

Bachelor's thesis  
Degree programme  
2018

Hemmo Helminen

# MEDICAL DEVICE SOFTWARE MAINTENANCE

– Process improvement

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technologies

2018 | 50 of pages, 2 pages in appendices

Hemmo Helminen

# MEDICAL DEVICE SOFTWARE MAINTENANCE

## -Process improvement

This thesis was commissioned by Evondos Ltd, which is a Finnish healthcare service company.

The objective of the thesis was to improve procedures and work methods in the Evondos software maintenance. An additional objective was to analyze how well standards are followed.

In order to achieve the objectives, the work was conducted in three phases. In the first phase, the key standards of medical software maintenance were studied. Information from the first phase was utilized in the second phase where current work methods and procedures are analyzed.

In the second phase, the current work methods at Evondos were analyzed as a whole. Activities and deliverables for one software release were analyzed thoroughly. Based on the findings, root causes were analyzed and improvement proposals were created.

In the third phase, the selected improvement proposal was designed and implemented.

A missing localization process was found to be the greatest single challenge in the software development. The purpose of the localization process is to describe the process and responsibilities for adding and updating the Evondos service text and voice messages to the customer languages.

As an end result, the localization process was designed and approved, the work methods and documentation were analyzed against the IEC 62304 standard. In addition, there were many other improvements identified and partly implemented.

The performance of the localization process will be followed and it will be further developed as seen feasible and appropriate.

## KEYWORDS:

software development, localization, software maintenance.

Hemmo Helminen

# LÄÄKINNÄLLISEN LAITTEEN OHJELMISTON YLLÄPITO

- Prosessien parantaminen

Opinnäytetyön tarkoituksena oli analysoida ja kehittää ohjelmistokehityksen toimintatapoja ohjelmiston ylläpitovaiheessa. Lisäksi tavoitteena oli tutkia, kuinka hyvin ohjelmistokehityksen käytännöt täyttävät keskeisimpien standardien asettamat vaatimukset lääkinnällisten laitteiden ohjelmistokehitykselle. Opinnäytetyö on tehty Evondos Oy:lle, joka on suomalainen terveysteknologiaan erikoistunut yritys. Yritys on kehittänyt automaattisen lääkeannostelupalvelun kotihoidon piirissä oleville pitkäaikaislääkityille.

Ohjelmistoversion tekemiseen tehdyt aktiviteetit sekä aikaansaannokset analysoitiin aikaisemmin julkaistua ohjelmistoversiota hyväksi käyttäen. Edellisen lisäksi, ohjelmistokehityksen dokumentointia ja työskentelytapoja analysoitiin IEC 62304 standardin tarkistuslistaa hyväksi käyttäen.

Analyysin perusteella isoimpina ongelmakohtina nousivat esiin ohjelmistoon loppuvaiheessa lisätyt muutokset, kehitystiimin tehottomuus muutosten toteuttamisessa sekä tärkeän testauksen painottuminen ohjelmistoprojektin loppupuolelle. Ongelmakohtia tarkemmin tutkittaessa selvisi varsinaiset juurisyyt ongelmakohtien takana. Pitkä julkaisuväli ohjelmistoversioiden välillä aiheutti paineen uuden sisällön saamiseksi meneillään olevaan kehitykseen. Ohjelmistokehityksen tehottomuuteen liittyi useita selittäviä tekijöitä, mutta isoimpana ongelmana nousi esiin käyttöliittymien käännöstyöhön liittyvät epäselvyydet. Testauksen painottuminen projektin loppupuolelle johtui testitapausten suuresta määrästä verrattuna käytettävissä oleviin resursseihin ja testausmetodeihin.

Löydösten pohjalta kehitettiin lokalisointiprosessi, joka kuvaa Evondos-palvelun tekstien ja puhuttujen viestien kääntämiseen tarvittavat aktiviteetit sekä aktiviteetteihin liittyvät roolit. Ohjelmistoversioiden julkaisuaikataulujen nopeuttamiseksi käynnistettiin laitteiden etäpäivitykset mahdollistava projekti. Testauksen kehittämiseksi aloitettiin testiautomaation käyttöönotto, jotta resursseja lisäämättä testausta voidaan tulevaisuudessa tehdä aikaisemmin ja nopeammassa tahdissa.

Opinnäytetyön isoimpina aikaansaannoksina olivat lokalisointiprosessin kehitys sekä muiden ongelmakohtien korostaminen, jonka johdosta useat korjaavat toimenpiteet saivat yrityksen johdossa riittävän prioriteetin asioiden eteenpäin viemiseksi.

ASIASANAT:

ohjelmistokehitys, ohjelmiston ylläpito, lokalisointi.

# CONTENTS

<b>LIST OF ABBREVIATIONS</b>	<b>7</b>
<b>1 INTRODUCTION</b>	<b>8</b>
<b>2 FRAMEWORK FOR THE MEDICAL DEVICE SOFTWARE DEVELOPMENT</b>	<b>12</b>
2.1 Standards and regulations	12
2.2 Medical device software – software life cycle processes	13
2.2.1 Software maintenance plan	16
2.2.2 Feedback monitoring, documentation and evaluation	17
2.2.3 Creating problem report and change request	18
2.2.4 Problem investigation and change analysis	20
2.2.5 Modification implementation	22
2.2.6 Quality assurance and releasing	26
<b>3 EVALUATING THE WORK METHODS</b>	<b>27</b>
3.1 Study of a release	27
3.1.1 Decision to start release testing	28
3.1.2 Changes during release testing	29
3.1.3 Other findings	34
3.1.4 Release 1718 conclusions	34
3.2 Root cause analysis and improvement proposals based in 1708-release	35
3.2.1 Testing	35
3.2.2 New content: New camera lens	35
3.2.3 New content: Patient user interface renewal	36
3.2.4 New content: The date and time inserting logic	36
3.3 Localization	36
3.4 History validation feature	37
3.5 Development team resourcing	38
3.6 Review of medical device Software maintenance process	38
<b>4 IMPROVEMENTS</b>	<b>40</b>
4.1 Localization procedures	40
4.1.1 Phase 1: Process discovery – What, Why and How	40
4.1.2 Phase 2: Understanding the As-Is Process	43

4.1.3 Phase 3: Designing the To-Be Process	44
4.2 Implementation of other improvements	45
4.2.1 Shortening the release cycle	45
4.2.2 Customer feedback	46
4.2.3 Requirements management	46
4.2.4 Software development	46
4.2.5 Resourcing	46
<b>5 CONCLUSION</b>	<b>47</b>
5.1 What is the significance of the results?	48
5.2 Lessons learnt	48
<b>REFERENCES</b>	<b>50</b>

## APPENDICES

Appendix 1. Identified improvements

## FIGURES

Figure 1. Release candidates for 1708 release.	28
Figure 2. Software changes during release testing.	29
Figure 3. Workflow illustrated as swim lane diagram [8].	43
Figure 4. As-Is process for changing existing text.	44

## PICTURES

Picture 1. Evondos service system architecture [2].	8
Picture 2. Evondos software development process overview [3].	10
Picture 3. Improvement process.	11
Picture 4. Software maintenance process [4].	15
Picture 5. Software maintenance process.	16
Picture 6. Example of defect report.	19

Picture 7. Maintenance phase code line management example.	25
Picture 8. Software changes divided into categories.	31
Picture 9. Checklist for small companies without a certified QMS [4].	39
Picture 10. Localizing the service process in the software lifecycle process	42

## **TABLES**

Table 1. Essential standards for medical device software development.	12
Table 2. Information sources.	18
Table 3. Changes after first release candidate.	32
Table 4. Localization changes.	33

## LIST OF ABBREVIATIONS

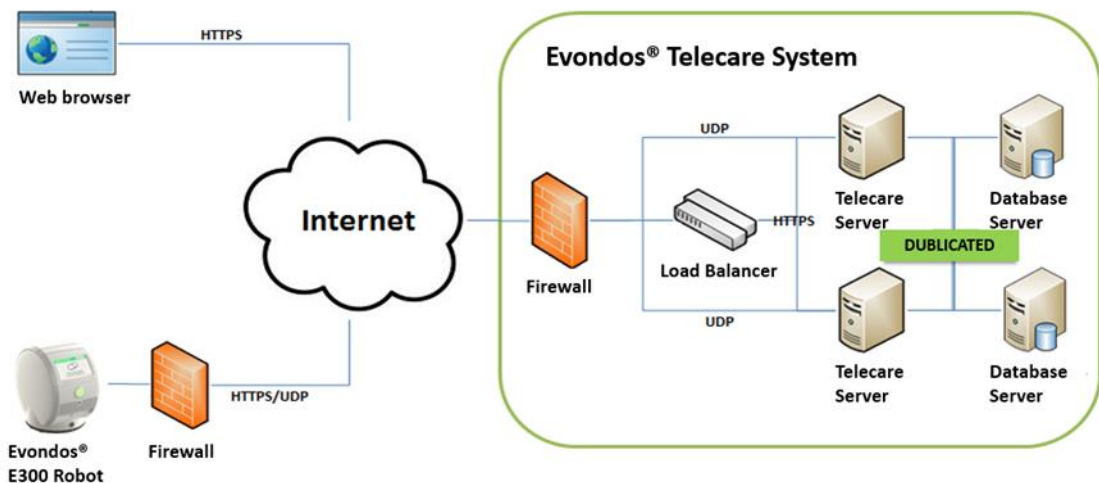
EPIC	Feature or customer requirements in Agile
FTA	Fault Tree Analysis
FMEA	Failure Mode and Effects Analysis
HTTPS	Hypertext Transfer Protocol Secure
Jira	Tools for software team to plan, track, and release software
MDD	Medical Device Directive
NPS	Net Promoter Score
QMS	Quality Management System
SMS	Short Message Service
SOUP	Software Of Unknown Provenance
Trac	Trac is an issue tracking system for software development
UDP	User Datagram Protocol
UI	User Interface

# 1 INTRODUCTION

This thesis was commissioned by Evondos Ltd, which is a Finnish healthcare service company. Evondos main office, including research & development and the factory is located in Salo. In addition, there are sales offices in Sweden, Norway and Denmark. Evondos employs around 60 persons.

Evondos has developed a unique solution for patients who receive home care and need long-term medical treatment. The Evondos® automatic medicine dispensing service enables patients in need of long-term medical treatment to receive the right medication at the right time and in the right doses – completely automatically. The service improves the patients' medical adherence and safety while introducing significant direct cost savings and quality benefits in healthcare. [1]

Picture 1 illustrates Evondos service architecture. The Evondos Service consists of Evondos® E300 Medicine Dispensing Robots and the web-based Evondos® Telecare System.



Picture 1. Evondos service system architecture [2].

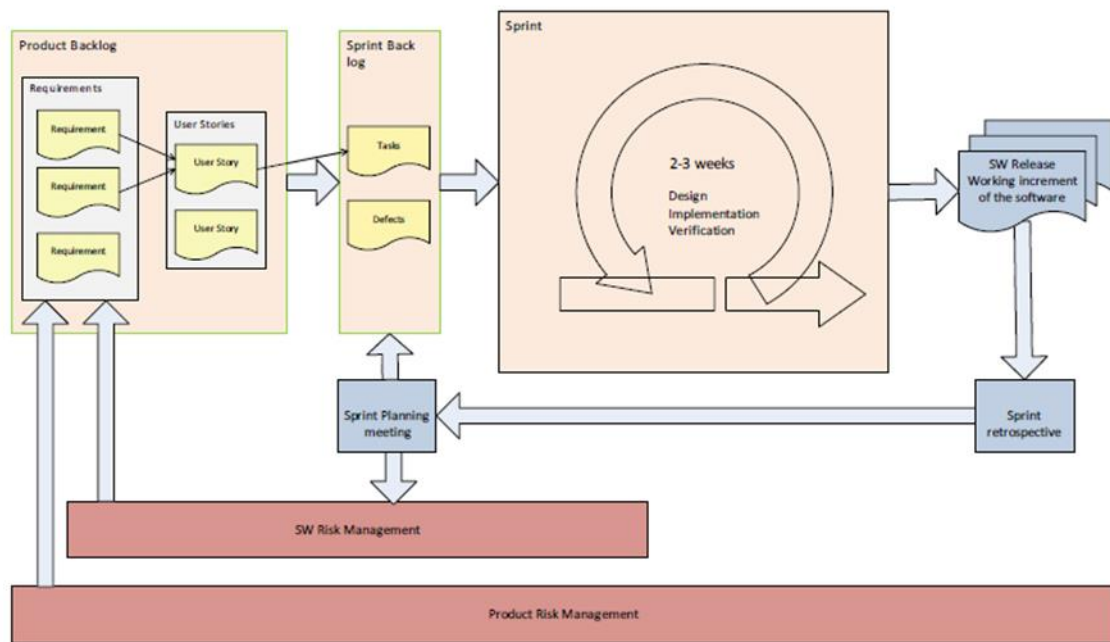
Telecare application servers are accessed with a browser over an HTTPS connection. Medicine Dispensing Robots communicate with the Evondos® Telecare System over a



secure HTTPS connection. In addition, UDP is used, but only for non-sensitive actions such as monitoring and keep alive messaging [2].

The software (SW) team is responsible for the software for both Medicine Dispensing Robots and the web based Telecare System as well as the communication mechanism in between. In addition to the software development, the SW team is supporting service operations in a problem solving and by advising customer support organization.

The Evondos SW development process is documented in a SW Development Process document. The Evondos SW development process is based on iterative and incremental software development process (Picture 2). Software is developed in 2-3 week sprints. The SW Development team is selecting user stories and bugs to the sprint backlog according to the product backlog priority. User stories are usually split into smaller pieces of work – tasks or subtasks. Since team members are not co-located, daily scrum meetings are not part of the development practice, however, team members are actively collaborating via chat. A sprint ends with a sprint review and a retrospective meeting. A sprint review is concentrating on what has been accomplished during the sprint whereas the purpose of the retrospective is to find what went well and what potential improvement areas are. In practice, the sprint review is taking most of the focus and a separate retrospective meeting seldom happens. In order to fulfill the Medical Device directive requirements, the software development process is extended with additional risk management activities [3].



Picture 2. Evondos software development process overview [3].

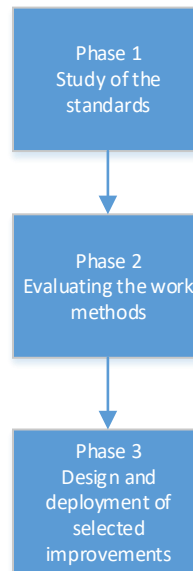
The Telecare service software and the robot software are developed in separate projects. The development teams are separate and have no common team members. Also, tools for handling tickets (user stories and bugs) are partly different. Trac is used both for the Telecare service and for the robot software; however, Telecare service tickets are also entered to the Jira. Actual development performed by the Telecare service development team is followed in the Jira tickets and the purpose of the Trac is just to have all tickets in one place to follow the software development as a whole. Trac tickets and Jira tickets are synchronized manually.

Software deployment to the existing robots in the field is carried out by sending software updates in memory sticks to the care organization. The care organization personnel is updating the new software to the robots. The update procedure requires a higher level of privileges, thus the privilege level of the person who performs the update must be raised.

The software team is facing many challenges when developing and maintaining software for the Evondos medicine-dispensing robot. Consequently, releases are late, deployments of new features are taking too much time, new requirements are introduced close to the release date and there are difficulties in predicting schedule versus content.

The objective for this thesis is to understand the main challenges in the robot software development more thoroughly, find the root causes of the challenges and based on the findings, to create improvement proposals and deploy selected improvements.

In principle, the work is divided in three phases illustrated in the Picture 3.



Picture 3. Improvement process.

In the study of the standards phase (phase 1), the key standards of medical software maintenance are studied. Information from the first phase is utilized in the second phase where current work methods and procedures are analysed.

In the evaluating the work methods phase (phase 2), the current processes and work methods are studied. Two different considerations are taken into account when analysing the current work methods. Firstly, one release is studied thoroughly to understand the concrete challenges in the creation of the software release. Secondly, the overall work methods and documentation are compared to the IEC 62304 (Medical Device Software life cycle processes) [4] to ensure that the common framework is followed. Problems or challenges found in the evaluation are then analysed further and root-cause analysis is performed.

In the design and deployment of selected improvements phase (phase 3) different options for improvement are studied and considered. Improvement proposals are presented to key stakeholders and an action plan is agreed. As an end result, there is list of actions and selected improvements are implemented as part of this thesis work.

## 2 FRAMEWORK FOR THE MEDICAL DEVICE SOFTWARE DEVELOPMENT

In this chapter the key standards of medical software maintenance are studied. Chapter describes an example maintenance process where these standards are taken into account.

### 2.1 Standards and regulations

While designing and modifying processes for medical device research and development, it is essential to ensure that legislation and regulations are followed. Medical device markets are highly regulated and while operating in European Union area, European Union directives need to be adhered to. Manufacturers shall ensure that products meet the requirements. The Evondos medicine dispenser and Telecare system are class I medical devices. Medical device classification is defined in European Union Council Medical Device Directive 93/42/EEC (MDD) [5].

All medical devices must comply with the essential requirements listed in Annex I of the MDD, ensuring that they do not compromise the health and safety of patients, users, and any other person and perform as intended by the manufacturer. Medical Devices bear the CE mark to indicate their conformity with the MDD [6].

The standards in Table 1 are setting the framework for processes and procedures for medical device manufacturers. For this thesis, the IEC 62304 (Medical Device Software Life Cycle Processes) is the most relevant standard.

Table 1. Essential standards for medical device software development.

IEC 60601	Medical electrical equipment, general requirements for basic safety and essential performance.
IEC 62304	Medical device software life cycle processes [4].
ISO 14971	Standard for the application of risk management to medical devices [7].
IEC 62366	Application of usability engineering to medical devices[11].
ISO 13485	Medical devices. Quality management systems. Requirements for regulatory purposes[10].

## 2.2 Medical device software – software life cycle processes

In case the software is an integrated part of the medical device functionality, the manufacturer shall ensure that a) it is understood what software is intended to do. b) the software fulfills the expectation without any unacceptable risks. For that purpose, international standard IEC 62304 (Medical device software - software life cycle processes) [4] provides a framework of processes for the medical device software development and maintenance. This standard defines the necessary requirements for process steps in the software life cycle.

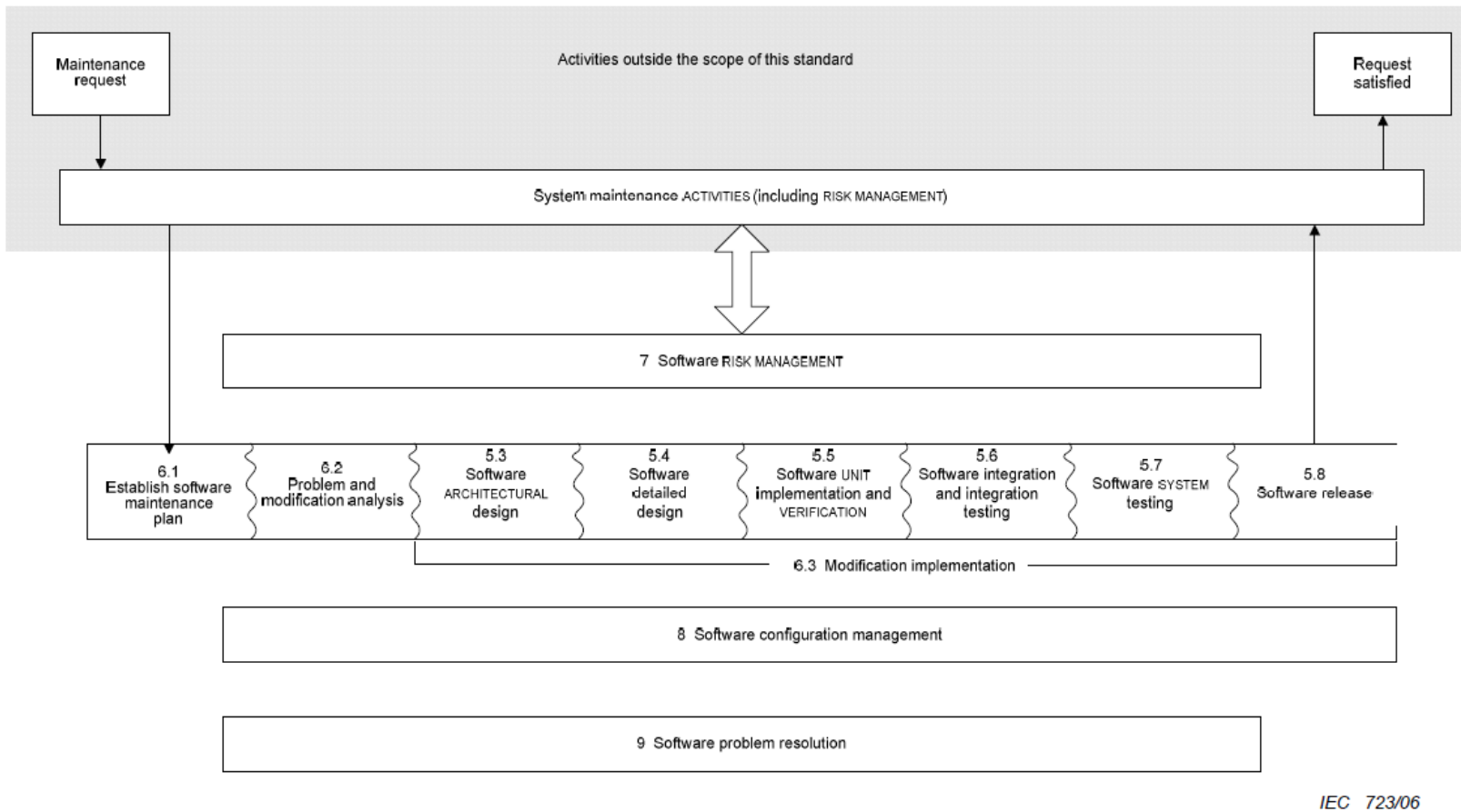
The standard describes software development and software maintenance requirements in separate chapters. Many areas in the software life cycle management are similar in the development and maintenance phase. The purpose of the software development process is to create a new software program to fulfill customer needs whereas the software maintenance purpose is to modify an existing software. The thesis focuses on the maintenance part of the life cycle processes.

A software program is in maintenance after it has been released to the customers, i.e., the software maintenance part is of the post-production activities. Software maintenance means that the existing software is patched or upgraded to its newer version. There are several different reasons for software updates, such as upgrade of the system to improve current functionalities or to fulfill new customer needs, to fix defects or deploy security patches to remove vulnerabilities.

The software lifecycle processes in the software maintenance are modification, implementation, software risk management, software configuration management, and software problem resolution as illustrated in the Picture 4. It can be noted that inside the modification implementation process, activities are directly referring to the software development lifecycle processes (Activities 5.3-5.8) [4].

Risk management activities are mandatory and shall comply with ISO 14971 (Application of Risk Management for Medical Devices) [7].

The software maintenance process should not only define how the activities are performed but also ensure that the organization has capacity and capability to react when issues are identified.



Picture 4. Software maintenance process [4].

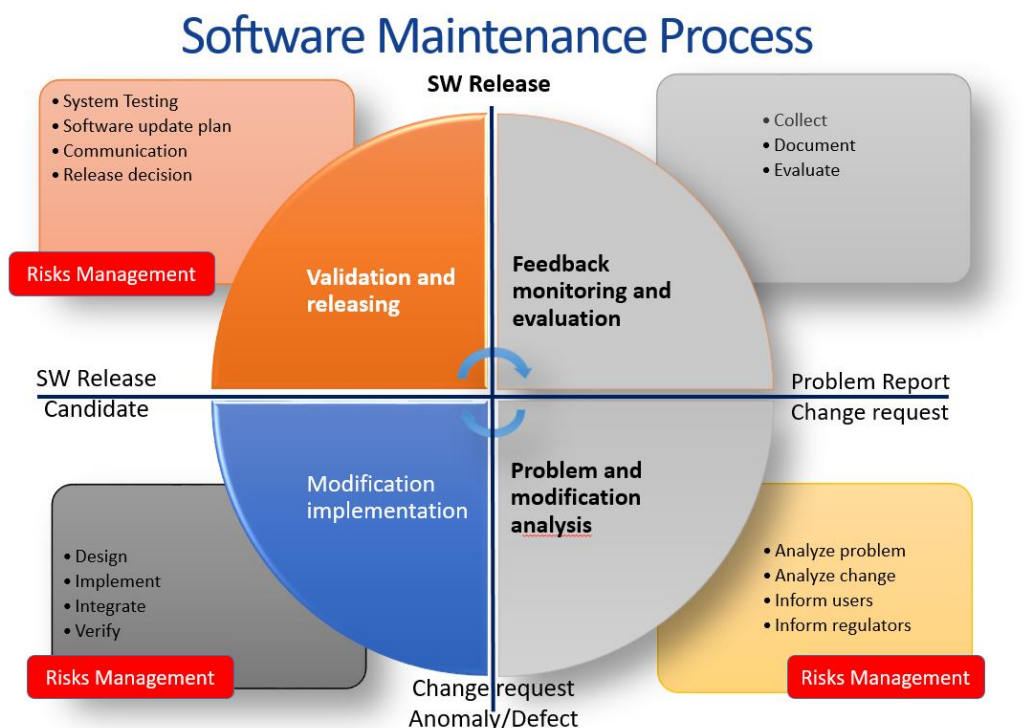
## 2.2.1 Software maintenance plan

A manufacturer shall create a software maintenance plan as defined in IEC 62304 (*Medical Device Software life cycle processes*) [4]. The plan shall cover all the areas from feedback to new software release.

The following areas shall be covered in the maintenance plan [4]:

- Procedures for receiving, documenting, evaluating, resolving and tracking feedback arising after the release of the MEDICAL DEVICE SOFTWARE
- Criteria to determining whether the feedback is considered to be a problem
- Use of the software RISK MANAGEMENT PROCESS
- Use of the software problem resolution process for analysing and resolving problems arising after release of the MEDICAL DEVICE SOFTWARE
- Use of the software configuration management process for managing modifications to the existing SYSTEM
- Process to evaluate and implement upgrades, bug fixes, patches and obsolescence of SOUP

The software maintenance can be considered as a continuous four-phase cycle and is illustrated in Picture 5.



Picture 5. Software maintenance process.



Software maintenance is a continuous process where feedback and customer needs are translated to software changes and finally a set of software changes will form a new software release.

### 2.2.2 Feedback monitoring, documentation and evaluation

Both IEC 60324 (*Medical Device Software life cycle processes*) and ISO 13485 (*Medical Devices Quality management systems. Requirements for regulatory purposes*) define that a company shall have documented procedures to receive and handle feedback.

The purpose of the activity is to gather and monitor information to understand how well the requirements are met and how the software system is performing in the field. A feedback mechanism and feedback procedures shall also make visible when there is a deviation to the expected behavior. All user groups and roles who use the software should be considered and feedback channels should be designed and implemented accordingly. Both external and internal users shall be taken into account. The following procedures shall take place after the software is released:

- Collect and receive feedback continuously
- Create visibility by documenting the feedback
- Analyze and evaluate feedback and update documentation for fact based decision making
- Provide input to risk management and problem resolution processes
- Provide input to continuous improvement for upcoming SW updates

A manufacturer shall define sources from where valuable feedback shall be collected. Documentation shall answer the following questions:

- How feedback is collected?
- How often feedback is collected?
- How feedback is documented?
- Who is the responsible person for feedback collection and documentation?

A responsible person must ensure that the data is gathered and documented.

Feedback channels should cover all possible feedback sources. Table 2 provides an example of possible information sources.

Table 2. Information sources.

Information Source	Frequency of update	Responsible person
Customer Feedback (NPS questionnaires, direct contact)	Continuous, NPS feedback rounds twice a year	Service Leader, Sales
Service Organization	Continuous	Service Leader
Customer Care – Customer complaints and Repair data	Continuous. Critical problem escalated immediately	Service Leader
Telemetry data from robots	Continuous	Software Leader
National Cyber Security Center alerts	Continuous	Software Leader
Hardware Development & Sourcing	Continuous	Hardware Leader

All feedback, negative and positive, is valuable and is great source for improvement ideas and new features. Regular reviews for all feedback and selection of the best ideas for further consideration should be part of standard operational procedures. [4]

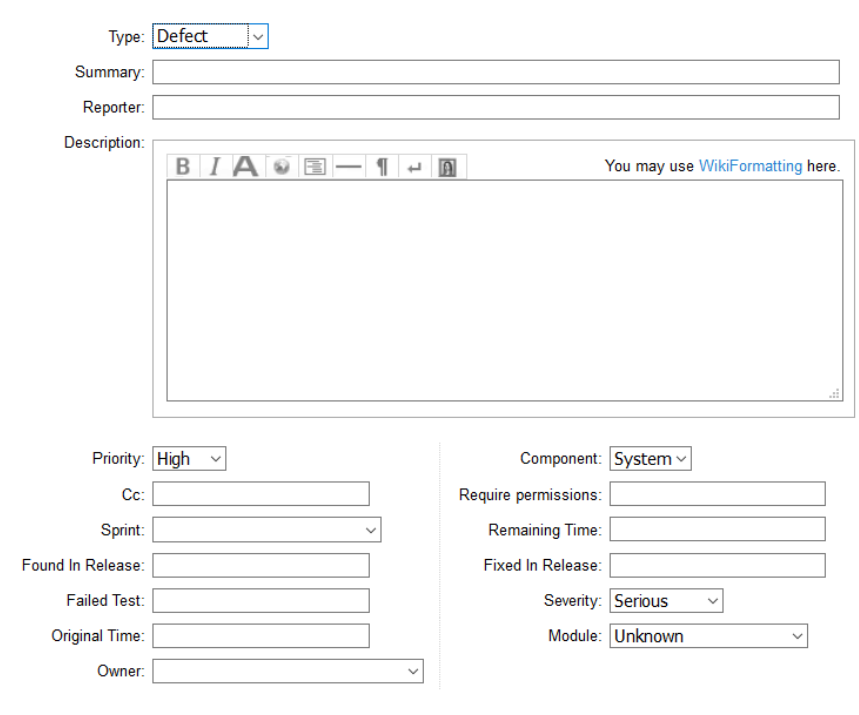
### 2.2.3 Creating problem report and change request

A problem report is created when feedback is evaluated and a problem is identified in a software system.

A problem report is documented as a defect (Anomaly) which is created to the ticketing system. A defect report example is shown in Picture 6. A problem report shall include all relevant information which is needed in problem investigation, problem resolution, problem verification as well as in risk management activities. One common pitfall is that the problem reporter has omitted some important information. Often the case is that at the time of reporting something is so self-evident for the reporter that information is left out. Later, when the developer is analyzing the report, he/she does not understand the

case thoroughly enough due to the missing information and more information must be requested from the problem reporter.

To improve the quality of the reports, feedback to the reporters shall be given regularly.



The image shows a web form for creating a defect report. At the top, there is a dropdown menu for 'Type' set to 'Defect'. Below it are input fields for 'Summary' and 'Reporter'. The 'Description' field is a large text area with a rich text editor toolbar and a note that says 'You may use WikiFormatting here.'. Below the description are two columns of fields. The left column includes 'Priority' (set to 'High'), 'Cc', 'Sprint', 'Found In Release', 'Failed Test', 'Original Time', and 'Owner'. The right column includes 'Component' (set to 'System'), 'Require permissions', 'Remaining Time', 'Fixed In Release', 'Severity' (set to 'Serious'), and 'Module' (set to 'Unknown').

Picture 6. Example of defect report.

A defect report has a title, a description and many fields or criteria. An example of the most important fields is described below:

- **Summary:** Title of the defect, it should describe the problem in one sentence
- **Description:** Describes the problem, including information such as:
  - How the problem is affecting the use of software or product
  - How the problem is visible
  - Steps to reproduce the problem state
  - How the software or system recovers from the problem state
  - Possible environmental factors which may affect the likelihood of the occurrence of the problem.

In the example picture, the estimated probability (or likelihood) of the problem to occur shall also be added to the description because the likelihood and the severity need to be known in order to calculate the risk index.

- **Severity:** How seriously the problem affects the use of the software. Example severity levels are categorized as Negligible, Minor, Serious, Critical and Catastrophic. Severity levels should be aligned with severity levels defined in the Manufacturers risk management.
- **Priority:** The risk management process will set the priority for the problem. Priority levels are categorized as Low, Normal, High and Critical.
- **Found in release:** on which version of software the problem is found.
- **Original time:** When the problem was detected. This is important information needed in the logs investigation.

In addition, there might be other fields to help to deal with the report and define responsibility:

- **Sprint:** To which development sprint correction work is allocated.
- **Owner:** Person who is responsible to fix the problem.
- **Fixed in release:** Software version in which the fix for the problem is released.
- **Remaining time:** How many work hours the implementation is estimated to take.

Manufacturer shall use a problem resolution process to address the problem.

In addition to changes caused by the problems, there can be other changes as well. Feedback evaluation may conclude that a change to the software system is needed to address, for example, customer feedback, regulation changes or information security alerts. In addition, there can be an innovation which causes a change request for a software system. The IEC 62304 standard does not explicitly define when feedback shall raise a change request if feedback is not found to be a problem. However, manufacturer feedback evaluation and product management activity shall be such that the process ensures regulation changes and customer-related needs are taken into account also in the post-production phase. [4]

#### 2.2.4 Problem investigation and change analysis

The main purpose of the problem investigation and change analysis phase is to utilize a problem resolution process to investigate problems further and identify possible root causes. In addition to the problem investigation, each change request shall be analyzed to understand the effect on the existing software system, including the interfaces.

The problem report is analyzed. A risk management process is utilized to evaluate the problem's relevance to the safety of the software system. Risks are recorded in a risk management file. Fault tree analysis (FTA) and/or failure mode and effects analysis (FMEA) are utilized and mitigation suggestions collected. The results of the risk analysis determine the priority for the problem. The results might also trigger a new change request to the backlog for risk mitigation. [7]

Depending on the local legislation, the relevant parties including regulators shall be informed about the found problem.

The defect shall be allocated to a developer for further investigation. Based on the investigation result, the next actions are agreed. A decision can be, for example:

- More information is needed from problem reporter.
- A test request to the test engineer (or other role) who is expected to reproduce the problem.
- The Defect is approved for fixing by setting a target milestone (Release) and is set to defect backlog according to the priority.
- After analysis, if the defect priority is lower than current bar for release under development, the defect is left lower in the backlog.
- The defect is found to be duplicate for an already known defect. The report will be closed as duplicate. The closed ticket will be linked to the original defect. The original defect priority is set higher if the new defect report has impact on the severity or occurrence. The work will continue on the original defect.
- The cause is found to be user error where the software or device had been used improperly or environmental factors have been out of the specifications. However, since it is a medical device in question, a risk assessment is needed to understand the severity and the probability of the harm. If the risk index is high, mitigation actions are needed even though the device and the software itself work as specified.

Change requests are analyzed by the software team. Often more information is needed to thoroughly understand the expected outcome. The risk management process shall be utilized to identify if a change can potentially contribute to a hazardous situation. The work estimate is updated. There shall be an agreed procedure to approve a change

request for implementation. This can be carried out, e.g., by setting a target release by an authorized party. [4]

### 2.2.5 Modification implementation

All modification implementations are based on approved changes. These changes are defects, anomalies or user stories of new or changing features or functionalities. The software development process will be applied when designing, implementing, integrating, and verifying the software.

Different methods can be used for an architecture specification, a user interface design, and a detailed design, depending on the scale and complexity of the change. The software development or maintenance process shall define which documents shall be created and maintained.

Software units are developed according to the specifications. It is good practice to review the code changes with peer developers before progressing further. There shall be a unit verification process which defines methods, procedures and acceptance criteria for software unit verification. Conducting a unit test is one approach to verify the software units. Unit test cases and procedure shall be evaluated for correctness before they are approved as an accepted verification method. The software units shall fulfill the acceptance criteria before they are integrated to the software.

The accepted software units are integrated to the software system. Software integration is verified to ensure that the accepted changes have been integrated to the software as planned. The software shall be integration-tested with verified procedures to ensure that the software item performs as intended. Regression testing shall be performed to ensure there are no defects introduced. The risk control measures shall be verified.

The problem resolution process shall be utilized for anomalies found during the testing. The test report shall include information about a) person who executed tests b) test results c) list of anomalies d) information needed to repeat the tests.

All changes to user interface texts are translated to supported languages and developers are updating translations to the software. Similar to other changes, there shall be a

method to verify the user interfaces after the translation changes to ensure texts are correct and shown properly in the user interfaces.

Software configuration management and version control shall be performed systematically. Software configuration shall be identified and version controlled. All changes to the software shall be traceable to the approved change request and accordingly each software change shall be traceable to a requirement or to a defect. The configuration shall also include SOUP configuration items. This covers all third party software, including standard libraries, and documentation.

The maintenance of a software system has some additional challenges compared to the situation where the software is developed for the first time from scratch. All existing devices in the field shall be taken into account. There can be several device or hardware versions having different versions of software. The software for a device shall work with all hardware versions or there shall be different versions of the software for the different hardware versions. The latter option runs the risk that effort needed to maintain several software versions in parallel increases over time. In addition, the software update process shall not add any unacceptable risk to the usage of a medical device.

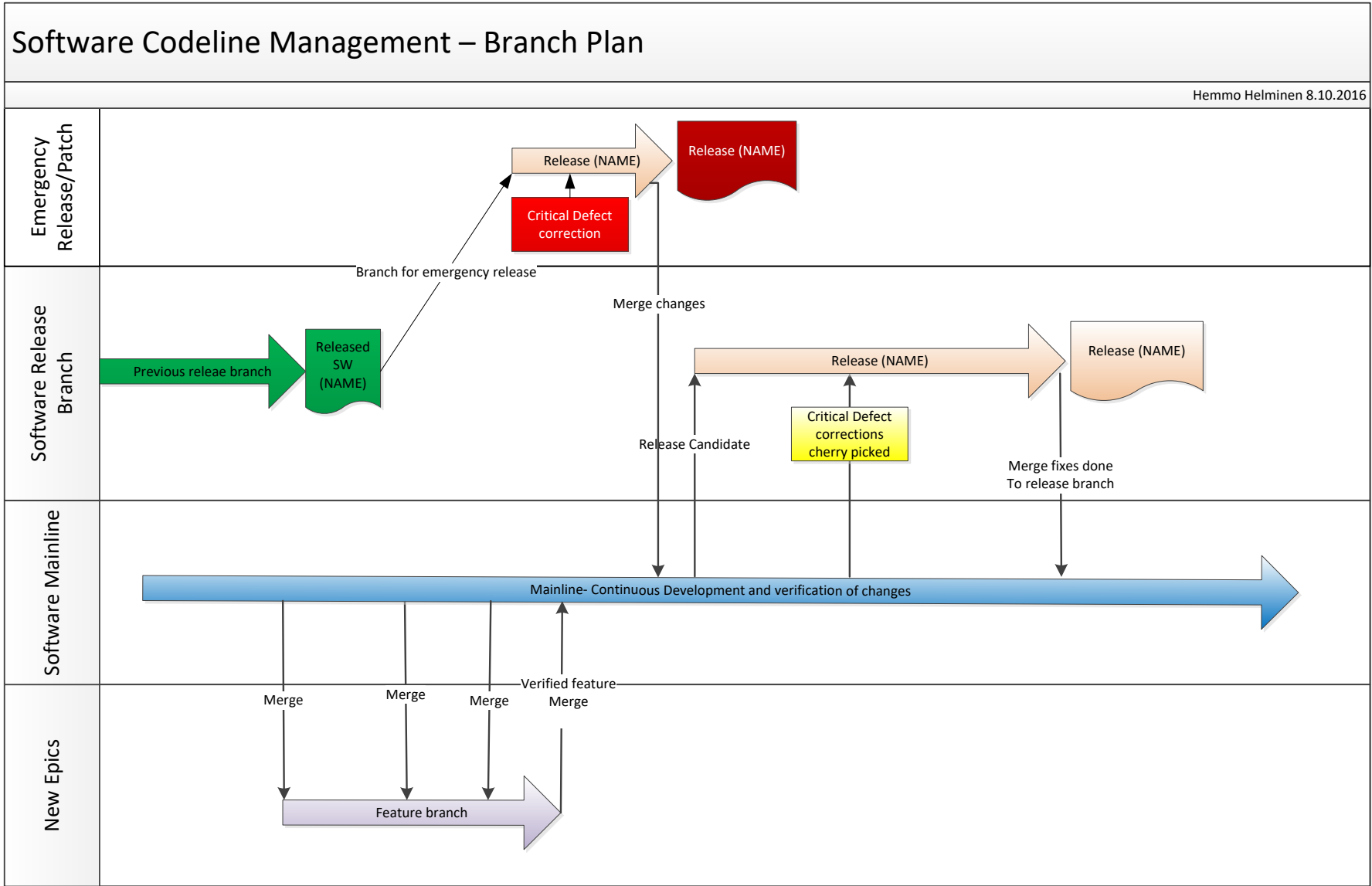
Picture 7 illustrates an example of the code line management in the maintenance phase. Changes are implemented in sprints and integrated to the software mainline in a continuous integration mode. Unit and integration testing are performed to ensure quality of the mainline. In the example, there is separate branch for new Epics, meaning that the new feature is developed in its own branch and only merged to the mainline once the feature is completely coded and passes tests.

Once all new features and changes are implemented, verified and acceptance tested in the mainline, a release candidate will be created. After the release candidate for the software is created, it shall be ensured that the software is isolated from changes. This means that if the new release candidate is needed, it includes only changes which are absolutely mandatory and have been formally accepted to be included in the software release. In the example, this is achieved by branching the software from the mainline to the release branch. In parallel, development for the later releases can continue in the mainline without adding regression risk to the software in the release branch.

In case there are changes implemented directly to the release branch, it is an extremely important task to merge the changes to the mainline. Otherwise, the coming releases may be missing an already implemented change.

The manufacturer shall have the codebase for all releases stored (without changes after releasing). These can be used as a basis if there is an urgent need to have a critical defect fixed or security patch deployed at short notice. [4]





Picture 7. Maintenance phase code line management example.

### 2.2.6 Quality assurance and releasing

The purpose of the quality assurance and releasing phase is to ensure that the modified software product is working as expected and can be re-released. Software system testing shall cover all software requirements. Testing can be combined to a single plan together with the integration testing. If there are software changes carried out during the system testing, tests shall be repeated, tests shall be modified or additional tests shall be added as seen appropriate. Similarly to the integration testing, the test report shall include information about a) person who executed test b) test results c) list of anomalies d) information needed to repeat the tests. Software system testing shall be verified to ensure that the test procedures are appropriate, the test cases can be traced to the requirements and all the software requirements are covered by the test cases. Accordingly, it shall be ensured that all the test cases have been executed and the results meet the required pass/fail criteria.

Before releasing the software following activities need to be completed:

- Completeness of all verification activities shall be verified
- Residual Anomalies shall be documented and evaluated
- Risk file shall be reviewed
- Software version documentation shall be updated
- Procedure and environment of software creation is documented

When all tasks have been completed, the release approval shall take place. The software development process shall define the roles and the acceptance criteria needed for the software approval. When all items are in acceptable level, the release is approved and the decision is recorded.

The software release along with the configuration items and documentation shall be archived for the life of the device or the time set by the regulators whichever is longer.

After software release has been approved, an engineering change note is created to indicate that the production shall start to use a new software in the production. It shall be ensured that software delivery to the production as well as to the customers is carried out without adding risk for corruption or unauthorized change. The software updates to a customer devices are started according to the software update plan. [4]

### 3 EVALUATING THE WORK METHODS

The Evondos software development processes and work methods are evaluated to understand the current status. Different sources such as self-observation, existing documentation, sprint and release reviews and retrospectives as well as interviews are used. Evaluation is divided in to two approaches. The first approach is to study the development work for one software release thoroughly to find concrete evidences for the challenges. The second approach is to evaluate processes and work methods against IEC 62304 (Medical device software – software life cycle processes) [4].

#### 3.1 Study of a release

Study of release is done for the 1708-release because of the following reasons:

- It was the latest released software.
- There had been an extensive delay in releasing.
- The software had been deployed and already updated to big portion of dispensers in the field.

Initially planned schedule for the release was mid November 2016, but was postponed at early state to be released on December due to the changes in the content. To the customers it was communicated that a software release with certain important quality improvements will be available by the end of the year 2016. In the beginning of December, there was high pressure to start the release testing in order to meet December releasing target, thus it was decided to create the release candidate and start the release testing even there was still some content missing. First release candidate was created at fifth of December and the release testing was started. Aim was to add the missing content as soon as possible, perform the release testing in three weeks and fix possible found defects and release the software before the end of the year. Actual progress did not correspond with the expectation. After 13 release candidates, the software was released on 23<sup>rd</sup> of February 2017. Figure 1 illustrates the progress from the release candidate point of view.

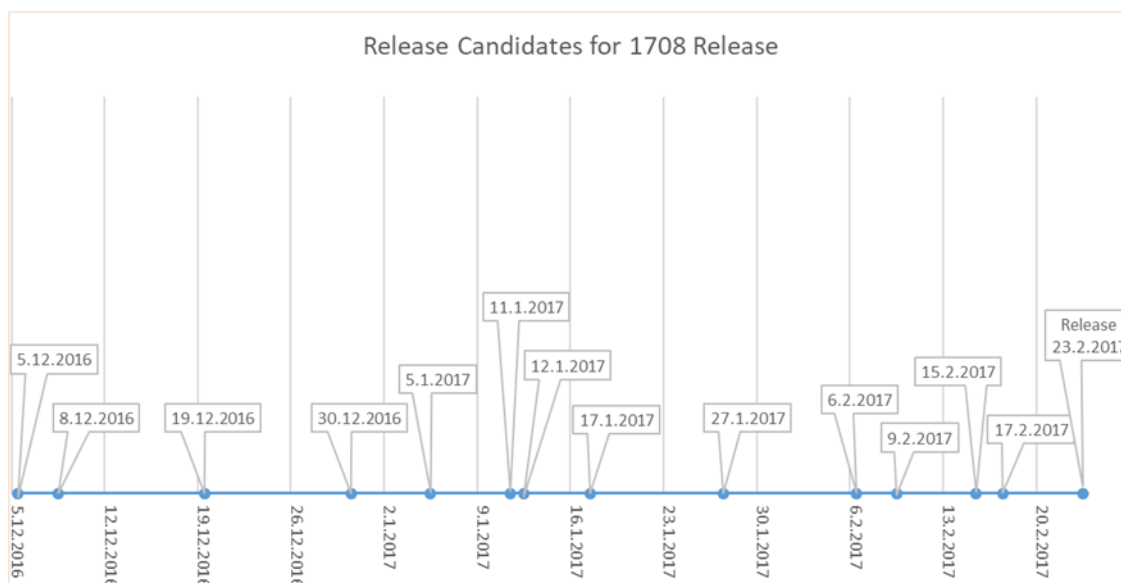


Figure 1. Release candidates for 1708 release.

During the release testing period, decisions to create a new release candidate was made based on the committed content and testing status with previous release candidate.

### 3.1.1 Decision to start release testing

Release candidate is a software build, which is expected to be ready for releasing to the market. All new features and changes should have been implemented, verified and acceptance tested. All bug fixes should have been verified.

Purpose of the release testing is to ensure that all requirements are met and to detect possible regression caused by changes to the software i.e. changes have not broken any part of the functionality or caused any unexpected side effects and furthermore cause a risk for the care continuity and the customer health.

Since the complexity of the robot and fact that the patient health shall not be compromised due to the malfunction of the robot, relatively extensive regression testing is required to minimize the possibility of unexpected behavior. For the 1708-release, the estimated release testing time was three weeks with the resources available.

Verification for the already implemented features and changes had been completed. There was known missing content: Part of the history validation functionality, Danish

translations and Irish Speech themes. In addition, there were ongoing discussion about five major functional changes but no decision to include any to the release.

The missing content was analyzed and concluded that the implementation would be ready within two weeks. Also concluded that the release testing could be started with the tests cases which would not be impacted even the defined content would be added after the release test start. Based on the information available, decision to create the release candidate and start the release testing was made. The release candidate was built on 5th December 2016.

### 3.1.2 Changes during release testing

Work estimates for the five change proposals were reviewed on the 8th of December and it was decided to implement the proposed patient UI changes to the release. The other four change requests were postponed to the future releases.

To understand all the changes taken into the software during the release-testing period, changes were listed and analysed. Software change information is available in the Git version control system. Commits from release branch was printed to text file with git log >> commits.txt command. Text file was imported to Excel and Excel pivot was used to create Figure 2 chart to illustrate software changes in a calendar time.

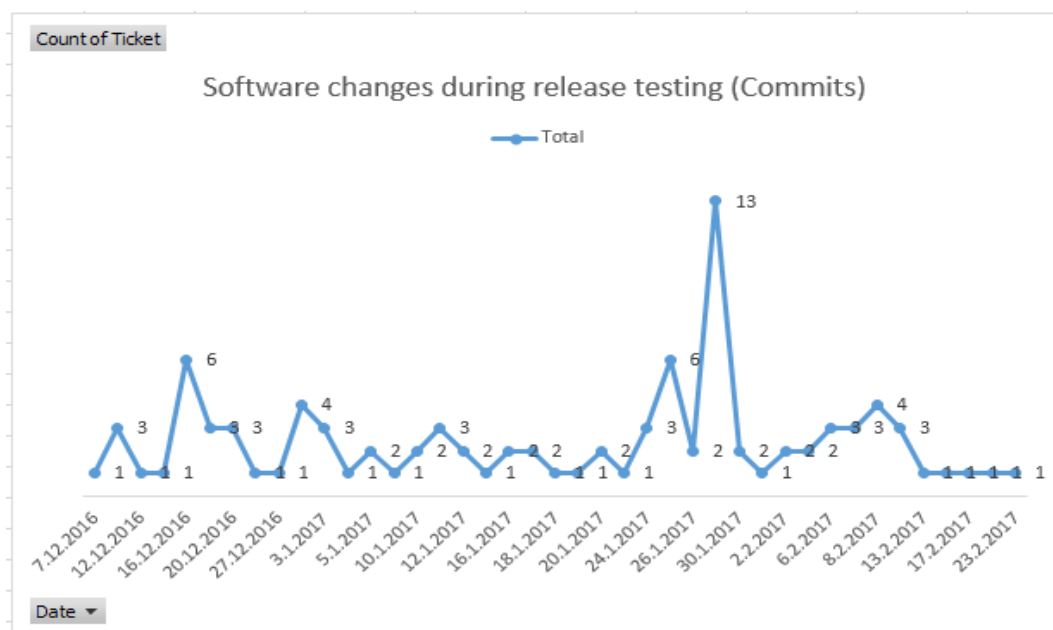
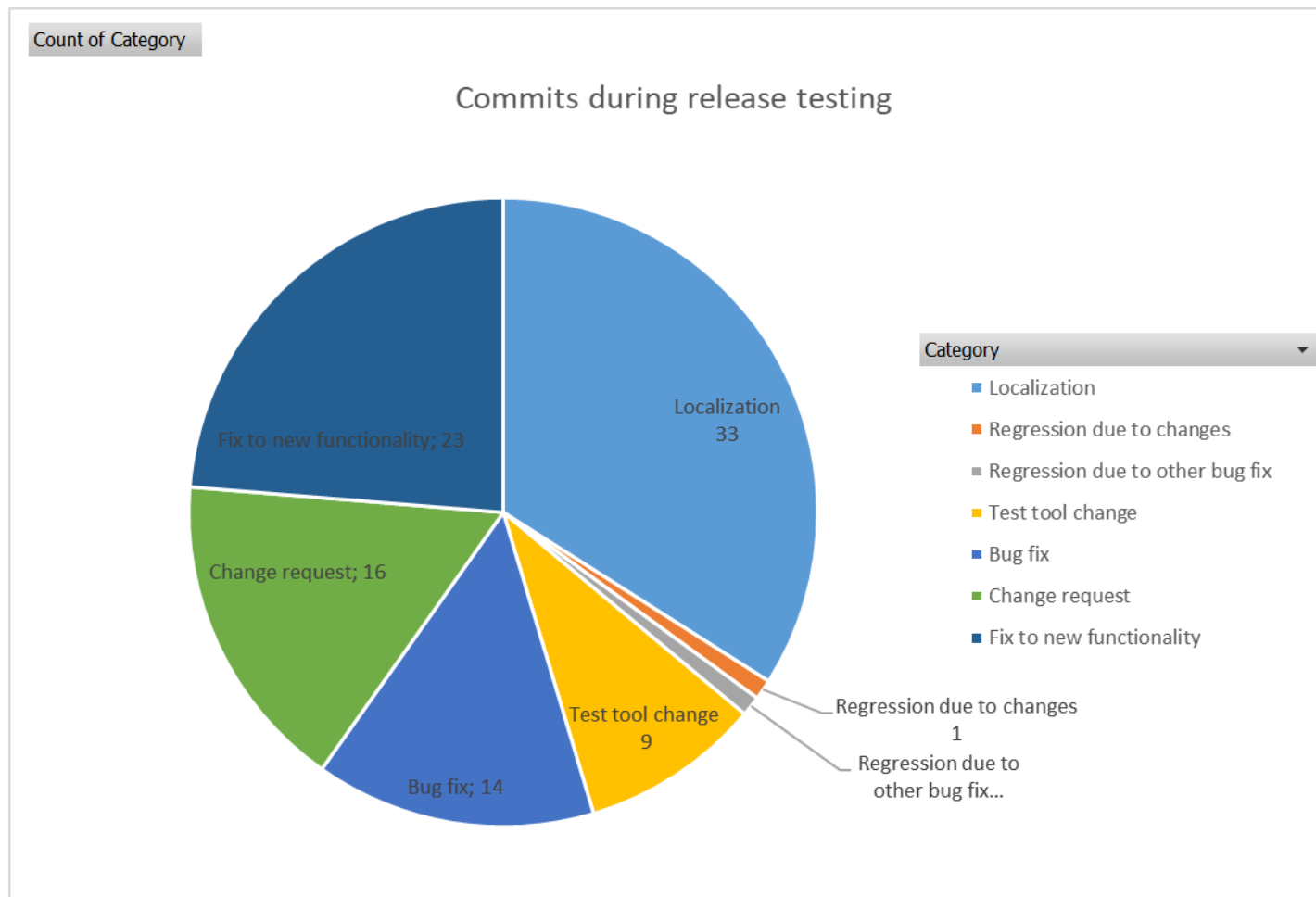


Figure 2. Software changes during release testing.

Altogether, there were 97 changes. In the Picture 8 changes are divided into the categories to understand the portion of different type of changes. It is visible that there are four dominating categories. The changes due to the change requests to the functionality (16 pcs), bug fixes (14 pcs), localization changes (33 pcs) and fixes to the new functionality (23 pcs).

The fixes to new functionality means that in the release testing it had been found that implemented functionality did not behave as initially expected or found that specification was poor and implemented functionality did not fulfil the need. Latter one means that a functionality is implemented according the original requirement, but when the functionality is demoed it has become evident that the requirement shall be changed to make the functionality more feasible for the customers.



Picture 8. Software changes divided into categories.

Changes due to change requests are listed in table 2.

Table 3. Changes after first release candidate.

Date of change	Changed functionality
7.12.2016	Medication history validation
8.12.2016	Irish speech themes
8.12.2016	Medication history validation
16.12.2016	Medication history validation
19.12.2016	Medication history validation
3.1.2017	Medication history validation
4.1.2017	New camera lens support
5.1.2017	New camera lens support
11.1.2017	Patient UI renewal
12.1.2017	New camera lens support
25.1.2017	Medication history validation
25.1.2017	Medication history validation
25.1.2017	Medication history validation
27.1.2017	Medication history validation
1.2.2017	Correction to English speech themes
3.2.2017	Unload date/time inserting change

It was known that some changes are required, even the release testing was already started. The Irish speech themes highlighted in table 2 were included to the content as expected. Implementation of additions and changes to the history validation functionality took much longer than expected. In addition, there were late business decision to add support for new camera lens, renew the patient user interface and change the date and time inserting logic for the unload use case. In addition, a small change to the terminology in the English speech theme was agreed.

Localization corrections and changes are listed separately since it became evident that they form a big individual problem area. The localization related changes caused much more work than initially expected. The missing Danish translations were added as expected, but the patient user interface changes and other localization related findings caused lots of work and was one big contributor for the extra delay. Amount of localization changes are listed in Table 4.



Table 4. Localization changes.

Date of change	Changed functionality
<b>Date of change</b>	<b>Changed functionality</b>
2016-12-08	Localization change
2016-12-12	Localization change (Danish translations)
2016-12-16	Localization change
2016-12-16	Localization change
2016-12-16	Localization change
2016-12-16	Localization change
2016-12-20	Localization change
2016-12-20	Localization change
2016-12-20	Localization change
2016-12-27	Localization change
2016-12-30	Localization change
2017-01-03	Localization change
2017-01-10	Localization change
2017-01-17	Localization change
2017-01-24	Localization change
26.1.2017	Localization change
26.1.2017	Localization change
27.1.2017	Localization change
27.1.2017	Localization change
27.1.2017	Localization change
27.1.2017	Localization change
30.1.2017	Localization change
30.1.2017	Localization change
2.2.2017	Localization change
6.2.2017	Localization change
8.2.2017	Localization change
8.2.2017	Localization change
9.2.2017	Localization change
9.2.2017	Localization change
15.2.2017	Localization change
17.2.2017	Localization change
21.2.2017	Localization change
23.2.2017	Localization change

In addition, Picture 8 includes nine change for test tools. The test tool changes are not changes to actual robot software instead those are changes to test scripts or manufacturing tools used to configure and verify the robot in the factory. The tool changes are visible in the commits because tools are maintained and version controlled together with the robot software.

### 3.1.3 Other findings

The developers had many simultaneous responsibilities disturbing the development work. The development work was interrupted due to the request coming from Customer support, Production support and Hardware team.

In addition, following problems were identified:

- Many tasks are dedicated to certain developer, it is not possible to share work
- Same developers are taking care of tools and development environments
- No back-up persons, during the absence tasks will be on hold
- The team is not co-located, which is making communication more challenging
- Many various and simultaneous tasks are ongoing
- Handling tickets in the parallel ticketing system is time consuming
- Team velocity is not known and used, the current Trac implementation does not support agile estimation

### 3.1.4 Release 1718 conclusions

The software team time was consumed to the following development and testing tasks during the release testing phase:

1. Implementation and testing of the new content
  - a. Support for the new camera lens
  - b. The patient user interface renewal
  - c. The date and time inserting logic for the unload use case
2. Implementing and testing of the localisation related changes and corrections
3. Implementing and testing of the history validation feature additions
4. Implementing and verifying of the bug fixes

The workload for above tasks together with the workload for all other tasks was the total workload. The total workload, team capability and velocity was not understood well enough to give reliable schedule estimate for the decision making.

### 3.2 Root cause analysis and improvement proposals based in 1708-release

In this chapter, the challenges identified in the 1708-release study are analysed further to understand root-causes. When the causes are known, actions for the improvements are identified. The improvement actions are mentioned in each chapter and in addition, listed in the Appendix 1.

#### 3.2.1 Testing

One of the four values in the Agile manifesto is to value responding to change over following a plan [9]. The value welcomes changes also in a late phase of the project. The Evondos integration and system testing process and capability does not support this value very well. There are hundreds of test cases and the test cases are not automated. The current work methods and testing capacity makes it difficult to respond late changes especially if big changes are happening in the release testing phase. In the analysis it was noticed that quite many bugs were found in the release testing. Most of these bugs should had been found before the release testing.

A nightmare scenario is that important testing is only performed at the end of a release or project, and if at that point a significant problem is found nothing can be done about it without extending timescale and costs [9].

Test automation along with process improvements should be introduced to improve the integration testing so that bugs are found earlier.

#### 3.2.2 New content: New camera lens

Support for new camera lens was required because a new lens type was replacing the old lens. The factory did have a buffer of lenses for a certain time, but estimated releasing date for the next release was so far in the future that it was not possible to postpone the lens support to the next release.

Better product management and requirement management processes should ensure that the requirements are added to the product backlog in an earlier phase. A shorter releasing cycle shall enable postponing similar late requirements to the next release.

### 3.2.3 New content: Patient user interface renewal

According to the customer feedback, the patient user interface in the robot was not optimal for the users. Improvement needs were documented and the new user interface was designed to address the feedback. The Company leadership decided that the change must be included in the release.

A better customer needs process should ensure that important requirements are added to the product backlog earlier and with higher priority. With a shorter releasing cycle and an easier software update procedure, late requirements could be postponed to the next release more easily.

### 3.2.4 New content: The date and time inserting logic

When the new functionality included in the release was demoed, it was found out that the date and time inserting logic for a last sachet was not aligned with the similar functionality in the other use cases. The modification of other use cases had been made to the release but this use case had been left out from the design. Due to the consistency, decision was to change the functionality as well.

The requirements analysis and software design phase should have more focus to take whole functionality and interfaces into account when modifying the software functionality and the user interfaces. Agile practises should be improved so that functionalities are demoed earlier to a larger audience. Development team collaboration and information sharing should be improved by introducing daily scrum meetings.

## 3.3 Localization

Localization for the 1708-release meant only translation related tasks since there was no a new market area nor a new language added. More than ten update rounds for the

translations in the 1708-release was performed. Brainstorming the localization work methods generated the following list of challenges:

- Responsibilities are unclear
- Varying methods to get feedback from the country organizations
- Cases where the translated text does not fit to a user interface component
- Cases where translator does not understand the context and a wrong translation is used
- Source texts are not up to date, causing that the English translation file must be used instead
- The QT-translation files are difficult to use since QT-linguistics tools is needed
- Different people have different opinions for wording and it is not clear who will do the final decision
- Risk that the wording is changed only for certain UI's and not aligned with other similar UI's
- Texts started to variate between different languages i.e. meaning of the phrases not aligned between the languages
- Country organizations are proposing fundamental changes within translations
- Mistakes when modifying the translations files with translation tool (QT-Linguistics)
- Difficult for the test team to generate the UI screenshots with all possible screen and text combinations

Responsibilities between different roles should be clarified. There should be a common way to collect feedback from the country teams. The user interface pictures should be available for the translators. The source texts should be updated. There should be a vocabulary for terms used in the Evondos system. Summarizing all above together, the whole localization process should be re-designed and documented.

### 3.4 History validation feature

The implementation of the medication history validation feature lasted eight weeks instead of initially estimated two weeks. The history validation feature had been specified in September and implementation started in October. The implementation was completed in the end of November and reviewed in the 2nd of December. In the review, it was found that the original requirements had not been complete and all use cases had not been taken into account.

More focus to the requirements analysis and software design is needed to ensure that the overall functionality and the interfaces are taken into account when modifying the software functionality and the user interfaces. Since it was difficult or even impossible to demo the feature after the sprints, the design documentation could have been used to explain functionality after each sprint. A better template for functional and design specifications should be created.

### 3.5 Development team resourcing

In the small team with dedicated competence areas a dynamic work allocation and sharing of tasks was found to be difficult. An increasing and a varying workload coming outside of the development project caused un-planned interruptions and delays in the development work.

The overall capacity of the team should be better aligned with the increasing workload. Additional resources would enable a better competence sharing and improve the team productivity. A shorter releasing cycle and an easier software update procedure would enable quicker deployment of bug fixes, which would decrease problem investigation and debugging workload accordingly.

### 3.6 Review of medical device Software maintenance process

The Evondos software maintenance process was reviewed against the Checklist for small companies without a certified QMS, Picture 9. Checklist is available in the Annex D of the Medical device software life cycle processes [4].

Checklist was created to Microsoft Excel where it was easy to record evidences and deviations. Each activity in the checklist was analyzed from two perspectives. Firstly, how the activity is described in the documentation and secondly how the actual work method follows the standard.

According to the analysis, activities defined in the standards seems to be followed well in the work methods and daily practices. Found deviations were mainly related to the documentation of the responsibilities and procedures. A serious deviation, which would

require immediate actions, was not identified during the analysis. However, there were some findings which are agreed to be corrected by the end of first half of the year 2019. A better documentation of the responsibilities and activities regarding the customer feedback collection was agreed to be the first improvement.

ACTIVITY	Related clause of ISO 13485:2003	Covered by existing procedure?	If yes: Reference	Actions to be taken
5.1 Software development planning	7.3.1 Design and development planning	Yes/No		
5.2 Software requirements analysis	7.3.2 Design and development inputs	Yes/No		
5.3 Software ARCHITECTURAL design		Yes/No		
5.4 Software detailed design		Yes/No		
5.5 SOFTWARE UNIT implementation and verification		Yes/No		
5.6 Software integration and integration testing		Yes/No		
5.7 SOFTWARE SYSTEM testing	7.3.3 Design and development outputs 7.3.4 Design and development review	Yes/No		
5.8 Software release	7.3.5 Design and development verification 7.3.6 Design and development validation	Yes/No		
6.1 Establish software maintenance plan	7.3.7 Control of design and development changes	Yes/No		
6.2 Problem and modification analysis		Yes/No		
6.3 Modification implementation	7.3.5 Design and development verification 7.3.6 Design and development validation	Yes/No		
7.1 Analysis of software contributing to hazardous situations		Yes/No		
7.2 RISK CONTROL measures		Yes/No		
7.3 VERIFICATION of RISK CONTROL measures		Yes/No		
7.4 RISK MANAGEMENT of software changes		Yes/No		
8.1 Configuration identification	7.5.3 Identification and traceability	Yes/No		
8.2 Change control	7.5.3 Identification and traceability	Yes/No		
8.3 Configuration status accounting		Yes/No		
9 Software problem resolution PROCESS		Yes/No		

Picture 9. Checklist for small companies without a certified QMS [4].

## 4 IMPROVEMENTS

This chapter concludes the proposed improvements. All improvement proposals are listed in the Appendix 1.

### 4.1 Localization procedures

Localization process definition was the largest effort of the thesis. For the localization procedures, there were no documentation describing what are activities in the process, who is expected to perform tasks and when an activity (in which order) should be done. Exact method for designing and describing the process was open. Internet and different books were examined for an advice about the best approach. Methodology described in the Workflow Modelling Tools for Process Improvement and Applications Development book [8] was selected. The work for the process improvement was organized in three phases as described in the book:

1. Establish process context, scope, and goals —includes identifying a set of related business processes, and for each, clarifying its boundaries, contents, and some aspects of the current implementation, performing an initial assessment, and setting to-be goals.
2. Understand the current (as-is) process —includes modelling its workflow and making initial observations on factors impacting process performance;
3. Design the new (to-be) process —includes finalizing an assessment of the process, devising and assessing potential improvements, selecting which changes (improvements) will be made, defining the important characteristics of the to-be process required to implement the changes, and designing the new workflow [8]

#### 4.1.1 Phase 1: Process discovery – What, Why and How

The starting point for the process development work is to understand the sole purpose and aim of the process. The purpose of the process is to describe the localization related procedures and responsibilities for the Evondos® service localizations. The process is named as Localize Service. The scope of the process is to, translate, implement, verify and release **software related items** which require localization. Current list of items:



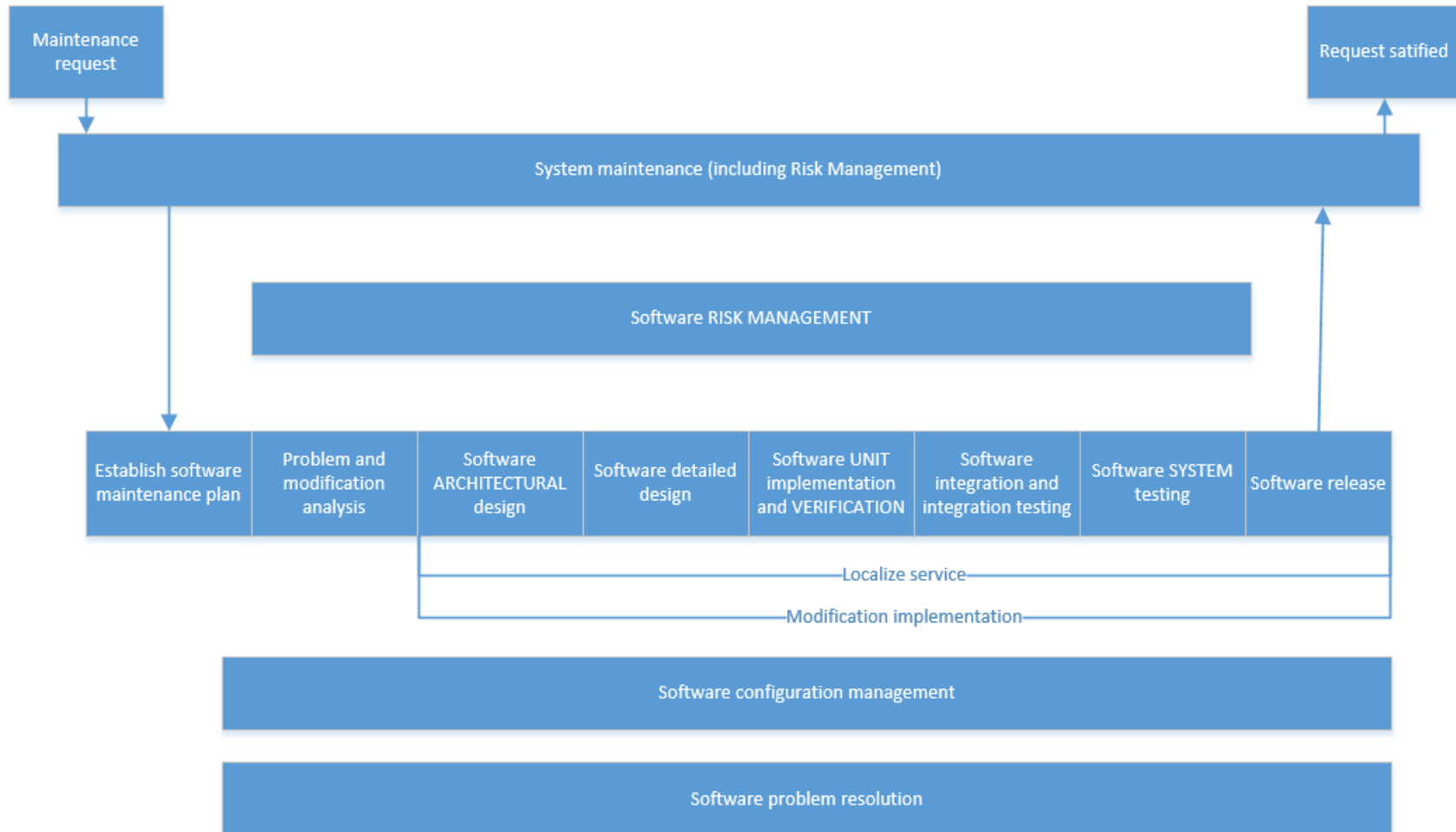
- Texts visible in the user interfaces
- Service voice messages in the robot
- E-mail messages
- SMS-messages
- Call voice messages

User guides and hardware related items are not in the scope of the process. Currently Finnish, English, Norwegian Bokmål, Norwegian Nynorsk, Swedish and Danish are the supported languages.

Above described purpose and scope was added as first chapter to the process description document.

The Localise service process is a sub process of the modification implementation in the software lifecycle processes as illustrated in the Picture 10.

As an input to the localize service process is the outcome of the Problem and modification analysis process. These outcomes are either problem reports or change requests. Outcomes of the localize service process are localized software components included in the software release. Exception is the call voice messages, which are not part of the software release and are updated separately to the voice call system.



Picture 10. Localizing the service process in the software lifecycle process

### 4.1.2 Phase 2: Understanding the As-Is Process

Purpose of the understanding the As-Is Process phase is to understand and describe the current flow of the work. As an end result there shall be a workflow diagram describing the activities and the roles performing the activities.

Instructions and examples in the book [8] proposed to draw swim lane picture of the process. In the swim lane diagram, you have both roles and activities in the same diagram. Roles are swim lanes and activities that a role is performing are drawn inside the swim lane. Activities are in chronological order from right to left. Example in the Figure 3. Workflow illustrated as swim lane diagram [8].

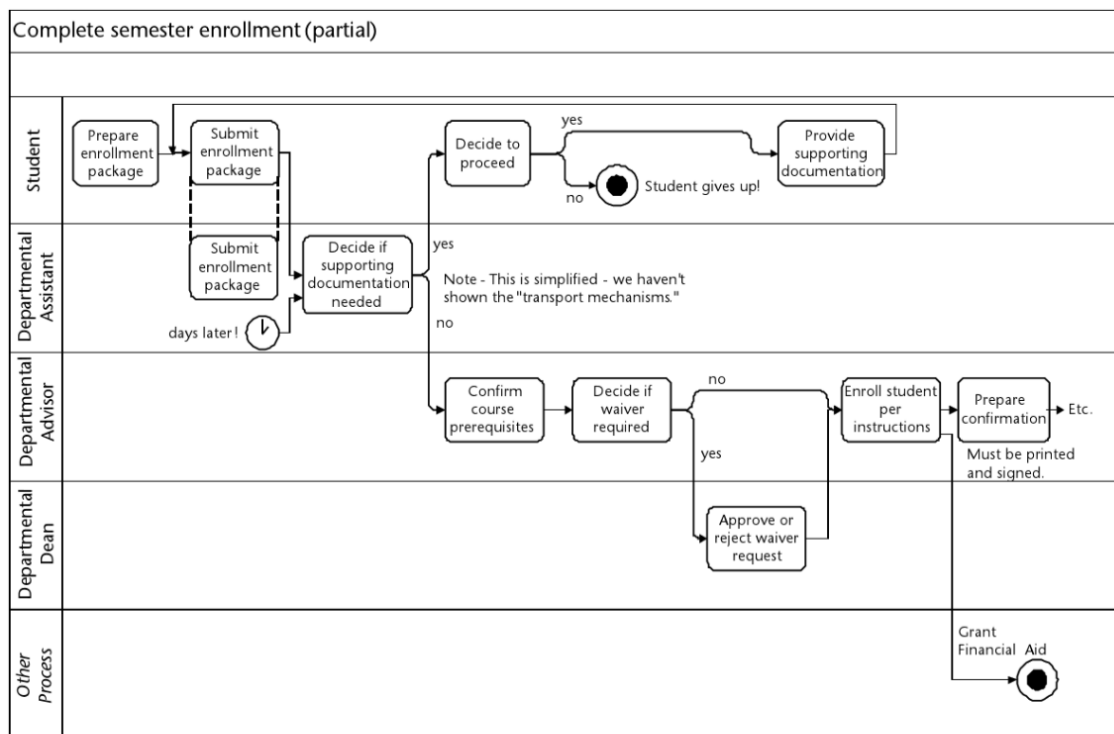


Figure 8.1 A simple swimlane diagram.

Figure 3. Workflow illustrated as swim lane diagram [8].

While drawing the As-is process it became evident that it was difficult or even impossible to use the swim lane diagram. One of the problem identified in the root cause analysis, was the unclear roles. When trying to draw the current workflow to the swim lanes it was impossible since one-day certain task had been performed by Service Project Manager and next time it had been Software Manager or Development Engineer.

Instead of a swim lane diagram, activities were drawn in the simple workflow-chart where the activities were listed in the chronological order but the roles were not defined for the activities. Figure 4. As-Is process for changing existing text illustrates how the As-Is state was described.

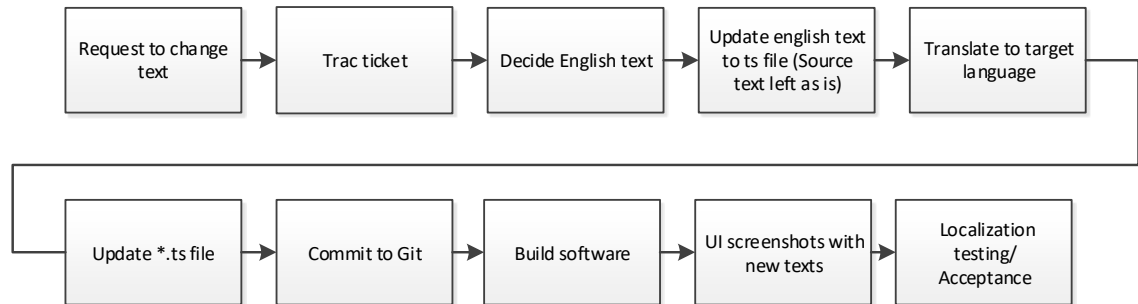


Figure 4. As-Is process for changing existing text.

As end results there were number of workflow diagrams describing the activities for different type of changes. Type of change here means that e.g. only translation for a language is corrected whereas other type is that a new user interface is added and all languages as well as the source text needs to be defined. In addition, there were separate workflows for the Robot and the Telecare system. To be noted that as a deviation to the original target, roles were not visible in the diagrams.

#### 4.1.3 Phase 3: Designing the To-Be Process

As an input to the To-Be process, we had: 1. The As-Is process workflow-diagrams 2. The list of challenges identified during the assessment of the current state 3. Out of the box thinking and willingness to improve.

The work proceeded by adding new activities to the flowcharts and by modifying the existing activities to address the problems identified earlier. For example, UI-design creation and delivering them to the translator were added to the process along with many other tasks. As a result, there was flowchart pictures describing the activities needed to run the process perfectly through. Results were reviewed a couple of times and after each review updated based on the review feedback. Few weeks' later it seemed that work is not progressing. In the review meetings different and partly conflicting feedback was given comparing to the previous reviews. The question which raised was that why

there were so many changes requested to the process steps which were previously agreed.

Maybe it was about the too detailed flowchart, some activities could easily be abstracted to higher level tasks and still maintaining the needed level for process description. Then about the roles, while designing the process, it had become clear which role should do which task, I thought. But the roles were not visible in the process pictures, the basic flowcharts were used, still. It would be good idea to use the swim lanes instead of the basic flowchart, but there was hesitation since amount of work would be rather big. In spite of the work amount the work proceeded by refactoring the process pictures to the swim lane format and at the same time raising the abstraction level for some of the tasks.

While drawing workflows with the swim lanes, major improvements were found when the roles and workflow between the roles was visible. For example, there was task where role was working only as a middle-man between two roles without adding any value, only causing possible delay. Some tasks were abstracted to a higher level tasks. The problem list which was created in the root-cause analysis was reviewed to ensure that all the localization process related problems had been taken into account in the process design.

The process description includes altogether eight different process workflows. Three workflows for the robot UI-texts, three workflows for the Telecare system UIs' and one for both the robot voice messages and the voice call messages. The e-mail and the SMS-message localizations are following the Telecare system UI-process as they are originating from the Telecare system.

## 4.2 Implementation of other improvements

In addition to the localization process improvement there were several other improvements defined.

### 4.2.1 Shortening the release cycle

Shortening the time between software maintenance releases has been set as a target. To achieve the target, improvements are required to the methodologies development team is using as well as improvement in the software update is delivery to the customers.

Enablers for shortening the release cycle are progressing. Test automation ramp-up has been started and target is to increase automation level significantly by the end of the year 2019.

To improve software update procedure, project has been initiated. As an outcome the remote software update functionality is expected to be available during the year 2019.

#### 4.2.2 Customer feedback

Importance of the customer feedback process as well as related tooling was highlighted to the key stakeholders. The customer feedback processing and decision making has been improved by the regular feedback reviews. Documentation of the feedback process and responsibilities has been proposed.

#### 4.2.3 Requirements management

Importance of the Requirements management process improvement as well as related tooling highlighted to the key stakeholders. Regular Product Management meetings has been started. Requirement management tool decision is still pending.

#### 4.2.4 Software development

A daily meeting practise has been started in cadence of two meetings per week in the robot software development and everyday meetings for the Telecare system software. Peer reviews has been added to the common practise. The robot software ticketing system renewal has been added to the targets for the year 2019.

Telecare system ticketing was moved to the Jira. The Telecare system existing tickets were closed in the Trac. To maintain the traceability, the tickets in the Trac were manually linked to the Jira tickets by adding the Jira ticket number as reference. By implementing this change, software lead workload decreased and unclarity due to not synched information disappeared.

#### 4.2.5 Resourcing

Development team resourcing was studied. Based on the identified competence and resource gaps, two scenarios for the resource plan was created. Implementation is waiting for the financial decision which will also set basis for the scenario selection.

## 5 CONCLUSION

Target for this thesis was to improve the software maintenance processes and work methods in the Evondos software development. The Evondos software development was thoroughly analyzed from different perspectives. The analysis highlighted the challenges the software team is facing when creating the maintenance releases for the robot software. Major problems in the high level were: the additional content close to the releasing target date, the team capability to react to the changes, and big portion of the important testing left to the late phase of the release project.

The additional content was added mainly by two reasons. First, the requirement management process was not optimal, causing that the important features did not get high enough priority in the product backlog. Second, when the importance of the features was understood the long releasing cycle caused that there was high pressure to get the changes included to the ongoing release content. As a solution, the requirements management process was improved by introducing the regular product management meeting, and activities for the enablers to shorten the releasing cycle were planned and started.

When the team capability was studied further, several reasons was found. Disordered localization activities were causing huge amount of extra work. The responsibilities and tasks in the localization area were clarified by introducing a dedicated process for the localization activities. The software development team daily practices were improved e.g. by improving the team member's collaboration between each other. The team resourcing did not support a dynamic work allocation i.e. responsibilities were fixed, causing that the implementation was interrupted if there were any absences or any tasks coming outside of the release project. A resource plan proposals were created to improve the resourcing situation.

The study of the software maintenance process revealed that the big portion of bugs were found during the release testing, not before as it should. The existing work methods and testing capacity makes it difficult to increase test coverage earlier in the release project. There are hundreds of test cases and testing is not automated. Test automation ramp-up was started and target is to increase level of automation significantly during the year 2019.



### 5.1 What is the significance of the results?

Overall the targets of this thesis were achieved with good quality. The results of the thesis are expected to have a positive impact to the Evondos software development performance. The localize service process alone is expected to clarify the responsibilities and clearly improve the work methods in the software development. When tasks and responsibilities are clear, it is expected that the needed effort for the localization-related activities will decrease. As iterations needed for the localizations are expected to decrease, this will have a positive impact on many areas in the software development. Such areas are change and configuration management, modification implementation as well as testing and verification.

While some of the findings might seem self-evident, documenting the findings made them more visible for the decision makers. The results from this thesis will be followed and further developed as part of the Evondos software development normal procedures.

The scope of thesis was quite Evondos specific and thus the results are not expected to give great benefit for a larger audience.

### 5.2 Lessons learnt

As the software maintenance for medical device is a huge area overall, it was occasionally difficult to keep the scope of the thesis focused. Adding to that, the one and half year time span which was used for thesis work.

Due to long time, the software release which was analyzed in the thesis had already become fairly old at the time when this thesis was finalized. Two releases have been released after that, however most of the challenges, especially in the localization area have been similar.

As the localization related difficulties were visible from the start, localization was kept as the major topic throughout the thesis work. The localization process is bringing new activities which have not been performed or have only seldom been performed. One example is the creation of UI screenshots for translators. Creating the screenshot will take some effort, however, the expected improvement is significantly greater as it is expected

that with screenshots the translator is able to better understand the context where translation is used as well as see how much space there is for the translated text.

## REFERENCES

- [1] Evondos® Service Product Brief and Architecture Overview M-files ID 3642 version 4 31.7.2017
- [2] Data security in Evondos platform M-files ID 14674 version 16 31.7.2017
- [3] Evondos Software Development Process M-Files ID 8486 R:N\_177008000 version 13 18.3.2016
- [4] IEC 62304:2006 Medical device Software life cycle processes
- [5] Medical Device Directive (MDD) 93/42/EEC (latest amended by directive 2007/47/EC).
- [6] 2009-06-03 MDEG – 2009–12-01 MSOG Class I\_Guidance Rev. 1\_2009-06 Compliance and Enforcement group
- [7] SFS-EN ISO 14971 :2007 Application of risk management to medical devices
- [8] Workflow Modeling Tools for Process Improvement and Applications Development Second Edition Alec Sharp, Patrick McDermott ISBN-13: 978-1-59693-192-3 © 2009 ARTECH HOUSE, INC. 685 Canton Street Norwood, MA 02062
- [9] Peter Measey and Radtac. (2015) AGILE FOUNDATIONS Principles, practices and frameworks. ePUB ISBN: 978-1-78017-256-9.
- [10] ISO 13485:2012 Medical devices. Quality Management systems. Requirements for regulatory purposes.
- [11] IEC 62366-1:2015 Medical devices -- Part 1: Application of usability engineering to medical devices

## Identified improvements

Improvement	Actions
Localization procedures	<ul style="list-style-type: none"> <li>• Define localization process</li> <li>• Create Evondos vocabulary</li> <li>• Create UI picture library</li> <li>• Update source text</li> <li>• Implement automated user interface screen shot generation</li> </ul>
Customer feedback handling	<ul style="list-style-type: none"> <li>• Define customer feedback collection and responsibilities</li> <li>• Replace multiple files with one tool for feedback</li> </ul>
Shorten releasing cycle	<ul style="list-style-type: none"> <li>• Introduce remote software update capability</li> <li>• Implement test automation to shorten release testing</li> <li>• Study other improvements for testing</li> </ul>
Software development process	<ul style="list-style-type: none"> <li>• Introduce daily meetings</li> <li>• Peer reviews shall be added to the development process</li> <li>• Consolidate software ticketing to one tool by migrating Trac to Jira.</li> <li>• Improve communication towards stakeholders during the modification implementation</li> <li>• Create UI-design or mock-up every time when there is change to the user interface</li> <li>• Consider development team co-location</li> </ul>
Feedback process documentation	<ul style="list-style-type: none"> <li>• Document feedback process and responsibilities</li> </ul>
Requirement management	<ul style="list-style-type: none"> <li>• Improve requirement management process definition</li> <li>• Deploy tool for requirement management</li> </ul>
Specifications	<ul style="list-style-type: none"> <li>• Create functional specification template</li> <li>• Improve architecture specifications</li> <li>• Improve practise to update and review design documentation during the iterations</li> </ul>

System event data visualization	<ul style="list-style-type: none"><li>• Import data to Power BI and implement needed dashboards</li><li>• Logs files does not include enough information about activities performed in the UI</li></ul>
Defect management process	<ul style="list-style-type: none"><li>• Update severity classifications</li></ul>
Telecare system ticketing	<ul style="list-style-type: none"><li>• Use only Jira for the Telecare system tickets</li></ul>
Resourcing	<ul style="list-style-type: none"><li>• Update development team resource plan</li><li>• Nominate dedicated person to transform customer feedback to business cases and requirements</li></ul>