TAMPERE UNIVERSITY OF TECHNOLOGY

TUOMO HEINONEN

# RISK MANAGEMENT SYSTEM FOR MEDICAL STANDALONE SOFTWARE

Master of Science Thesis

Examiner: Prof. Jari Hyttinen
Examiner and topic approved in the Computing and Electrical Engineering Council meeting on 9 November 2011

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
**HEINONEN, TUOMO**: Risk Management System for Medical Standalone Software
Master of Science Thesis, 66 pages, 9 Appendix pages
December 2011
Major: Medical Informatics
Examiner: Prof. Jari Hyttinen
Supervisor: Dr. Samuli Niiranen
Keywords: MDD, CE mark, standalone software, ISO 14971, IEC 62304, ISO 13485, medical standalone software, risk management system, usability

According to update of Medical Device Directive (MDD) by European Union in 2007, the software as such can be a medical device. The direct consequence of the change of the directive is, that now depending on the intended use of software, the software might be regulated according to the MDD. If the software is classified to be a medical device, manufacturer has to fulfill the requirements of MDD to get CE mark for software. Most of the requirements are fulfilled by using three standards: IEC 62304 Software life cycle processes, ISO 14971 Application of risk management to medical devices and ISO 13485 Quality management systems.

The purpose of the thesis is to discuss the influence of regulation to medical device, classified as software, globally and in Europe and also the influence of three mandatory standards. The risk management standard is processed in more detail and the development of risk management system is based on it. The risk management system was constructed according to characteristics of medical standalone software. The goal of the thesis was to model the risk factors in the software environment and build the risk management system around the model.

The risk management system is based on the Risk factors model, which is developed in this thesis. In the model, the use of software was divided into seven factors that together or alone could contribute a hazardous situation in using the software. The developed risk management system consisted of four parts: preliminary planning, software development, post-production use and operation, and production and post-production information collecting system.

The risk management system is one of the essential requirements to launch new medical device. For a software, which is classified as a medical device there is no established a way to fulfill the regulative requirements of risk management system, because the change in the MDD is new. The thesis presents one approach to fulfill the requirements and produce more safe software.

# TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
**HEINONEN, TUOMO**: Riskienhallintajärjestelmä lääkinnälliselle erillisohjelmistolle
Diplomityö, 66 sivua, 9 liitesivua
December 2011
Pääaine: Lääketieteellinen informatiikka
Tarkastaja: Professori Jari Hyttinen
Ohjaaja: Tekniikan tohtori Samuli Niiranen
Avainsanat: MDD, CE merkki, erillisohjelmisto, ISO 14971, IEC 62304, ISO 13485, lääkinnällinen erillisohjelmisto, riskienhallintajärjestelmä, käytettävyys

Euroopan unionin uuden vuonna 2007 lääkinnällisistä laitteista annetun direktiivin mukaan ohjelmistot voidaan katsoa sellaisenaan lääkinnällisiksi laitteiksi. Suora seuraus direktiivistä on se, että riippuen ohjelmiston käyttötarkoituksesta, ohjelmisto voidaan säädellä lääkintälaitedirektiivin mukaan. Mikäli ohjelmisto luokitellaan lääkintälaitteeksi, valmistajan tulee täyttää lääkintälaitedirektiivin vaatimukset saadakseen ohjelmistolle CE-merkin. Suurin osa vaatimuksista voidaan täyttää soveltamalla kolmea standardia: IEC 62304 Ohjelmiston elinkaariprosessit, ISO 14971 Riskienhallinnan menetelmä lääkintälaitteisiin ja ISO 13485 Laatujärjestelmät.

Diplomityön tarkoitus on käsitellä regulaation merkitystä ohjelmistoiksi luokiteltujen lääkintälaitteiden osalta globaalisti ja Euroopassa sekä kolmen pakollisen standardin merkitystä. Riskienhallintastandardiin keskitytään tarkemmin ja se toimii pohjana riskienhallintajärjestelmälle. Riskienhallintajärjestelmä rakennettiin lääkinnällisen erillisohjelmiston erityispiirteiden mukaisesti. Tavoitteena oli mallintaa riskitekijät ohjelmistoympäristössä ja rakentaa riskienhallintajärjestelmä mallin ympärille.

Riskienhallintajärjestelmä perustuu Riskitekijä-malliin, joka on kehitetty diplomityössä. Mallissa erillisohjelmiston käyttö jaetaan seitsemään tekijään, jotka yhdessä tai erikseen voivat aiheuttaa vaarallisen tilanteen ohjelmistoa käytettäessä. Kehitetty riskienhallintajärjestelmä koostuu neljästä osasta, jotka ovat alustava suunnittelu, ohjelmiston kehitys, tuotannon jälkeinen käyttö ja toiminta, ja tuotannon ja tuotannon jälkeisen informaation keräysjärjestelmä.

Riskienhallintajärjestelmä on yksi välttämättömistä vaatimuksista uuden lääkintälaitteen markkinoille tuomiseen. Lääkintälaitteeksi luokitellulle ohjelmistolle ei ole olemassa vakiintunutta tapaa täyttää säädetyt riskienhallintajärjestelmän vaatimukset, koska lääkintälaitedirektiivimuutos on uusi. Diplomityö esittää erään lähestymistavan täyttää vaatimukset ja tuottaa turvallisempi ohjelmisto.

# PREFACE

I spent the July in office writing my master thesis. Most of other co-workers were on holiday, so the coffee was solely mine and the silence forced me to write intensively. Due to good pre-work, the writing process was quite smooth.

The process of constructing this thesis was extremely interesting. I had an awesome opportunity to write my master thesis for a company, which really needed the outcome of my work. That made it possible for me to study also a completely new industry.

I want to thank Dr. Samuli Niiranen about the discussions we had. Professor Jari Hyttinen and professor Ilkka Korhonen gave also invaluable feedback that made it possible to improve the outcome.

Finally, I want to thank my wife Minna, who supported me to finish my studies and patiently tolerated the hours I was studying and working.

--

Tampere, July 2011

Tuomo Heinonen

**Contents**

## ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| CE mark | Manufacturer's declaration that the product meets requirements of the directive. |
| DMR | Device Master Record |
| FDA | Food and Drug Administration |
| GHTF | Global Harmonization Task Force |
| HCI | Human Computer Interaction |
| IEC | International Electrotechnical Commission |
| ISO | International Organization for Standardization |
| LIS | Laboratory Information system |
| LEO | Lyons Electric Office |
| MDD | Medical Device Directive by European Union |
| MHLW | Minister of Health, Labor and Welfare |
| OOPSLA | Object-Oriented Programming, System, Languages, and Applications |
| QMS | Quality Management System |
| PACS | Picture Archiving and Communication System |
| SABRE | Semi-Automated Business Research Environment |
| SAGE | Semi-Automated Ground Environment |
| SFDA | State Food and Drug Administration |
| SOUP | Software of unknown provenance |
| TDP | Therapeutic Products Directorate |
| TGA | Therapeutic Goods Administration |
| TR | Technical Report |
| UML | Unified Modeling Language |
| WHO | World Health Organization |
| XP | Extreme Programming |

# 1 INTRODUCTION

Software engineering industry is a rapidly expanding industry (Boehm 2006, p. 12). It is not an easy task to describe what is software engineering because of many types of software engineering. Boehm (2006, p. 12) lists dichotomies like large or small, commodity or custom, embedded or user-intensive and casual-use or mission-critical.

It seems that variation and expanding creates challenges in emerging methods that can be used to apply theory to practice fast enough. However, Yang and Mei claim that quality and productivity of software engineering have increased and at the same time cost and risk have been decreased. In China software engineering started in 1980 and in 1999 revenue of Chinese software industry was 5.3 billion USD and in 2004 it was already 27.8 billion USD (Yang and Mei 2006, p. 2). Even though Yang and Mei's study is related to China, it shows the spirit of software engineering. There is no limit in expanding.

Naturally, in the beginning electrical engineering was in bigger role than software engineering. For example war industry has been the pioneer developing systems consist of hardware and software. There was a project called Semi-Automated Ground Environment (SAGE) for U.S. and Canadian air defense. The project introduced sequenced waterfall type software engineering model (Boehm 2006, p. 13; Everet et al. 1957; King 2010). Also in the medical device industry, medical devices were mostly hardware. Later, more software involved and medical device systems became more complex.

Medical devices have public interest because of intended use of medical devices is to cure people by affecting to human's physical or mental state. Thus governments became aware about the risks involved in medical devices. European Union harmonized laws concerning medical devices in 1993 (Council Directive 1993). Major change happened in medical device industry in 2007, when software alone was interpreted as a medical device (Council Directive 2007). Still, there are few contradictions concerning this interpretation, such as calling software as a device even it is clear that software is only a data consist of combinations of "0" and "1".In standards it is also assumed that the medical system consist of both hardware and software. After all, it is not totally clear what is appropriate for standalone software and what is for embedded system. For software manufactures it is really a moment of thinking how to apply standards economically and same time the safety of patient is increased.

## 1.1 Purpose of thesis

Starting point of this thesis is to determine the situation in the field of medical devices considering a standalone software system. In this thesis it is discussed about the systems relating to medical device industry from viewpoint of software. Scope is an administrative one, covering software engineering history and regulation that together have produced several standards for regulation purposes. Generally, three complementary standards have to be adopted so that CE mark can be granted and CE mark is important because without it manufacturer cannot put product to market (Council Directive 2007).

Proper topic of this thesis is to create a risk management system according to risk management (ISO 14971:2007). Risk management system will be established from scratch and applied to company environment. Risk management is an essential part of quality system and software life cycle and thus significantly important to work properly.

This thesis is written in co-operation with company that produces medical standalone software and there is yet no risk management system in place. Thus there is a strong practicality in addition to academic work. As standards focused to the systems contained both hardware and software, the situation in the scope of this thesis is totally different. That is because of standalone software that is manufactured in the company and the requirements for manufacturing process or risk management process don't meet the reality.

The two main problems discussed in this thesis are the following:

1. The first one is more general concerning to everything discussed in this thesis. It's about clarifying the reasons why and how European Union wants not only embedded software but all software subordinate to regulation in the field of medical devices.
2. The second and more explicit one is that how it is possible and above all practical to apply risk management system described in risk management standard to software system (ISO 14971:2007).

The first problem is more general and is discussed mainly in chapter 2 and 3 that together presents the history of software engineering and the regulation of medical software. Those two chapters trying to answer the question why the regulation is like what it is now. Chapter 4 builds the basis for risk management system and the answer for the second question. Finally chapter 5 presents the answer to the second question.

## 1.2 Structure of the thesis

Chapter 2 discusses about history of software engineering. Stress has been given to quality approach because fundamental idea behind all software engineering is in the end the quality of software. Point of view of software industry is also included.

Chapter 3 discusses the regulation in the field of medical devices. First the meaning of regulation is defined generally: why and what is regulated. Then the regulation is viewed globally. Regulation relating to European Union and CE mark is examined particular. Also local laws in Finland have been reviewed briefly, partly because it's the common way to act in European Union regarding to directives. All standards that relate to CE mark are also presented. Finally there is made a summary of historical and regulative aspects of the field of medical devices.

Chapter 4 presents the scientific side of development of risk management system. First is presented the characteristics of medical standalone software and risk management that relates to it. Then there is presented the risk factors model, which is used to identify hazardous situations relating to medical standalone software. Finally the chapter 4 describes the essential requirements of risk management system.

Chapter 5 presents the developed risk management system and assessment of it. Also a case is presented concerning to use of risk management system.

Chapter 6 discusses a little about the future of the industry and also about how appropriate all standards are for standalone software to apply. In the end is presented the possibilities of future research.

Chapter 7 presents the conclusion of this thesis.

# 2 QUALITY DEVELOPMENT IN SOFTWARE QNDUSTRY

## 2.1 Scope of Analysis and Definition of Terms

Mahoney presents three disciplines that are concerned to be computer-related: 1) electrical engineering 2) computer science and 3) software engineering (Mahoney 1988). Each of disciplines is central and it's difficult to examine them separately because of their influence to each other. However, in this thesis only discipline of software engineering is concerned and perspective is limited mostly to quality. From the viewpoint of regulator, software certification has concentrated to processes because it is difficult to measure software (Tripp 1996, p. 146).

**Software engineering** means according to IEEE 610.12 following: "The application of a systematic disciplined, quantifiable approach to the development, operation, and maintenance of software". Software engineering is more than only programming the software. It includes the whole software life cycle. Depending on the nature of produced software, different parts of life cycle presented in definition are emphasized. In the later presented Nato conference the software engineering was described as a work and management of programmers.

**Quality** in software engineering is understood according to IEEE 610.12 as following: "1) The degree to which a system, component, or process meet specified requirements, 2) the degree to which system, component, or process meets customer or user need or expectations and 3) quality comprises all characteristics and significant features of a product or an activity which relate to the satisfaction of given requirements." Simplified, software quality depends on how the software fulfills its requirements. However, there are two targets that the software has to meet, requirements and customers. Even though the software meets specified requirements, the requirements don't necessary satisfy the user's expectations. It may be because of poor quality but also because of poor design. Thus, it is a good custom to keep in touch with the customers and communicate about customer's needs and expectations. Traditionally it's understood that software quality is equal to product quality but later the focus shifted to systems development quality (Chiravuri 2003, p. 53). In other words focus is shifted from result to process.

**Reliability** means an ability to perform intended functions of software and hardware as expected without any failures.

**Efficiency** means the fulfillment of a purpose without wasting resources. There are also several other quality factors. However it's not always possible to specify certain quality factors in an unambiguous way (Sommerville 2001).

**Usability** is associated with five attributes: learnability, efficiency, memorability, errors and satisfaction. Learnability means that the system should be easy to learn. Efficiency means that the system should be efficient to use, and high level of productivity is possible after learning the system. Memorability means that the system should be easy to remember. Errors means that the system should have a low error rate and if users make errors they can easily recover from them. Satisfaction means that the system should be pleasant to use. (Nielsen 1993).

**Selected Sources**

Main sources that have been used are listed here. Selection was made according to the academic position of author and the specified field of software engineering.

*Boehm (2006):* Barry Boehm started as a programmer in 1955. He is professor emeritus in the University of Southern California.

*Brooks (1986):* Frederick Brooks's publication "There is no silver bullet" is a classic in the field of software engineering.

*Caminer (2001):* David Caminer was working for Lyons in the time LEO was built, thus having first-hand information about the case.

*Mahoney (1988):* Michael S. Mahoney was a professor of history in Princeton. He divided his research between development of mathematical sciences and the recent history of computing and information (Princeton 2008).

## 2.2 History of Software Engineering

This chapter presents the history of software engineering. All chosen milestones influenced the different industries somehow significantly by software. There are addressed equally to methods, papers and software projects. The meaning was to show the diversity of software engineering by selecting different industries that contributed to software engineering.

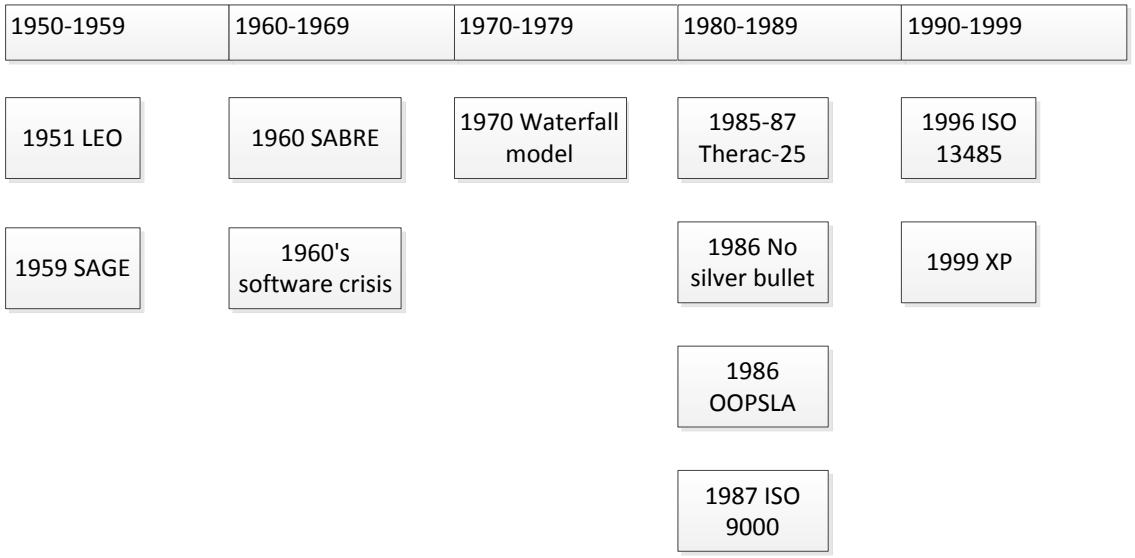


**Figure 1: Milestones relating to history of software engineering from perspective of medical software from 1951 to 1999.**

In Figure 1 all the chosen milestones are presented. Different milestones do not relate straightforwardly to each another.

### 1951 Lyons Electric Office

The company Joe Lyons founded in 1894 was first a teashop, and later on they built a chain of teashops. Next step was to install advanced bakery machinery. To be productive, they needed more output than for the bakery itself. In a few years every shop was stocking Lyons products. However, expansion brought some problems, e.g. how to organize nationwide distribution, how to control all movements of goods or how to keep account huge amount of small transactions. Thus, they recruited John Simmons from Cambridge University in 1923, and Lyons started mechanize their office. Later on there came up an idea of building a computer, LEO. (Caminer 2001).

LEO was carried out in November 1951 being ready to use and two years later full-scale LEO business machine was ready. It was recognized as the first computer that ran business application. The application was based on two management techniques introduced by John Simmons. The first one was internal market where there were several units that had their own cost center. The second one was standard costing calculation. LEO raised also the level of reliability comparing to the state of art. (Caminer 2001).

In 1954, after the completion of LEO 1, it was also able to handle all payrolls in Lyons. (Caminer 2001). Handling payrolls and book-keeping related applications are also more general the first applications in the business environment.

### 1959 SAGE - Semi-Automated Ground Environment

Boehm describes SAGE project the most ambitious information processing project of the 1950's. There were together radar engineers, communication engineers, computer engineers and nascent software engineers developing the system. Boehm estimates, that there were thousands of programmers participating the project. The purpose of SAGE was to detect, track and prevent aircrafts to bombing the land. (Boehm 2006).

Boehm claims that by 1960's people found out that software phenomenology differed from hardware phenomenology and many software applications became more people intensive than hardware intensive. Thus also SAGE was more dominated by psychologists addressing attention to HCI issues than by radar engineers (Boehm 2006). There were two direct HCI contributions: interactive CRT displays and light-pen I/O devices (King 2010).

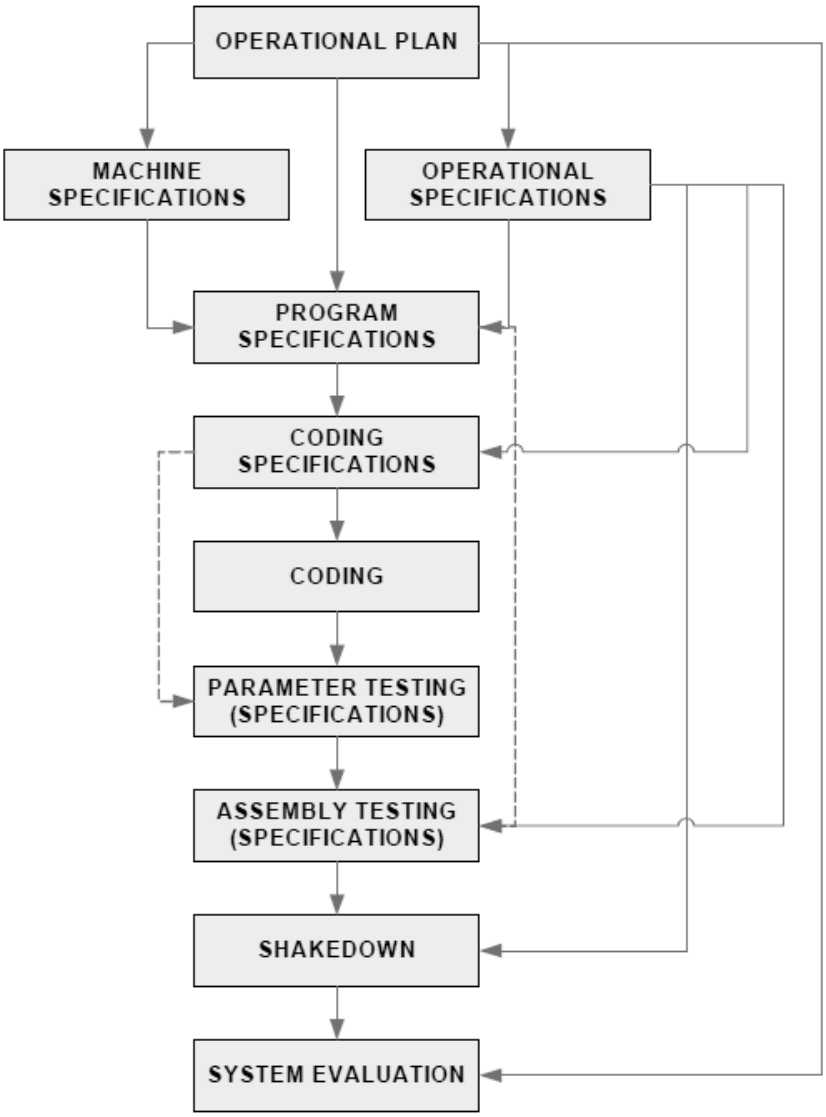


**Figure 2: The SAGE software development process in 1956 (Boehm 2006).**

Figure 2 presents the software development process that was used in SAGE project in 1956. It is very similar comparing to later presented Royce's waterfall development model or V-model.

SAGE led to broader development of computing field and new applications of information technology e.g. SABRE. Also IBM benefited a lot from SAGE with the product IBM Systems/360. Project SAGE was shut down in 1980's. (King 2010).

**1960 SABRE - Semi-Automated Business Research Environment**

In 1953, president of American Airlines and senior sales representative for IBM had a conversation about travel industry and the idea for a data processing system sparked. Six years later American Airlines and IBM jointly announced their plans to develop SABRE. (Sabre holdings 2011).

SABRE was the first passenger reservation system in airline industry and at the same time it was the first real-time business application. The system automated one of the Sabre's key business areas. The system was also a drastic technological leap forward for the airline industry. (Sabre holdings 2011).

**1960's Software Crisis**

There are three essential occurrences that can be found to lead to software crisis:

1. Hardware capabilities increased, which led to the act that the complexity of software increased.
2. Hardware cost decreased and cost of creating and maintaining software increased.
3. Software systems grew too large, and complexity of managing large groups of programmers led to failures.

For electronics industry it was possible to expand the limits of speed and memory setted by vacuum-tube circuitry and same time drastically lower the cost of hardware (Mahoney 1988). That was one of the reasons why programming could take practical advantage of research into programming languages and compilers (Mahoney 1988). It's obvious that when the cost of hardware gets lower and with the same time tools are improved, it's possible to do more complex software and it takes more labor. When more labor involves and constructed software is more complicated, also costs increase.

Large programming projects were behind schedule, over budget and below specifications. Because the situation was like that all over the industry, Nato Science Committee convened an international conference to address it. (Mahoney 1988).

Conference was held in Germany in 1968. There were both programmers and programming work managers. Participants agreed that the situation is a software crisis. As a

solution, participants accepted new discipline of software engineering without defining its content. Even in conference held in Rome in 1969, participants were not able to reach consensus about the core techniques of this new. Software engineering as a term is accepted to describe the work and management of programmers. (Haigh 2010). The term software engineering was chose before the conference in 1967 as a deliberately provocative term, implying the need to be based on theoretical foundations and practical disciplines (Naur and Randel 1969).

By the early 1970's the US Department of Defense, the single largest procurer of software of United States, had declared a major stake in the development of software engineering as a body of methods and tools for reducing the costs and increasing the reliability of large programs (Mahoney 1988). That was e.g. SAGE. Mahoney claims that efforts to define the content of software engineering constituted much of the history of computing during the 1970's (Mahoney 1988). Later, in 1986 Brooks published an article where he spoke out his opinion that there is no single technique that could drastically reduce costs.

**1970 Waterfall Model**

Winston W. Royce wrote in 1970 his article "Managing the development of large software systems" (Royce 1970). In that article he presents the waterfall model, which is still the fundamental basis for the software engineering.

There are several rules by Royce for software development:

1. Complete program design before analysis and coding begins.
2. Documentation must be current and complete.
3. Do the job twice if possible.
4. Testing must be planned, controlled and monitored.
5. Involve the customer. (Royce 1970).

Even though through the software crisis it came clear that there are essential differences between software engineering and electrical engineering Royce combines both branches in his waterfall model. E.g. rule 3 is understandable, because it is not possible to do everything right and completed at the first time. Actually, this is the major problem nowadays, because software systems are very large and at the first time there is not enough information for constructing without mistakes. However rarely nothing is started again, but fixing existing one. Later introduced spiral development method tries to enhance this aspect.

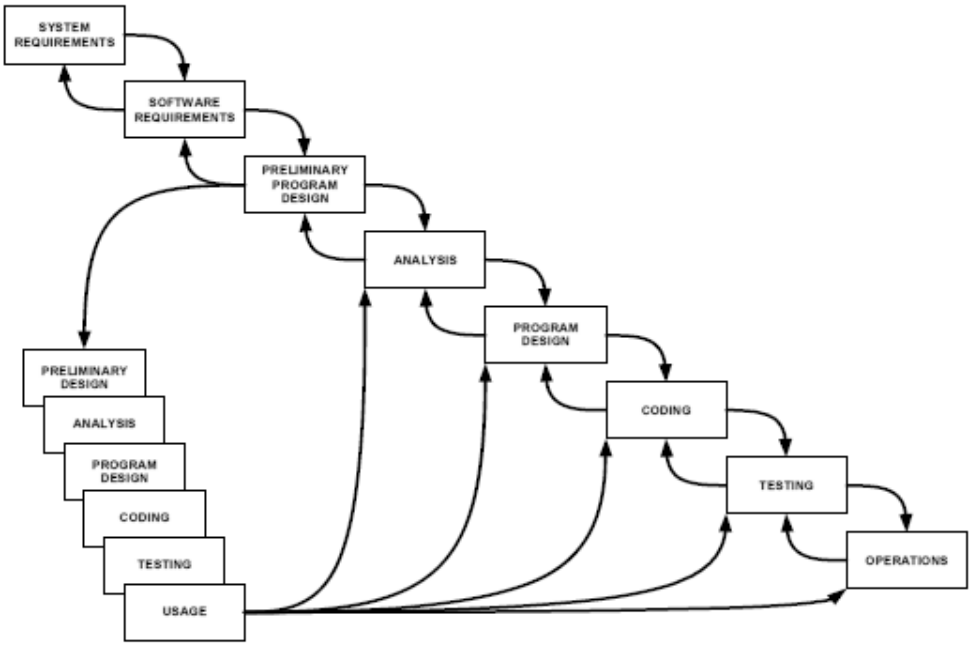


**Figure 3: Waterfall software development mode from W. Royce (Royce 1970)**

Figure 3 presents the waterfall development model for software. There are presented two times phases from preliminary program design to usage because of rule 3.

### 1985 – 1987 Therac-25

Therac-25 was a radiation therapy machine controlled by software and was used to treat people who had cancer. Six patients got massive overdoses of radiation because of a software error. Investigation revealed that in some circumstances machine display indicated no dose given and that was why operators repeated overdoses. The result of the excessive radiation exposure was the death of three patient. (Kopec and Tamang 2007; Leveson and Turner 1993).

Leveson and Turner claim that the Therac-25 accidents were the most serious computer-related accidents in the history until 1993 (Leveson and Turner 1993). The accidents probably contributed also to regulation and are one of the reasons why risk management is seen so important.

### 1986 No Silver Bullet

Fred Brooks has written an article “No silver bullet” in1986. In that article he tries to indicate the problems of software engineering. Brooks use the metaphor of werewolves which transform from familiar into horror and with bullets of silver it’s possible to magically lay them to rest forever.

The main message from Brooks is that there is no technology or management technology that alone promises even one order-of-magnitude improvement within a decade in

productivity, in reliability or in simplicity (Brooks 1986). There are several interesting parts of this message. First of all, it includes the aspects that are in definition of software engineering including both programming work and management work. Secondly, there are presented concepts like productivity, reliability and simplicity. Productivity and reliability are those essential concepts of software industry that are pursued. The new concept is simplicity that is definitely a countermove to complexity, to which software was going from the 1960's.

Brooks claims that there is wished a silver bullet at least by non-technical managers that would make software cost drop down as rapidly as hardware costs do (Brooks 1986). Brooks approach the answer for this demand through the nature of software. He presents that there is no problem in software progress speed but in extremely high speed in computer hardware progress (Brooks 1986).

Brooks also claims that a hard part of constructing software is the specification, design and testing of this conceptual construct. There are always syntax errors, but the most significant ones are conceptual errors in most systems. The easy part is representing and testing the fidelity of the representation by labor. (Brooks 1986).

Brooks (1986) presents four inherent properties of modern software system:

1. complexity
2. conformity
3. changeability
4. invisibility

Complexity of software is essential property, and it is not increasing linearly with size. Complexity develops unreliability because of difficulty to enumerate all the possible states of a program. In management side complexity causes the difficulty to communicate among team members. Also, conformity to other interfaces increases complexity. The software is constantly subject to pressure for change partly because it is so easy and because it embodies the function that is sensitive to the pressure for change. Software is not easy to visualize, because the reality of software is not inherently embedded in space (Brooks 1986).

In trying to capture software to geometrical form, there will be various graphs superimposed on upon another. Graphs may represent control flow, data flow, dependency pattern, time sequence or name-space relations. However, all these graphs are not hierarchical. (Brooks 1986).

**1986 Oopsla**

The first Object-oriented programming, system, languages, and applications OOPSLA conference was held in 1986. Later Oopsla introduced many well-known methodologies like Agile or UML. (Oopsla 2011).

**1987 ISO 9000 Quality Management standard Series**

There were many quality problems during the World War II in the Great Britain. Bombs went off in factories and solution to it was to require factories to document all their manufacturing processes and keep records that they had followed the processes. Standard was called BS 5750. (ISO Quality Services). According to John Seddon, British Government persuaded ISO to adopt BS 5750 as an international standard in 1987 (Seddon 2000). However, in 1987 was released ISO 9000:1987, which was delivered from BS 5750. There were also other defense standards that influenced ISO 9000:1987 (ISO Quality Services). The newest version of this standard is ISO 9000:2005. ISO 9001:2000 combines ISO 9001, ISO 9002 and ISO 9003 into same standard. The newest version is ISO 9001:2008.

ISO 90003:2004 includes guidelines for the application of ISO 9001:2000 for software. Last guidelines for software were done in 2004. It can be claimed that those guidelines are not following the latest trends of software industry anymore.

**1996 Quality Management Standard ISO 13485**

In the field of medical devices there is a standard ISO 13485:2003 which has same elements than ISO 9001. The main differences between ISO 13485 and ISO 9001 are some conceptual goals. ISO 13485 is harmonized with ISO 9001 having same structure. (Basler and Pizinger 2004).

EN 46001, which is an application of ISO 9001:1994 to medical device manufacturers, became the route to achieve CE mark in European Union. In 1996, ISO issued ISO 13485:1996, “Quality systems – Medical devices – particular requirements for the application of ISO 9001”. (Basler and Pizinger 2004).

In 2000, when ISO 9001 was revised, continuous improvement, customer satisfaction and training effectiveness came to that standard. Because this new standard was industry nonspecific, many regulatory agencies didn’t accept it. The new standalone standard ISO 13485 was approved in 2003 and became mandatory in 2006. There are some differences to ISO 9001, such as continuous improvement or customer satisfaction. Those concepts are not appropriate to heavily regulated medical device industry. (Basler and Pizinger 2004).

**1999 Extreme Programming**

Extreme programming (XP) is designed to work with projects of following demands:

1. The project can be done by teams of two to ten programmers.
2. There should not be sharp constraints in the project.
3. Tests can be done by a reasonable job in fraction of a day.

Some characteristics that distinguished it from other methodologies are its early and continuing feedback from short cycles, incremental planning approach and flexibly to responds changing business needs. (Beck 1999).

XP is reliant on automated tests, customers to monitor the progress of development, oral communication, evolutionary design process, and close collaboration of programmers. (Beck 1999).

There are four control variables:

1. Cost
2. Time
3. Quality
4. Scope

It's possible to pick up any three of the variables to being controlled. XP's solution is to make all four variables visible. After that it is possible to consciously decide which variables to control. (Beck 1999).

It seems that some techniques in XP are founded in early 1950's when computers were slow and it took a lot of money to use a computer. Boehm describes how the high hour price taught good practices like desk checking, buddy checking, and executing software manually before running it (Boehm 2006). XP makes it even more extreme by pair programming. (Beck 1999).

## 2.3 Software Industry

Software is essentially used for modeling physical world problems (Yang and Mei 2006). There are various problems and many ways to categorize them. What makes software industry so special is that software is found from everywhere. As it was seen in case of conference sponsored by Nato there were many industries with similar problems than software. Even though it is possible to use software for many purposes, there is no difference in development process. This phenomenon is demonstrated also by Leo, which was made for bakery industry for management accounting purposes and after few years SAGE, which was made for military purposes and SABRE which was essentially based on SAGE but which was used for travel industry.

Software engineering has developed rapidly. In 1950, there was a handful of specially designated machines and a handful of specially trained programmers but by 1955 there were about a 1000 general-purpose computers requiring about 10000 programmers and by 1960 the number of devices had increased fivefold, but the need of programmers had increased sixfold (Mahoney). Today even the amount of programming languages and different programming environments is a vast.

**Technology Driven Industry**

There are four driving forces of software technology recognized by Yang and Mei (2006):

1. Utilizing hardware capabilities better.
2. Pursuing a computing model that is both expressive and natural.
3. Making heterogeneity possible and facilitating interoperation.
4. Abstracting commonalities to promote reuse. (Yang and Mei 2006).

Hardware develops faster than software. That makes it possible to reach more and more complex information systems because there is more calculating power to run them. At the same time better computing models has to be pursued so that they could facilitate software development and maintenance. At the same time also tools improved, making it possible to benefit about improved computing models.

Because of the free trade the heterogeneity of software gains. In some fields like healthcare there are several big companies that manufacture incompatible products e.g. in Finland there are several EPRs used by hospitals. Even though it is not possible, gaining co-operation with different companies in the same field could really emerge better software.

Inside the company there are chances to promote reuse of software components, but it is more difficult to do with other companies. However, open source and licensing have been getting more popular last years and maybe in the future there are even more abstract classes that can be used commercial, for example in markets of user interface components there already are multiple alternatives to choose.

**Project Management**

All the driving forces identified by Yang and Mei (2006) were positively driven forces. There are also other forces that influence software industry, mostly negatively. The bigger the project is the bigger is complexity. There are also easily more delays because the forecasting is not possible to do accurately with the big and complex projects. The same problems exist from 1970’s to this day. Some humorous folklore in software industry is that initial software project time budget should multiply by pi and add 30% (Haikala

2006). It's partly true because it's not easy to foresee how long the development of a computer system takes even though all budgets and timetables exist.

Making software is always a project, even though it's not ending like orthodox project but it needs patches and fixes after release. Management of programmers as it was presented in 1968 is essential part of project management. There are nowadays also other aspects but still the basis for software engineering is management of labor and work. However, nowadays also business aspects are more and more involved in the project management.

Development models are quite much involved to project management. One of the purposes of the project management is to produce an environment where it easy and possible for software engineers to develop software. In 1970's it became clear that development of software is fundamentally different than development of hardware. New methods and tools have been developed, but there are still the same fundamental principles of development models that were introduced in 1956 and in 1970.

**Programming languages**

Raccoon has divided programming languages to statement-oriented, function-oriented, module-oriented, object-oriented and framework-oriented programming languages (Raccoon 1997). Classification is straightforward even though framework-oriented programming is kind of special case of object-oriented programming. The main difference between framework-oriented development and traditional development is that in framework-oriented development there is no need to create everything from scratch but there are artifacts available (Flores 2008). Also the difference between module-oriented and object-oriented exists, but it is more like an interphase of transformation from function-oriented to object-oriented. Out of this clarification are graphical programming languages e.g LabVIEW, even though they are partly located in framework-oriented programming.

Evolution of programming languages produced more complex and larger software systems. High-level programming languages made it possible to think bigger entities. Also programming environments consisted of tools that together helped programmers to improve their productivity (Raccoon 1997, p. 93). Finally, object-oriented languages allowed building software systems that consist of other systems, and because of interfaces, it was possible to do packages of entities and make them to interact. In 1988 Mahoney presented that three of the most commonly used languages are also among the oldest: Fortran, Cobol and Lisp (Mahoney 1988). Those all were high level programming languages. In the beginning of 1980's Ada was introduced by United States Department of Defense. Ada is also standardized in ISO 8652. Ada has many safety-critical support features. The first programming language that was considered to be object-oriented was Simula67 (Raccoon 1997, p. 90).

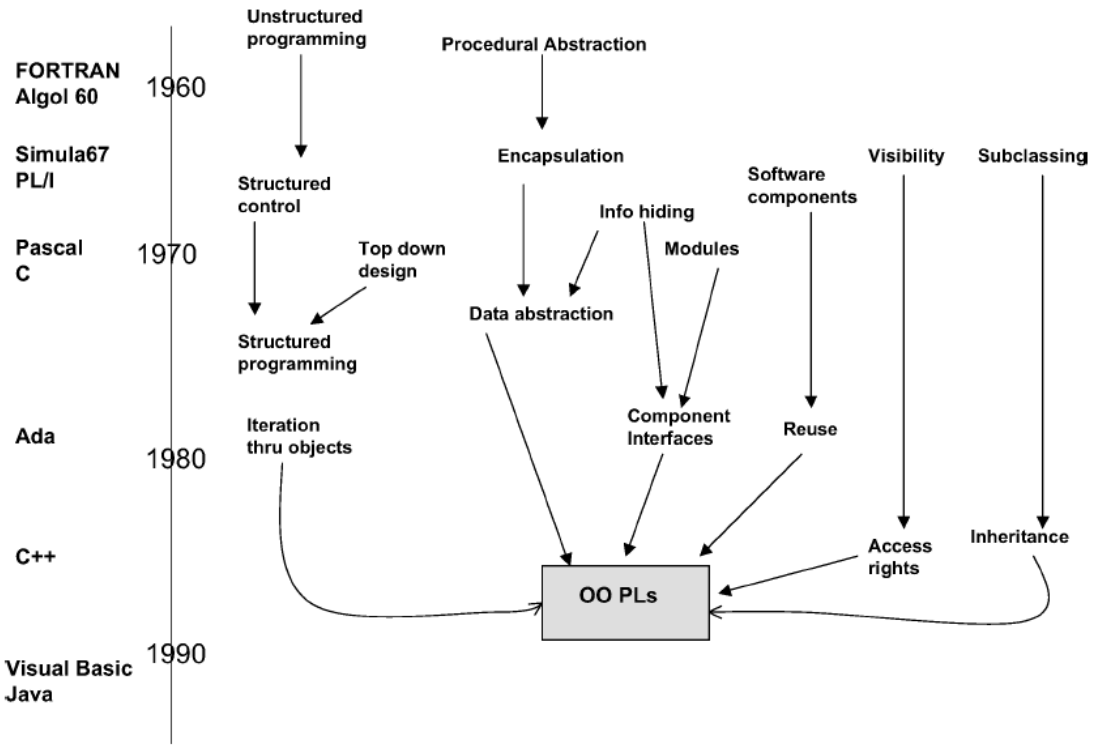


**Figure 4: Some aspects of development of object-oriented programming languages (Ryder, Soffa and Burnett 2005).**

Figure 4 presents various concepts like reuse, component interface and inheritance that made possible to develop large software system. In 1980's, before the article of Brooks there were Ada and C++ that really made different to software developing. Those languages were so high level languages that it was possible to develop new methods for software engineering. Still, Brooks wrote his article and tried to argument why it's not possible to invent a method that could change the direction of industry.

# 3 REGULATION IN THE FIELD OF MEDICAL DEVICES

## 3.1 Meaning of Regulation

Regulatory systems are intended to ensure a high level of protection of public health and safety (Australian Regulatory Guidelines for Medical Devices 2010). There are three key elements in medical device regulation to protect public health:

1. Safety
2. Effectiveness
3. The quality of products (Miura 2007; Global Harmonization Task Force 2011).

Runciman et al. claim that safety is just one dimension of the quality in healthcare and there are also dimensions like access, timeliness, efficacy, efficiency, appropriateness and acceptability. (Runciman et al. 2006). Runciman et al. probably observe the quality from the larger viewpoint of the whole healthcare industry.. However, the same dimensions are important regardless the taxonomy used.

Safety is the major concern in the field of medical devices. In Europe safety is stressed quite solemnly by dedicating complete standard for the risk management system. There are also instructions in software life cycle standards about risk management.

**What Is Regulated**

Figure 5 presents the phases of regulation by government. Figure 6 presents the roles of stakeholders relating those phases presented in Figure 5. There is also presented product stages that are under the regulation.

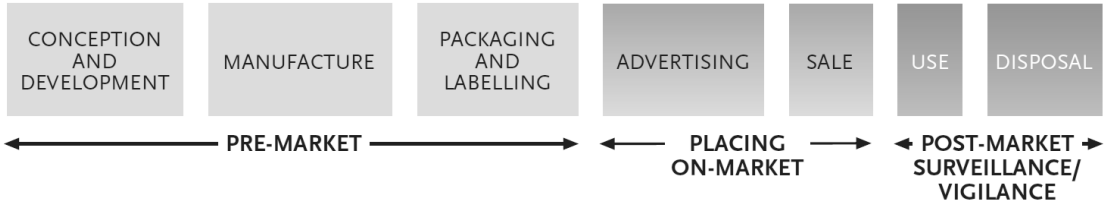


**Figure 5: Phases of government regulations (WHO 2003).**


**Figure 6: Roles in government regulations (WHO 2003).**

The content of regulation depends on the region where the medical devices are manufactured and intended to sell. If those regions are different, a manufacturer has to fulfill all the requirements even though the regulation is based on totally different system. Later in chapter 3.2, regulators in some of the large regions are presented.

### Regulation vs. Standards

From the viewpoint of industry, difference between regulation and standards is presented in Figure 7.

| *Standard* | *Regulation* |
|---|---|
| Voluntary | Mandatory |
| Provide sufficient information | Minimum requirements based on established technology |
| Certification is based on the result of audit | Nonconformity against regulation is violation |
| Is developed for voluntary use | Development more difficult |

**Figure 7: Selected differences between standard and regulation from industry viewpoint from conference paper (Miura 2007).**

As it can be seen in Figure 7, standard is voluntary and regulation is always mandatory. What is significant to notice is that standard is developed for voluntary use. Of course there are all standards in the field of medical devices constructed thinking about regulation purpose. However, it's very complicated to write detailed standard for devices that vary from simple thermometer with a few code lines to large standalone software systems.

World Health Organization has presented four purposes and benefits for standards:

1. Providing reference criteria for a product, process or service.
2. Providing information for enhancing safety, reliability and performance.
3. Assuring consumers about reliability or other characters
4. Giving consumers more choice by allowing one firm's products to be substituted for, or combined with, those of another. (World Health Organization 2003).

Giving some benefits, there are also some problems that relate to standards. The regulation of medical devices is often complicated by legal technicalities. Also legal terms and their meanings are sometimes non-uniform even within one regulatory system. (World Health Organization 2003).

**Development of a standard**

There are few models used to describe standardization process. World Health Organization has introduced a process of twelve steps that is presented in Figure 8. International Organization for Standardization has presented a process of six steps.


**Figure 8: Standardization process (World Health Organization 2003).**

To be a good standard, four attributes for developing process are identified:

1. The development has been overseen by a recognized body
2. The development process has been open to input from all interested parties and the resulting document based on consensus.

3. Good technical standards are based on consolidated results of science, technology and experience.
4. Standards do not hinder innovations and must be periodically reviewed to remain in tune with technological advances. (World Health Organization 2003).

## 3.2 Regulation Globally

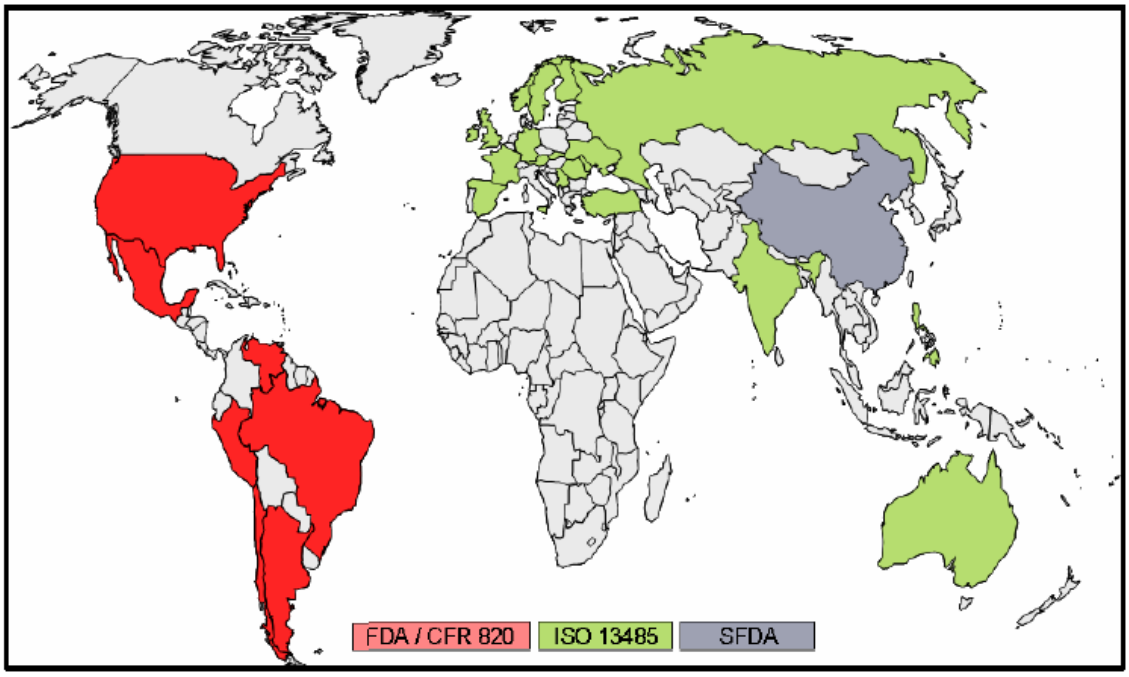


**Figure 9: Medical device regulatory landscape (Shenvi 2010).**

Figure 9 presents the quality management systems that are used in different regions. In China the State Food and Drug Administration formulate policies and programs on the administration of medical devices (SFDA 2011). ISO 13485 is used also in other countries.

### The Global Harmonization Task Force

Purpose of GHTF is to unify all continents to similar standards. The organization was conceived in 1992 and it is a voluntary group of representatives from national medical device regulatory authorizes and the regulated industry. The founding members are representatives from Europe, the United States of America, Canada, Japan and Australia. (Global Harmonization Task Force 2011).

The following requirements and practices are promoted by GHTF:

1. Promote the safety, effectiveness/performance and quality of medical devices.
2. Encourage technological innovation.
3. Foster international trade.

4. Serve as an information change forum. (Global Harmonization Task Force 2011).

**Australia**

Therapeutic Goods Administration administrates the federal Therapeutic Goods Act 1989. Medical devices are regulated under that act. (Jamieson 2001). The Australian regulatory guideline for medical devices is developed to:

1. Provide guidance to assists manufacturers of medical devices in meeting the regulatory requirements.
2. Help ensure that medical device applications to the TGA meet all the necessary legislative requirements.
3. Enhance the clarity and transparency of the processes. (Australian Regulatory Guidelines for Medical Devices 2010).

The guideline has two parts: pre-market and post-market. Medical devices cannot be imported, supplied in, or exported from Australia unless they are accepted to the Australian Register of Therapeutic Goods. In a post-market, when medical device is accepted to ARTG the device must continue to meet all the regulatory, safety and performance requirements. (Australian Regulatory Guidelines for Medical Devices 2010).

**Canada**

Health Canada is a federal department that reviews medical devices to assess their safety, effectiveness and quality before being authorized for sale in Canada. The Therapeutic Products Directorate (TPD) has a role as the federal regulatory authority in Canada. It applies the medical device regulation under the authority of the Food and Drugs Act. (Health Canada 2011).

The TPD plays also a role in monitoring medical devices after they are licensed to ensure safety and effectiveness (Health Canada 2011).

**Europe**

European Union has decided that all medical devices in Europe must compliance with Medical Device Directive. The directive is applicable for medical devices that are defined as follow:

“Any instrument, apparatus, appliance, software, material or other article, whether used alone or in combination, including the software intended by its manufacturer to be used specifically for diagnostic and/or therapeutic purposes and necessary for its proper application, intended by the manufacturer to be used for human beings for the purpose of:

- diagnosis, prevention, monitoring, treatment or alleviation of disease,

- diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap,
- investigation, replacement or modification of the anatomy or of a physiological process,
- control of conception,

and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means;" (Council Directive 2007/47/EEC).

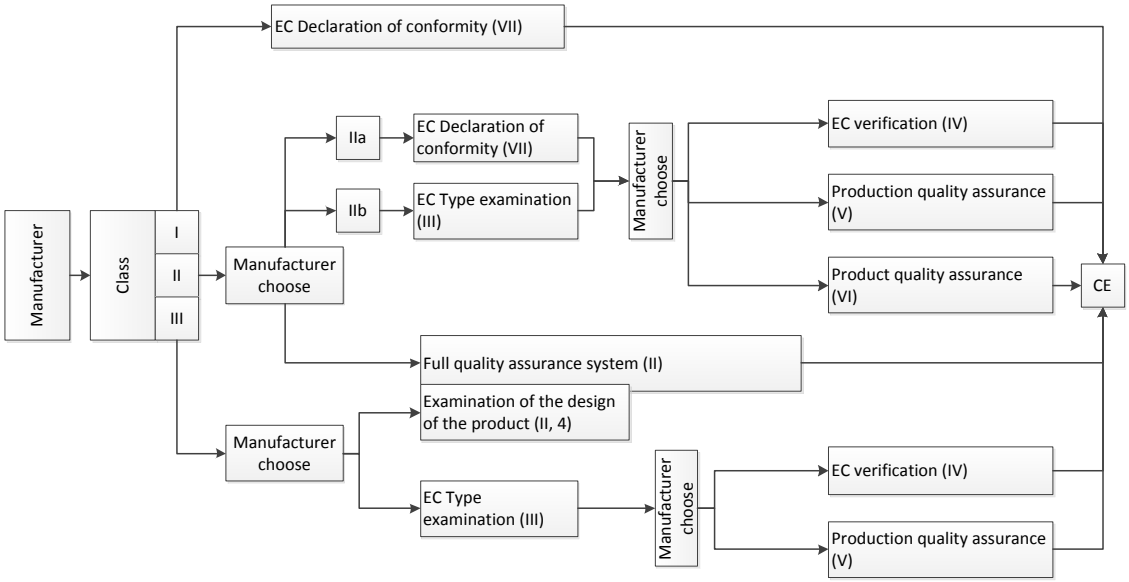


**Figure 10: Alternative routes for manufacturer to get a CE mark (Pöyhönen and Hukki 2004).**

Figure 10 presents the alternative routes to get a CE mark. With CE mark manufacturer can place the product on the market. The device class I, II, or III depends the manufactured medical device.

In Europe, ISO 13485 is accepted to be used as a quality system standard. There are accredited Notified Bodies that are objective side in the process of getting CE mark.

## Japan

Minister of Health, Labor, and Welfare (MHLW) is the authority in Japan. In 2002 MHLW passed legislation to revise Pharmaceutical Affairs Law to harmonize it more closely with those in EU, Australia, Canada, and the US. In 2005 MHLW established Pharmaceuticals and Medical Devices Agency to create a more efficient and transparent review process. (D'Eramo 2007).

Japan's Pharmaceuticals and Medical Devices Agency conducts inspections for foreign companies. Passing those inspections is essential to get medical device registered in Japan. There are two main inspections: accreditation inspection and QMS compliance

inspection. For accreditation inspection manufacturers are required the submission of several documents relating to: information on the staff, information on the product and information on the manufacturing site. The QMS compliance inspection is tied to manufacturing site. If the QMS is approved, it expires after five years and manufacturer must apply a renewal. (Gross and Minot 2007).

The Japanese QMS is similar to ISO 13485. The key difference between ISO 13485:2003 and Japanese QMS is the production and maintenance of a document called Device Master Record. The purpose of DMR is to detail the relationship between the individual product's specification and overall quality system. (Gross and Minot 2007).

In the industry side in 1984, Japan Federation of Medical Devices Associations was founded. JFMDA has contributed to the improvement of welfare and health care. There are four main functions in JFMDA: 1) government relations 2) information services 3) study and research activities and 4) international affairs. JFMDA consists of 20 associations representing about 5000 companies and over 130 individual companies are registered to sponsor JFMDA's activities. One aspect of JFMDA's vision is that they want to ensure further safety of medical devices. (JFMDA 2011).

**United States**

The Food and Drug Administration is responsible for supervision of medical devices in US. The FDA Modernization Act of 1997 was legislation focused reforming the regulation of food, medical products and cosmetics. The FDAMA focused to risk-based regulation in case of medical devices. In 2007, the Food and Drug Administration Amendments Act was reauthorized and expanded the Medical Device User Fee and Modernization Act. (FDA 2011).

Medical devices are classified into Class I, II, and III. Most of Class I devices are exempt from Premarket Notification 510(k). Most of Class II devices require 510(k) and most of Class III devices require Premarket Approval. The quality system is regulated under 21 CFR Part 820. (FDA 2011).

**Conclusion**

Figure 11 presents the conclusion of different regions and the quality systems used there.

| Region | Organization | Act | Quality system regulation |
|---|---|---|---|
| Australia | TGA | Therapeutic Goods Act | ISO 13485 |
| Canada | TPD | Food and Drugs Act | ISO 13485 |

| | | | |
|---|---|---|---|
| Europe | EC Unit F3, N.B. | Medical Device directive | ISO 13485 |
| Japan | MHLW | PAL | Japanese QMS |
| United States | FDA | FDAMA, MDUFMA | 21 CFR 820 QSR |

**Figure 11: What is the regulation for medical devices in different regions.**

Japan and United States are using their own quality system regulation. However, the actual content of quality system regulation is quite similar in all the regions. Organization refers to the authority of region who supervises the compliance of regulation. In the all other regions there is a law that determines the content of regulation. In Europe the situation is a little different because the directive only guides the local lawmaker. However, e.g. in Finland the local law refers straight to MDD.

## 3.3 Regulation in European Union for Medical Devices Containing Software

There are three standards that together satisfy the requirements for CE mark. This chapter introduces standards briefly. Structures of standards are presented in appendices. Risk management standard ISO 14971 will be the basis for risk management system that is constructed in chapter 4 and 5. For those medical devices that do not contain software, IEC 62304 is not used. Instead, there are IEC 60601 standard family for that kind of medical devices.

### 3.3.1 ISO 13485 - Quality Management System

Quality management system is based on the management commitment, from where is e.g. quality policy presented. Management also must to address needed recourses. By default QMS covers the whole company and all the functions, but it is possible to exclude some parts.

In QMS customers are very important and the products have to fulfill the requirements of customer. QMS states in the high level the production process but especially in the case of software, the software life cycle processes standard is used.

Structure of the standard is presented in Appendix 1.

### 3.3.2 IEC 62304 - Software Life Cycle Processes

Software development plan defines the software development life cycle model. The standard supports all kind of life cycle models e.g. waterfall development model, V development model and agile development models. However, applying an agile develop-

ment model is not simple because of requirements for software requirements, implementation, testing and traceability.

The software life cycle covers the software development, maintenance, risk management, configuration management and problem resolution processes. Risk management process is overlapping with SIO 14971, but also appends some software-related details.

Structure of the standard is presented in Appendix 2.

**Criticism**

The industry side has negative criticism for this standard. For example company Sakura Finetek Japan presents that IEC 62304 is too aggressive and complicated for regulatory requirements. To fulfill all the requirements is huge burden especially for small manufacturers. They claim that it is not suitable for regulatory audit criteria. (Miura 2007).

More generally, the problem with IEC 62304 and ISO 14971 is that they are not developed for complex software systems though they might work well with simple software systems.

### 3.3.3 ISO 14971 – Application of Risk Management to Medical Devices

Risk management is an essential part of quality management system.

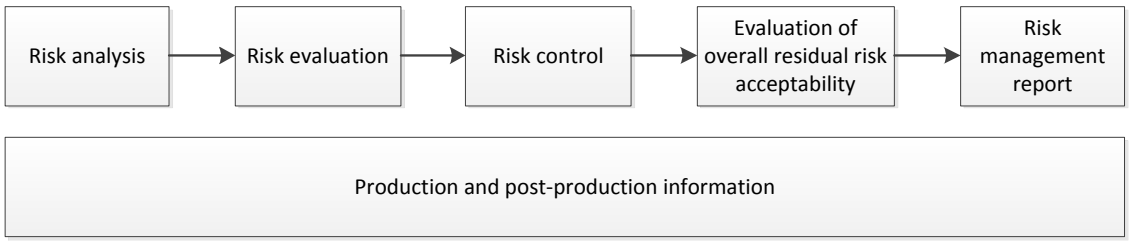


**Figure 12: Risk management system according to ISO 14971.**

Figure 12 presents the parts of risk management system according to ISO 14971. Production and post-production information shall be collected in all phases of medical device life cycle.

Risk management has to be done in all phases of software development. There has to be risk management plan including the risk management and verification activities, review requirements, and the personnel involving the risk management activities and the process. Also the production and post-production information collecting system is addressed in the risk management plan. The outcome of risk management process is risk management file.

Structure of the standard is presented in Appendix 3.

## 3.4 Regulation in Finland

The Medical Device directive is implemented in Finland's legislation and updated on June 2010. The purpose of the law is to sustain and develop the safety and use of medical device. (L 24.6.2010/629).

According to Valvira, the National Supervisory Authority for Welfare and Health, the medical devices containing software that are placed on the market before 1.7.2010 are not automatically in the scope of the law, if no modifications are done to software. (Valvira 2011).

All over the European Union the regulation in the field of medical device is similar comparing to Finland. That is because of the directive, which has to be implemented in all the member countries to local legislation.

Valvira's purpose is to supervise and provide guidance to healthcare providers. The medical devices must be accepted and registered to Valvira's register. (Valvira 2011). Depending on the device class the manufacturer might be forced to use Notified Body. The company may choose among all the accredited Notified Bodies. In Finland there is a company called VTT Expert Services Oy as a Notified Body.

## 3.5 Summary of the Historical and Regulative Aspects

The software engineering industry is a rapidly expanding industry. In 60 years software engineering has revolutionized also all the other industries by bringing the software tools into use. In 1951, LEO was used to business management to make national wide distribution of bakery goods possible.

In 1960's, the SAGE project introduced the usability aspects of software development. There was used also a software development model similar to waterfall model introduced by Royce in 1970. SAGE was used later as a basis for passenger reservation system SABRE for American Airlines.

In 1968, a conference was organized due to software crisis. Conference was sponsored by NATO and gathered many professionals together. After the conference term software engineering was used. The term was developed to be a little bit provocative implying the need to be based on theoretical foundations and practical disciplines.

In 1986 Brooks published an article, where he claimed that there is no single technological solution for the problems that were identified early in the 1960's software crisis. In 1987 ISO 9000 series was founded. It is the basis for quality management systems like ISO 9001 or ISO 13485.

Regulation in healthcare exists to provide safety for citizens. Regulation is in form of standards that have to be applied by the manufacturer of a medical device. Because of GHTF, which was established in 1992, the regulation in all large regions of the world is quite similar. Safety, performance and quality of medical devices are seen important all over the world. It's significant to notice that after Therac-25 accidents in 1985-1987 a lot of development in the field of medical device regulation was done. Only five years after the last death of a patient, the GHTF was established. After seven years, in 1994, there was an application of quality system for medical devices, and after nine years in 1996 the quality system for medical devices ISO 13485 was established.

# 4 DEVELOPMENT OF THE RISK MANAGEMENT SYSTEM FOR MEDICAL STAND-ALONE SOFTWARE

## 4.1 Terms and Definitions

It's important to notice that in chapter 4 and 5 the term medical standalone software is used to separate it from the medical device. Current legislation makes no difference between pure standalone software and software that is used with hardware. However there are essential problems to apply current standards and principles as such for standalone software.

**Harm**: "physical injury, damage, or both to the health of people or damage to property or the environment" (IEC 62304).

**Hazard**: "potential source of harm" (IEC 62304).

**Hazardous situation**: "circumstances in which people, property, or the environment are exposed to one or more hazards" (ISO 14971).

**Intended use** is a regulative term for describing how the software should be used. Intended use is the key for classifying the software in different classes as described in Figure 10.

**Medical standalone software** means software system that is developed for medical purposes. Medical standalone software could be used to manage medical information or assist physicians in decision making e.g. laboratory information system or other information systems. Standalone software that is intended to be used with some hardware, which operates to patient, is not medical standalone software.

**Misuse** is a regulative term intended to mean incorrect or improper use of the medical device (ISO 14971).

**Risk**: "combination of the probability of occurrence of harm and the severity of that harm" (IEC 62304).

**SOUP** is a third party software component that is used in a medical device (IEC 62304).

## 4.2 The Nature of the Faults in Medical Device Software

Wallace and Kuhn (2001) have analyzed the causes why manufacturers had to recall the medical devices included software back. Their analysis included two datasets from 1983 to 1991 and from 1992 to 1997. The total amount of software recalls from 1983 to 1997 was 383, but because of limited information, only 342 failures were discussed in their study. (Wallace and Kuhn 2001).

The study identified the fault classes of medical devices containing software and generic problems for each type. They also proposed prevention and detection methods for each generic problem. Figure 13 presents those the fault classes. Wallace and Kuhn asked the two obvious questions: "Why are logic and calculation the prevalent types?" and "What can prevent or detect them before product release?" (Wallace and Kuhn 2001).

In Wallace and Kuhn study the configuration management was only 1 % share of all faults and initialization and interface related faults were each 2 % share of all faults. The third largest fault class was change impact with only 6 % share. It can be seen that most relevant faults really are the first and the second one.

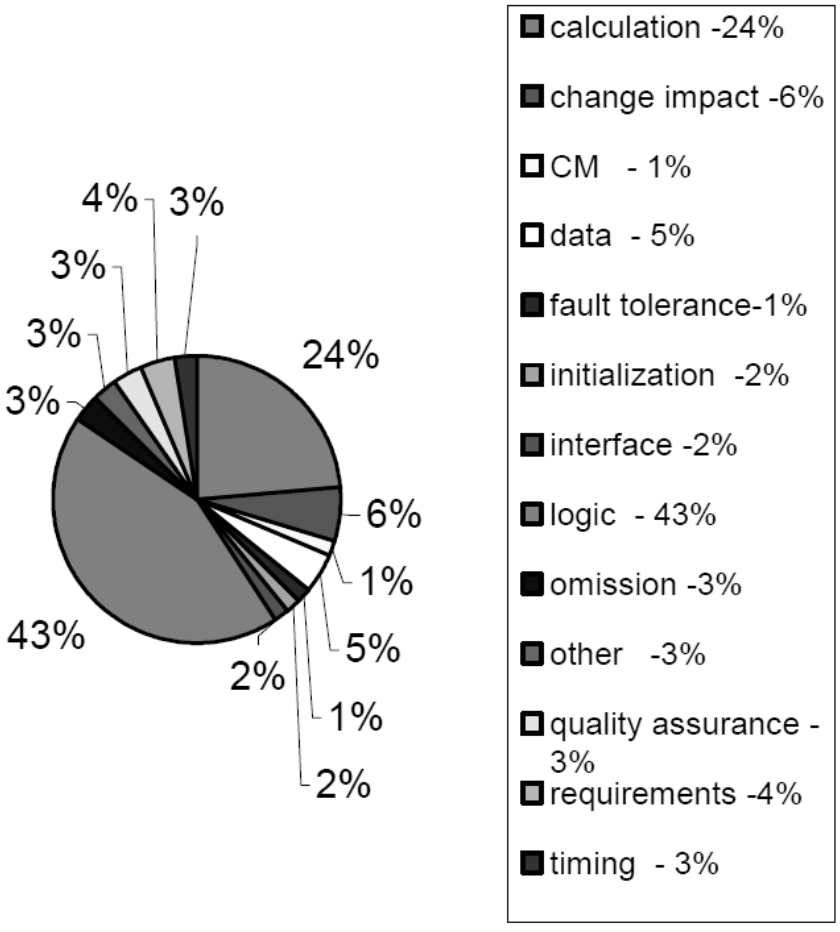


**Figure 13: Fault class distribution (Wallace and Kuhn 2001)**

In the study, no distinction was made between different types of software. Most probably the medical devices of the study weren't information systems. However, the proposed principles are practical for all software because of the generalization of fault classes and problems. Proposed prevention and detection methods of fault classes are presented in Appendix 4.

Size of different classes may be only directional. At least in omission, logic and calculation classes the source may be locating in the requirement specification. Requirements class demonstrates the need to develop, verify and validate a requirement specification. (Wallace and Kuhn 2001).

Logic problems might have resulted from incorrect, incomplete, or inconsistent requirements or designs. The source of the problems could have been requirements, design, or code. (Wallace and Kuhn 2001). They presented design and code review as a prevention method. Also walkthrough the software implementation against design was suggested. Being a major fault class, most of the problems could have been solved by rigorous software development process.

There are also few fault classes that do not relate in any way to the information system. Fault tolerance class relates to safety-critical systems that should tolerate abnormal or anomalous conditions. Timing class relates to real-time applications.

It is significant to notice that review of the artifact is suggested almost in all fault classes either as a prevention method or as a detection method. Even though there are some technical solutions to prevent data faults like data validation, still most of the prevention methods are relating to development process phases and not to any technical solution. That is significant also when risk control measurements are considered.

## 4.3 Medical Standalone Software

The threat of injuries or death characterizes medical device and also medical standalone software. Without any connection to individual human no threats appear either. That makes the difference between medical standalone software and other software in regular use. For example, in the industry of electrical devices there are strict regulations how to build a device that uses electric power. CE marks are used in that industry too. The reasons are same than in medical device industry: to minimize the threat of injuries or death of human beings. From that viewpoint it is understandable why also standalone software in medical device industry wanted to put under the regulation.

**Differences between Standalone Software and Embedded Software**

Generally, software can be divided to two different categories**: s**tandalone software and embedded software. Embedded software is in interaction with hardware and together with hardware it consists of embedded system. The embedded system specifies the

functionality of the embedded software. The functionality of the embedded software is essential for using the embedded system properly. In the medical device industry, embedded systems consist of the hardware providing a medical functionality and the assisting software in it. Standalone software is a software system that is used either alone or with some electric machine providing a medical functionality.

Medical standalone software can be a large information system that is e.g. used to consultative purposes. Medical standalone software can be used over the Internet by a web browser or as standalone software running in the user's workstation. The difference between standalone software and medical standalone software is that medical standalone software is alone both the system and software system, but standalone software can be part of system that consists of standalone software and some medical purpose hardware.

Few examples of medical standalone software and medical device with embedded software are presented in Figure 14.

| Medical standalone software | Medical devices with embedded software or with standalone software |
|---|---|
| • Laboratory information system (LIS)<br>• Cardiology information system<br>• Picture archiving and communication systems (PACS)<br>• Radiology information systems | • Glucose meter<br>• Digital ECG<br>• Blood pressure monitor<br>• Coagulation meter |

**Figure 14: Dichotomy between medical devices with medical standalone software and medical device with embedded software is listed in the table by two columns.**

### Differences between Software and Hardware in Risk Management

ISO 14971 is made for all medical devices. Later, when more software appeared, the ISO 14971 was still instructed to be used also with software even though it was not designed to it. Because of differences between software and hardware, sometimes it is difficult to apply ISO 14971 literally. There is IEC/TR 80002-1, which is technical report for guidance on the application of ISO 14971 to medical device software. Even though there are many improvements and a lot of help to understand how to apply standard to the software, still medical standalone software aspect is missing. The reason for that probably is that in the time the TR 80002-1 was completed there were no examples from real life how to apply ISO 14971 to medical standalone software. Because of the long preparation time for the technical report, not all the MDD changes affected to final technical report.

| | Software | Hardware |
|---|---|---|
| Major common hazard | Error (bug) | Not common hazard |
| Location of the hazard | Unknown | Known |
| Risk assessment | Difficult | Not difficult |
| Drastic risk control | Eliminate error | Reduce risk for each hazard |
| Risk control timing | All stages | Early stage |
| ISO 14971 | Difficult to apply as it is | Should be applied |

**Figure 15: Selected differences between software and hardware from industry viewpoint from conference paper (Miura 2007).**

Figure 15 presents the built-in perception that is in software life cycle standard and risk management standard relating to risk control timing. Both of the standards and later TR 80002-1 explain that developing architecture is the most significant stage to control risk. In software and especially in standalone software the architecture is important but in most cases not much can be done in that phase alone. Like Miura presents, common hazards in software are bugs and the architecture does not affect at all to those hazards. Also Human-Computer Interaction (HCI) has to be considered even more carefully than with no standalone software contained medical devices.

## 4.4 General Principles of the Risk Management System

### Safety of Medical Device

Ensuring the safety is a top priority in the field of medical devices. According to WHO the safety and performance of a medical device depends on two critical and one important element. The critical elements are product and use, and the important element is representation of the product. (World Health Organization 2003).

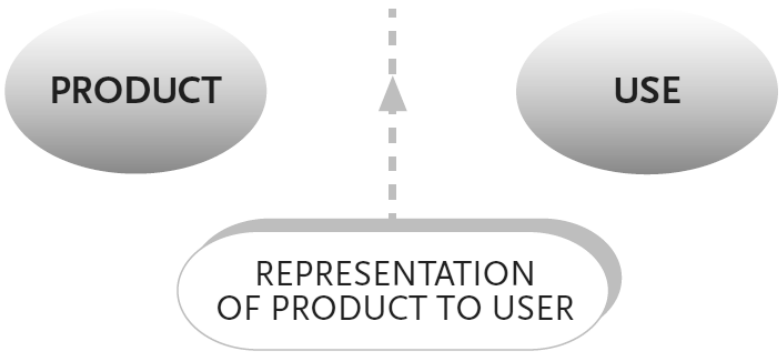


**Figure 16: Elements of medical device safety (WHO 2003).**

Pre-market review contributes to product control and post-market surveillance so that medical device in use is safe and effective (World Health Organization 2003). As it was presented in Figure 5 and Figure 6, the manufacturer is responsible for the product in the pre-market phase and user is managing the post-market phase. Representation of product to user is controlled through labeling and training (World Health Organization 2003). Representation is important, because the user might not have enough information about the medical device without training. In case of software, the labeling is placed to user manual or into the software, for example as a form of information boxes.

Sommerville presents three complementary ways to achieve safety in software:

1. hazard avoidance
2. hazard detection and removal
3. damage limitation

The system should be designed so that hazards are avoided. If hazards still exist, the system should be designed so that hazards are detected and removed before they result in accidents. If an accident could happen, the system should include protection features that minimize the damage that might result from the accident. (Sommerville 2001). The first and the second way are related to product element, and the third way is more related to the use element.

**Risk acceptability**

A medical device must be safe to use before it can be purchased. The risk management standard ISO 14971 is developed for that purpose. The main idea in IEC 14971 is that potential harms must be minimized. For this purpose, three levels for risks are presented:

1. Function level
2. Function in system
3. System

In function level the risk is identified. For all identified risks, risk control measures are considered. Whether it is possible or not to implement needed risk control measure, there still is the residual risk in the system. If it is possible to eliminate the risk completely, of course there is no residual risk in that case.

In the second level the residual risk shall be evaluated. If the residual risk is not acceptable, manufacturer shall perform risk/benefit analysis. However, in this level the residual risk is examined in system level. The questions that can be asked are: is the function really needed in the system, or what can be done to minimize the influence of identified risk to the surroundings.

In the system level, the whole medical device shall be evaluated. The overall residual risk must be acceptable before the medical device can be purchased. At this level, the residual risks have to be evaluated together to finding out if the system is safe enough to use.

**Risk Management Report**

Risk management report concludes the risk management process. In the report manufacturer shall carry out a review of the process. There are several important things to ensure:

1. The risk management plan has been appropriately implemented.
2. The overall residual risk is acceptable.
3. Appropriate methods are in place to obtain relevant production and post-production information.

Part one requires to reviewing the risk management file against the risk management plan and checking that all the requirements presented in the risk management file have been fulfilled. Part three requires the existence of the production and post-production information collecting system.

**Traceability**

Traceability is an essential dimension of a risk management system. Traceability makes it possible to be sure that the development process is complete. However, traceability is very difficult to do because of the massive amount of artifacts, combining both risk management artifacts and software engineering artifacts. The matrix that has to be produced is vast and requires unambiguous identification system.

In risk analysis there shall be documented the following trail according to ISO 14971:

1. From the hazard to foreseeable sequence of events
2. From the foreseeable sequence of events to hazardous situation
3. From hazardous situation to harm

Risk management standard is not the only one that gives instructions to risk management. In software life cycle processes standard there are also several mentions about risk management. The following trail relating to hazard shall be documented according to IEC 62304:

1. From the hazardous situation to software item
2. From the software item to specific software cause
3. From the software cause to the risk control measure
4. From the risk control measure to the verification of the risk control measure.

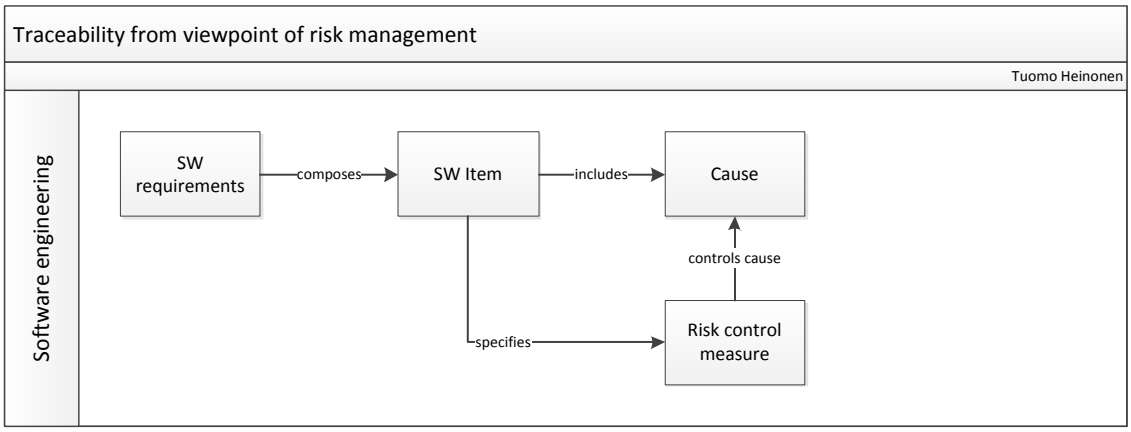


**Figure 17: Traceability and factors from software engineering viewpoint**

Figure 17 presents how the required trail relates to software specification and implementation. There are two aspects in risk management: traceability and risk control measures. Risk management system should take care of traceability. From the viewpoint of risk management there are several aspects that make things difficult. There can be errors everywhere and the error is difficult to find. Also the need of risk control measure might be difficult to realize and after realizing in some cases even more difficult to implement effectively.

### Software Safety Classification

Software safety classification is actually not the task of the risk management system. The purpose of the classification is to affect the development process of the software and is regulated in the software development life cycle standard.

According to IEC 62304, software architecture can be divided into software items. Those items should be decomposed into further items. When it is not possible to decompose a software item into further items, the last item is called software unit. Every software item should have its own safety class.

The safety classes are defined as follows:

Class A: No injury or damage to health is possible.
Class B: Non-serious injury is possible
Class C: Death or serious injury is possible

Decomposing and classification is a top-down method. Manufacturer shall first assign safety class to software system. After decomposing software system into software items each software item inherits the safety classification of the original software item or system. Items may be, however, classified separately if the item is segregated. In that case segregation has to be implemented by hardware and manufacturer has to document the rationale that explains the segregation.

The safety class affects the development of the software item. Class A software item may be developed with less documentation than class C software item. Actually, the safety class is basis for whole IEC 62304 standard and relating to every requirement of that standard there is defined to which classes the requirement concern.

In case of medical standalone software it is not possible to do any hardware segregation. Also, because medical standalone software handle only patient data, all functionalities relate somehow to patients, whether directly or indirectly.

## 4.5 Risk Factors Model for Medical Standalone Software

It's critical for medical standalone software that no hazardous situations are introduced. Unfortunately, it is not possible to guarantee that. It is mainly because there are too many factors and even more actors who use the software.

The software life cycle standard IEC 62304 is used as a starting point to classify the potential risk factors relating to medical standalone software. In the standard there are five potential causes of software item contributing to a hazardous situation presented:

1. Incorrect or incomplete specification
2. Software defects in functionality
3. Failure or unexpected results from SOUP
4. Hardware failures or other software defects resulting unpredictable software operation
5. Reasonable foreseeable misuse. (IEC 62304).

In the second situation the term defect when speaking of software is difficult to understand. The most probably it refers to incorrect implementation. The third situation relating to unexpected operation of SOUP is difficult for software manufacturer to ensure. SOUP is always developed by someone else and it is not always possible to get any anomaly list published by supplier of the SOUP.

A model was created concerning to risk factors in medical standalone software, based on those five situations. Figure 18 presents the model that describes how the use of medical standalone software can be understood and how the possible threat of harm will emerge. It is important to notice that there are usually one or more human decisions before any contact to patient. Some medical standalone software systems, however, can have straight contact to patient by SMS or email. In those cases there is only one human that is the patient not the physician to make the decision whether to believe the software or not.

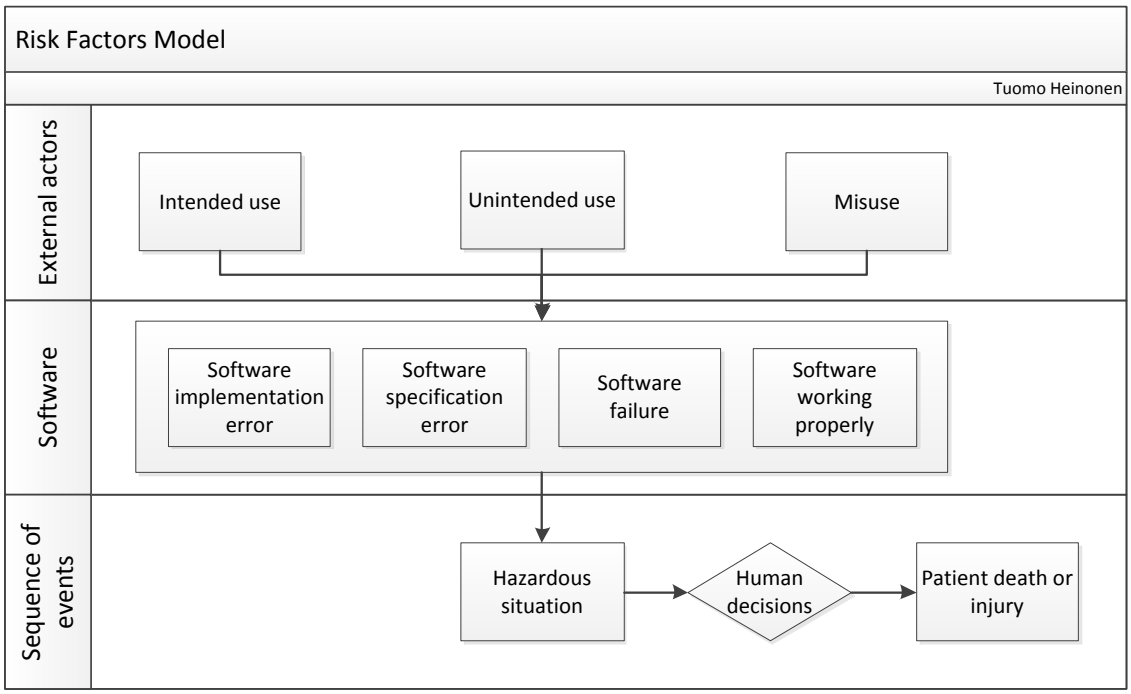


**Figure 18: Factors in software risk management.**

Figure 18 presents the potential sources of hazard and sequences of event leading to hazardous situation and harm. Potential sources of hazard are:

1. Intended use
2. Unintended use
3. Misuse
4. Incorrect specification
5. Incorrect implementation
6. Software failure
7. Software working properly

The use of software is divided in three groups and the operation of software is divided in four groups from the viewpoint of hazardous situation. External actors could be either humans or machines.

All of the factors can contribute a hazardous situation, whether alone or in combination with others.

### 4.5.1 Intended Use

Intended use is defined in standard ISO 14971 as a use for which a product, process or service is intended according to the specifications, instructions and information provided by the manufacturer (IEC 14971).

Intended use is equal with the case where the software is working properly, emphasizing the interfaces. Intended use itself of course is not a source of hazardous situation and is necessary for using software.

However, intended use is important from the viewpoint of regulation. Intended use defines the class of medical device and the direct consequence is that intended use defines whether the software is medical at all. Another point of view, the regulator might exclude the software from medical devices according to intended use.

### 4.5.2 Unintended use

Term unintended use is not based on regulation. Unintended use can be defined as following: using the software as intended but because of some reason, the user fails to use the software as intended. When intended use means that user uses software correctly and misuse means incorrect use, unintended use means that user wants to use software correctly but fails in it.

There are several foreseeable causes for unintended use:

1. Interrupts
2. Tiredness
3. Stress
4. Hurry
5. Carelessness, etc.

Those factors can contribute to the user to do something unintended. The most obvious reason, how the unintended use could happen is that the user does not recognize something that is intended to be recognized. Another case could be situation where the user mixes up something e.g. lines of array.

It is difficult or impossible to prevent all unintended use cases in the software. However, usability engineering techniques are effective tools to affect user's actions in positive way conducting more safe use. Techniques that can be used are presented in the chapter 4.7.1.

### 4.5.3 Misuse

Misuse is incorrect or improper use of medical standalone software. However, in case of medical standalone software, misuse is a little ambiguous concept. It is possible to identify some misuse relating to medical device that is connected to human body, but software system is so abstract concept that it makes also the term misuse abstract.

The conceptual problem in misuse with medical standalone software in general is that, the user inputs data to software and gets some data out. The user cannot control software to do things that are against its intended use, but he can input wrong data to soft-

ware, i.e. if it is intended that a user should input his name to a textbox and the user inputs his friend's name instead, the software does not recognize the wrong name, but acts like the name that was inputted was the user's own name. Whether this is a misuse or not, it does not affect the development of the software, because it is not possible to control this kind of semantic misuse.

On the other hand, there could be some accessories used with medical standalone software. In this kind of situation, misuse is related to the use of the interface. If the interface is correctly done, no harm arises, but if it is not, unexpected situation might appear.

### 4.5.4 Incorrect Specification

If the specification is incorrect, no one can guarantee if hazardous situation occurred there or not. However, incorrect specification is difficult to recognize. Customer requirements and software requirements should be consistent. It is actually a classic problem in software engineering that customer and manufacturer do not understand each other.

Often the situation is that customer wants a new feature to be implemented. The hazardous situation might appear, if the requirements are understood incorrectly and customer does not recognize it.

Also, as it was presented in chapter 4.2, the logic and calculation fault classes might have their source in the incorrect specification. It means that the incorrect specification factor might be really important to realize and to consider, because the specification defines the outcome of software development process.

### 4.5.5 Incorrect Implementation

In the field of software engineering, the incorrect implementation is very common problem. The complex software systems contain huge amount of code lines and only one mistake is needed for incorrect implementation. Snooke (2004) presents three possible implementation errors:

1. Logical
2. Algorithmic
3. Semantic.

In general, the incorrect implementation means that the software is implemented erroneous against specification. If the specification is unambiguous, the reason for incorrect implementation is programmer's mistake.

As it was presented in chapter 4.2, the largest fault class was logic faults. Requirements, design or code has the source of logic faults. Actually, design and code are in the scope of incorrect implementation. Design and code review are prevention methods for logic

faults. Walkthrough the implementation against the specification was also suggested as a prevention method. Code reading, inspection and testing were suggested as a detection method.

### 4.5.6 Software Failure

Even though software failure is mentioned as one of the causes for hazardous situation, there is nothing that can be done for it in software development process. That is, because software cannot fail in the context of medical standalone software. In general, content of software failure is also quite difficult to define. According to Snooke three causes of failure for software:

1. Abnormal input value to the software
2. Failure in the hardware that affects to software
3. Logical, algorithmic or semantic error in the implementation. (Snooke 2004).

It's possible to ensure that no abnormal values are inputted. If abnormal values, however, are inputted, whether it is a software failure or incorrect specification is more difficult question. To be consistent, it is defined to be a cause of incorrect specification because specification defines software.

Hardware failures, instead, relate more to embedded software development process than medical standalone software development process, and nonetheless, it is very difficult to get ready for hardware failures within software development process and in most cases it is even impossible. For medical standalone software, hardware failures are, however, responsible of operation administrator.

Snooke presents also the error in the implementation as a software failure. However, in this thesis it is divided to be in its own category. There might be a correct implementation of an algorithm in a software but that algorithm is not specified in the specification. In that situation it is clearly incorrect implementation against specification instead of a software failure. The difference between software failure and implementation error is that cause of the software failure is not because of the programmer but implementation error is.

### 4.5.7 Software Working Properly

There might be some hazardous situations either because of residual risk or unforeseeable risk. That is because of the nature of medical device. There are three causes to that:

1. Hazardous situation is not foreseeable and thus no risk control measure is possible to implement.
2. There is no need for risk control measure, because the identified risk was acceptable after evaluation.

3. There is a risk control measure implemented. Whether the residual risk was acceptable after the evaluation or after the risk/benefit analysis, the residual risk still exists.

### 4.5.8 Summary

The seven factors were identified that can contribute to hazardous situation. To prevent hazards to happen, all the factors shall be eliminated, because there are several combinations that can produce hazardous situation, but even one factor can produce it. The risk management system should consider all of these factors somehow.

It is obvious that there is no way to eliminate intended use or the state where software is working properly, even though they might contribute to a hazardous situation. These two factors shall be considered at least in the documents relating to medical standalone software. The software failure is difficult to eliminate, because it does not include in the development process of the medical standalone software. However, the factor shall be considered in the use of medical standalone software.

## 4.6 Risk Assessment

There are risks in different levels relating to medical standalone software. For example software units, items and system could all be contributing to a hazardous situation.

Following risk analysis methods described in Figure 19 are identified in ISO 14971. Not all of those methods are applicable to standalone software risk analysis. Methods have been chosen to standard obliviously because of covering all the stages of development.

| Method | Development stage | What can be found? |
|---|---|---|
| PHA | Early in the development process. | Identify hazards, hazardous situations, and events causing harm. |
| FTA | Early in the development stages. | Identification and prioritization of hazards. |
| FMEA, FMECA | Design matures. | Identification of effect or consequences of individual components. |
| HAZOP, HACCP | Latter stages of the development. | To verify and then optimize design concepts or changes. |

**Figure 19: Risk analysis methods presented in standard ISO 14971.**

What makes software so special is that there are not only different development phases but also different layers in the product. Different methods should be applied to different layers in the software to control and analyze the risks.

Also the persons that relate to risk analysis activities do matter. Lindholm and Höst made a study, where group of physicians, group of developers, and group of medical device developers analyzed risks individually regarding to new patient monitoring system risk scenario. The assumption was that multiple roles will affect the list of identified risks and the list will be more complete. The research conducted conclusion that the different experiences affect the risk identification and prioritization of the risks. However, there wasn't distinct difference in the kind of identified risks between those groups, expect that physicians did not identify any risk relating to development risk. (Lindholm and Höst 2009).

## 4.7 Risk Controlling

The standard ISO 14971 presents three risk control options. Those three options are listed in the priority order:

1. Inherent safety by design
2. Protective measures in the medical device itself or in the manufacturing process
3. Information for safety

According to Technical Report, the first option could involve:

- Eliminating unnecessary features
- Changing the software architecture to avoid hazardous situations
- Simplifying the user interface to reduce the probability of human errors in use
- Specifying software design rules to avoid software anomalies.

With standalone software it is difficult to apply the changing of the software architecture. Of course something can be done, but there is not much difference with different architectures from the viewpoint of safety. In some cases the attention to safety might produce even more complex architecture. The simplifying of the user interface relates directly to unintended use factor, and software design rules specification relates directly to incorrect implementation factor. (Technical Report 2009).

In the second option, the protective measures in the medical device itself are almost impossible to apply with medical standalone software. The Technical report clarifies that protective measure should be independent of the function to which it is applied. This can be done if hardware protective measure is applied to software. If the protective measure is implemented in software and applied to software, it is important to avoid multiple failures arising from one cause. Using software as a risk control measure is not helping to prevent implementation errors because it is not possible to be sure that im-

plemented risk control measure does not have another implementation error. (Technical report 2009). Protective measures in the manufacturing process are not described in detail in the Technical Report, but those can be used in incorrect specification factor and incorrect implementation factor. Rigorous manufacturing process is thus acceptable risk control measure.

The third option is always possible in case of software. According to Technical report, information for safety could mean everything between simple on-screen warnings and complex user manuals with defined training courses in case of software. The complexity of such written material can be reduced by good user interface design. (Technical Report 2009)

### 4.7.1 Usability Engineering as a Risk Controlling Method

**Usability Engineering**

In the field of usability engineering there is a lot of research done and conducted to usability engineering principles and guidelines. During last ten years usability has got increasing attention also in the context of information systems in healthcare.

Also the technical report mentioned user interface multiple times. Actually, all of the three risk controlling options included a mention about the user interface and usability.

**Guidelines for Usability Engineering of Clinical Information Systems**

One usability program was made within MIRACLE project by Philips Medical Systems. Program included heuristic evaluation, two usability tests and weekly interviews with participants. The program resulted following guidelines for conducting usability programs co-operation with medical professionals:

1. Focus user requirements-gathering on user goals and tasks
2. Perform heuristic evaluation of the software to identify and correct obvious problems before usability testing
3. Budget extra time and recourses for participant recruiting, scheduling and orientation, both early and throughout the project
4. Provide the most accurate, up-to-date data as possible, even in the test setting
5. Use methodologies that permit some flexibility in study design or implementation. (Rosenbaum, Hinderer and Scarborough 1999).

**Usability Attributes and Their Relation to Risk Management**

These attributes are at same time the characteristics of medical device mentioned in 4.2 (ISO 14971). Also the definition of usability is part of that. Actually there are main levels from the definition of usability and second-levels from these attributes.

| Attribute | Relation to Risk Management |
|---|---|
| Learnability | - Employees should learn the system as fast as possible; otherwise something important might not be recognized in the use of medical standalone software that contribute a hazardous situation.<br>- Production is going on without learning phase, so for the new employee there is no time to spend to learn the system.<br>- Physicians and other staff might be unwilling to spend their time receiving training (Rosenbaum, Hinderer and Scarborough 1999) |
| Efficiency | - Psychological issue if the system is too slow: users become frustrated.<br>- If the system is too slow, users might start thinking that the system is somehow broken and do something unpredictable and unintended.<br>- Risk is not easily measurable<br>- Included in regulation |
| Memorability | - A holiday can't cause the situation where employee forgets how to use the system.<br>- Proper training might be missing.<br>- It might not be possible to spend much time reading manual. |
| Errors | - The possibility not to recognize the error might cause a threat for patient safe.<br>- Employees do not have spare time to re-do things |
| Satisfaction | - Psychological issue: user becomes frustrated.<br>- Risk is not quantifiable and not measurable. |

**Figure 20: Usability attributes and their relation to risk management.**

Figure 20 presents usability attributes and their relation to risk management. The standard IEC 62366 "Application of usability engineering to medical devices" also discusses about usability issues relating to software and expands the risk management standard ISO 14971.

### Use of Usability Heuristics

Usability heuristics can be defined as a basis for a systematic inspection of a user interface to find its usability problems (Nielsen 1993).

A heuristic evaluation was done in MIRACLE project independently by three usability specialist in the beginning of the program. It was possible to identify obvious usability problems and recommend changes to the user interface. (Rosenbaum, Hinderer and Scarborough 1999).

There are several usability heuristics to use. Nielsen presents following ten usability heuristics (Nielsen 2003):

1. **Simple and Natural Dialogue**
   - User interfaces should be simplified as much as possible, because it is possible to misunderstand everything.
   - There should be exactly the information the user needs and no more.
   - Colors should only be used to categorize, differentiate and highlight, not to give information.
2. **Speak the User's Language**
   - Terminology in the user interfaces should be based on the user's language and not on system-oriented terms.
3. **Minimize User's Memory Load**
   - Software should take over the burden of memory from the user as much as possible.
4. **Consistency**
   - The same information should be presented in the same location on all screens and should be formatted in the same way to facilitate recognition.
5. **Feedback**
   - The system should continuously inform the user about what it is doing.
   - Feedback is especially important in case of long response times.
   - Informative feedback should be given in case of system failure.
6. **Clearly Marked Exits**
   - The system should offer the user an easy way out from as many situations as possible.
7. **Shortcuts**
   - It should be possible for the experienced user to perform frequently used operations especially fast.
8. **Good Error Messages**
   - Error messages present opportunities for helping the user understand the system better.
9. **Prevent Errors**
   - Better than having a good error messages would be to avoid the error situation in the first place.
10. **Help and Documentation**
    - Fundamental truth is that most of the users don't read manuals.

## 4.8 Production and post-production information

There shall be established, documented and maintained a system for the production and post-production phases of medical device according to ISO 14971.

There are also other requirements for the system:

1. Manufacturer shall collect and review information about medical device or similar devices in the production and post-production phases.
2. Manufacturer should consider the mechanisms by which information is collected and processed. The operator or the user might generate this information. Information might be generated by those accountable for the installation, use and maintenance of the medical device.
3. Manufacturer should collect and review publicly available information about similar medical device on the market.
4. Manufacturer should consider new or revised standards.

Collected and reviewed information shall be evaluated for possible relevance to safety.

Manufacturer should consider at least following conditions in the evaluation:

1. Previously unrecognized hazards or hazardous situations are present.
2. The estimated risks arising from a hazardous situation are no long acceptable.

If any of the above conditions occur, manufacturer shall do following actions:

1. The impact on previously implemented risk management activities shall be evaluated.
2. The impact on previously implemented risk management activities shall be fed back as an input to the risk management process.
3. A review of the risk management file for the medical device shall be conducted; the impact on previously implemented risk control measures shall be evaluated, if there is a potential that the residual risks or its acceptability has changed.

# 5 THE DEVELOPED RISK MANAGEMENT SYSTEM

## 5.1 Developed Risk Management System

The risk management system affects the whole lifecycle of medical standalone software. In the beginning, customer requirements are collected and analyzed in case of foreseeable hazards. In the software development phase the risk management system affects to process itself, architecture and function development. After release post-production use and operation phase will be applied. When a new product develpment is starting according to this risk management system, the information should be produced from the whole life cycle of medical device.

The risk management system has to cooperate with different functions of company relating to software manufacturing. Manufacturer might be responsible for post-production operation of the software system. The environment, where the medical standalone software is running on, might be e.g. one kind of cloud service or simple workstation. Within this thesis the post-production phase is not covered, even though Figure 26 presents a possible draft about the post production phase. The risk management standard does not specify if the post-production operation is in scope of risk management system activities or not.

Figure 21 presents the developed risk management system that consists of four phases: preliminary planning, software development, post-production use and operation, and production and post-production information collecting system. Each of the phases relates to some manufacturing phase in the company. The production and post-production information collecting system presented in Figure 27 collects information as presented in chapter 4.8.

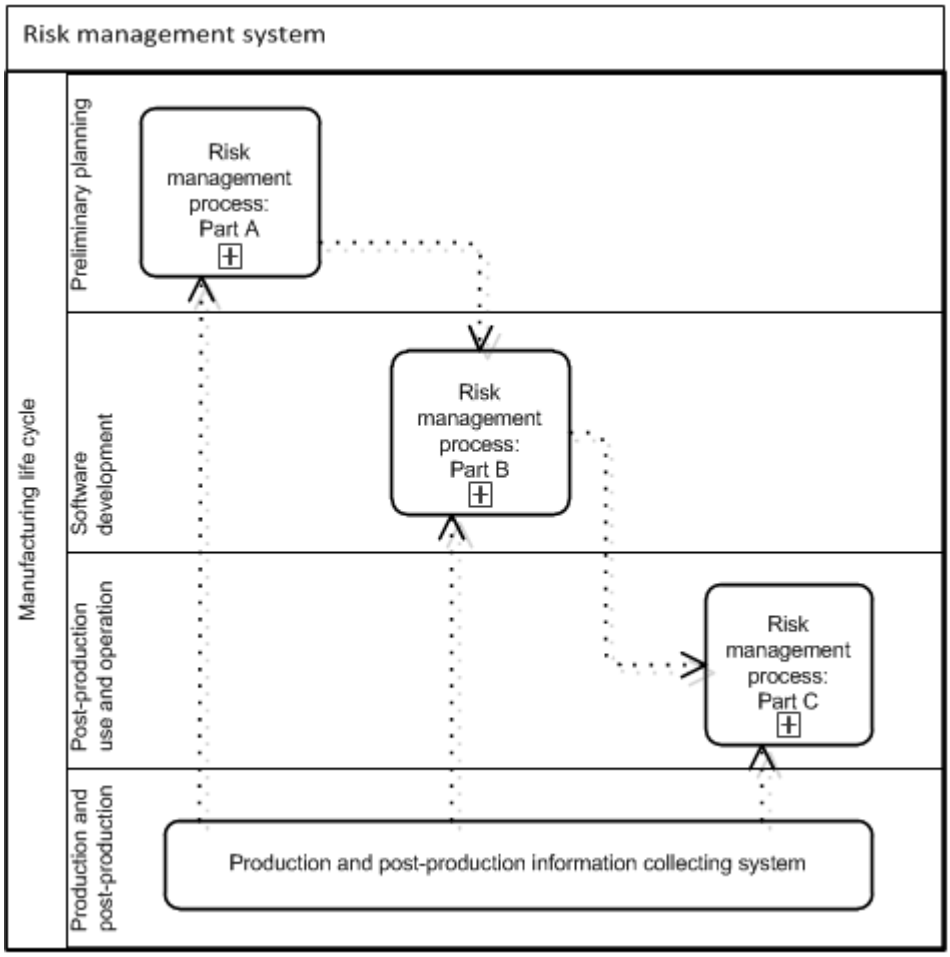


**Figure 21: Developed risk management system.**

Risk management system is developed in accordance of ISO 14971. It means that all the requirements of that standard, presented in chapter 3.3.3, are fulfilled. The traceability is handled by using a matrix, where all information about risk assessment and risk controlling are placed.

Production and post-production information collecting system is used as a hub that connects all the parts of risk management process together. The production and post-production information collecting system can be used also to deliver information between different company functions.

### 5.1.1 Risk Management Process

Risk management process consists of three parts: Part A, Part B, and Part C. It is simple to have one risk management process containing different procedures in different phases. Otherwise three processes shall be considered. In Part A and Part C the standard ISO 14971 is applied as appropriate, but in Part B all the requirements are fulfilled.

Part A is about preliminary planning. In this phase no line of source code is written. Part A is active in the planning phase of a new product. The risks that are analyzed and iden-

tified are more business risks, and not risks relating to hazardous situation and harm. However, some important knowledge might emerge that is used by software development team.

Part B is about software development. In that phase all the software is developed and released. That phase has to be integrated to software development model tightly.

Part C is about post-production use and operation. That phase is active after software release.

**Part A: Preliminary planning**

In this phase the project is starting and customer requirements are asked. The risks involved here are more business risks than patient risks. However, the characteristics related to safety and hazard identification is started in this phase, because those can serve in Part B when software architecture is started to develop and customer requirements are started to interpret to software specification. Figure 22 presents the Part A.

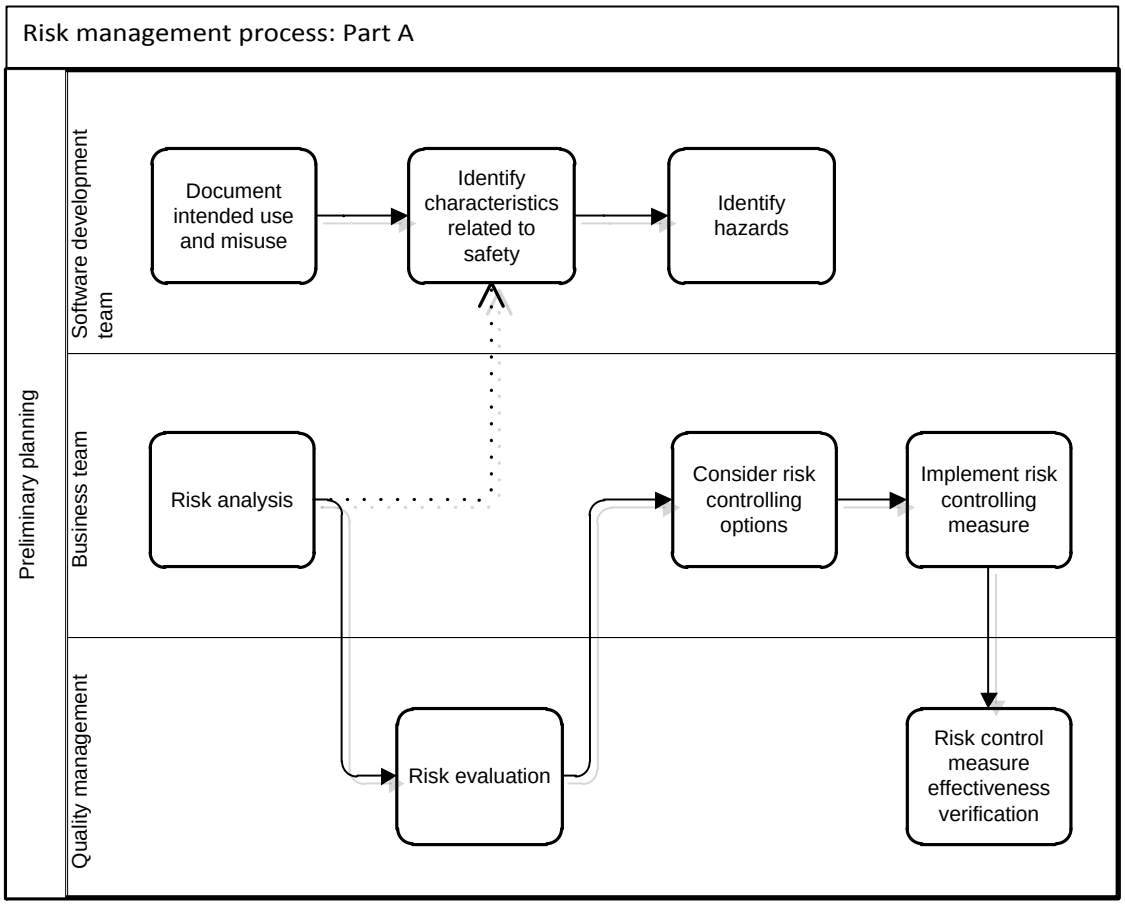


**Figure 22: Risk management process: Part A**

*Inputs*: Customer requirements

*Outputs*: Risk assessment and proposed risk controlling measures of business risks. Preliminary risk analysis of software development relating to Part B.

**Part B: Software development**

This phase includes the software development process from software requirements analysis to software release. Software development phase begins from constructing of software specifications from scratch. All the risks involved in this phase are patient risks.

The structure of Part B demonstrates the three levels of risks, and how the risks flow from function level to system level. The B-1 presented in Figure 23 is for the function level risks, the B-2 presented in Figure 24 is for the function in system level risks, and the B-3 presented in Figure 25 is for the system level risks.

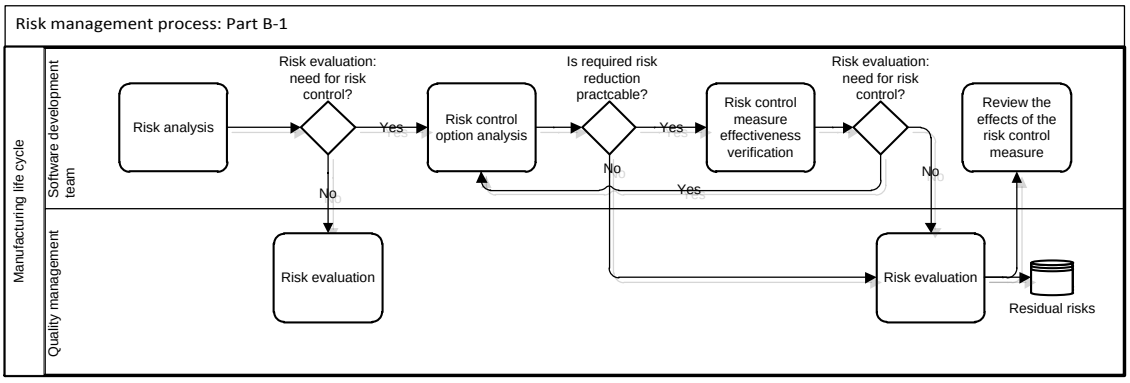


**Figure 23: Risk management process: Part B-1**

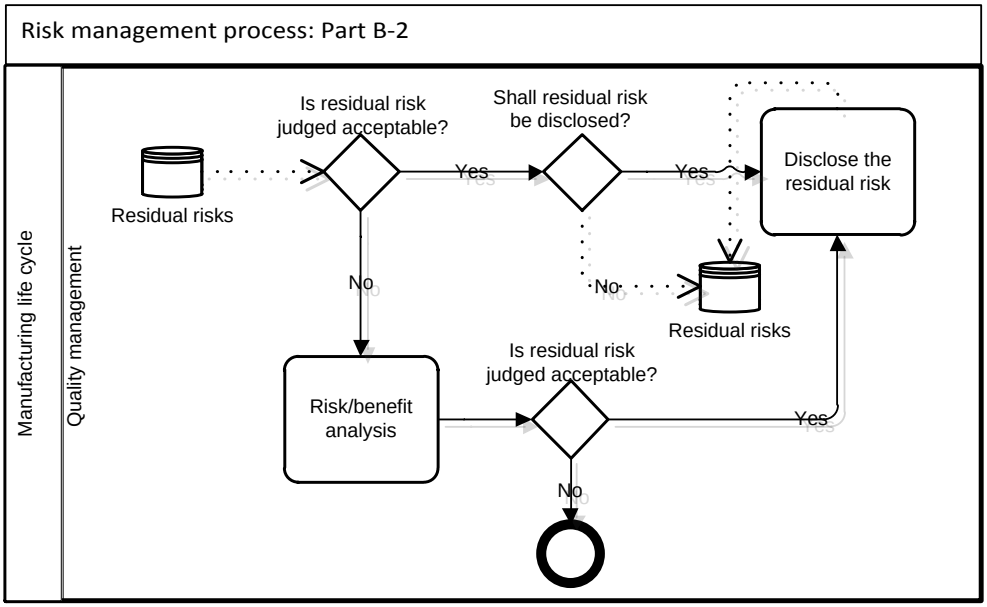


**Figure 24: Risk management process: Part B-2**

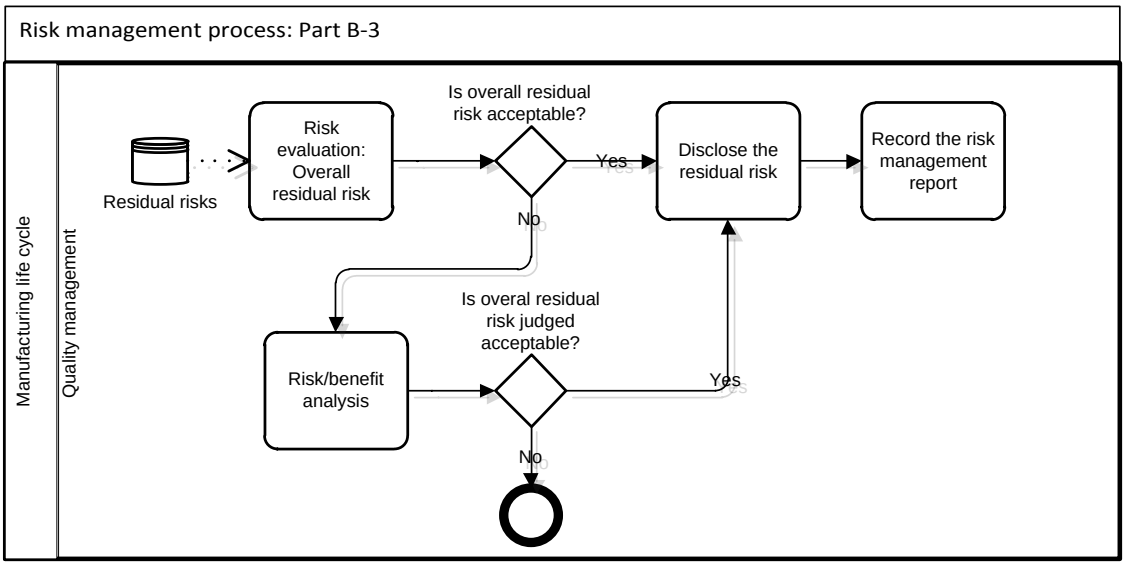


**Figure 25: Risk management process: Part B-3**

*Inputs*: Preliminary risk analysis from Part A.

*Outputs*: Risk assessment and risk control measures, risk management report and risk/benefit analysis.

### Part C: Post-production use and operation

This phase includes software operation environment. In this phase software is released, and risk management process concentrates to hazards, which might contribute a software failure. The risks involved here are both business risks and patient risks. Figure 26 presents the post-production use and operation phase.

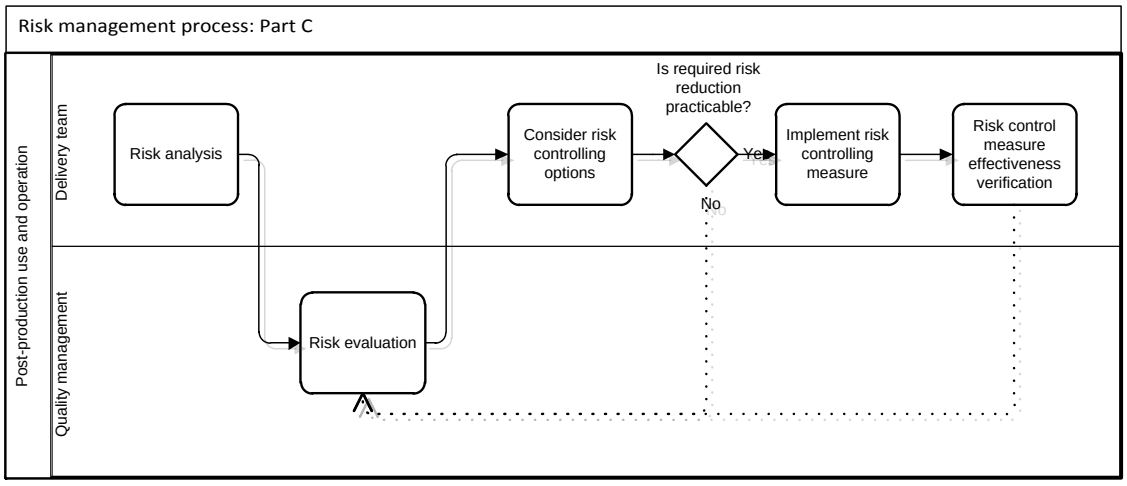


**Figure 26: Risk management process: Part C-1**

*Inputs*: Documentation created for post-production use and operation.

*Outputs*: Risk assessment and risk controlling measures.

### 5.1.2 Production and post-production information collecting system

Production and post-production information collecting system is developed based on the requirements relating to IEC 14971. Information flow to information collecting system includes all phases relating to medical device manufacturing, and also the information that is possible to get outside of the company. Also, new or revisited standards are observed. The information collecting system is presented in Figure 27.

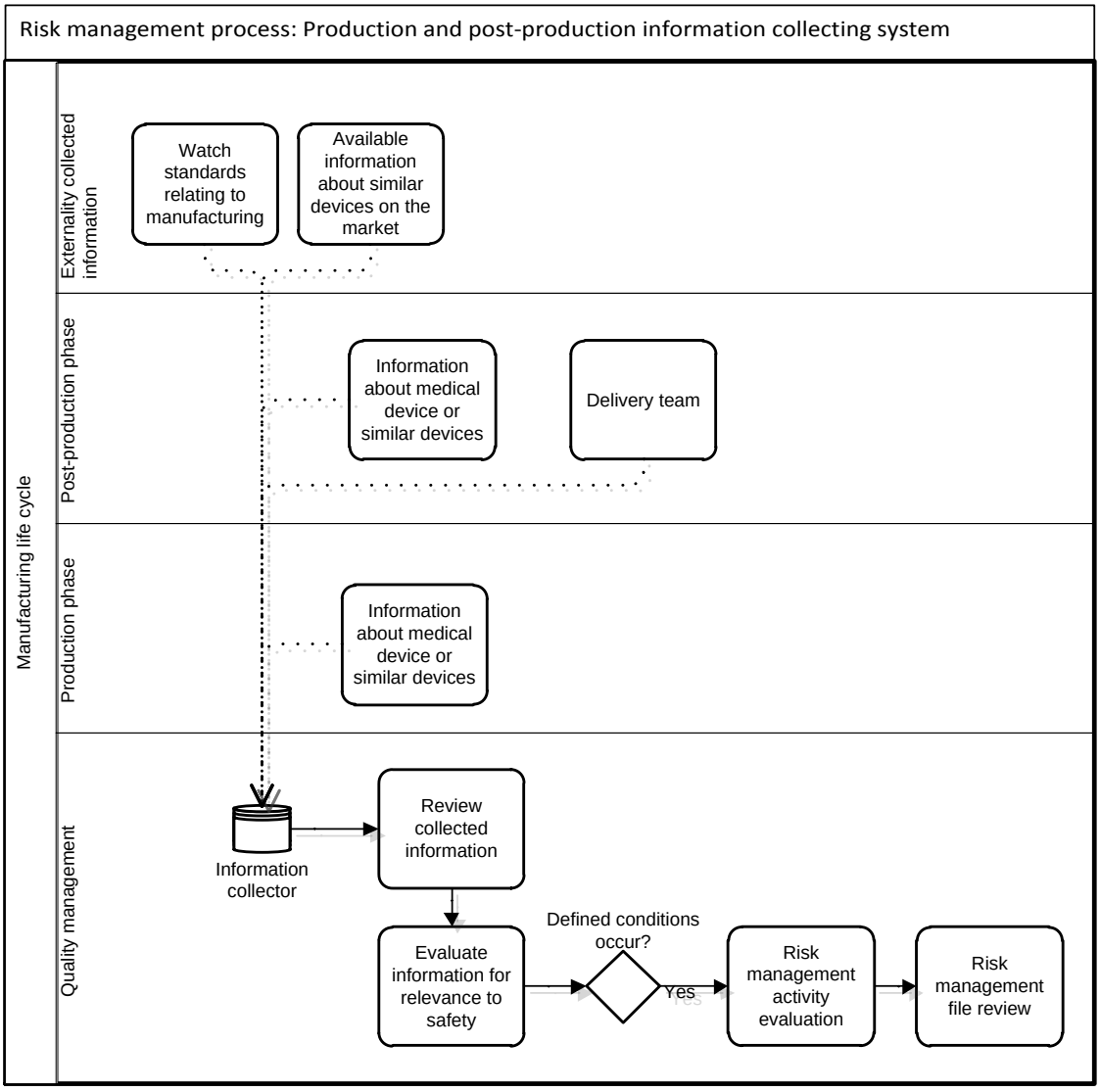


**Figure 27: Production and post-production information collecting system.**

The instance of the production and post-production information collecting system is quite difficult to emerge. In interphase the information collecting system constructed so that company use old active reporting systems and only defines the information streams relating to those old systems. Later, all the separate reporting systems can rationally be replaced with one system if needed. The system should be able to handle all the information streams relating to the medical device and its lifecycle.

### 5.1.3 Risk Assessment and Risk Controlling Methods

Chapter 4.5 presents the factors that could contribute hazardous situations in case of medical standalone software. Those factors can appear alone or in combinations. For total safety, all of those factors have to be prevented or somehow be treated. This chapter presents how to get prepared against them.

Not all of these factors are appropriate to every risk management phases. Figure 28 presents for each factor, in which phase the factor is applicable and will be considered. Generally, the factors in the phases are divided according to possibility of risk control activities. If the factor is not possible to control in the phase, it is not applicable. If the factor can be controlled in the phase, it is applicable.

| Factor | Preliminary planning | Software development | Post-production use and operation |
|---|---|---|---|
| Intended use | Applicable | Applicable | N/A |
| Unintended use | Applicable | Applicable | N/A |
| Misuse | Applicable | Applicable | N/A |
| Incorrect specification | Applicable | Applicable | N/A |
| Incorrect implementation | N/A | Applicable | N/A |
| Software failure | N/A | N/A | Applicable |
| Software working properly | N/A | Applicable | N/A |

**Figure 28: Risk factors that should be considered in risk analysis in presented phases.**

It is significant to notice that even though risk factor is applicable in some phase, it is not always active there. For example, incorrect specification is applicable in software development phase only when new customer requirements are given or the specification against customer requirements is changed. That is because specification has to be compared against customer requirements, which is the input to correct specification.

In the preliminary planning phase, there is no software development as such. That is why implementation, failure and software working properly are not applicable there. Of course there can be some ideas about possible failures, but actually those are implemented in post-production use and operation phase. Or there might appear intended use

that could contribute hazardous situation, but in the fact there is no software developed so the factor cannot be applicable.

Software failure is the only one in the post-production use and operation phase that is applicable. Of course other factors can be appeared in the post-production use and operation phase, but the corrective actions can be done for them only in software development phase. Vice versa the hardware-related things that are caused by software failure factor can be considered in post-production use and operation phase and not in the other phases.

#### 5.1.3.1 Intended Use

**Risk analysis**

This risk factor includes all those hazards that are not foreseeable or are relating somehow to characteristics of medical standalone software.

When constructing a software specification, intended use shall be considered in case of hazards. However, hazards relating to intended use are those that are identified in case of unintended use.

**Risk control measures**

This factor's risk control measures are used to disclose the residual risk.

Instructions and information shall be placed to the user manual and help section in the software, in order to inform, motivate and enable the user to use the device safely.

#### 5.1.3.2 Unintended Use

**Risk analysis**

Usability principles shall be applied over the software user interface trying to find usability problems that could contribute a sequence of events resulting in a hazardous situation.

Foreseeable hazards: poor usability containing complicated and unnatural dialogue, not speaking the user's language, user memory load, inconsistency, no feedback, unclearly marked exits, no shortcuts, bad error messages, no error prevention, and no help nor documentation.

**Risk control measures**

Proposed modifications to user interface based on applied usability heuristics.

### 5.1.3.3 Misuse

**Risk analysis**

There are no general foreseeable hazards relating to misuse.

**Risk control measures**

If it is possible to identify some misuse, the risk control measures that can be used are the same than in case of intended use.

### 5.1.3.4 Incorrect Specification

**Risk analysis**

Formal risk analysis is not appropriate for this risk factor.

Foreseeable hazards: customer requirements are different comparing to software requirements; customer wants new functionality that is wrongly specified by manufacturer.

**Risk control measures**

Requirement and design review: reviewing customer requirements to software requirements in case of differences or inconsistencies.

Traceability between customer requirements, software specification, software implementation, and testing.

### 5.1.3.5 Incorrect Implementation

**Risk analysis**

Formal risk analysis is not appropriate for this risk factor.

Foreseeable hazards: incorrect implementation.

**Risk control measures**

Rigorous software development process that includes validation activities, verification activities, and testing activities.

Unit testing and integration testing. There should be testing scenarios at least for critical components of the software.

Design and code review: reviewing design against software specification and code against software specification and design.

Walk through the software implementation against design.

#### 5.1.3.6 Software Failure

**Risk analysis**

For software failure factor suitable risk analysis tools are FTA or PHA.

Foreseeable hazards: hardware break (memory, cpu, hard disk), wrong operation settings.

**Risk control measures**

General system administration and configuration activities (duplicated hardware, backups, etc.).

#### 5.1.3.7 Software Working Properly

Same principles shall be applied than in case of intended use. The difference between this risk factor and intended use is that this is from the viewpoint of the software and intended use is more from the viewpoint of user.

## 5.2 Case: How to Use the Risk Management System

The risk management system process activities must be trained for employees. The risk management system is integrated as a part of the quality management system and thus before training, the guideline for risk management process must be made. The basic principle is that employees should follow the instructions presented in the quality management system.

According to the risk factor model, all the possible risk factors shall first be considered relating to developed software system. The results of the consideration shall be recorded to risk management file. Here is presented how to apply afterwards risk management system for medical standalone software.

It is not possible to identify the foreseeable misuse. That is because of the applied definition of misuse. Also software failure factor is not considered, because of the development phase of the software system.

Both incorrect specification and incorrect implementation is not possible to consider. The software is developed by using the software development model that might not be as strict as it is instructed in the new risk management system. However, not much can be done for that. Incorrect specification is also quite difficult to consider, because the traceability exists only partly, and in few years the customer requirements are merged to the software specification.

Intended use and the state where software is working properly will be considered while finalizing the user manual. Also, with all the produced specification after the starting point of risk management system, these two factors could be considered.

Unintended use is the only factor that could be considered immediately. The analysis can be conducted by using Nielsen's ten heuristics presented in chapter 4.7.1. The software architecture allows analyzing the user interface components separately. It is easy to do in the beginning a heuristics evaluation for user interface components separately.

## 5.3 Assessment of Risk Management System

There are several important attributes relating to risk management system. The purpose of the risk management system and other regulation is to ensure and provide safety, effectiveness and quality for the medical device. The most important objective of risk management system is to ensure safety.

To assess the risk management system in accordance to this particular objective is extremely difficult. The testing environment should contain two similar medical device development processes, with and without the risk management system. The extreme assessment of the results of those two development processes is not possible to do. All that matters is minimizing the injuries of the patients. However, this is not possible to test.

Some assessments could be done by examining the identified hazardous situations and the risk controlling measures relating to them. However, in this thesis it is not possible to do any tests, where two adjacent risk management systems would be used. There is no truthful way to assess, how the developed risk management system really assists the safety development of the medical device or how reliable it is regarding to risk reduction.

In the chapter 4 the nature and principles of the risk management system are presented and in chapter 5 presents the developed risk management system. The risk management system was developed according to the principles that were presented in the chapter 4. As it was described in chapter 5.2, not all the parts of developed risk management system were possible to implement or to use. However, the objective assessment of those parts that were possible to implement and use, is not possible to conduct.

# 6 DISCUSSION

## 6.1 Benefits of Regulation

**Big Companies**

Development that has been described in this thesis might cause a situation where small companies lose some of their advantage in agility. That might be caused because of fulfilling all requirements from regulation needs some employees that are especially dedicated to do quality and regulation related things.

The question that arises is why European Union wanted all the companies in software industry that develop medical software under the control. For big companies it is clear that the more regulation there is, the more difficult for small competitors it is to do business. Another point of view, the more regulation produces most probably more safe software.

However, it seems clear that those big companies who afford to put people doing risk and quality management and whose processes are already in order will benefit about this regulation. It may be even so that in some cases the big companies need not to have necessarily to change anything because they might already have the essential requirements fulfilled.

**Patients**

Regulation forces companies to do their best effort to make all software safe. Before MDD 2007 there was a small gap for those software systems that were not in straight contact with patients. There was not any regulation for finding out the possible risks and fix them. For example, a laboratory information system assists doctors to choose right drugs for patients. If the system has some problem, it makes it possible to prescribe wrong drugs.

**Society**

When all companies in medical devices field including software system manufactures have to apply for CE mark in Europe to launch their software product, it is definitely nice for hospitals both in public and private sector to know by sure that the system they are going to buy is accepted by public authority.

## 6.2 Usability Engineering in Risk Management

Now there is in one mention about usability engineering in the software requirements chapter in the standard IEC 62304. As it was presented in this thesis, usability has to have much more attention in software development of medical standalone software. For example Philips has already in 1999 give attention (Rosenbaum, Hinderer and Scarborough 1999) to usability and they found out essential principles for development of medical standalone software. Actually, the need is identified for framework to identify, classify and prioritize errors in the healthcare context (Sarnikar and Murphy 2009).

There is a mention about usability engineering in IEC 62304. The standard references to another standard IEC 60601-1-6 which is replaced with IEC 62366. Regarding to what is discussed in chapter 4.7.1, one possible branch of developing is to require fulfilling standard IEC 62366 because it is huge impact to safety of medical standalone software.

## 6.3 Post-production Operation

According to ISO 14971 the requirements of the standard are applicable to all stages of the life-cycle of medical device. This is the only reference to operation stage of medical device and actually the standard assumes that after release the manufacturer is not in responsible for the medical device operation phase. That indicates, for example, the need to do risk management report before release and after that only production and post-production system collects information that should be considered.

In Finnish legislation there is mentioned in 12 § that manufacturer is responsible for design, manufacturing, packaging and marking of the medical device. There is no mention about post-product operation.

It seems that everything depends on contract between manufacturer and the medical care provider that uses the medical device. The contract should consider about the situation where patient risk appear.

## 6.4 Risk Management Development in the Future

### 6.4.1 Some Difficulties Relating to Risk Management

There are quite many difficulties incorporated to risk management system. The major difficulty is that risk management system is separated from the software development model. However, in real life the risk management system is still in close relationship with the software development model. One solution to this problem is to combine software life cycle processes standard and risk management standard to one standard for medical standalone software.

The second difficulty in general is the traceability. Even though traceability is possible to do, the benefit of it might stay minimal. This is not the problem of any standard, but as long as traceability is required, there should be some solution to administrate the traceability. The trivial solution is that some company manufacturers the software for medical standalone software requirement management. However, the traceability is still an important risk control measure relating to incorrect specification.

Relating to the secondly identified difficulty, the lack of requirement management is an actual source of hazards. In that management system there should be also the configuration management. What is even more challenging is to apply agile methods in developing of medical standalone software.

### 6.4.2 Standard for Medical Standalone Software Development

Current standards are not appropriate enough for special requirements of medical standalone software. Especially the software life cycle processes standard IEC 62304 is not developed for standalone software that is operated by manufacturer after release.

The software safety classification that was presented in chapter 4.4 is a rational way to apply different levels of rigorous for software development processes. Unfortunately, in case of medical standalone software, the classification is totally irrelevant. As it was said, no hardware segregation is possible to do.

In the future, there must be a specified standard for medical standalone software development. The standard should be in close relationship with the risk management. The incorrect implementation factor in risk factors model is managed by rigorous software development process. The software safety classification would make sense, if the rigorous level of software development could relate the software item development as it is also in current standard. In future, there should not be any requirement of hardware segregation. The difference between embedded software and standalone software is that in standalone software the risk does not flow to other software items.

One possible way to analyze the software item is to construct some formula with two attributes: the amount of people who are related to development process of specified software item, and the amount of sequential software items.

### 6.4.3 Risk Factors Model

As it was said already, all of the factors of risk factors model can contribute a hazardous situation whether alone or in combination with others. Further studies should be conducted at least with following:

1. What are the most usual combinations of risk factors?

2. How the risk management system could affect to clinical decision making to prevent medical errors relating to medical standalone software.
3. Usability studies among health care staff to identify and learn the most severe problems.

The Risk factors model does not deeply concentrate to post-production use and operation. Further studies should be conducted in accordance to be sure the real effects of post-production phase to medical standalone software safety.

# 7 CONCLUSION

There were two purposes for doing this master thesis. The first one was to find out what the reasons are for European Union to put all the software under the regulative control and how they are doing it. The second one was to develop a risk management system for medical standalone software.

To reach the first goal, first the history of software engineering and quality development, and the regulation for medical devices was viewed. The case like Therac-25 definitely affected to development of regulation somehow. The Global Harmonization Task Force was conceived to uniform the regulation system, but also to serve as a discussion forum.

The second goal was more specific. Chapter 4 presents the aspects relating to risk management system. The detailed attention was given to medical standalone software and its differences to other medical device. The detailed information was received from study concerning the recalls of medical devices containing software.

The risk factor model was developed in this thesis. With this model it was possible to divide possible causes of medical standalone software. Also according to the model, it was possible to study the possible risk control measures. The major finding within this thesis was the use of usability engineering to control risks in medical standalone software.

Finally, the developed risk management system for medical standalone system consists of four parts: preliminary planning, software development, post-production use and operation, and production and post-production information collecting system. The first three parts constitute the risk management process.

The developed risk management system was taken in place after it was completed. Until now, no feedback has emerged yet. The assessment of the risk management system, partly because of that, is quite challenging.

During the project it became clear that risk management standard is not specified for medical standalone software. The most probably in the future a new risk management standard takes place. The standard should be specified for medical standalone software, and the best solution would be, if the risk management standard and the software life cycle processes standard were merged into one standard.

# REFERENCES

Australian Regulatory Guidelines for Medical Devices. 2010. Therapeutic Goods Administration. Referenced 25.7.2011 from http://www.tga.gov.au/pdf/devices-argmd.pdf.

Basler, R., Pizinger, R. 2004. The Arrival of ISO 13485:2003 – New standard shifts quality system from procedure-based to process-based. Medical Product Outsourcing. Referenced 12.7.2011 from http://www.fdatraining.com/Noblitt-Rueland_ISO13485-2003_Part1.pdf and http://www.fdatraining.com/Noblitt-Rueland_ISO13485-2003_Part2.pdf.

Beck, K. 1999. Extreme Programming Explained. First edition. ISBN: 0201616416, p. 224.

Boehm, B. 2006. A View of 20th and 21st Century Software Engineering. In Proc. ICSE'06. ACM Press. pp. 12-29.

Brooks, F. 1986. No silver bullet – Essence and accidents in software engineering. Proceedings of the IFIP Tenth World Computing Conference. H.-J. Kugler, ed, Elsevier Science B.V.,Amsterdam, NL. pp. 1069-76.

Caminer, D. 2001. LEO and the Computer Revolution. In Conference Proceeding of the Business Computing the Second 50 years. The Guildhall Conference for business leaders. Referenced 6.7.2011 from http://is2.lse.ac.uk/leo/archive/caminer.pdf.

Chiravuri, A., Ambrose, P. 2003. Investigating the effects of downsizing on software professionals' self-efficacy and its consequences on software development quality. In Proceedings of the 2003 SIGMIS conference on Computer personnel research: Freedom in Philadelphia--leveraging differences and diversity in the IT workforce (SIGMIS CPR '03). ACM, New York, NY, USA, pp. 52-57.

Council Directive 1993/42/EEC. Medical Device Directive.

Council Directive 2007/47/EEC. Medical Device Directive.

D'Eramo, P. 2007. Japan's Pharmaceutical Affairs Law (PAL): Opportunities and Challenges. Referenced 25.7.2011 from http://www.ispe.org/cs/regulatory_review_archive/february_2007_japans_pharmaceutical_affairs_law_pal_opportunities_and_challenges.

Everett, R. R., Zraket, C. A., Benington, H. D. 1957. SAGE: a data-processing system for air defense. In Papers and discussions presented at the December 9-13, 1957, eastern joint computer conference: Computers with deadlines to meet (IRE-ACM-AIEE '57 (Eastern)). ACM, New York, NY, USA, pp. 148-155.

FDA. 2011. U.S. Food and Drug Administration. Referenced 25.7.2011 from http://www.fda.gov.

Flores, N., Aguiar, A. 2008. Patterns for understanding frameworks. In *Proceedings of the 15th Conference on Pattern Languages of Programs* (PLoP '08). ACM, New York, NY, USA, Article 8, p. 11.

Global Harmonization Task Force. 2011. Welcome to the Global Harmonization Task Force Website. Referenced 12.7.2011 from http://www.ghtf.org/.

Gross, A., Minot, J. 2007. Japanese Audits and Accreditation for Foreign Medical Device Manufacturers. Referenced 21.7.2011 from http://www.medicaldevices.org/sites/default/files/JapanAuditLayoutOct07_000.pdf.

Haigh, T. 2010. Dijkstra's Crisis: The End of Algol and Beginning of Software Engineering, 1968-72. Draft for discussion in SOFT-EU Project Meeting. Referenced 8.7. from http://www.tomandmaria.com/tom/Writing/DijkstrasCrisis_LeidenDRAFT.pdf.

Health Canada. 2011. Health Canada. Referenced 25.7.2011 from http://www.hc-sc.gc.ca/index-eng.php.

IEC 62304:2006 Medical device software – Software life cycle processes.

ISO 13485:2003 Medical devices – Quality management systems – Requirement for regulatory purposes.

ISO 14971:2007 Medical devices – Application of risk management to medical devices.

ISO Quality Services. Referenced 5.7.2011 from http://www.isoqsltd.com/html/iso_history.html.

JFMDA. 2011. The Japan Federation of Medical Devices Associations. Referenced 7.7.2011 from http://www.jfmda.gr.jp/e/.

King, J. L. 2010. Project SAGE, a half-century on. Interactions 17, 5, pp. 53-55.

Kopec, D., Tamang, S. 2007. Failures in Complex Systems: Case Studies, Causes, and Possible Remedies. Inroads – The SIGCSE Bulletin 39. 2. Pp. 180-184.

L 24.6.2010/629. Laki terveydenhuollon laitteista ja tarvikkeista.

Leveson, N., Turner, C. 1993. An Investigation of the Therac-25 Accidents. IEEE Computer. 26. 7. Pp. 18-41.

Lindholm, C., Höst, M. 2009. Risk Identification by Physicians and Developers - Differences Investigated in a Controlled Experiment. SEHC'09, May 18-19. Vancouver, Canada. Pp. 53-61.

Mahoney, M. 1988. The History of Computing in the History of Technology. Annals of the History of Computing 10, pp. 113-125.

Miura, S. 2007. Industry view point on software. 11th Conference of the Global Harmonization Task Force. Referenced 5.7.2011 from http://www.ghtf.org/meetings/conferences/11thconference/E/MIURA.pdf.

Naur, P., Randell, B.1969. Software Engineering – Report on a conference sponsored by the Nato Science Committee. Referenced 11.7.2011 from http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.

Oopsla. 2011. Oopsla history. Referenced from http://www.oopsla.org/oopsla-history/.

Princeton. 2008. Michael S. Mahoney, historian of science and devoted faculty member, dies. Referenced 11.7.2011 from http://www.princeton.edu/main/news/archive/S21/70/15G51/index.xml?section=topstories.

Pöyhönen, I., Hukki, K. 2004. Development of risk-informed requirements specification process of software.Espoo. VTT. Research Notes 2263. p. 47.

Raccoon, L. B. S. 1997. Fifty years of progress in software engineering. Software engineering notes 22, 1, pp. 88-104.

Rosenbaum, S., Hinderer, D., Scarborough, P. 1999. How usability Engineering Can Improve Clinical Information System. Reprint of paper delivered at UPA 99, sponsored by the Usability Professional's Association. p. 5. Referenced 18.7.2011 from http://www.teced.com/PDFs/upa99sr.pdfo.

Royce, W. 1970. Managing the development of large software systems. Proceedings, IEEE WESCON, August 1970, pp. 1-9.

Ryder, B., Soffa, M., Burnett, M. 2005. The Impact of Software Engineering Research on Modern Programming Languages. ACM Transactions on Software Engineering and Methodology 14. 4. pp 431–477.

Sabre Holdings. 2011. Sabre history. Referenced 7.7.2011 from http://www.sabre-holdings.com/aboutUs/history.html.

Sarnikar, S., Murphy, M. 2009. A Usability Analysis Framework for Healthcare Information Technology. Sprouts: Working Papers on Information Systems. 9. 62. Referenced 18.7.2011 from http://sprouts.aisnet.org/9-62.

Seddon, J. 2000. The quality you can't feel. The Observer. Sunday 19 November. Referenced 5.7.2011 from http://www.guardian.co.uk/money/2000/nov/19/workandcareers.madeleinebunting.

SFDA. 2011. State Food and Drug Administration. Referenced 12.7.2011 from http://eng.sfda.gov.cn.

Shenvi, A. 2010. Medical Software: A Regulatory Process framework. ISEC'10, February. Mysore, India. pp. 119-124.

Slaughter, S. A., Harter, D. E., Krishnan, M. S. 1998. Evaluating the cost of software quality. Communication of the ACM 41, 8. pp. 67-73.

Sommeville, I. 2001. Software engineering. Pearson education, 6 ed., p. 693.

Technical report. 2009. IEC/TR 80002-1 Medical device software – Part 1: Guidance on the application of ISO 14971 to medical device software. IEC, Geneva, Switzerland.

Tripp, L. L.1996. International standards on system and software integrity. StandardView 4, 3, pp. 146-150.

Wallace, D., Kuhn, R. 2001. Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data, International Journal of Reliability, Quality, and Safety Engineering. 8. 4. pp. 351-171.

Valvira. 2011. Valvira – National Supervisory Authority for Welfare and Health. Referenced 26.7.2011 from http://www.valvira.fi.

World Health Organization. 2003. Medical Device Regulation – Global overview and guiding principles. ISBN 9241546182. Referenced from http://www.who.int/medical_devices/publications/en/MD_Regulations.pdf

Yang, F., Mei, H. 2006. Development of software engineering: co-operative efforts from academia, government and industry. In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, pp. 2-11.

# APPENDICES

**APPENDIX 1:** ISO 13485 – Quality Management Systems

**APPENDIX 2:** IEC 62303 – Software Life Cycle Processes

**APPENDIX 3:** ISO 14971 – Application of Risk Management to Medical Devices

**APPENDIX 4:** Summary of fault classes, generic problems, prevention and detection methods (Wallace and Kuhn 2001).

## APPENDIX 1

### Chapter 4: Quality Management System

In this chapter general requirements and documentation requirements are presented.

### Chapter 5: Management Responsibility

Management responsibility includes:

1. Management commitment
2. Customer focus
3. Quality policy
4. Planning
5. Responsibility, authority and communication
6. Management review

### Chapter 6: Resource Management

Resource management includes:

1. Provision of resources
2. Human resources
3. Infrastructure
4. Work environment

### Chapter 7: Product Realization

Product realization includes:

1. Planning of product realization
2. Customer-related processes
3. Design and development
4. Purchasing
5. Production and service provision
6. Control of monitoring and measuring devices

### Chapter 8: Measurement, analysis and improvement

Measurement, analysis and improvement include:

1. General
2. Monitoring and measurement
3. Control of nonconforming product
4. Analysis of data
5. Improvement

## APPENDIX 2

### Chapter 4: General Requirements

IEC 62304 presents that manufacturer shall apply a risk management process complying with ISO 14971 and shall demonstrate the ability to provide medical device software that consistently meets customer requirement and applicable regulatory requirements. In practice requirements mean that both standards ISO 14971 and ISO 13485 shall be applied.

The standard also defines a software safety class. There are three classes: A, B and C. Class A means that no injury or damage to health is possible and class C means that death or serious injury is possible. Safety class instructs what requirements must be performed in life cycle processes.

### Chapter 5: Software Development Process

There are several activities to do:

1. Software development planning
2. Software requirement analysis
3. Software architectural design
4. Software detailed design
5. Software unit implementation and verification
6. Software integration testing
7. Software system testing
8. Software release

### Chapter 6: Software Maintenance Process

Software maintenance process consists of several activities:

1. Establish software maintenance plan
2. Problem and modification analysis
3. Modification implementation

Modification implementation includes practically software development process activities 2-8. According to IEC 62304 forewords activity 2 is not belonging to modification implementation. However, only in the case of implementation error there is no need for activity 2 of software development process. In other cases the specification is changed somehow.

### Chapter 7: Software Risk Management Process

Software risk management process consists of several activities:

1. Analysis of software contributing to hazardous situations
2. Risk control measures
3. Verification of risk control measures
4. Risk management of software changes

**Chapter 8: Software Configuration Management Process**

Software configuration management process consists of several activities:

1. Configuration identification
2. Change control
3. Configuration status accounting

**Chapter 9: Software Problem Resolution Process**

Software problem resolution process consists of several activities:

1. Prepare problem reports
2. Investigate the problem
3. Advise relevant parties
4. Use change control process
5. Maintain records
6. Analyze problems for trends
7. Verify software problem resolution
8. Test documentation contents

## APPENDIX 3

### Chapter 3: General Requirements for Risk Management System

General requirements consist of several components:

1. Risk management process
2. Management responsibilities
3. Qualification of personnel
4. Risk management plan
5. Risk management file

Risk management plan explains how manufacturer arranges risk management system.

Risk management file is the essential part of risk management system. Risk management file contains all the records produced by activities of the risk management process.

### Chapter 4: Risk Analysis

Risk analysis consists of several components:

1. Identification of intended use and characteristics of medical device
2. Foreseeable misuse
3. Identification of hazards
4. Risk estimation

### Chapter 5: Risk Evaluation

Risk evaluation is decision whether risk reduction is needed or not. The criteria defined in risk management plan shall be used.

### Chapter 6: Risk control

Purpose of risk controlling is to choose risk control measurements that reduce the risk to be acceptable. Risk controlling consists of several components:

1. Risk control option analysis
2. Implementation of the risk control measure
3. Residual risk evaluation
4. Risk and benefit analysis
5. Risks arising from risk control measures
6. Completeness of risk control

### Chapter 7: Evaluation of Overall Residual Risk Acceptability

The overall residual risk has to be evaluated in order to ensure that medical device is safe to use. The overall residual risk should be acceptable using the criteria defined in risk management plan.

If the overall residual risk is not acceptable using the criteria, manufacturer should view the overall residual risk from broader perspective. If the research conducts the conclusion that overall residual risk is acceptable and medical device is safe to use, manufacturer may disclose the overall residual risk.

**Chapter 8: Risk Management Report**

Before releasing the medical device for commercial use, manufacturer shall carry out a report of the risk management process. That report should include at least the information that risk management plan is appropriately implemented, the overall residual risk is acceptable and appropriate methods are in place to obtain relevant production and post-production information.

**Chapter 9: Production and post-production information**

There shall be established, documented and maintained a system for the production and post-production phases of medical device. The system shall collect and review information about medical device or similar devices.

## APPENDIX 4

| Fault class | Generic problem | Prevention | Detection |
|---|---|---|---|
| Calculation | • Constants incorrectly coded<br>• Precision problem | • Code review<br>• Code reading<br>• Low level design review | • Code reading<br>• Inspection<br>• Unit test |
| Change impact | • Logic<br>• No verification against original design specification | • Traceability analysis<br>• Change impact analysis | • Inspection<br>• Regression test<br>• Traceability analysis |
| Configuration management | • Incorrect configuration for non-domestic systems<br>• Software incompatible with other components | • Use of CM tools<br>• Traceability analysis | • Verify usage of Cm tools for all changes<br>• Inspection of requirements for component interfaces<br>• Verification of changes<br>• Regression test |
| Data | • System failed due to invalid input data<br>• Inconsistency of data retrieved from database and that expected by the program<br>• Database corruption | • Assertion for invalid values<br>• Checks for ranges that imply incorrect data<br>• Design: set criteria of input data validation<br>• Code: implementation of input data validation<br>• Assertions on validity of da- | • Review for completeness of data specification<br>• Review that all data specifications are included in the user instructions<br>• Inspection<br>• Test against invalid data<br>• Testing focused on data retrieval |

| | | | |
|---|---|---|---|
| | | ta retrieved from database<br>• Database administration | • Error handing routine in software |
| Fault tolerance | • Excessive use of the program causes failure<br>• Incorrect action due to unexpected condition<br>• Incorrect action due to operator error | • Fault tolerance such as warnings to operators<br>• Fault tolerance to protect against human error | • Stress/volume test<br>• Test against boundary and abnormal conditions |
| Initialization | • Lack of initialization of the runtime environment<br>• Executing the software first time it fails to store necessary initialization values | • Use assertions for initialization<br>• Document initial conditions for both initial run and consecutive run | • Inspections<br>• Code review<br>• Test against initial conditions<br>• Stress test |
| Interface | • Software does not properly interface with external device or other software component | • Trace requirements through design through code<br>• Examine the specification for each interface | • Inspections<br>• Reviews<br>• Integration test |
| Logic | • Incomplete or incorrect control logic<br>• Improper handling of boundary conditions<br>• Improper data validation<br>• Programming error | • Design review<br>• Walk through the software implementation against design<br>• Verify logic for all conditions<br>• Code review | • Code review<br>• Inspection<br>• Testing |

| | | | |
|---|---|---|---|
| Omission | • Vital system function are missing<br>• Lack of documentation<br>• Improper documentation | • Trace requirements through design through code<br>• Trace into user and test documentation<br>• Proper release procedure<br>• Traceability | • Inspections, reviews examining traceability of functions<br>• System test<br>• Verify completeness by examining trace<br>• Inspection |
| Other | • Out of compliance with the performance standard<br>• A typographic error in software algorithm causes incompatibility between two devices | • Simulation<br>• Design review<br>• Code review<br>• Code reading against algorithm specifications | • Performance test<br>• Unit testing<br>• Walkthrough |
| Quality assurance | • Test plan was not implemented or executed appropriately<br>• Regression test was not performed on modified software<br>• No validation before initial release<br>• No validation on software changes | • Project management oversight<br>• Change impact analysis<br>• Specified procedures regarding testing before product release | • Project status review<br>• QA process checks |
| Requirements | • Exceptional conditions were not specified in the requirement specification<br>• Functions missing in the requirement specification | • Modeling<br>• Analysis<br>• Traceability<br>• Requirement review<br>• Design review | • Interface analysis<br>• Requirement review<br>• Design review<br>• System test |
| Timing | • Real time clock | • Simulation | • Timing analy- |

| | | | |
|---|---|---|---|
| | was not accurate<br>• Scheduled event did not occur due to timer failure | • Design review<br>• Code review<br>• Fault tolerance | sis<br>• Integration test<br>• System test |