson [84]. This was done by informing all participants of the relevant facts regarding the study before their inclusion as well as sending back notes taken during the interviews for consent before using any of the information in the study, if they so desired. The study did have scientific value which Singer and Vinson states a study should to bring people in to participate and exposing themselves. Confidentiality is also of great importance and was considered when performing interviews and managing the subsequent notes. All notes were saved on the author's computer and scheduled to be deleted after six months from the date of the interview. The beneficence of the study is a balance of risks, harms and benefits which was considered to be positive in this study due to the low level of risks and harms and the benefits it could bring.

In addition, all interviewees as well as the name of the case company were anonymised and sensitive data excluded. After these actions were taken, no other ethical risks were found according to the author.

Chapter 4
Result

The following chapter presents the results of the interviews held for the background research as well as the findings of the case study from the different versions, the final review, the interviews, and the case study protocol.

4.1 Background research

4.1.1 Literature Study

The result of the literature search can be found in Section 2 and includes various methods, techniques, processes, and approaches for how to develop critical and business critical software as well as describes the terms.

4.1.2 Interviews

The three interviews held at the case company gave input and information regarding how business-critical software is viewed and developed at the company and worked as a valuable compliment to the literature study.

The general view of the term business-critical was that the term is somewhat used, but that it is rather the terms of safety and security that is thought of and in focus. The risk manager, J3, stated that the management of business risks are not as clear as those of safety and security, but that the business criticality is important when prioritising projects and that there exists a connection between safety/security and business since security flaws can have major financial consequences. J3 also seemed to be more familiar with the term and concept of critical systems than the two people working with IT, J1 and J2.

J2 stated that they actually work with several business-critical systems after given some thought on the matter, and that many systems are in fact business-critical. The interviewee did not however, give much thought or reflect on working with such systems and mentioned

that one sometimes can be a bit blunted with them stating *"In the beginning you are afraid of everything and of making mistakes but now as time goes you may be more careless"*.

All three were asked about various methods, techniques, standards, laws, and frameworks which they use or follow to make the software or system safe and secure. J3 described that they try to start and proceed from ISO standards, especially ISO 31000 and ISO 27001, and that a risk process and NIS directives are used at the company. The risk process is not a stand-alone process but a support process integrated and used in other processes where various tools and templates for conducting a risk analysis exist as well as a risk matrix for the risk evaluation. Checklists for typical risks exist as support, but no process or project is the same and therefore adaptions of methods and tools need to be made for their different purpose. The interviewee also mentioned that the rapid digitisation and technology development increases the risks and the time put on constructing a risk analysis. Security measures may therefore become insufficient but it is highly likely that managing risks properly will pay off in the long run.

The use and implementation of various steps of the risk process, ISO standards, and NIS directives could be confirmed by one or both of J1 and J2 and were some of the methods mentioned when asked which types of methods, techniques or standards they use to create safer and more reliable systems. Other methods mentioned from J1 and J2 were two-factor authentication, hyperconverged infrastructure, unit-tests, using backups, physically locked environments, conducting background checks on personnel, segmented networks, code signing, redundancy, and having different systems for development, test, and production. J1 also stated that they get in contact with the department for risk and security to get their help and input when they believe they have a high risk for a specific project. The department also do research on new programming languages and tools before they include them in their systems as a security measure, as stated by J2.

When asked when they consider to have reached *safe enough*, J2 answered that it is more of a gut feeling that no more testing can or needs to be done. J3 answered that the conclusions from a risk analysis are primarily a recommendation and part of a more extensive decision basis but that the risk analysis are judged by a manager who approves it, or that the responsibility lies with other roles in other processes.

4.2 Case study

Below the results of the case study are divided by the four different versions created and concluded by the final review of the program.

4.2.1 Version 1

The first version of the program was based on the specifications given by the case company and was written in C# along with Windows Forms as the graphical user interface. It was solely written by the author of the thesis with occasional help from various people at the IT-department. The project consisted of one repository containing two different clients with slightly varying specifications.

Interviews

The two interviews held raised several concerns regarding the code, both smaller and larger. When asked how dependable the code was on a scale of 1-10 the answers varied. The more senior interviewee, I1, rated the code to a 3 much due to the lack of unit tests and exception handling. Comments were also made on the division of the two clients as they consisted of primarily identical code and I1 stated that having almost the same application in two different versions was not reasonable.

The more junior interviewee, I2, gave the same version grade 7 and did not comment the lack of unit tests or exception handling. I2 commented on possible security and dependability issues with the program having to do with server problems and issues with dependability between the clients, and asked how the program would react and respond in those cases.

Both interviewees mentioned hardcoding, and not creating a config-file, as a dependability issue. They also both stated that they had too little knowledge about some of the thread handling to make comments or have an opinion about it but that it might cause problems if not handled properly. Other comments made where that I1 mentioned that it was somewhat difficult to assess the reliability and I2 stated that their knowledge regarding the GUI was not the greatest, but that they most likely would have written a program quite similar.

Case study protocol

Several thoughts, experiences, and comments were written down in the case study protocol during the course of developing the first version of the program. A comment made early was regarding having the program to be inside the case company's own networks and not accessing the internet and the simplicity, and security, it adds.

A concern was raised regarding the use of open source packages and libraries which were intended to be, or were, used in the program. The packages and libraries were eventually presumed to be solid due to their reputation, number of downloads, and status in the field.

Thoughts regarding the GUI also arose and became a recurring theme. The introduction of the GUI initially made the program not thread safe and significantly lowered the safety and dependability of the program. It was later resolved which increased the dependability and also lead to the use of the model-view-presenter pattern. The use of MVP was in the case study protocol considered to provide a probable increase in safety and reliability due to the addition of structure, readability, and interchangeability to the code. However, concerns still remained regarding the thread safety since the program was the first time the author used C# and Windows Forms and was not considerably confident in the programs thread management.

On the question whether the code had improved compared to the week prior, the answers during the weeks of the first version was that it was more or less the same from week to week. Some issues from the week before could have been resolved but then new ones had arisen instead. An example of this was between week 2 and 3 where the MVP pattern had been introduced but thread issues still remained. A thought presented in the protocol was also that although minor or major issues could have been resolved by the following week, the author also gained knowledge and experience regarding business-critical software and the programs and tools used, which might have led to the author being more critical and seeing more possible security flaws than the week before.

4.2.2 Version 2 (Code analysis)

The first check of the program using the SCA tool returned a result for three of its four evaluation values, see Table 4.1. Since no unit tests were conducted and included, the value for test coverage gave a percentage of zero and will not be discussed further.

The majority of the issues given by the tool were resolved and the changes made to the program were: breaking out of the recursion and infinite loop as well as updating and removing the unnecessary comments, parenthesis, parameters, and definitions suggested by the tool. The updated code was once again uploaded to the SCA tool where new evaluation values were given and these along with the old evaluation values can be seen in Table 4.1.

The final value of the program's complexity given for the files cyclomatic complexity landed on 6% which was the same value as the first upload. This was due to the tool not giving any information as to where the cyclomatic complexity resided and how to minimise it, and therefore no effort was put to solving it.

The percentage of duplicated files were originally 28 % and decreased to 9% after modification had been made according to the comments from the tool. However, this was without any active actions taken to minimise the duplication since making a generic class instead of having two versions, was considered to big a project to be carried out in time. The location of the duplication was also not specified by the tool which did not facilitate a solution to the issue.

The value for the issue percentage landed at 13% and included minor, medium, and major issues. Four major issues were given and several medium and minor ones. The major issues consisted of: changing the visibility from const to readonly for a variable, breaking out of recursion in a loop, updating a static field from a non-static field, and changing the visibility of a static variable. The medium and minor issues included removing unnecessary comments, parenthesis, parameters, and definitions.

The tool also gave a rating of the quality for each file in the repository on a scale of 1-6 and displayed the issues in the files in a structured manner as well as stating how long it would approximately take to resolve them.

Table 4.1: The result of the evaluation values given by the SCA tool before and after issues were resolved.

Evaluation value	Before	After
Complexity value	6 %	6 %
Duplication	28 %	9 %
Issues	13 %	2 %
Test coverage	0 %	0 %

Interviews

Both interviewees gave the same answers to the first three interview questions after version 2 which were that the reliability, rating, and vulnerabilities were the same as the prior version and that the differences were minor. I1 mentioned that it was a good thing that the recursion in a loop was broken which increased the dependability rating the interviewee gave the

project. The same interviewee also mentioned that it was an improvement that some parameters had changed from public to private but that I1 was hesitant to the impact changing to or from static had. I1 also thought that the SCA tool should, but did not, catch and comment on hardcoded values which was considered almost as bad as not breaking out of recursion.

I1 gave the program grade 4 while I2 gave it grade 7.

Case study protocol

Comments written in the case study protocol addressed that the suggestions and issues given by the tool were mostly medium or minor and felt somewhat unnecessary, such as removing comments or unnecessary parenthesis. However, it also stated that it helped the code become more clean and readable which might have increased the safety and dependability of the code but how and in what ways was not known. The issues given also reminded of errors received in code editors such as Visual Studio Code which made the author of the case study protocol question why an additional tool would be needed. Although the issues resembled each other, they were not identical and the tool helped to find additional issues.

The case study protocol included comments on the tool's smooth connection with git and that each push was automatically forwarded to the tool and reviewed. This made the tool easy to use in that one quickly and automatically got a review of the latest commit. The tool did however, not have support for editing and making changes directly in the tool. Instead one had to go back to one's own editor to fix the issues. This was stated to be a bit tedious since one had to find the right file and line for every minor fix and it was particularly enervating since the repository had a few lines of duplicated code.

The overall evaluation and opinion about the tool was that it did not increase the security or dependability considerably and that unit tests or a code review might have been better than using a SCA tool. Comments and issues mentioned from the interviews held after version 1 and version 2 was considered contributing more to the program than the code analysis itself.

It could conclusively be concluded from the protocol that the second version was better than the first due to the changes made, even though they were relatively few and small.

4.2.3 Version 3 (Risk analysis)

The risk analysis identified three primary risks for the program; risks regarding the personnel, instructions, and computer communication. The risk analysis was followed by the risk assessment which gave varying results, where the risks regarding instructions and computer communication received a higher value for their probability of occurrence than the risks for personnel. All three risks had the same high value for consequence which ultimately lead to the risk value being higher for instructions and computer communication than for personnel.

The risks regarding personnel and instructions were not risks which could be minimised or resolved by changing anything in the program or the code and thus no changes or measures were taken due to them. Risks regarding personnel considered sudden illness or deviation from the workplace which was considered to have a low probability while the risks concerning instructions included vulnerabilities in regards to inaccessible, missing or incorrect instructions which would lead to erroneous or non-existent management of the system and had a higher probability.

The risks concerning computer communication regarded cyber attacks, bugs in the system or the human error which would lead to consequences such as the system going down and that information would not reach those involved. This lead to measures taken such as better error handling regarding the connection with the server. It included that the program visualised the connection and its status, showed error messages when called for, and reconnected in a better manner.

Interviews

The results of the interviewees were to a large extent similar. Both interviewees increased the rating of the program by 1 step compared to the version before, but they still thought that vulnerabilities remained. Interviewee I2 mentioned that the dependability had increased now that you get an indication of issues with the connection, which was consistent with the answer of I1 who said that it is safer now when they know, and get indications of, if things are alive or not. I2 also brought up that visualisation is good and that one strives for errorless programs but that simply visualising issues goes a long way.

Final thoughts from I1 was that the interviewee thought that the risk analysis would include more points and that it would have been a better risk analysis if an employee from the IT-department, with their experience, had joined in. The grades for version 3 were 5 from I1 and 8 from I2.

Case study protocol

The case study protocol for the third version stated that the program had improved after the risk analysis due to the updates made. The risk analysis lead to the author going through scenarios regarding the consequences of the risks with computer communication, and tried to find solutions to minimise or completely eliminate those. However, the only solutions found that were feasible within the time frame and given resources, were solutions regarding connection and error messages. The protocol stated that a few other potential consequences were considered but once they were troubleshooted, they were considered already solved.

It was also mentioned in the protocol that although the risk analysis did not result in a numerous amount of changes, it did bring up risks which the IT-department might not have thought of, such as the risks with personnel and instructions. Even though those risks did not lead to actions, it was considered valuable and important to mention and reflect around. It was reviewed as a good complement to risks already thought of by the IT-department.

Another thought raised was an issue with not knowing if a client had been hacked or not but were initially thought to be too big a problem to solve in time.

Overall, the risk analysis was considered helpful and increased the dependability making version 3 better than the previous one.

4.2.4 Version 4 (Architecture redundancy)

The architecture redundancy resulted in two different versions of the model, which both used the same library and its functionality for the connecting to the server. The models were around the same size where model 1 consisted of 168 lines of code and model 2 of 204 lines. Both models met the specifications and worked as intended.

Some alterations had to be made to the presenter of the program to incorporate the models as it was not clear from the specifications what format the output would be on. The output from model 2 was therefore reformatted to match that of model 1 and the outputs were then compared. If the outputs did not match, the program would not work as intended.

The comparison of the outputs did not apply to a disconnection from the server which was an intentional decision. If a connection of one of the clients would go down and not the other, the comparison of the outputs would not work but no warning would be seen. Due to this, and given the severeness of a disconnection-event, if one of the clients would disconnect it would be displayed even though the other client would still be connected. However, this incident never happened during the course of testing and running the program.

It was not only the output from the models which were duplicated, but also a functionality of locking the work station for the user. The two models solved this in different ways where model 1 started a new process disconnected from the program while model 2 kept the process within the program keeping the control over it.

Interviews

The interviews held yielded some similarities and differences. Both interviewees stated that the security and dependability had increased, but not by the same amount. They also said that N-version programming would probably have added more value if the program was bigger since as it was, the models were fairly small and similar. Interviewee I1 raised the grade by 2 points while I2 kept it the same grade as before. I1's comment on the grade was that it is important that the locking of the work station is done in the right way and that the process is kept internally within the program.

I2 considered the dependability the same as before, as the vulnerabilities from the last version remained but did consider the double-checking of the model's output and keeping the process internal as good contributions even though keeping the process internal was not so revolutionary that it would raise the grade. The same interviewee also said that the models felt like two different styles of the same answer and that it does detect anomalies, but that it would have been more valuable in a larger program. On an added question to I2 regarding if the person had used N-version programming before, the interviewee answered that they had not but that one maybe should, however when was unknown. I2 believed that there are better "sanity checks" such as asking a colleague and having a mini code review than using N-version programming.

I1 expressed that both models were similar and that it was hard to know what they would have done differently themself to make the program safer and that the program required more try/catch-statements and better logging. It was stated that N-version programming did add value but that it was a small program and it might have been more optimal if the second model had a superb error handling because then one could have found issues not currently found due to the similarity of the models. I1 believed that the method was on too small a scale to show an effect but that it might have had one if N-version programming had been used for all steps and parts of the project. The interviewee also stated that they believe in N-version programming since more developers are better than one.

The grades given from both interviewees were grade 7 from interviewee I1 and grade 8 from I2.

Case study protocol

The case study protocol regarding the fourth version stated thoughts and experiences regarding how troublesome it can be to try to incorporate another persons code to your already existing program. The author experienced that it took longer and was more difficult than expected, which by a large part was due to the specifications not being clear enough. The output of the different models were therefore not in the same format and had to be converted to match each other.

A second thought was how much the method contributed to the project. On one hand it did, by finding a better and more secure solution to the locking of the work station and double-checking that the code and connection was reasonably made, but on the other hand the difference and safety enhancements were not that great. However, it was a good and comforting check for the author as a student and not very experienced person in the field yet and to the project, by having a lifeline for making sure that the messages and connection was working properly. It also gave the author reassurance that model 1 was not way off in its solution.

The author also thought that it was good and valuable that the solutions for locking the work station were different because it helped show security flaws with the initial model. This was however, an issue raised by I1 during the interview after version 1 of the program.

Overall, the author stated that the code for version 4 was better than the code for version 3.

4.2.5 Final review

The final review of the methods showed that the interviewees found the code analysis contributing the least to the project which was also seen in the case study protocol. I1 mentioned that the code analysis helped find basic issues but did not find hardcoded values while I2 stated that the code analysis mostly gave syntax errors. The case study protocol described that programs such as Visual Studio Code already give a great deal of help and error detection like the ones received from the tool.

The method contributing the most to the project was, according to I1, N-version programming which the interviewee believed was a good method but would give more value to a larger project. I2 also thought N-version programming was a good method to use but was unsure how much it contributed to this given project. It was more of a “sanity check” which the interviewee stated might be what N-version programming is, and that Google says the same thing regardless of who is googling, referring to the use of N-version programming with two different persons.

> “*Google says the same thing regardless who is googling.*” (I2)

The same interviewee also expressed that it was unfortunate writing in two different programming languages could not be done and that it would have been good to see the complexity of the programming languages. I2 also stated that there is a reason for writing different programs in different languages since various programming languages have their advantages and that version 4 was the method the interviewee preferred the most out of the four since it was good for robustness and identifying issues.

The case study protocol also mentioned that writing in a different programming language would probably have given more to the program and project and that N-version programming for this specific case was unnecessary, it might have been enough and more valuable to ask a colleague for a review. Using a code review was mentioned as a very good method and as a method that might have contributed more than all three methods combined.

The case study protocol considered the risk analysis as the highest contributor to the program out of the three methods implemented since it gave a broader view of the problem and raised issues not thought of before.

On the question what you take with you, I1 answered that they believe building a business-critical system requires multiple iterations of the code where one gets continuous input and review. I2 answered that they will not use any of the methods, but that the conversations about the project and code have contributed a lot. The case study protocol stated that it can be good to have the methods used in the back of your mind, but that code analysis to some extent is already built into other tools and that N-version programming is too big a project and expensive for some situations. The risk analysis on the other hand, could be a very valuable method if the system is critical enough to require one.

When asked when they think the program is safe enough and would have given it grade 10, I1 replied that one might think they have written a program of grade 10, but then a second person reviews it and disagrees which is why using N-version programming or a code review is a good thing. It was further stated that grade 10 does not exist since you yourself have limitations and that there are always room for improvement. The code might become a 10 but then perhaps the readability is set to a 3 since the program becomes too complicated. I1 concluded by saying that a program is good and decent if you have a grade 8 or 9. I2 answered on the same question that it is safe enough after thorough testing and test deployment, but that it is hard to know which time frame to set. The answer applied to both when *enough* had been reached and when the program would get grade 10. However, the interviewee then asked themself a rhetorical question if one ever has a program of grade 10.

Regarding the term business-critical, I2 mentioned that one has to ask a lot more questions when it is critical and that one wants to know and have control over as much as possible.

The interviewee's final grade of the program was asked one last time which resulted in I1 giving it grade 6 and I2 grade 8 where I1 stated that there were still improvements needed to be made. The grades for the different methods and final review can be seen in Figure 4.1. The case study protocol also stated that the program had improved in safety and dependability but that issues still remained.

After the final interviews were conducted, the use of an AI based chat bot to duplicate the model for the N-version programming method was tested. This was done to see potential similarities and differences both in time and performance, between the model produced by M1 and by the chat bot. Both models were very similar and connected to the server and locked the workstation in the same way. The chat bot produced a solution in a few minutes while M1 did so in a week and a half in parallel with other tasks. M1 stated that if they would have solely focused on the model, it would have taken M1 one to two days to write the model.

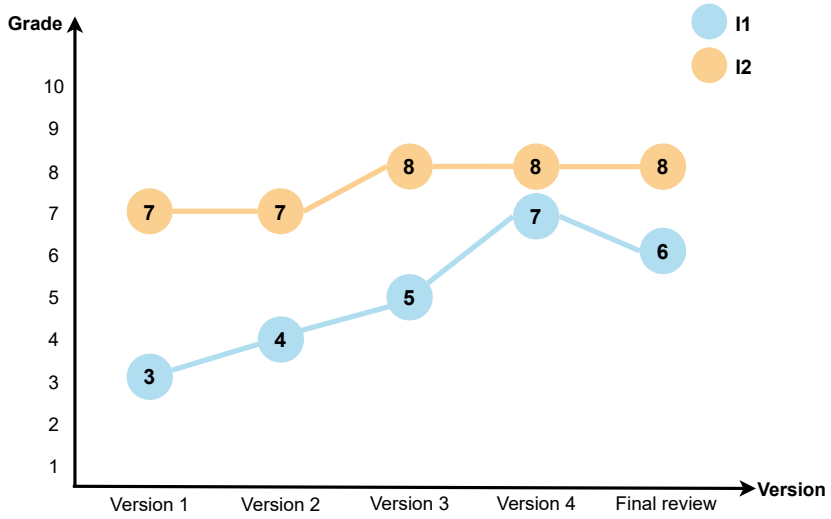

Figure 4.1: A diagram of the interviewee's grading from version 1 until the final review.

Chapter 5
Discussion

This chapter discuss the results presented in Chapter 2 and 4. Firstly, terms and findings from the literature study is discussed followed by the choice of method for both the literature and case study. Lastly, the three different methods implemented and their results are analysed as well as the final review and the choice of the methods implemented.

In the following sections, discussion about the validity of the thesis will be incorporated. The validity states how trustworthy the results are [76] and can be viewed from four different aspects: construct validity, internal validity, external validity, and reliability [85]. Construct validity is an aspect of whether the researcher or the case, has a sufficient set of measures and if the measures represents what was investigated. Internal validity concerns the causality of the case and its results and if all factors affecting the results have been considered. External validity regards if the results can be generalised and also apply outside the scope of the given study. Lastly, reliability concerns whether a later researcher can follow the steps taken to conduct the case study and get the same results and conclusion.

5.1 Business-critical

The term business-critical can be applied to various areas but appears to be significantly smaller and less used than the other types of critical systems where safety-critical is the term primarily mentioned in the literature. This is something which the interviews confirmed where the employees at the case company stated that they seldom use the term even though one interviewee, after some thought, realised that they work in several business-critical systems. Neither the interviews nor the literature study provided any answers to why that is. It could have to do with several different things such as the term not being as well-known, that there is no use for it, that most systems are safety- or mission-critical, that all systems are put under one roof as safety-critical systems or that most business-critical systems are safety-critical in one way or another.

5.2 Business- vs safety-critical systems

Given the answers from the interviews stating that the terms safety and security are in focus rather than the term business-critical, a conclusion may be that business-critical systems are developed in the same way as safety-critical systems. The risk manager's opinion that there exists a connection between safety/security and business since security flaws can have a major financial consequences, contributes to that thesis as well as the major difference in search hits between business- and safety-critical. On the other hand, one could also draw the conclusion that business-critical software is developed in the same way as non-critical software since the term is seldom used according to the interviews and literature.

There are several resemblances between business- and safety-critical systems and software as mentioned in Section 2.2.2 and some systems may have the consequences of both. How these then are categorised was not found in the literature and unfortunately not asked in the interviews. They may then be classified as safety-critical systems as it is the more severe classification of the two and with more critical consequences but they may also be classified as both. Classifying a system which has both safety- and business-critical consequences as solely business-critical seems less likely due to the consequences being less severe. The difference in severeness and criticality may however also be discussed and depends on the viewer. Some may say that loosing several trillions of dollars is more severe than a smaller chemical leakage affecting the surrounding environment for a lesser period of time, or that any loss of life or property is invaluable and cannot be measured in money.

Given the definitions of the two terms, it could be argued that all safety-critical systems at a company are business-critical too, since any failure leading to loss of life, significant property damage or damage to the environment would probably damage the company's reputation and result in a high cost for the business. However, the other way around does not have the same implication.

In the literature study, methods for safety- and business-critical systems were investigated to find as many methods for developing a critical system as possible. Both types of critical systems were considered to be reasonably similar and since the term safety-critical was exceedingly more widespread than any of the other types of critical systems, the searchers including the term safety-critical gave a broader basis. The searchers including only the term business-critical did not contribute to near as many methods found as including safety-critical and would not have given enough foundation for the thesis to build off. The reason for that could have to do with the term not being as well-known and widespread and therefore not used as much, or that systems may be safety- and business-critical but only referred to as safety-critical. It could also be because there are far fewer business-critical systems than safety-critical ones, that methods for business-critical systems are very few or them not at all being the same as the methods used for safety-critical systems.

5.3 Safe or reliable enough

A question that was raised and also asked during the interviews for the literature study was *what is enough?*; when is the system safe, dependable or trustworthy enough or can such a question even be asked. Different answers were given such as it being a gut feeling or that it is when a higher or more qualified manager accepts the level the program is at. The question

was also found in the case study protocol as a concern and something that had been thought about. The program did increase in dependability from version 1 to version 4 according to the interviewees and case study protocol, but when would the program be good enough; was it when reaching grade 8 or 9 or only when reaching grade 10 and will a program ever reach the highest grade. These were questions and thoughts also raised by the interviewees during the final review, who stated that grade 8 or 9 is good and decent but that a program might never reach grade 10.

Furthermore, it is interesting to consider how one should reach safe enough or the highest grade possible. Should one implement as many business-critical methods as possible or implement one to perfection. The answer to that seems to be that it is an iterative process and that working and aiming for high security is a continuous and ongoing process.

5.4 Method

5.4.1 Literature study

The literature study focused primarily on business-critical software and systems but included safety-critical systems and their methods as well. Both mission- and security-critical systems were left out of the larger search for methods, the reasons for which have been previously explained. However, mission- and security-critical systems might have been included and investigated further to perhaps discover more methods relevant for the area of business-critical systems and to get a better understanding of critical systems, their methods, and how they can be developed.

Solely examining business-critical systems could have been done for the literature study which was initially done, but after several days of searching for literature and trying to find methods it was realised that solely using the term business-critical would not contribute with sufficient basis for the thesis. The interviews also stated that the term is seldom used and indicated that people developing these kind of systems do not give much thought to specific methods they use but rather try to make them as safe and secure as possible. This made it difficult to pinpoint specific methods strictly linked to business-critical systems but rather gave information regarding how they try to make systems more secure. However, the interview question concerning which methods or techniques the interviewees use was asked in regard to *to achieve the security level you want* and not in regard to business-critical systems in specific. This was done so to obtain as many methods as possible from the interviews, but the question could have been asked in hindsight with regards to business-critical systems as well. Still, the question asked as it was gave the interviewees more leeway to answer the question and potentially contributed with more methods. Not asking specifically regarding business-critical systems could be a threat to construct validity since the measures did not represent what was investigated.

The method for conducting Table 2.3, Table 2.4, Table 2.5, and Table 2.4 could have been done in a more structured and methodical way. As it was done, by writing down key areas, keywords, and methods found based on the author's own understanding of the abstract and the relevance of the keywords and methods, is difficult to replicate. If another person would find keywords and method based on the abstract of the first 100 article's found by each main search, the tables would probably be different. If a more senior person or someone with

more knowledge in the field of critical systems were to conduct the same type of tables from the main searches, he or she could potentially include more or fewer terms, or completely different ones. This could definitely have been a threat to the reliability of the thesis since it could be difficult for a later researcher to follow the same steps and get the same result. However, the key areas, keywords, and methods found were considered by the author to be relevant to the literature study, but perhaps additional terms could have been found if the procedure was done in a more methodical way or by a more knowledgeable person.

Some of the methods in the literature study were investigated in more depth than others. This had to do with the methods occurrence in the literature, the interviews, and dialogues with supervisors and how modern, well-used, and easily integrated they seemed to be according to the author. For example, formal models were described as they appeared frequently in the literature and N-version programming seemed to be one of the easier methods to implement given the limitations of the project. One might have had a more structured and consistent plan regarding which methods to investigate to minimise bias, but a sift of methods had to be made for the literature study to be of reasonable size and the reasons mentioned above seemed to be a good starting point.

As mentioned earlier, the methods presented in this report in no way cover all methods and techniques that exist in the scope of business-critical systems and critical systems in general, but is an attempt of presenting which techniques exist for developing business-critical software.

The interviews held for the literature study contributed with a clearer picture of which methods are used in the corporate world and how the use and image of critical systems and business-critical is. It would however, been beneficial if more interviews were conducted and especially with people working at other companies than the case company. It would have contributed with a broader view of the terms and provided more support to the conclusions and assumptions made.

5.4.2 Case study

The method and steps taken to conduct the case study were based on convenience and time where the three methods chosen seemed like a reasonable workload. If there had been more time and resources for the project, other methods might have been examined and implemented such as SDLC methods and organisational methods for quality assurance, another safety architectural pattern, paying hackers or unit tests. However, formal methods would either way not have been implemented due to their complexity and, what it seems from the interviews and conversations with the supervisor, low usage and anchoring outside of literature. Not including methods on an organisational level could be a threat to construct validity for the case study by not including sufficient measures.

Validation and verification were two terms frequently appearing in the literature, primarily for safety-critical systems but the process was not implemented for the case study. This was due to the fact that V&V concerns the testing and evaluation of specifications and requirements for the various life-cycle phases, not all of which were included in this project. It was also considered that testing and evaluation of specifications would to some extent be included in other methods.

The interviews contributed immensely to the case study and thesis with their input regarding business-critical software, the methods used, and the final review. Although the case

study protocol and interviews were in agreement on much, the interviews contributed to new thoughts and opinions from people not involved in writing the code and with more experience in the field. For example, the interviewees and the author did not agree on which method contributed the most to the program and gave varying importance to certain improvements in the code. An opinion and thought that also applied to the previous interviews or the literature study, were that the interviews could have been recorded and transcribed for better accuracy and allowing the interviewer to be more involved and engaged in the conversations. The interviewer did believe they were involved and engaged, but might have been more so if they did not have to focus on taking notes simultaneous. This could be a threat to internal validity since it could have affected the result of the interviews.

Asking the interviewees to grade the program occasionally seemed to be a difficult question given that the question is so definitive, one has to answer a specific grade and cannot be vague. It is also a question not usually asked regarding a program and hence whether it is a good question to ask is debatable. However, it did pinpoint and confirm the interviewees' opinions regarding each method and structured the results in a good and comprehensible way.

Input and comments from the interviews in some ways became a mini code review, which in itself is a method used for developing critical systems. To not include an additional method or affect the result of the other methods, input and comments made from the interviews regarding the code was not taken into account when the succeeding versions were developed. However, it can not be guaranteed that the author was not biased by the comments made. A better approach to reduce any potential impact from the interviews would have been to have an outside person conducting them or to conduct all interviews once the final version was done. For the last approach, all versions could then have been finished before the interviewees would start to review the code and get interviewed regarding the different versions and methods. This was not thought of in advance, hence the procedure turned out as it did.

The overall opinion of the case study protocol were that it was a valuable tool for writing down and remembering the author's thoughts, experiences, and opinions. This was shown, for example, once the notes were compiled and the author believed they remembered their opinions and experiences but it turned out they did not. It was also a good tool to force the author to continuously write down and analyse each method and step of the process.

5.5 Code analysis

The result given from the SCA tool after the changes made based on the tool's own recommendations, were an improvement in duplication from 28 to 9 % and a decrease of issues from 13 to 2 %. The reduction in duplication was puzzling given that no specific measures had been taken to reduce it since a total reduction of duplication, i.e. making a generic client instead of having two versions, was considered to big of a project to be carried out in time. Minor changes were made to the program but not as major as the reduction seen. It could have had to do with the removal of miscellaneous program files, a bug in the tool or, which is less probable, the minor changes made to the program. The issue reduction, on the other hand, was a result of active actions taken after the tool's comments and recommendations and was considered a decent reduction. The last two percent consisted of four varying issues where two were in a Windows-file not constructed by the author and not considered an issue

needing to be fixed, and the other regarded defining two classes as partial, which was a suggestions given by Visual Studio as well. The class definition was considered minor and since given by the code editor, it was not considered a major issue and therefore not resolved.

An early analysis made regarding the code analysis, and which was found in the case study protocol, was that another tool or method would have been preferred over the code analysis. The SCA tool used gave many but not very relevant error messages and the case study protocol mentioned that such messages to some extent are already given in code editors. The necessity of a SCA tool is therefore debatable, but it could have to do with the specific SCA tool used or the size and complexity of the program. A larger and more complex program might have benefited more from the tool, but it is purely a guess and a given opinion regarding SCA tools are that they generate a lot of output so that one feels as though you have received value for money. The specific tool used could be a threat to the internal validity of the case study since the result of the tool could primarily be based on the specific tool itself, and the use of another SCA tool could have led to a different result. This applies to the risk analysis and N-version programming as well. If a different type of risk analysis or different type of diversification (for example on programming languages) had been used, the result of those methods might not have been the same.

Unit tests were mentioned several times throughout the process, both by the interviewees and in the case study protocol and would according to both have been a better choice of method than the code analysis. It was sifted out due to time constraints but is one of the methods the author would have used if redoing the project.

The duplicated lines of code could not be pinpointed but was most likely due to the two client versions being vastly similar. Since the location of the duplicates could not be seen and due to the substantial work of making the clients generic, it was decided not to be fixed or solved even though it was also mentioned by I1 after the first version. It was considered being to time consuming and not giving enough value to the security of the program for the time it would take to solve.

5.6 Risk analysis

The risk analysis contributed with the findings of additional risks. Some, which had not and most probably would not, have been thought of by the author or the interviewees I1 and I2 since some of the risks and their consequences were not mentioned by either one prior to the risk analysis despite having had two interviews each. The majority of the risks and their consequences were not issues that could be resolved solely by the code or the program and the risk analysis can therefore be considered to have contributed more to the project than to the specific program or code. This also confirms the division and placement in Figure 2.2 where risk management methods are included among the project methods.

The risk analysis could have benefited from having a more senior and experienced person from the field of IT which was confirmed by the comment made by I1 regarding having a person from the IT-department join in on the risk analysis. Several more risks and consequences would then probably have been discovered as the author had never before conducted a risk analysis, worked in the specific IT-environment at the case company or developed a program like the one developed. The risk manager also had limited insight and experience with the IT-environment and programming in general, and was therefore unable to assist as

much regarding the technical risks as probably needed.

More could have been done to the program because of the risk analysis which could be a threat to the internal validity since all factors affecting the result might not have been considered. It could also be a threat to the construct validity since a sufficient set of measures did not exist. An issue given were *bugs in the system* which is a rather vague and unspecific comment and hard to solve since one does not actively leave known bugs in the program. Bugs found are solved, and bugs not found will be solved once found, it is just a matter of finding them all. Finding them could have been done by deploying the program in a test environment before deploying it to its actual and intended environment or by having code reviews by others. However, the actual deployment was not in the scope of the project and therefore not thought of. Thoughts and solutions as these were raised in retrospect which can indicate that the risk analysis could have contributed even more to the project if a more thorough analysis was made, for example with the help of more senior employees, or more time had been given to its analysis and management.

The risk analysis was the only method where both interviewee's raised the grade which could indicate that the method contributed the most to the program and project, and that it is a good and solid method to use. However, they both expressed that vulnerabilities remained after the risk analysis and the changes made due to it were not major. The only change made because of the risk analysis were a better error handling for the connection which was an issue which could have been found and solved by another method as well. Hence, the role and impact of the risk analysis based on the rating is a bit difficult to determine. This can however, apply for several of the methods and issues resolved. It can be considered that it is not so much about the particular method used but rather the *finding of* issues and errors. The method best for that might vary and in this particular case it was the risk analysis, but it could be another method as well. It is hard to tell since once a risk or issue was found with the help of one method, it was resolved and was therefore not found by another method. More data and studies on the topic are needed to determine which method contributes the most to making a program as secure and dependable as possible, and it may be that it is different depending on which system or branch one works in or the level of experience and expertise the developer has.

5.7 Architecture redundancy

An attempt of writing the primary model in a different programming language was made both in Java and in Python. However, due to time and framework limitations, only the model and not the GUI were to be rewritten which lead to problems. The problems arose due to Windows Forms being written in C# and the use of .NET where the compatibility and integration with other programming languages were neither good nor simple given the complexity and limitations of the program. This is can be an issue that occurs quite often, especially considering that most programs and projects have some form of limitations and restrictions initially given.

Why diversifying on programming language was chosen in the first place had to do with the simplicity and convenience of having only one person writing the program. It was then only one person's time that needed to be claimed and who had to familiarise themselves with the problem and program. Writing in a second programming language could also have been

a good way to detect flaws and potential bugs with the solutions of the different languages. As I2 said during one of the interviews; there is a reason for writing different programs in different languages since they have their advantages, and it would have been interesting to examine that further.

The final variant of N-version programming used were diversifying on the implementation where two versions of the model was written. An initial discovery from the implementation was that it can be difficult to find a colleague who has time to help writing a version. The person initially thought of for writing a second version got urgently busy with another project and could therefore not assist within the time frame. There were several others at the case company who could help, but it still shows an issue that can arise when needing a second person to assist in a project. Having a second person writing a version can be a greater problem if people tend to work alone on projects and are not very familiar with other colleagues' projects, as the time to get started and familiarise yourself with the project will be longer. However, if a team nearly always work with the same tools, programming languages, and environments it might not be that difficult to acquaint oneself with a colleague's project but if not, the time needed to be spent on the project would increase. Another issue could be that the project might not be the other person's main concern and they may not make it a priority.

A solution to the issues with having more than one person writing a version could be to use a virtual assistant or an AI based chat bot. This was tested and the solution produced by the chat bot was extremely similar to the models already produced and it solved the issue with locking the workstation in the same way as model 2 did. For this case, the chat bot would have helped in the same ways as having another person writing a version and it would have been faster and smoother. The chat bot took a few minutes to produce the code compared to M1, who took a week and a half when doing it in parallel to the person's other tasks and without a rush.

However, using a chat bot can be polarised and problematic in some aspects which will not be discussed in this report, but the use of it for this particular case would have been quite optimal. No other person would have had to be involved which would have saved time and resources and the model itself was small and easy enough to implement so that a chat bot could be used. It might have been more difficult to use a chat bot if one were to diversify an entire repository with hundred or thousand of lines of code, files, and dependencies.

An interesting find from the result was the difference in grade given by I1 and I2 regarding version 4, where the first raised the grade by 2 points and the latter not at all. This was primarily due to the different opinions regarding the impact of the locking of the workstation which could be explained by their different years of experience or that there are different views on the importance of keeping the process internal.

A recurring theme and comment made by both interviewees were that N-version programming would have contributed more if the program had been bigger since as it was, both models were small and similar. The more lines of code you write, the higher the probability of making mistakes or writing error prone code and a larger program would, most likely, have lead to a bigger difference in the models and potentially more errors, making N-version programming contributing more than for a smaller program. Despite this, the method found interesting and valuable bugs and errors in the few lines of code that the models contained. However, if N-version programming were to be tested on a larger program for the sake of the method then contributing more, it could have turned out that it did not and then even

more time and resources would have been put into the method without it contributing to the desired extent.

In this particular case, the N-version programming felt somewhat unnecessary, which was confirmed by both interviewees and by the case study protocol where it was stated that it was a troublesome process to integrate the two versions and that the difference and safety enhancements were not that great. Although, the case study protocol did contain several positive comments regarding the method as well. Nevertheless, the author's overall feeling regarding the method was that it was not worth the time and energy the method required for the outcome it produced for this particular case.

5.8 Final review

Both interviewees claimed that N-version programming contributed the most to the program, even though only one of them raised the grade after the method. That both thought it contributed the most could have had to do with N-version programming being the method contributing to the biggest change in the program in terms of the line of codes being added or modified, to it finding the most critical issues or that it confirmed that the program was reasonably written. Why one of them did not raise the grade was due to them not finding the method contributing enough, but they still valued the method the highest which was a bit contradicting.

The risk analysis was the only method where both interviewees raised the grade and the method contributing the most according to the case study protocol. Why the author of the case study protocol, and not the interviewees, held the risk analysis the highest could have to do with the author being the only one of the three who attended the risk analysis and could be a threat to the internal validity. It was also the authors first time conducting a risk analysis which was not the case for I1 and I2 who had been involved in several risk analysis before. The author may therefore have placed more emphasis on the method and felt that it contributed more, while I1 and I2 already had some of the information and knew some of the risks that usually emerges during a risk analysis.

Which method that in the end were the best one is hard to determine and perhaps one should not stare blindly at the grades given, the result also stated that both interviewees occasionally found it difficult to give a grade. It may be better to analyse and base the results on the interviewees' comments instead, although the grading followed their comments fairly well.

The case study protocol expressed that the mini code reviews that came about during the interviews and which were not taken into account, would probably have given more to the security and dependability of the project than any of the other methods. This was confirmed from I2 who expressed that the conversations about the project and code had contributed a lot and that a code review is better than using N-version programming. There were also several issues and comments left after the final version from the interviewees which the three methods implemented did not help resolve. If the research could choose the methods implemented once more, code review would most likely be one of them.

An interesting find was that I1 gave version 4 grade 7, but then lowered the grade to 6 during the final review even though the program had not been changed, see Figure 4.1. This was one of the reasons for asking the question again when conducting a final review since an

initial thought from the author was that it might be easy to raise the grade for every version, as something had always been improved, but that the overall grade was not as high when taking a step back and viewing the program.

5.9 Choice of methods

The use of three methods contributed greatly to the thesis and RQ2. Simply implementing one method would not have contributed enough to understanding and seeing the advantages, disadvantages, and problems that exists with different methods. Having three different methods from three varying levels gave a good view of what one might encounter when implementing such methods or similar ones. All methods contributed to the program in one way or another and helped discover different flaws and issues.

The choice of methods was somewhat limited, as it probably is in most cases since many projects and programs, both those that exist and are being planned, have limitations or frameworks they must follow. Thus, it might not be up to the software developer or the project team to decide which methods to use or not. Perhaps the team leader or the board has decided that only the Waterfall Model should be used, with no or limited room for other methods to be implemented. This thesis has given the author a lot of freedom to choose which methods to implement, but there have also been limitations.

It would have been interesting to investigate and implement a method on a project or organisational level. For example, the V-process or Waterfall model could have been tested since they are recommended by standards such as IEC 61508 and IEC 62278. Several people and employees tend to be involved in projects, which may limit the use and possibility of methods on a personal level, and rather increase the use of methods on a project or organisational level. Hence, testing such methods would have been of interest.

The three methods implemented were fairly simple to implement, however, the risk analysis was already in place at the company and the N-version programming was implemented on a small program, both of which probably helped and made the process and execution of each method easier.

The order in which the methods were implemented may have affected the result and may also be a threat to the internal and external validity. The result of the case study might not have been the same if the order of the implementations were different. For example, switching the order of N-version programming and the risk analysis may have resulted in N-version programming finding and solving the same issues as the risk analysis did. Then the use of the risk analysis might have decreased and not been given the same grade. This example applies to all three methods and may have affected the result and affected whether the result can be generalised or not.

Whether the result can be generalised or not, i.e, the external validity, is also a concern when discussing the size of the program which may not be representative of an average program at a company. The program developed for this thesis was developed by a student not very experienced in the field during 9 weeks and will most likely not be near the size and complexity of most programs or systems developed and implemented at a company.

Chapter 6
Conclusion and Future Work

The scope of the term business-critical software is not the largest, nor does it contain a complete manual on how systems and software within the scope could or should be developed. This thesis have presented and implemented several methods and techniques for how to develop business-critical systems and software and what experiences can be found.

Business-critical software can be developed at different levels and layers, or base on different point of views where various methods, techniques, standards, and directives can be used, alone or in conjunction.

The experiences drawn from the implementation of the three different methods showed that each method, to different extent and in different ways, contributed to making the system safer or more reliable, however, none of them managed to resolve all security issues found in version 1. The methods contributing the most were the risk analysis and N-version programming, though it should be noted that both the case study protocol and the interviews highlighted the not used method code review as a good method that could potentially have contributed to the same extent as the three methods combined.

The findings of this thesis can help businesses and organisations looking to develop business-critical software as they provide insight into various methods and techniques. The report contributes to the understanding and selection of appropriate methods and provides valuable insights for others to take informed decisions.

Further research can build upon the findings and divisions made in this report to further investigate the scope of the term business-critical and the methods implemented. Investigating and implementing more methods and conducting additional interviews touching on the topics of: how widespread the term business-critical software is, how the term is used, how critical systems are categorised (if done so at all) and what methods are used at companies, would be of great value. Finally, it would be interesting to carry out more case studies where the same or other methods are implemented and evaluated.

References

[1] M. R. Mamdikar, V. Kumar, P. Singh, and S. Chandra, "Availability and security analysis of business-critical systems: A case study of e-commerce business process," *Quality and Reliability Engineering International*, vol. 38, no. 4, pp. 2218–2232, 2022.

[2] I. Sommerville, *Software Engineering. 10th Edition*. Boston: Pearson Education Limited, 2016.

[3] I. Sommerville, "Systems engineering for software engineers," *Annals of Software Engineering*, vol. 6, no. 1-4, pp. 111–129, 1998.

[4] J. Hadderingh, "Software versus systems engineering, is there a difference?," in *INCOSE International Symposium*, vol. 18, pp. 428–435, Wiley Online Library, 2008.

[5] A.-L. Carter, "Safety-critical versus security-critical software," in *5th IET International Conference on System Safety 2010*, pp. 1–6, IET, 2010.

[6] I. Stoian, E. Stancel, S. Ignat, and S. Balogh, "Federative scada-solution for evolving critical systems," *Journal of Control Engineering and Applied Informatics*, vol. 12, no. 3, pp. 52–58, 2010.

[7] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th international conference on software engineering*, pp. 547–550, 2002.

[8] J. Augusto, Y. Howard, A. M. Gravell, C. Ferreira, S. Gruner, and M. Leuschel, "Model-based approaches for validating business critical systems," in *System Testing and Validation Workshop*, pp. 225–233, IEEE Press, 2003.

[9] C. Areias, N. Antunes, and J. C. Cunha, "On applying FMEA to SOAS: A proposal and open challenges," in *International Workshop on Software Engineering for Resilient Systems*, pp. 86–100, Springer, 2014.

[10] A. Bertolino, G. D. Angelis, and A. Polini, "A QOS test-bed generator for web services," in *International Conference on Web Engineering*, pp. 17–31, Springer, 2007.

[11] T. Lauritsen and T. Stålhane, "Safety methods in software process improvement," in *European Conference on Software Process Improvement*, pp. 95–105, Springer, 2005.

[12] R. M. Savola, H. Pentikäinen, and M. Ouedraogo, "Towards security effectiveness measurement utilizing risk-based security assurance," in *2010 Information Security for South Africa*, pp. 1–8, IEEE, 2010.

[13] M. v. d. Brand, P. Klint, and C. Verhoef, "Core technologies for system renovation," in *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 235–254, Springer, 1996.

[14] J. Toresson, "It-attacken mot coop – detta har hänt." `https://www.svt.se/nyheter/inrikes/it-attacken-mot-coop-detta-har-hant`, 2017-07-14. Accessed: 2022-11-20.

[15] R. Verma, A. Kaushik, and V. Yadav, "Design of web-based information systems–new challenges for systems development,"

[16] P. H. Carstensen and L. Vogelsang, "Design of web based information systems-new challenges for systems development?," *ECIS 2001 Proceedings*, p. 8, 2001.

[17] P.-Y. Piriou, O. Boudeville, G. Deleuze, S. Tucci-Piergiovanni, and Ö. Gürcan, "Justifying the dependability and security of business-critical blockchain-based applications," in *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*, pp. 97–104, IEEE, 2021.

[18] O. Cawley, I. Richardson, X. Wang, and M. Kuhrmann, "A conceptual framework for lean regulated software development," in *Proceedings of the 2015 International Conference on Software and System Process*, pp. 167–168, 2015.

[19] P. Ochsenschläger and R. Rieke, "Abstraction based verification of a parameterised policy controlled system," in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pp. 228–241, Springer, 2007.

[20] S. Gerasimou, D. Kolovos, R. Paige, and M. Standish, "Technical obsolescence management strategies for safety-related software for airborne systems," in *Federation of International Conferences on Software Technologies: Applications and Foundations*, pp. 385–393, Springer, 2017.

[21] A. E. Haxthausen, "An introduction to formal methods for the development of safety-critical applications," *Technical University of Denmark*, 2010.

[22] J. Lockhart, C. Purdy, and P. Wilsey, "Formal methods for safety critical system specification," in *2014 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 201–204, IEEE, 2014.

[23] S. Madan, "Techniques to facilitate development of safety critical software systems," in *CCECE'97. Canadian Conference on Electrical and Computer Engineering. Engineering Innovation: Voyage of Discovery. Conference Proceedings*, vol. 1, pp. 249–252, IEEE, 1997.

[24] J. Bowen, "The ethics of safety-critical systems," *Communications of the ACM*, vol. 43, no. 4, pp. 91–97, 2000.

[25] J. Jacky, *The way of Z: practical programming with formal methods*. Cambridge University Press, 1997.

[26] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM computing surveys (CSUR)*, vol. 41, no. 4, pp. 1–36, 2009.

[27] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How amazon web services uses formal methods," *Communications of the ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[28] F. Alaydrus, T. Raharjo, B. Hardian, and A. Prasetyo, "Approaches in determining software development methods for organizations: A systematic literature review," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pp. 1–6, IEEE, 2021.

[29] P. M. Khan and M. M. Sufyan Beg, "Extended decision support matrix for selection of SDLC-models on traditional and agile software development projects," in *2013 Third International Conference on Advanced Computing and Communication Technologies (ACCT)*, pp. 8–15, IEEE, 2013.

[30] K. S. Church, P. J. Schmidt, and G. Smedley, "Casey's collections: A strategic decision-making case using the systems development lifecycle—planning and analysis phases," *Journal of Emerging Technologies in Accounting Teaching Notes*, vol. 13, no. 2, pp. 31–81, 2017.

[31] M. S. Durmuş, İ. Üstoğlu, R. Y. Tsarev, and J. Börcsök, "Enhanced V-model," *Informatica (Slovenia)*, vol. 42, no. 4, pp. 577–585, 2018.

[32] X. Ge, R. F. Paige, and J. A. McDermid, "An iterative approach for development of safety-critical software and safety arguments," in *2010 Agile Conference*, pp. 35–43, IEEE, 2010.

[33] J. A. Børretzen and R. Conradi, "Results and experiences from an empirical study of fault reports in industrial projects," in *International Conference on Product Focused Software Process Improvement*, pp. 389–394, Springer, 2006.

[34] D. Stefanović, D. Nikolić, D. Dakić, I. Spasojević, and S. Ristić, "Static code analysis tools: A systematic literature review.," *Annals of DAAAM & Proceedings*, vol. 7, no. 1, 2020.

[35] L. M. Rao Velicheti, D. C. Feiock, M. Peiris, R. Raje, and J. H. Hill, "Towards modeling the behavior of static code analysis tools," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pp. 17–20, 2014.

[36] D. Nikolić, D. Stefanović, D. Dakić, S. Sladojević, and S. Ristić, "Analysis of the tools for static code analysis," in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–6, 2021.

[37] C. Ebert, J. Cain, G. Antoniol, S. Counsell, and P. Laplante, "Cyclomatic complexity," *IEEE Software*, vol. 33, no. 6, pp. 27–29, 2016.

[38] B. Chess and J. West, *Secure programming with static analysis*. Pearson Education, 2007.

[39] M. F. Aniche, G. A. Oliva, and M. A. Gerosa, "What do the asserts in a unit test tell us about code quality? a study on open source and industrial projects," in *2013 17th European Conference on Software Maintenance and Reengineering*, pp. 111–120, IEEE, 2013.

[40] K. Buffardi, P. Valdivia, and D. Rogers, "Measuring unit test accuracy," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, p. 578–584, Association for Computing Machinery, 2019.

[41] N. Setiani, R. Ferdiana, and R. Hartanto, "Developer's perspectives on unit test cases understandability," in *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 251–255, 2021.

[42] M. Zielinski and R. Groenboom, "Using advanced code analysis for boosting unit test creation," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 279–283, 2021.

[43] S. Lee, X. Bao, and T. Zhao, "A safety-critical software development strategy based on theory of diverse design," in *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pp. 694–699, IEEE, 2011.

[44] R. Khoury, A. Hamou-Lhadj, M. Couture, and R. Charpentier, "Diversity through n-version programming: current state, challenges and recommendations," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 4, no. 2, p. 56, 2012.

[45] J. Niño, "An overview of programming language based security," in *Proceedings of the 47th Annual Southeast Regional Conference*, ACM-SE 47, (New York, NY, USA), Association for Computing Machinery, 2009.

[46] C. Skalka, "Programming languages and systems security," *IEEE Security Privacy*, vol. 3, no. 3, pp. 80–83, 2005.

[47] U. D. Kumar, "Reliability analysis of N-version programming with deadline mechanism," *International Journal of Quality & Reliability Management*, 2000.

[48] T. Harmon and M. R. Lowry, "N-version programming in wcet analysis: Revisiting a discredited idea," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pp. 157–160, 2010.

[49] J. Wu, "Software fault tolerance using hierarchical n-version programming," in *IEEE Proceedings of the SOUTHEASTCON'91*, pp. 243–247, IEEE, 1991.

[50] S. P. Kumar, P. S. Ramaiah, and V. Khanaa, "Architectural patterns to design software safety based safety-critical systems," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pp. 620–623, 2011.

[51] J. Jaafar, U. I. Janjua, and F. W. Lai, "Software effective risk management: An evaluation of risk management process models and standards," in *Information Science and Applications*, pp. 837–844, Springer, 2015.

[52] B. W. Boehm, "Software risk management: principles and practices," *IEEE Software*, vol. 8, no. 1, pp. 32–41, 1991.

[53] E. Smith and J. Eloff, "A new perspective on risk assessment techniques," in *Proceedings of the Fifth International Network Conference (INC 2005)*, pp. 227–234, 2005.

[54] J. Deng, L. Song, and X. Wu, "Information security risk assessment methods for the transportation industry," in *International Conference on Smart Transportation and City Engineering*, vol. 12050, pp. 605–611, SPIE, 2021.

[55] Z. Ying, Q. Li, S. Meng, Z. Ni, and Z. Sun, "A survey of information intelligent system security risk assessment models, standards and methods," in *Cloud Computing, Smart Grid and Innovative Frontiers in Telecommunications*, pp. 603–611, Springer, 2019.

[56] K. Georgieva, A. Farooq, and R. R. Dumke, "Analysis of the risk assessment methods–a survey," in *International Workshop on Software Measurement*, pp. 76–86, Springer, 2009.

[57] S. Ni, Y. Zhuang, J. Gu, and Y. Huo, "A formal model and risk assessment method for security-critical real-time embedded systems," *Computers & security*, vol. 58, pp. 199–215, 2016.

[58] A. Hakemi, S. R. Jeong, I. Ghani, and M. Ghanaatpisheh Sanaei, "Enhancement of vector method by adapting octave for risk analysis in legacy system migration," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 8, no. 6, pp. 2118–2138, 2014.

[59] I. E. Iacob and A. Apostolou, "A quantitative risk analysis framework for bow-tie models," in *2015 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pp. Y–43, IEEE, 2015.

[60] L. Xiaosong, L. Shushi, C. Wenjun, and F. Songjiang, "The application of risk matrix to software project risk management," in *2009 International Forum on Information Technology and Applications*, vol. 2, pp. 480–483, IEEE, 2009.

[61] H. Yong, C. Juhua, R. Zhenbang, M. Liu, and X. Kang, "A neural networks approach for software risk analysis," in *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*, pp. 722–725, IEEE, 2006.

[62] B. Karabacak and I. Sogukpinar, "ISRAM: information security risk analysis method," *Computers & Security*, vol. 24, no. 2, pp. 147–159, 2005.

[63] E. A. Addy, "A framework for performing verification and validation in reusebased software engineering," *Annals of Software Engineering*, vol. 5, no. 1, pp. 279–292, 1998.

[64] D. R. Wallace and R. U. Fujii, "Software verification and validation: an overview," *Ieee Software*, vol. 6, no. 3, pp. 10–17, 1989.

[65] G. Gallardo, J. May, and J. C. Gallardo, "Assessment of data diversity methods for software fault tolerance based on mutation analysis," in *Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006)*, p. 6, IEEE, 2006.

[66] P. Kumar, L. K. Singh, and C. Kumar, "Software reliability analysis for safety-critical and control systems," *Quality and Reliability Engineering International*, vol. 36, no. 1, pp. 340–353, 2020.

[67] Y. Yang, "Test based safety-critical software reliability estimation using bayesian method and flow network structure," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of risk and reliability*, vol. 233, no. 5, pp. 847–856, 2019.

[68] B. S. Medikonda and S. R. Panchumarthy, "A framework for software safety in safety-critical systems," *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 2, pp. 1–9, 2009.

[69] G. Karmakar and Y. Nirgude, "AERB SG D-25 and IEC 60880 for certification of software in safety systems of indian npp," in *Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN-2013)*, pp. 309–316, Springer, 2013.

[70] C. Lalonde and O. Boiral, "Managing risks through ISO 31000: A critical analysis," *Risk management*, vol. 14, no. 4, pp. 272–300, 2012.

[71] C. Steglich, A. Majdenbaum, S. Marczak, and R. Santos, "A study on organizational it security in mobile software ecosystems literature," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 234–241, IEEE, 2020.

[72] R. Bell, "Introduction & revision of IEC 61508," *Measurement and Control*, vol. 42, no. 6, pp. 174–179, 2009.

[73] C. Smidts and M. Li, *Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems*. US Nuclear Regulatory Commission, 2000.

[74] B. Shojaie, H. Federrath, and I. Saberi, "The effects of cultural dimensions on the development of an isms based on the iso 27001," in *2015 10th International Conference on Availability, Reliability and Security*, pp. 159–167, IEEE, 2015.

[75] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," tech. rep., Technical report, ver. 2.3 ebse technical report. ebse, 2007.

[76] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[77] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems," *MIS quarterly*, pp. 67–93, 1999.

[78] A. S. Lee, "A scientific methodology for MIS case studies," *MIS quarterly*, pp. 33–50, 1989.

[79] C. Robson, *Real world research: A resource for social scientists and practitioner-researchers*. Wiley-Blackwell, 2002.

[80] G. Pervan and H. Maimbo, "Designing a case study protocol for application in is research," in *Proceedings of the ninth pacific asia conference on information systems*, pp. 1281–1292, PACIS, 2005.

[81] M. Alles, D. Crosby, B. Harleton, G. Pattison, C. Erickson, M. Marsiglia, and C. Stienstra, "Presenter first: Organizing complex gui applications for test-driven development," in *AGILE 2006 (AGILE'06)*, pp. 10–pp, IEEE, 2006.

[82] A. Singh and N. Jeyanthi, "Mvp architecture model with single endpoint access for displaying covid 19 patients information dynamically," in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 471–476, IEEE, 2020.

[83] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[84] J. Singer and N. G. Vinson, "Ethical issues in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 12, pp. 1171–1180, 2002.

[85] R. K. Yin, *Case study research: Design and methods*, vol. 5. sage, 2009.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-03-16

EXAMENSARBETE How to develop business-critical software - a case study on a small system
STUDENT Sara Hult
HANDLEDARE Martin Höst (LTH)
EXAMINATOR Emma Söderberg (LTH)

How to choose the right method for developing a business-critical system

POPULÄRVETENSKAPLIG SAMMANFATTNING **Sara Hult**

Methods and techniques for developing business-critical software are many and diverse. This thesis has carried out a literature study of available approaches and implemented three of them to obtain and present knowledge and experience about when, how and why they can or should be used.

The term business-critical systems describes systems whose failure may lead to loss of business or damage to reputation in the market. It is easily overshadowed by the much more well-known term safety-critical and is far less used and widespread. Why that is, and how business-critical systems and software can be developed is examined and discussed in this Master's thesis along with a case study implementing three methods for developing business-critical software.

Various methods, techniques, laws, and point of views exists in the area of critical systems and this thesis has tried to divide the different terms and methods to achieve a better understanding of how critical systems, and primarily business-critical systems, are and can be developed.

Three of the methods found were implemented on a case at a company in Sweden that develops business-critical software. The methods were chosen due to convenience, time, and occurrence in the literature and the interviews conducted. The methods consisted of a code inspection where a static code analysis tool was used, the company's risk analysis, and a method for architecture redundancy where N-version programming was chosen.

The method for code inspection did not contribute a great deal to the project while the risk analysis and N-version programming had a more clear impact. The results show that none of the methods implemented solved all the issues found but they all, to different extent and in different ways, contributed to making the program safer and more reliable.

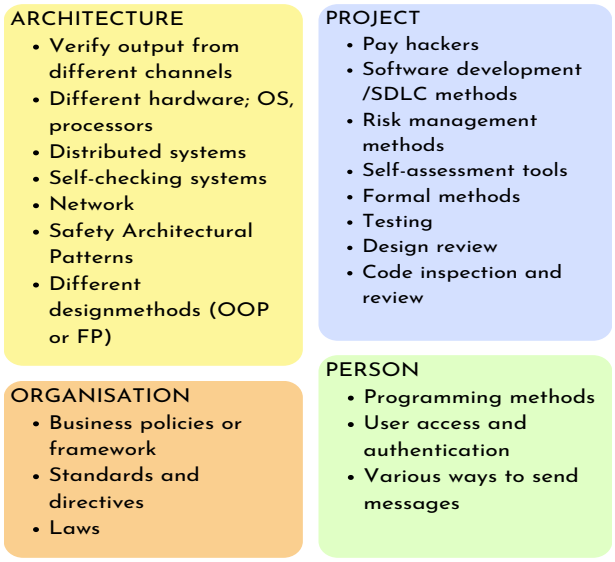

Figure 1: An overview of methods to use in a business-critical system divided by architecture, organisation, project, and person.