Självständigt arbete på grundnivå

Independent degree project - first cycle

Datavetenskap
Computer Science

A new approach for medical device product documentation
Usage of software version control system Git as a combined software and hardware tool in order to achieve compliance with medical regulatory demands

Elisabet Hamnede
A new approach for medical device product documentation
Elisabet Hamnede

MID SWEDEN UNIVERSITY
Department of Information Systems and -Technologies

Examiner: Leif Olsson, leif.olsson@miun.se
Supervisor: Håkan Sundberg, hakan.sundberg@miun.se
Supervisor NEP: Anders Söderbärg, anders.soderbarg@nepartner.se & Fredrik Karlsson, fredrik.karlsson@nepartner.se
Author: Elisabet Hamnede, elha0804@student.miun.se
Main field of study: Computer Science
Semester, year: Spring, 2017
Abstract

Hardware and software developers rely on different tools for document management, product data management (PDM) and software configuration management (SCM). As more and more products include components of both types there is a growing demand for one collaborative system. This becomes even more critical in the medical sector, where a device is under regulatory demands for document management to even be allowed in to the market. Combined systems become more complex and are generally based on PDM-principles rather than SCM. Current development of SCM tends instead towards simpler systems focused on pure version control (VCS) that are easy to use and economically available to small- and medium sized enterprises (SME), which is not the case with generic PDM-systems or combined systems.

This study explored the possibility to extend the usage of such a VCS and include hardware documentation as well as software. The aim was to further our understanding of the SME perspective on product documentation for the medical device field. The method was a case study, collaboration with a SME development company. The scope was to explore possible usage of a chosen VCS (GitLab) and to compare it with a generic PDM-system and with existing manual system.

The results showed that for several of the hardware document types there are special made Git-solutions to find within the open source community. However, none of the ones tested in the study was deemed good enough with respect to functionality and reliability. Instead the case study used direct storage of the files in their binary format and focused on testing different VCS functions and on how to organize in order to best gain the advantages of using the system.

The conclusions showed that hardware documents can be stored in the same iterative manner as software but with limited Git functionality. Compared with a PDM system GitLab can offer the same level of revision control and communication around the specifications but lacks classification of parts and detailed product structures. GitLab offers better iteration history than both a PDM system and the existing manual system does. But not being able to use full Git functionality the organization needs a collaboration strategy to handle the decentralized storage. If the collaboration strategy matches the organization development practices, GitLab is a useful alternative for medical device documentation.

Keywords: Medical device product documentation, PDM, CSM, VCS, SME, Git, GitLab
Acknowledgements

I’d like to dedicate thanks to NEP for letting me work with their idea for this case study. Thanks for the open minded cooperation, for the technical knowledge and for the ever present drive for improvement.

I’d also like to thank my family for all their patience and support throughout this period and my supervisor at Mid Sweden University.
Table of contents

Abstract............................................................................................................. iii
Acknowledgements............................................................................................ iv
Table of contents............................................................................................... v
Terminology........................................................................................................ vii

1 Introduction...................................................................................................... 1
  1.1 Background............................................................................................... 1
  1.2 Problem discussion .................................................................................. 3
  1.3 Objectives................................................................................................ 4
  1.4 Demarcations............................................................................................ 4
  1.5 Outline...................................................................................................... 4
  1.6 Contributions............................................................................................ 5

2 Theory.............................................................................................................. 6
  2.1 Related work............................................................................................ 6
  2.2 PDM......................................................................................................... 6
    2.2.1 PDM – state of the art ....................................................................... 6
    2.2.2 PDM concepts................................................................................... 8
  2.3 SCM......................................................................................................... 9
    2.3.1 SCM – state of the art ....................................................................... 9
    2.3.2 SCM and Git concepts...................................................................... 10
  2.4 PDM – SCM integration........................................................................... 12
    2.4.1 System comparison.......................................................................... 12
    2.4.2 Approaches for integration............................................................ 13
  2.5 Medical device regulations....................................................................... 14
    2.5.1 Swedish Medical Products Agency............................................... 14
    2.5.2 EN ISO 13485:2016.......................................................................... 14
  2.6 Case study Company............................................................................... 15
    2.6.1 NEP.................................................................................................. 15

3 Methodology................................................................................................... 16
  3.1 Description............................................................................................... 16
  3.2 Case study............................................................................................... 17
  3.3 Comparison............................................................................................. 17
  3.4 Participant observation.......................................................................... 17
  3.5 Method criticism..................................................................................... 18
    3.5.1 Reliability........................................................................................ 18
    3.5.2 Validity............................................................................................ 19

4 Results.......................................................................................................... 20
  4.1 Existing manual system........................................................................... 20
  4.2 GitLab usage........................................................................................... 21
    4.2.1 Evaluating special made solutions................................................. 21
    4.2.2 Case study....................................................................................... 23
  4.3 Comparison summary............................................................................. 27
A new approach for medical device product documentation
Elisabet Hamnede

5 Discussion ........................................................................................................ 28
6 Conclusions ..................................................................................................... 31
7 Social and ethical considerations ................................................................. 32
8 Future research ................................................................................................ 33
9 References ....................................................................................................... 34
## Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product Data Management (PDM)</strong></td>
<td>Systems for managing the creation, change and archive of all information related to a hardware product.</td>
</tr>
<tr>
<td><strong>Software Configuration Management (SCM)</strong></td>
<td>Systems for managing the different items and version of a software product.</td>
</tr>
<tr>
<td><strong>Version Control System (VCS)</strong></td>
<td>System for managing the version of a software product, can be part of a SCM or stand alone.</td>
</tr>
<tr>
<td><strong>Git</strong></td>
<td>Stand alone VCS using decentralized principles available on all mainstream development platforms.</td>
</tr>
<tr>
<td><strong>Small- and Medium sized Enterprises (SME)</strong></td>
<td>Defined by the European Commission as companies with less than 250 employees.</td>
</tr>
<tr>
<td><strong>Medical device</strong></td>
<td>Any product intended for use in all areas of health care.</td>
</tr>
<tr>
<td><strong>Medical regulations</strong></td>
<td>National or international regulations of products that are to be marketed as medical devices.</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Hardware development and software development rely on different types of systems for document management.

For hardware development the tool is Product Data management systems (PDM). PDM-systems focus is on managing the creation, change and archive of all information related to a product. It tracks and controls data related to a particular product and serves as a central repository for product history. The data usually involves specifications of the product, material specification, parts lists, drawings and specifications for manufacture. PDM stems from traditional engineering design activities that created product drawings and schematics on paper and using CAD tools to create parts lists (Miller, 1998). However, PDM-systems can be costly to install and complicated to implement, drawing on both economical and human resources to maintain. This aspect is particularly important to small and medium sized companies (SMEs) that have limited resources and potentially higher vulnerability to implementation failures. Instead of investing in PDM-systems many SMEs rely on manual effort to achieve control of documentation and revisions (Waldenmeyer & Hartman, 2013).

For software development the tool is Software Configuration Management systems (SCM). SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it. If a configuration is working well, SCM can determine how to replicate it across many hosts. SCM came in use in the 1950’s when hardware configuration management was applied to software as well (Pressman, 2009). Since then the systems have evolved greatly and must now be considered a discipline on its own, separated from the hardware practices it once evolved from (Do & Chae, 2011). In the most recent development we see standalone software focused on version control, often referred to as Version Control Systems (VCS). VCS support collaboration between several developers in the same file, the possibility to revert changes and to compare different versions of software with one another (Candrlic et al., 2007). One major aspect of VCS is the strong ties to the open source community where several of the modern but still mature systems are developed (Spinellis, 2005; Spinellis, 2012).

The number of products with embedded software increases across all application areas continuously and, with that, the complexity between the hardware and software steadily increases (Hopf & Ovtcharova, 2013; Shenvi, 2010). This leads to a demand for control of the total product, including the software components (Crnkovic et al., 2001). Even more so when the product is a medical device and the supplier has to comply with region specific regulatory demands in order to even place the product on that market (Lepmets et al., 2014). In Europe and USA suppliers have to keep a complete technical file over their medical device. The file shall include not only all product documentation but
also the result of risk analysis and motivated descriptions of the design solutions selected to comply with the regulatory demands in question (www.lakemedelsverket.se). Most of the information which must go into the technical file is readily, if not only, available in the design stage. Ensuring compliance with regulations in retrospect has been shown to be a time-consuming and costly exercise. The designer of a medical device should document essential aspects of the design as early as possible in the development stage (Romer & Stuyt, 2007).

As a consequence more and more companies look for product management systems that can cover all aspects of the complete product, both software and hardware. In the integration of these processes, many difficulties are encountered because of the different natures of the processes and the different approaches made to the problem. SCM and PDM are used to solve similar problems in different ways and are overlapping more than complementing one another (Crnkovic et al., 2001).

The common approach for creating one complete system has been to, in different ways, incorporate SCM in to existing PDM-systems. However there are downsides to this approach.

The first is that software development is evolving quickly and new solutions constantly emerge to enhance the usefulness of software development tools, especially in the open source community. Big, complicated systems, such as PDM-systems, are to slow to develop to be able to keep up with new best practices that are based on these solutions (Suciu et al., 2012).

The second downside is that if PDM-systems generally are inaccessible to SME due to high cost for implementation and maintenance (Waldenmeyer & Hartman, 2013). This will apply even more so to systems that incorporates even more functions, thus making them even more complicated.
1.2 Problem discussion

As the number of medical devices with embedded software increase there is an ever growing need to integrate SCM and PDM to support multidisciplinary development environments. But the attempts of incorporation SCM in to PDM have not been successful enough to convince the market that this is the best practice for complete product documentation. Even if that approach had yielded better results the cost and implementation difficulties of an advanced PDM-system still makes the solution unavailable to a large part of companies in the development sector.

In this study the problem of achieving one complete system for product documentation will be looked at from the opposite angle. The idea is that instead of incorporation SCM into PDM it could be possible to incorporate PDM into SCM. This is a under researched approach and no literature on targeting exactly this approach has been found. Therefore this study will have to be of an exploratory nature, selecting a case study for practical experiment and report on the results, drawing conclusions on what can be possible rather than what is proven, hopefully opening the niche for further study.

In order to do this, Git has been chosen for evaluation. Git is one of the most widely used modern VCM on the marked today. It is a distributed system available on all mainstream development platforms through a free software license (Spinellis, 2012). This makes it technologically and economically available to
all kinds of companies regardless of size. Git is by now so well established on the market that research projects have focused on the possibility of integrating it into the teaching of computer science (Binas, 2013 ; Brinnen & Nord, 2013 ; Haaranen & Lehtinen, 2015). Even other applications other than pure source code, such as medical records, have been attempted (Neumann et al, 2012). All of the above factors makes Git a suitable system for this study.

The Git repository can be accessed remotely by a number of different services such as GitHub, GitLab and Bitbucket. Due to personal experience in the company chosen for case study, GitLab will be the access service of choice for this study.

1.3 Objectives

The aim for this study is to further our understanding of the SME perspective on product documentation for the medical device field. This will we accomplished by exploring to what extent a software version control system such as GitLab can be used to integrate all types of product specifications. The purpose of the integration is to make a complete technical documentation that would meet the demands of regulatory standards. This will be done by looking in to the following research questions:

- How can hardware specifications such as drawings, cad-files and build instructions be stored and controlled in GitLab?
- How does revision handling of such documents in GitLab compare to that of a standard PDM-system?
- How does revision handling of such documents in GitLab compare to that of a company that’s has not implemented a PDM-system?
- To what extent can the documentation and revision control possible in GitLab meet the demands of regulatory standards for medical devices?

1.4 Demarcations

This study is limited to the evaluation of GitLab as a tool for storage and control of a limited number of hardware specifications types (defined in detail in the Theory chapter) during the development phase of the product.

1.5 Outline

Chapter 2 explains the theoretical framework for the study in greater detail. Chapter 3 presents the chosen methodology, the motivations for it and the limitations. Chapter 4 reports on the result of the study. This chapter is divided in to three sections, the first documents the existing manual system, the second re-
ports on the case study of GitLab usage and the third compares the three system
types present in the study. Chapter 5 discusses the result of the study and com-
pares it with theoretical framework described in chapter 2. Chapter 6 states the
conclusions made from the study. Chapter 7 discusses social and ethical consid-
erations and chapter 8 makes suggestions for future research.

1.6 Contributions
This study was conducted by the author in cooperation with hardware and soft-
ware developers at NEP. The aim for the research originates from the company
but the research questions have been prepared and formulated by the author as
well as the theoretical material collection. The case study has been planned and
conducted as cooperation by the company and the author, the results have been
documented and analysed by the author.

No part of this study has been previously published.
2 Theory

This chapter will lay out the theoretical framework for the study. It will summarize the background on PDM, SCM and approaches for combining the two system types. It will also summarize medical device regulations and introduce the case study company in question.

2.1 Related work

The study of similarities and differences between PDM-systems and SCM-systems is not a new one. Estublier et al. (1988) and Westfechtel & Conradi (1998) were early in the field, both mainly focusing on comparison and reporting on the differences found in regards to how the two system types use for example product models, process models and version operators.

After that several studies were made to find a technical approach to combine the two systems. Crnkovic et al. (2001) provided frameworks for SCM and PDM integration and considered integrated product data models to be a critical factor for the success. Dahlqvist et al. (2003) proposed an integrated product structure model which includes both hardware and software products. Krebs et al. (2005) provided an integrated product configuration model. It enables customers to specify their own product configurations of both software and hardware products. El-Khoury (2005) suggests a model based approach that also combines the two systems with Product Lifecycle Management. Do and Chae (2011) proposes a PDM-architecture that supports the extension of the data model version items for SCM systems. Vendors of commercial systems also try to find solutions to the problem, for example Siemens PLM which can interface corresponding version items in IBM ClearBase database in order to manage complete product documentation (Do & Chae, 2011).

2.2 PDM

This section will introduce PDM-systems and PDM concepts in further detail.

2.2.1 PDM – state of the art

PDM is used to manage data for hardware development and contains primarily two types of data; specification data and meta data. Specification data describes the characteristics and properties of a product or part. Meta data describes the characteristics of other data; information types, predefined values in data sets or organizational responsibilities for data items. PDM systems keep track of the masses of data required to design and manufacture hardware products (Miller, 1998). The systems originate in manual handling of documents and product
structures and were first developed to automate those manual processes, both for efficiency and for improved security, but the possibilities of computerization led to process management of different sorts being incorporated into the systems (Armstrong, 2001). While this can be a great advantage to an organization, helping management to organize and control the daily work, it can also be a drawback and a hindrance if the processes build in to the system don’t fit with the organization in question (Otto, 2012). This is especially problematic for smaller organizations that may have neither the economical nor the personnel resources to implement a complex system and reorganize in order to create a perfect fit (Waldenmayer & Hartman, 2013).

Traditional PDM systems have been developed and implemented for one organization at a time but products are seldom handled by one organization only. There is often a need to share product data with manufacturers, consulting designers and other types of partners. For this reason, and the rapid development of Internet, there has been a trend towards network connection and web-based interfaces (Kim et al., 2006). The information security issue this raises is the same as for other information systems and does not seem to have generated any amount of research connected specifically to security of PDM-information.

Instead it has raised questions on standardization of data modelling, one developed and maintained by the ISO technical committee TC 184 is the ISO 10303 Standard for Exchange of Product model data (STEP). STEP PDM Schema handles typical product-related information such as geometry, drawings, project plans, part files, assembly diagrams, bills of material and many more (Panetto, 2011).

Hand in hand with the development of different standardizations of data modelling there has been a development of having other information systems use information stored in PDM system. This is both enabled by the standardization of data modelling and also puts pressure on development to continue and to improve (Panetto, 2011). The most common system type to have usage for PDM information is Enterprice Resource Planning systems (ERP), where it is used to plan material and production and manage different levels of sourcing (ibid).

Tightly connected to PDM-systems, and usage of PDM information by other system types, is the development of Product Life-cycle Management systems (PLM). PLM stands for integrated management of product-related information through the entire lifecycle and PDM information is one of its key components. Other components are ERP, Customer Relations Management and computer aided design systems (El Kadiri & Kiritsis, 2015). Whether Software and SCM is a part of PLM seem to vary, some researchers identifies it as a component (El-Khoury, 2005) while others leave it out (El Kadiri & Kiritsis, 2015), could be because it is not considered necessary at all or it is seen as a part of the PDM component.
2.2.2 PDM concepts

Data vault and document management

The backbone of any PDM-system is the records database where all specifications are stored, automatically named/numbered and individually controlled. Controlling document status ensures that the right version of a document is distributed to the users and that older versions of documents are stored in their original form, giving the user a full revision history. The centralized storage enables users to access documents without knowing where or how they are stored and file back-up in a controlled manner. The meta data that describes the characteristics of the documents is also managed by the vault (Armstrong, 2001).

Access and workflow management

Basing system access on individual log-in a PDM-system can control which users have access to what documents and in what fashion (read, write or both). Changes made in the system can be logged as to when and which user. In some systems tasks and ownership of different records can be delegated to individual users. Then communication between user and system as well between different users can be handled by mark ups or notifications enabling management of the design or change workflow (Armstrong, 2001).

Product structure / configuration management

A PDM-system can handle multilayer relationship between assemblies, parts and associated data, giving the user a models or bill of material for different levels of assembly of the product. This can in some systems be used for defining product baselines, variants and options (Armstrong, 2001).

Classification / search ability

Classification of parts and documents enables good search ability and the more data that is stored in the system the more important this aspect becomes, especially for enabling reuse of components or methods (Armstrong, 2001).

Figure 4: PDM components
2.3 SCM

This section will introduce SCM-systems and SCM concepts in further detail.

2.3.1 SCM – state of the art

SCM is used to control the evolution of complex software, it identifies the product component and their related documents, it keeps the product structure and control continuous changes, status and auditing. Traditionally it has mainly contained source code and not the binary files that are compiled from it since they have been so easy to reproduce. But as the development process has become more complex so this becomes less certain. The scope of the development process covered by SCM also increased so that it not only supported the implementation but also earlier phases (requirements and design) and later phases (maintenance and support) and also documentation. From this perspective the more complex SCM became less software-specific and more close to a general configuration and document management support (Crnkovic et al., 2001).

Parallel to, or perhaps in response to, the development of more complex and less software specific systems the open source community was more focused on collaborative implementation and from this several reliable systems for pure Version Control (VCS) have evolved. Spinnelis (2005) writes that adopting a VCS need not be expensive but could prove the most important tooling improvement an organization could undertake. Otte (2009) concludes that there is no doubt that VCSs simplify software development.

One of the bigger disputes when it comes to VCS is the centralized versus decentralized approach and a lot of research has gone in to looking at this from different aspects. The centralized approach is often represented by Subversion that was initiated 2000 but builds upon Concurrent that was initiated 1984. The decentralized by Git that was initiated 2005. They are now the most widely used systems within their approaches (Otte, 2009; Spandel & Kjellgren, 2014). Since the centralized systems were first out and rather well adopted by the community when the decentralized approach was presented this was for a long time in their favor (Otte, 2009). But the decentralized approach has grown fast and the community collaboration it offers makes it a very attractive alternative (Spinellis, 2012). Spandel & Kjellgren (2014) concluded that both systems approaches are easy to start using but that the decentralized approach, represented by Git, encourages developers to work more similar to what is describes as best practice.

Coming from the open source community Git is easily available for continued development. One example of this is a version control system for medical records that need be tracked both on a unit level and on a collection level that is built on Git foundation (Neumann et al. 2012).
2.3.2 SCM and Git concepts

Repository
The repository can be compared to a folder containing all the files of a project. Since Git is a decentralized VCS there can be several repositories set up to share files between them. The common practice is to have a server with a remote repository as well as a local repository for every user working on the project. The remote repository is the server located repository that is shared between all authorized users. The remote repository is created by one user and files are added to it when pushed from local repositories. The local repository is the client located repository that is worked on by a single user. The local repository can either be created from scratch at local level (when starting a new project) or cloned from the remote repository (when continuing on an existing project). The work is performed at local level and new files and changed files are then pushed to the remote repository when ready to be shared with other users.

![Figure 5: Decentralized file sharing](image)

Commits
A commit can be compared to a snapshot of the current state of the project in that repository. The user chooses when to save these snapshots and can attach messages or tags to them. A commit message is generally used to give a short summary of the changes since last commit. A commit tag can be added to a specific commit marking that commit as being especially important. Typically this is used to mark release points.

Branching and merging
A branch is track along which development continues. The first branch is called “master” and branching out from master allows development to be performed in simultaneous side tracks. The main benefit of branching is that development targeting a special feature or bug fix to be collected in the same branch and handled separately until it is ready to be part of main development. A branch other than master contains code that has been diverged from the main development track.

Merging is the action where a branch is reconciled with another; often with the master when the feature or bug the branch was targeting is developed to a degree where it is ready to be part of main development. Merging is initiated manually but the merging of the files, i.e. what parts of what files should be used to constitute the new files, is solved automatically as long as no new addi-
tions collide with one another. A merge conflict can occur if the same area of the same file has been edited in both branches that are to be merged and this has to be solved manually.

![Figure 6: Branching and merging](image)

**Git commands**

Add - A command to add files that are kept in the same folder as the local repository to that repository.

Status - A command to show the user the status of the files in the repository and the folder where the repository is placed compared with latest commit in that branch. Will only show which files that have changed and not how.

Diff - A command to show in detail how files have changed between the current state of the repository and one specific commit or between two different commits (Chacon & Straub, 2014).

**File types**

A text file is a computer file that only contains text and has no special formatting or images. The ASCII character set is the most common format for English-language text files. A binary file is a computer file that is not a text file. Binary file formats contain formatting information in binary form that is specific for the file type in question (http://opentechschool.github.io/python-data-intro/core/text-files.html).

Git can store any kind of file but is developed for tracking text files. This means that commands such as merge and diff can’t be guaranteed to work to its full extent with binary files. An attempt to perform git diff command on a file that git isn’t able to analyze renders the result “Binary file” instead of a list of differences (Chacon & Straub, 2014).
2.4 **PDM – SCM integration**

This section will introduce the different approaches for PDM and SCM integration in further detail.

2.4.1 System comparison

The biggest similarity between the two systems types is that both specify product definitions with their documents. The physical properties of products, or the source codes for programs, are examples of these specified product definitions. The two system types differ when it comes to meta data management for hardware parts compared to software items. Also, only SCM supports source code comparison between different versions of items and branch/merge operations for managing version items.

![Figure 7: Differences in system structure](image)

PDM manages detailed meta data with the product objects and attaches document as associated objects. SCM on the other hand directly manages source code in document objects without the objects for meta data used by PDM. These differences in use of meta data makes an integration between the system types difficult. The amount of meta data for version items, in SCM, is not sufficient enough to manage the development of complex, software intensive hardware products, which is the main functionality of a PDM system (Do & Chae, 2011).

SCM supports branch/merge operations on its source code but few PDM systems support similar operations on their documents. Source code comparison is another functionality crucial to SCM but generally lacking in PDM.

Both PDM and SCM represent their product structures with relationships among different hardware products or software items. Also, they both specify
product configurations. The models for making product structures are however not identical between the two system types. PDM typically handles product configuration as combinations of predefined product modules, usually modules that are standardized across an organization and sometimes shared with other departments. As such, these modules perform as standard information units for communication in the organization. SCM on the other hand has not needed such an advances approach in order to build product configuration, it simply selects version items directly, without the configuration modules. The different levels of moduling makes it difficult to integrate the two system types (Do & Chae, 2011).

PDM generally relies on more expressive product data models than SCM but instead it lacks functionalities crucial to software development.

2.4.2 Approaches for integration

PDM and SCM are both complex systems based on several crucial components. The different approaches for integration have focused on different component as the main focus for this integration.

Crnkevic et al. (2001) consider integrated product data models to be a critical factor for the successful PDM and SCM integration. Their example introduced the representation of meta data for software version items to enable the integration.

Dahlqvist et al. (2001) proposed an integrated product structure model which includes hardware and software products. However, product configurations and engineering changes are not taken into account. Both need consistent product data across the hardware and software products.

Krebs et al. (2005) provides an integrated product configuration model that includes both hardware and software components. Their proposed rule based product configuration model enables customers to specify their own product configurations of both hardware and software products. However their research focuses primarily on the product configuration management which is only one part of the PDM and SCM integration. It does not deal with representation of meta data, version operations for source codes, sharing product data model, product configuration management and engineering change management (Do & Chae, 2011).

El-Khoury (2005) focuses on the model-based approach that allows for integration of various engineering tools for hardware and software development. It also introduces a commercial PLM system in order to integrate PDM and SCM. However, they postponed the implementation of SCM operations for version items, in the integrated environment, to the future work, a decision that until solved renders the approach useless to software development.

Commercial PLM systems provide interfaces to commercial SCM systems in order to achieve the integration. They provide special types of data objects that can interface to version items in the target SCM. However, their approach
builds on management of independent PDM and SCM databases with heterogeneous applications which are interfaced to one another. These interfaces can cause data mismatches in the two independent databases, thus providing hardware and software engineers with heterogeneous user environments (Do & Chae, 2011).

Do & Chae (2011) proposes an architecture that can manage integrated hardware and software products for product configurations and engineering changes. Since both product configurations and engineering changes are in need of consistent product data across the hardware and software products, it extends the existing product data model for PDM in order to integrate SCM operations such as version control and branch/merge.

Focusing on all existing aspects of both PDM and SCM systems renders even more complex product architecture, and thus even more complex product, than either of the two systems alone. Focusing on one key aspect for integration can create a system that is less complex to learn and to work with but instead lacks functionality. However, complete functionality does not have to be a key issue for all organizations (Waldenmeyer & Hartman, 2013). Focusing on one key aspect has however favoured PDM aspects such as complex meta data over SCM aspects such as source code comparison.

### 2.5 Medical device regulations

This section will give a summary of the medical device regulations that have been taken into account for this study.

#### 2.5.1 Swedish Medical Products Agency

MPA is an agency under the Swedish Ministry of Social Affairs with the mission to promote the Swedish public health. MPA is responsible for supervision of manufacturers and devices in the medical field. The term medical device refers to any product intended for use in all areas of health care, ranging from patches and syringes to pumps for drug delivery, tools for expiratory supervision and pacemakers.

The Swedish regulations for medical devises are based on EU-directives and the manufacturer shall in their technical documentation be able to report the result of risk analysis and motivated design solutions chosen in order to meet the regulations (www.lakemedelsverket.se/)

#### 2.5.2 EN ISO 13485:2016

ISO 13485 is an international standard aiming to facilitate compliance with medical regulatory demands by specifying requirements for quality management system that can be used by an organization involved in one or more stages
of the life-cycle of a medical device. It is an independent standard but based on ISO 9001:2015.

Requirements for document control include that documents, both new and changed ones, are to be reviewed and approved before publication, that status and revision of a document can be identified, that relevant version of a document is accessible for usage and that both usage of outdated versions and loss of documents is prevented.

Requirements for record control include identification, safe storage, archiving, protection of confidential health information, possible identification of changes to a record and that product specific records shall be kept for the duration of the life time of the medical device as defined by the organization, minimum of two years after product release (SS-EN ISO, 2016).

2.6 Case study Company
This section will introduce the company where the case study for this research has been conducted.

2.6.1 NEP
The chosen case study is NEP, an electronics development company with less than 20 employees with an aim for robust design solutions and tight cooperation for their customer. The business strategy is attractive to product owner companies both within the SME sector and the medical device sector making high quality but inexpensive documentation a key issue for NEP.

Except for source code the specifications made during design phase are of the following types:

- Electrical schema made in Cadence.
- PCB layout made in Cadence.
- 3D presentation of assembly made in SolidWorks.
- Instructions made in Word.
- Bill of material made in Excel.

All the specifications are binary files of different sorts. When released, those specifications that will be sent to customer or used in printed form within the company are printed to .pdf to enable all users to access them. This also is a binary file format.
3 Methodology

Methodology refers to the scientific approach towards the intended topic and how the subject is dealt with. The methodology affects and permeates the entire research and it is crucial that the chosen method is suitable for the research questions and material that’s available on the subject.

Quantitative methods are of a numerical sort who handles data that can be translated into exact numbers and analysed statistically. Qualitative methods are not, the data can’t be translated into exact numbers but has to be handled in a less structured manner. The analysis relies more on interpretation and description (Holme & Solvang, 1997). Therefore quantities methods are suited for studies where there is a large set of homogenous data to be analyzed while qualitative methods are better suited for areas where the phenomenon to be studied is still relatively unknown (Björkqvist, 2012). For this study, where there is no earlier material to build on, the data cannot be relied upon to be homogenous or large. This study will take a predominantly qualitative approach and the methods had to be chosen accordingly. Classification and quantification are primarily quantitative methods and would not be useful. Testing a hypothesis or building a model could have been useful had the research questions been focused on a predetermined way of using Git, but that was not the case here. Instead the questions are of a explorative and comparative nature and are best answered by a combination of description, case study and comparison.

Data collection technique refers to the way in which the researcher collects the material to describe and make conclusions from. There are several different techniques available. For this study the challenge was the lack of earlier material to work with. Literature review is a useful technique when the data can be collected from written sources, but that was not the case here. Interviews and polls are useful when the data can be collected by asking people with experience on the subject, but that was not the case either since no one with experience in this particular field could be found. In order to gather the data needed it first had to be generated during a staged experiment that could be observed and documented. Therefore the best technique to use is participant observation.

3.1 Description

Description is a fairly basic method where the researcher accounts for details and connections in the area to be described. Description is itself empirical but the investigation may be initiated for evaluating reasons (Ejvegård, 2003).

In this study description will be used to account for revision control of hardware documents in manual systems in order to be able to compare these with the experiments conducted with VCS. The method will not be used to answer any of the research questions on its own but will together with the case study provide material for the comparison method.
3.2 Case study

The case study is a commonly used method in several scientific disciplines. Its purpose is to take a small part of a big process and by the study of the smaller part gain a better understanding of the process as a whole. Since only a smaller part of the process has been studied the conclusions can’t be taken as evidence but as rather as indices or suggestions of what is possible (Ejvegård, 2003).

In this thesis the case study will be the main focus of the research. It will be used to answer the first research questions; how hardware specifications can be stored in GitLab, as well as the forth question; to what extent this method for storage can meet medical regulatory demands. The case study will also provide material for the comparison method.

The case study will be divided into two phases. In phase one the focus will be on searching for and testing different special made solutions for the document types used in the case study company. The conclusions made from this will be used to evaluate what storage options will then be used in the second phase. The second phase will focus on testing the chosen storage solutions in an actual development project. The conclusion made from this will not only regard the technical solutions as such but also how they can be used by the company in question. During this phase the existing manual system will also be documented.

3.3 Comparison

Comparison is a scientific method used when two or more alternatives are to be compared with one another. Due to the difficulties of finding comparable units between alternatives the method has to be used with caution. It is important that:

- The units used have to be comparable or they have to be transformed into comparable units.
- The alternatives have to be generalized to focus on the entities that are to be compared.
- Similarities as well as differences have to be described.

In this study comparison will be used in order to evaluate the results of the experimental case study with the description of revision control in PDM-systems and manual systems. It will be used to answer the second and third research question; how revision handling in GitLab compares to that of a standard PDM-system and to that of existing manual system.
3.4 **Participant observation**

Participant observation is a data collection technique when the researcher describes a phenomenon or a process she participates in or an organization she has insight into. The advantage of participant observation is the possibility of depth analysis and true understanding of different stages and events. One disadvantage is that the outcome of the studied process can’t be known in advance and there is a risk that the researcher in retrospect might find that no greater insight was gained. Because participant observation is very time consuming and the final phase of the research may not be so useful, so it fits best in long-term projects that extend over a year or more and it is difficult to obtain materials about otherwise (Ejvegård, 2003).

This study targets an area that has not been subjected to a vast amount of research which means that written material is difficult to obtain, both in the scientific world and in the industrial. Due to this participant observation is chosen in spite of it usually being a time consuming technique. The case study to be observed is not a constantly ongoing process but a staged experiment and can therefore be planned and limited in order to maintain a possible time frame.

3.5 **Method criticism**

In order to reach a scientifically useful result certain special aspects must be considered regarding the method and techniques used. If these demands are not met with the research result can’t be considered to have any scientific value (Ejvegård, 2003).

3.5.1 **Reliability**

Reliability indicates the reliability and usability of the chosen measuring device and measurement units which is something to be cautious about in all sciences. Since the measuring instrument is often constructed by the researcher in question, such as questionnaire or interview questions, there is a risk that the measuring instrument reliability becomes low. An instrument that is stable and gives repeatable results is considered reliable. The reliability of a quantitative measurement can be ensured either by comparative test with an equivalent instrument or by retesting the same individual with the same instrument. In both cases the smaller the difference between the two measurements the higher the reliability. Qualitative methods are generally more complicated than quantitative when it comes to ensuring reliability but comparative testing and retesting can still be considered good guidelines (Ejvegård, 2003).

For this study the measuring device is entirely dependent on the researcher in question, since the data collection will be done by participant observation and no relevant literature on this particular area has been found to compare with. In order to achieve as good reliability as possible all tests of special solutions have either been complemented by comparative test or retest. Comparative test has
been made either with different mock up documents or with mock up document and a real specification. Retesting has been made with the same mock up document. The chosen case study has been made with both mock up documents and real specifications.

Also the results of the study have been continuously discussed with the involved staff at the case study company in order to find and eliminate eventual bias on the researcher’s side.

### 3.5.2 Validity

Validity indicates that the research method and technique actually measures what it is intended to measure. The key is to properly understand the units measured and to use them consistently (Ejvegård, 2003).

As with all case studies this one focuses on a small part in order to suggest what might be applicable for a larger population. In this case the study is performed on one company and one system in order to try if the system can achieve a level of control and usability that is acceptable to this particular company. The results cannot be assumed to be valid for all SME; there are too many variables to how a company functions. The analysis of the results will however take a more general perspective in order to reach conclusions that can be valid for the SME sector as a whole.
4 Results

This results section will be divided into three parts, the first part describes the manual system for document revision control currently in place, the second part reports on the practical experiments conducted to evaluate GitLab as a potential new revision control system and the third makes a comparison summary between the three system types of the study.

4.1 Existing manual system

Unique document numbering

Unique document numbering is achieved through a basic number generator that creates a document number based on generic prefix, date, document type and serial number. In order to receive a number the user has to add information to the system, which project the document belongs to and a short description. It is possible to postpone the creation of a document number until the first draft is ready for release stage which means that any saved iterations of the first draft has to be manually named with the number retroactively if they are to be correctly marked.

The marking of the document with the number is made in the document head and sometimes in the naming of the file but it is not uncommon that the files are named in a more descriptive manner.

Storage

All development is project based and for each project there is a folder with standardized named subfolders where all the files are kept. Every released version of a document is stored as a single file and whether iterations of the “in work” document are kept is a matter of personal choice of the developer in question. When the saved versions of a file become numerous enough to make it difficult to find the right version it is common to create an “Old” folder where the versions kept for traceability are kept.

The folders are kept on a server hard drive where backup is performed regularly.

Document status and revision

The status and revision of a document is handled in the number generator and marked out in the document head before the document is printed to .pdf, the revision usually in the naming of the file and the .pdf file. The status of a created document number is always “in work” until manually changed to released and the revision automatically follows the status so that if a document of “released” status is changed the revision is raised by one. Since the number generator is independent both from the program where the files are generated and the file manager where they are stored it is however possible to change the information in the document head but not in the system resulting in the system
information being incorrect. It is not uncommon that the document is updated more frequently during development and the system is updated before the whole project is released.

![Figure 8: manual system components](image)

Operator traceability and access
The number generator uses personal log-in so creation of number as well as changes to status and revision has the information about when and who attached to them in a manner that is easy to find for any operator.

All parts of the server hard drive are open to all employees but even with the standardized subfolders finding a document created by another developer takes some time and based on the number of iterations and revision saved it is not always clear which file should be used. It is common practice to double check with the colleague in question in order make sure the right version of a file is used.

4.2 GitLab usage
The part on GitLab usage is divided into two sub parts, the first sub part reports on attempts to find special made solutions for these specific programs or file types enabling more Git functionality than basic storage of binary files. The second subpart reports on how the possible solutions are used in this specific case study.

4.2.1 Evaluating special made solutions
Cadance documents
For Cadence documents one possible option for full Git integration was found for evaluation. It’s an open source software designed especially for the integration of Git with Cadence, CdsGit (https://github.com/cdsgit/cdsgit). Several Git features seem to be successfully integrated to the point where they can be accessed by dropdown menus in Cadence which is positive from a user perspective. Diff and Merge functionality is however not developed to a point where they can be used for both schemas and layouts which is too much of a drawback for the option to be further tested.
SolidWorks documents
For SolidWorks no special made solutions could be found.

Word documents
.docx is a binary file format even though a large part of the file can be presented as text. The first option evaluated was to convert the binary .docx file to a text file in Markdown which is a markup language with plain text formatting syntax designed to be easy-to-read and easy-to-write (http://daringfireball.net/projects/markdown/). There are several Markdown editors on the market but favoring user-friendliness the choice fell on Writage (http://www.writage.com/) which is a plugin enabling the user to work in Microsoft Word and directly save the file as .md instead of .docx. Once converted to .md Git could be used to its full extent and all commands usable for source code can be used in the exact same manner for instructions.

Testing of the concept however showed a few flaws. The file was not unaltered once it had been converted, pushed to the remote repository and cloned to a local repository at another location. The font was changed, all headlines were the same size, underlines and colors were removed and, most importantly of all, included pictures were lost. Since instructions rely heavily on pictures and educational layout this was not deemed good enough.

The second option was to store the .docx file directly without any conversion. A blog on the subject (http://geekswithblogs.net/mapfel/archive/2015/06/24/165317.aspx) had suggested that Git would be able to perform diff and merge on .docx files in spite of them being in binary format.

Testing the concept showed mainly positive results. All forms of textual contents, even empty lines, were tracked as though they were a text file, enabling full usage of commands such as diff and merge. Changes in layout such as adding color or bold text could be tracked as being a difference but not detailed as to what the difference was. Adding and removing pictures could also be tracked as a difference but not detailed. For these aspects diff and merge could not be equally useful. The document was not altered in any way when being pushed to the remove repository and cloned elsewhere.

Excel documents
.xlsx is a binary file format but it is based on zipped up directories of XML-files, which in turn are text files (https://msdn.microsoft.com/en-us/library/dd922181%28v=office.12%29.aspx).

The first option evaluated was to handle the .xlsx file as a collection of text files and thus take full advantage of the capabilities of Git. This can be done by always unzipping the file before placing it in the repository and then zipping it again before being able to work on it. This does however require several manual steps and full understanding of the file formats and the zipping process,
knowledge that is not generally part of the hardware development world, so this method was not considered user-friendly enough and no test was performed.

The second option for evaluation is a plug-in designed for excel (https://xltools.net/excel-version-control/), enabling committing and diffing directly from a menu in excel but not handling any branches except master. During testing it turned out that the local git-repository generated by the plug-in could not be made to communicate with a remote repository at GitLab.

**Basic storage**

When using Git without any special made plug-ins for specific programs the usage of it will be very similar to how most software developers use it, from the file manager program by means of an interface. There are many interfaces available, both command line interfaces (CLI) and graphical interfaces (GUI), several of them are free open source.

As previously stated storing of a binary file does not allow for usage of the more advanced Git functionalities that depend on Git being able to understand changes made to a file. Since branching strategy builds heavily upon these functionalities branching out a binary file from master seems an obvious risk.

![Figure 9: Storage without branching](image)

Without branching as a part of normal practice the important commands will be Commit, Push, Pull and Clone. The task was then to find an interface where these commands could be executed in a user friendly way as possible. The choice fell upon Tortoise Git (https://tortoisegit.org/), which is a plug-in to Windows file manager, enabling the user to make the commands by right clicking in the folder and choosing from the normal file manager menu.

**4.2.2 Case study**

**Avoiding Merge conflicts**
Hardware developer have the same need as software developers for collaboration on the same file or file package and since Git is a decentralized VCS there is no way for a developer to see from the remote repository if a colleague has a local repository where changes have been made. So without any kind of collaboration strategy the risk for merge conflict is still there. During this experiment the problem was solved by a hybrid with the old manual system. Instead of placing the local repository with the files to be worked on in a user specific folder it is placed in one of the standardized folder in the project folder where it is possible for all employees to see if someone is just then working on it, to continue work themselves and to commit and push the changed files to remove repository for storage.

Choosing repository strategy

Before storage was attempted on a real product there had to be a strategic decision on how to use the repositories, would one repository correlate to one product and store all documents for that product or would one repository correlate to one document storing only that one? The first option would make it easy to find all relevant specifications and see what revisions and iterations of the different parts that belonged together, i.e. were stored in the same commit. By using tags to certain commits it could mark released revisions or baselines for the whole product. But without branching there would be no way to keep commits for different documents separated from one another and they would all end up in one long chronological line where it would be very difficult to find commit comments referring to one specific document.

The second option, to keep separate repositories for separate documents, would enable a very clear revision history for each document but would not keep the documents together in the same obvious way as the first option. It would however make it possible to name the repository exactly as the document number of the document it referred to and the commits and tags could be used to mark revision or iteration and status of that document. That could render the independent number generator obsolete.

Figure 11 shows the two different repository strategies. The left side shown the product repository strategy. Here all commits are in one chronological line and the comment has to state which document they refer to. The right side shows the document repository strategy. Here commits for different documents are separated into individual chronological lines. The commit messages would not have to state which document they refer to. That’s added in the figure to better visualize the different strategies.
For a medical device it is important both to keep control of a total product revision and a detailed development history of the different parts but there is a slight difference as to when the two are important. Once the product has reached the marked it is crucial that the manufacturer can say exactly what revision is sold when and what revision of parts it is made up from. During development however, when the documentation will be used to be allowed to place the product on the market, it is crucial to be able to show how potential risk has been handled during development of all the different parts that constitute the device.

This particular case study evolves around the development phase and the strategic decisions will be made accordingly, so one repository represents one document, the commits will be used to get easy iteration history and tags will be used to mark document status and revision.

**Document and repository naming**

With one repository representing one unique specification the naming of the repository can be used as the document number. Since no two repositories on the same server can have the same name the uniqueness of the name will be guaranteed. The name has to be assigned manually and for the experiment we created a document number based of external customer name, project name, date, document type and a serial number. Together with the read me for the repository this makes it easy for all operators to find the repository in question and also to understand which repositories belong to the same project and what type of document they contain.

**Document status and revision**

When used for software development changes to a project are handled by branching out. A new feature or a bug is handled in a separate branch until it is developed and verified to a level where it can be merged with the master branch. Without branching the commits will still give a good change history and show
the different iterations and released revisions can be marked out using tags which make them more visible than if the information is in the commit comment only.

![Figure 12: Tags show released status and revision name](image)

**Creating a complete product file**

After deciding that one repository should represent one specification we faced the task of connection all specifications to one complete product. This was done by creating a product repository where all the other repositories could be added. The solution worked well out of user perspective, all levels of the product is available from one source and it’s easy to click ones way to the specific document or issue of interest. It is however not possible to connect the sub repositories in such a way so that it points out which commits (i.e. iterations and revisions) belong to a certain iteration of the product repository.

![Figure 13: gathering document repositories in a product repository](image)

**Issues**

A useful function in GitLab is the possibility of creating Issues. This was not tested to any great extent during the experiment but they are commonly used to report bugs and make change requests to a project and this can be used for hardware specifications in the same manner as for software development.
4.3 **Comparison summary**

<table>
<thead>
<tr>
<th></th>
<th>PDM</th>
<th>GitLab</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>Central database, automatic handling</td>
<td>Web based database, manual start, automatic continuation</td>
<td>Shared hard drive, manual handling</td>
</tr>
<tr>
<td><strong>Document numbering</strong></td>
<td>Automatic</td>
<td>Manual naming, documents can't be stored without proper name</td>
<td>Manual naming, naming independent of storage</td>
</tr>
<tr>
<td><strong>Revision handling</strong></td>
<td>Automatic revision handling, iterations not stored</td>
<td>Iterations automatically stored, revisions manually marked out</td>
<td>Manual storage, manual marking, risk for loss of revisions</td>
</tr>
<tr>
<td><strong>Configuration management</strong></td>
<td>Detailed product trees</td>
<td>Projects or repositories in repositories can be used to achieve a basic level</td>
<td>No</td>
</tr>
<tr>
<td><strong>Change management</strong></td>
<td>Build into system</td>
<td>Issues functionality can be used</td>
<td></td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>Individual user log-in</td>
<td>Individual user log-in</td>
<td>All users</td>
</tr>
<tr>
<td><strong>User traceability</strong></td>
<td>Actions can be traced to specific user</td>
<td>Actions can be traced to specific user</td>
<td>No</td>
</tr>
<tr>
<td><strong>Tasks / communication</strong></td>
<td>Build into system</td>
<td>Issues functionality can be used</td>
<td>No</td>
</tr>
<tr>
<td><strong>Part classification</strong></td>
<td>Build into system</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Table 1: Comparison between PDM system, GitLab and existing manual system*
5 Discussion

In this chapter the results are discussed and compared with the theoretical framework for the study. The chapter is divided into five sections, one for each research question.

*How can hardware specifications such as drawings, cad-files and build instructions be stored and controlled in GitLab?*

The experiments show that there are solutions available that are made to make Git functions work with specific binary files, especially in the open source community. But when tested they all lacked in some respect, either not all functionality was implemented or reliable enough or they were not deemed useful. It is however really interesting to see how many different solutions that were available just for the file types tested in this experiment. The open source community has proven itself very capable when it comes to development of version control tools (Spinellis, 2005; Spinellis 2012) and keeping in mind the continuous increase of products with embedded software (Hopf & Ovtcharova, 2013) it is not unlikely that more mature solutions for different binary files will reach the market within a few years.

The experiments also showed that storing binary files without Git functionality enabled by some special solution is possible and quite useful. Even if the files can’t be compared in the same manner as a text file can the advantage of iterative and commented versions can still be used.

*How does revision handling of such documents in GitLab compare to that of a standard PDM-system?*

Many of the functions that constitute a basic PDM-system are available in GitLab as well. There is the focus on individual naming, of control over revisions and the possibility of communication around the specifications stored in the system. What GitLab completely lacks in comparison to PDM is the focus on handling of parts. In GitLab the specifications are documents but in a PDM-system they are presentations of different parts and the tools for handling them take this into consideration. When a product structure is created it shows not only which parts a product consists of but also how many of each part. This is of great help to a person who is going to purchase material for the product or build it. The parts can also be classified into categories that enables a search ability that is helpful for a developer who wants to know if older parts can be reused for a new product, or a purchaser who wants to send all mechanic specifications to a possible new supplier.
But this complexity is also what makes PDM-system both expensive and time consuming to learn, something we know is problematic for SME (Waldenmeyer & Hartman, 2013). The usage of Git in the software development community has almost exploded (Spinellis, 2012) and this would not have happened had it not been an easy system to learn. For a company with only a few products that are continuously developed or a company that develops for others and reuse of components is not an issue many of the PDM-functionalities are never fully used. The fact that GitLab has less functionality than a basic PDM-system can be both a drawback and an advantage depending on the needs of the SME.

The key issue in using GitLab as a substitute for a PDM-system is probably the decentralized management which, with binary files that can’t be diffed or merged, has to be handled by some sort of collaboration strategy within the organization. Depending on how well this strategy fits with the organization as a whole GitLab can either be very useful or rather risky to use.

How does revision handling of such documents in GitLab compare to that of a company that’s has not implemented a PDM-system?

Compared with the old manual system one big difference is the system support for storage and unique naming of saved iterations and the change history that is built in by the commit comments. Having the document control as part of the storage instead of an independent system helps the developers use it continuously instead of having the risk that is handled as one of the things that has to be fixed before the project is done.

There is also an advantageous difference in how the documents are stored, the “read me” that belongs to every repository helps users find the right repositories and with the history handled in the system the older versions of a file don’t take up space in the folder and can’t confuse any user looking for the current file.

From a medical regulatory point of view the possibility to create issues connected to specific documents is a huge advantage as this enables the developer to document risk analysis throughout the development cycle, thus avoiding the time and cost associated with documentation in retrospect. As Romer & Stuyt (2007) points out as an important aspect of successful medical device development.

But the drawback is, as when compared to a PDM-system, the switch from a centralized management to decentralized without being able to utilize the tools that make this an advantage.

To what extent can the documentation and revision control possible in GitLab meet the demands of regulatory standards for medical devices?
Two key issues for meeting regulatory demands for medical devices are good document control and risk management.

With the continuous saving of iterations throughout development, the comments connected to every commit of the repository and only the latest version of the documents up front GitLab is an excellent tool for keeping good document control. With binary files, where all Git functionalities can’t be used, the system will however not be better than the usage of it and the collaboration strategy of the company in question.

During this experiment branching was completely unused but this was a matter of choice based on the risk detected in this company at this time. Branching is not forbidden or impossible when it comes to binary files, it just creates parallel versions of a repository and the documents in it that Git functions can’t be used to sort out. But if a company’s way of working fits with this branching out for new development and merging back after verification it could be used in same manner as it is done with software, giving an even better control of document status and history.

The configuration management could also be improved by allowing usage of branching. If the repository represents the complete product, separate branches can be made for each individual document. Master branch would represent the releases at product level and the other branches keep the iteration history for the different documents. The document number could be represented by the name of the branch instead of the name of the repository. For the configuration management to handle the complete product this would however have to include the software development into the same repository as all the other documents and this would go against common practice for software development where one repository is one part. Testing this approach would be a more extensive experiment than the one for this study.

However, the configuration management becomes more critical as the medical device reaches its marked and is implemented with users, and by then the most intense development phase of the product should be over. After all, that is what the regulatory demands are about; preventing devices from reaching the marked before they are proven stable enough. Maybe iteration history and compiling the technical file continuously during development are more important than making every effort to having the configuration management work automatically. The manual effort needed in marking out certain commits with tags could also be used as a verification of second person audit or some other toll gate of the quality assurance.

Continuous risk management can easily be documented in a combination of issues, commit comments and extra documents, such as test reports, connected to the commits where they are relevant.
6 Conclusions

This chapter states the conclusions made from the study.

*How can hardware specifications such as drawings, cad-files and build instructions be stored and controlled in GitLab?*

Hardware specifications can be stored in GitLab in the same iterative manner as software but mainly as binary files with no or very limited possibility to use the more powerful Git functions. Even though there are special made solutions for a lot of different file types to be found within the open source community the ones tested in this experiment all lacked in some respect, functionality, usability or reliability.

*How does revision handling of such documents in GitLab compare to that of a standard PDM-system?*

GitLab offers the same level of revision control and communication around the specifications as a PDM system. It also offers an iteration history that is not available in a general PDM-system. Instead it lacks the handling of components as parts, the classification of them that enables search ability and the detailed product structure. Being decentralized but not being able to use its merge functionality also means that it lacks the automatic collaboration management that a PDM system offers.

*How does revision handling of such documents in GitLab compare to that of a company that’s has not implemented a PDM-system?*

GitLab offers a well-structured iteration history of a document. It enables comments by the developer to be connected to the different iterations and to the repository. It also enables other users to communicate bugs, risks and improvement suggestions by using issues. All of these are advantages compared with the existing manual system. Being decentralized it does however need a collaboration strategy that the shared hard drive of the manual system does not demand.

*To what extent can the documentation and revision control possible in GitLab meet the demands of regulatory standards for medical devices?*

The commented iteration history makes a good document control on document level. Different solutions with branching or repositories in repositories can be used to ensure configuration management. Not being able to use full Git functionality the organization instead needs a collaboration strategy that ensures the decentralized approach does not create problems. However, developing for the medical sector relies on regulated practices and if these practices can be made to fit with the capabilities of GitLab the system will be of good use for medical device documentation.
7 Social and ethical considerations

As a researcher, one must constantly review one’s own work and with critical eyes consider what implications the research might have, not only for the field in which the research is conducted but also from an ethical and social perspective. The medical sector is one of high ethical and social importance, a patient seeking medical help is already in a vulnerable position and it is critical that devices used for examination and treatment function as little smoothly as possible in order not to make the patients situation even harder. Therefore it is crucial that devises are developed under good regulation. But it is also crucial that the regulations are not so difficult to work with that development doesn’t take place at all. An undeveloped device will cause no risk that is granted, but nor will it be of any help for the patient in need of treatment. For those reasons I find it important from both a societal and an ethical point of view to conduct research in the medical development sector, asking questions on how development can be made both safe and efficient in order to convert medical devices ideas into safely working equipment used for patient examination and treatment.
8 Future research

As with all case studies this one has been able to evaluate only a small portion of the options available to SME within the development sector and many other perspectives would be interesting to investigate.

One perspective that was in the original plan for this study was the usability aspect, whether hardware developers find Git as user-friendly as software developers do. The time frame for the perspective was however too short to be able to include this perspective as well as the others. A researcher who wants to study the usability should probably make this the prime focus of the study.

Git is based on decentralized principles which have to be handled by the company way of working in order to not create more problems than gain. There are also VCS build on centralized principles on the market ad it would be interesting to perform a similar experiment using one of those.

As this study has shown there are special made solutions for binary files available in the open source community but neither of the ones tested here was deemed mature enough to be used for development of medical devices. Adding a scientific approach to the development of one of these projects might render good insight into why the currently available solutions have not succeeded and hopefully suggest improvements that could lead to more useful tools in the future.

This study suggests that using software VCS for other file types than it was originally intended may be a useful approach to a SME that can match the drawbacks and advantages of the system with the working ways of the company. SME’s are generally described as more flexible and higher adaptability than larger companies (Harrigan et al., 2009). It would be interesting to study if this affects their choices of software tools, what kind of tools are typically chosen and to what extent the usage of them is modified in order to match the SME.
9 References


Brinnen D & Nord R (2013) “En värdtjänst för mjukvaruutvecklingsprojekt UTVECKLING AV ETT VERKTYG FÖR ATT EFKEKTIVISERA PROGRAMMERING” DEGREE PROJECT, FIRST LEVEL KTH Royal Institute of Technology


Cdsgit “Cadence Virtuoso Git Integration written in SKILL++” https://github.com/cdsgit/cdsgit retrieved 2017-03-10


Läkemedelsverket ”Medicintekniska produkter” https://lakemedelsverket.se/malgrupp/Foretag/Medicinteknik/ retrieved 2017-03-10
A new approach for medical device product documentation  
Elisabet Hamnede  
2017-05-21


Otte S (2009) (Version Control Systems” Computer Systems and Telematics; Institute of Computer Science Freie Universität Berlin, Germany


36
A new approach for medical device product documentation
Elisabet Hamnede


Tortoise Git “Windows Shell Interface to Git” https://tortoisegit.org/ retrieved 2017-04-03


Writage “Markdown plugin for Microsoft word” http://www.writage.com/ retrieved 2017-02-28