Application Lifecycle Management for Safety-Critical Software Development

Harri Honkanen

Master's Thesis
May 2015
Master's Degree Programme
in Information Technology

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

ABSTRACT

Tampere University of Applied Sciences
Master’s Degree Programme in Information Technology

Harri Honkanen:
Application Lifecycle Management for Safety-Critical Software Development

Master’s Thesis 68 pages
May 2015

Safety-critical software development imposes many requirements for the development methods and processes used. A lot of formal documents and design considerations have to be done. This results in longer development times and more expensive overall costs for software development projects. Having these requirements is still important because in the applications where the usage of safety-critical software development methods are required the possibility of failures causing major damage to people or material is high.

Application lifecycle management is an activity under which the whole development process from first idea to end of maintenance is governed. Application lifecycle management (or ALM for short) is a version of product lifecycle management concept specifically used for the management of software projects. ALM is typically realized in the form of software suite of tools for governing the requirements, design, progress tracking and testing of software project. Also defect tracking and user feedback can be included under its scope.

This thesis aims to apply the application lifecycle management concept to safety-critical software development. The aim is to introduce an ALM solution to Rocla software development in as agile and easy-to-use way as possible. The aim is to be able to fulfill the requirements of safety-critical software development of forklift trucks with minimum effort without compromising compliance with standards and regulations.

During this thesis the specific requirements from standard applying for Rocla software development are analyzed. Rocla currently has 2 different types of software projects and the possible solution should be able to fulfill the needs of both of these types. In the scope of this thesis those development processes are studied and several different possible solutions ALM software suites that could be used with both of them are reviewed. Several personnel at Rocla R&D were involved in this evaluation. At the end of this thesis a selection of ALM suite based on experiences gained during evaluation workshop is made and a pilot project of trying JIRA and its sister products is started.

Key words: Application lifecycle management, safety-critical software development, agile methods

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Master's degree programme in Information Technology

Harri Honkanen:
Ohjelmiston elinkaaren hallinta turvallisuuskriittisessä ohjelmistokehityksessä

Opinnäytetyö 68 sivua
Toukokuu 2015

Turvallisuuskriittisten ohjelmistojen kehittäminen asettaa monia vaatimuksen kehityksessä käytettäville menetelmille ja prosesseille. Kehityksen eri vaiheet on dokumentoitava tarkasti ja virallisesti ottaen huomioon erityisesti turvallisuusnäkökohdat. Tämä yleisesti tarkoittaa pidempää kehitysprojektien kesto ja huomattavasti kalliimpia kokonaiskustannuksia verrattuna normaaleihin ohjelmistokehityshankkeisiin. Näiden sääntöjen huomioon ottaminen ja noudattaminen on kuitenkin tärkeää koska turvallisuuskriittisten ohjelmistojen ongelmatilanteissa vahingot materiaalille ja laitteille puhumattakaan ihmishengistä voivat olla mittavat.

Ohjelmistojen elinkaaren hallinalla tarkoitetaan toimintaa joka kattaa koko ohjelmiston kehitysprosessin ensimmäisestä ideasta alkaen aina ohjelmiston ylläpidon lopettamiseen. Tyypillisesti ohjelmiston elinkaaren hallintaan käytetään jotain ohjelmistokokonaisuutta joka kattaa vaatimusten hallinnan, suunnitelmien hallinnan, työn edistymisen seurannan ja testauksen hallinnan koko ohjelmistoprojektille. Myös julkaistun ohjelmiston vikailmoitukset ja käyttäjäpalautteet yleensä kirjataan tällaiseen järjestelmään.

Tämän työn tarkoituksena on soveltaa ohjelmiston elinkaaren hallinta menetelmiä ja ratkaisuja turvallisuuskriittisen ohjelmistokehitykseen. Tavoittaa on tutkia eri vaihtoehtoja ohjelmiston elinkaaren hallinta järjestelmäksi ja valita jokin niistä käytettäväksi Rocla Oy:n ohjelmistokehitysprojekteihin. Työn tavoitteena on löytää ratkaisu jolla nämä vaatimukset voitaisiin täyttä mahdollisimman vähällä lisätyöllä verrattuna nykyisiin toimintatapoihin kutenkin täyttäen kaikki standardien määräykset.

Työssä analysoidaan Rocla Oy:n kehitykseltä vaadittavat seikat. Rocla Oy:ssä tehdään tällä hetkellä kahdentyyppisiä ohjelmistokehitysprojekteja ja mahdollisesti valittavan järjestelmän on kyettävä täyttämään näiden molempien erityiset vaatimukset. Useita eri ratkaisuja ohjelmiston elinkaaren hallinta järjestelmäksi tutkittiin useiden henkilöiden voimin. Tämän arvioinnin perusteella päädyttiin aloittamaan pilotointi projekti JIRA:N käyttöönotosta.

Avainsanat: ohjelmiston elinkaaren hallinta, turvallisuus kriittinen ohjelmistokehitys, ketterät menetelmät

FOREWORD

I want to thank Jani Mähönen from Rocla Oy for providing me with the opportunity to write this Master’s thesis and I want to thank all my colleagues for open-mindedness and enthusiasm in working towards the development of new and more efficient processes.

I also want to thank Erkki Hietalahti from Tampere University of Applied Sciences for feedback and guidance during the writing process.

Last but not least I want to thank Suvi for support during the writing process.

Tuusula, May 2015

Harri Honkanen

CONTENTS

1 INTRODUCTION 9
1.1 Customer 10
1.2 Problem Statement 10
1.3 Project Goals 11
2 SOFTWARE DEVELOPMENT MODELS 12
2.1 Waterfall model 12
2.2 The V-model 13
2.3 Agile software development 15
2.3.1 Overview 15
2.3.2 Iteration and feedback 16
2.3.3 Pitfalls of agile software development 18
2.3.4 Agile adaptation 19
3 APPLICATION LIFECYCLE MANAGEMENT 21
3.1 Overview 21
3.2 Key components 21
4 DEVELOPMENT OF SAFETY-CRITICAL SOFTWARE 25
4.1 Overview 25
4.2 SFS-EN ISO 13849-1 requirements for software development process 27
4.3 Requirements for software-based parameterisation of truck 32
5 IMPLEMENTATION 34
5.1 Overview 34
5.2 Current status 36
5.2.1 Software projects 36
5.2.2 Truck embedded software projects 39
5.3 Problems 40
5.3.1 New version of EN1175 and the adoption of EN ISO 13849-1 development standards for software development 40
5.3.2 Service tool development 41
5.3.3 Truck embedded software development 42
5.4 Improvements 43
5.4.1 Service tool development 44
5.4.2 Truck embedded software development 44
5.4.3 Conclusion 46
5.5 Options for application lifecycle management solution 46
5.5.1 Modifying Trac and Tarantula to fulfil new requirements 47

5.5.2 IBM Rational Doors ... 48
5.5.3 Polarion ... 49
5.5.4 Redmine ... 50
5.5.5 Microsoft Team Foundation Server ... 50
5.5.6 JIRA, Confluence and other products in Atlassian ALM tool family ... 50
5.5.7 Tool selection ... 52
5.6 Development process with Atlassian ALM solution ... 54
5.6.1 Using Confluence for requirements and design documents ... 54
5.6.2 JIRA for issue tracking ... 57
5.6.3 Test management with Zephyr plugin ... 59
5.6.4 Links between design artefacts and the big picture ... 60
6 CONCLUSIONS AND FUTURE WORK ... 63
6.1 Finalization of the selection of test management plugin ... 63
6.2 Automated report generation and storage in Aton ... 63
6.3 Using JIRA as general documentation tool and communication platform ... 64
6.4 Process and ALM re-evaluation after final version of EN1175 is published ... 64
6.5 Final word ... 65
REFERENCES ... 66

ABBREVIATIONS AND TERMS

ALM	Application Lifecycle Management
Aton PDM	Product data management system made by Modultek. Very popular in Finnish manufacturing companies.
CAN bus	Controller area network bus. Typically used in vehicle applications.
MCFE	Mitsubishi-Caterpillar Forklift Europe B.V.
MN	Mitsubishi-Nichiyu Forklift CO., LTD.
MSDN	Microsoft Developer Network. Website that provides resources in the form of documentation and downloads of Microsoft products. Special subscriptions are also available. Under these subscriptions developers can freely download and use a plethora of Microsoft tools. These tools can include, depending on the subscription level, e.g. Windows operating systems and different development tools.
PDM	Product data management system. Common acronym used for systems used for storage, version control and distribution of product data.
PL	Performance level. Performance level is measurement scale for safety-criticalness of a feature defined in EN ISO 13849-1. From least to most severe these levels are a, b, c, d and e. In documentation these levels are commonly referred with the abbreviation PL. (EN ISO 13849-1, 2008)
PLM	Product lifecycle management
PLr	Required performance level. See description of PL. PLr abbreviation is commonly used in standard documentation in context where a certain PL is required. (EN ISO 13849-1, 2008)
SaaS	Software-as-a-service. Common term when referring to software that is provided on subscription base and is accessible via internet. Typically also not installed on customer premises but in providers cloud.

SIL	Safety Integrity Level. Level used in some standards to imply the severity of requirement. Similar to PLr.
SLM	Software lifecycle management
SRP/CS	Safety-related part of a control system. This abbreviation is commonly used in standard documentation when talking about parts of control system. E.g. a single controller in distributed system. (EN ISO 13849-1, 2008)
TFS	Team Foundation Server. Common acronym used when talking about Microsoft ALM solution.

1 INTRODUCTION

Agile software development has been established as de-facto standard of software development for the whole industry. First implementations of agile emerged in the 90's which lead to publication of the Agile Manifesto in 2001 after which almost everyone has strived to change their methods toward more agile ways. One of the main concepts of agile development is to produce working software in small increments. In practice this means that the software has to be usable and working after implementing each feature or set of features. This enables testing and feedback on what to improve in very tight loops enabling quick responses to changes and customer needs. This is in contrast to old waterfall model where a working product is usually delivered so late in the project that no real changes to it can be done anymore. (Wikipedia, Agile software development, 2015)

Lifecycle of a software starts with initial project planning and ends when the customer stops using the product. In-between these points there are several different phases with possibly different resources working on the project. Without good process definition and working documentation this can cause a lot of extra work. Application lifecycle management (ALM) is an idea of handling and documenting the activities during the lifecycle of a product in an efficient and controlled way with predefined tool or set of tools. This idea has been driven by tool vendors who provide software suites that integrate all these activities under one product. (Kääriäinen, 2011)

In safety-critical software development the ALM is especially important due to the requirements imposed on the product by different standards and regulations. The safety of the product has to be proven with documented design requirements, specifications and then proven with test results against the initial requirements. In practice this traditionally means using waterfall model with monolithic documentation in different phases of project with complex and cumbersome software suites. In recent years this traditional model has been challenged and several companies have started studying and implementing agile methods in safety-critical software development to increase productivity. (Vuori, 2011)

This work examines how to combine the principles of agile methods, ALM and safety-critical software development.

1.1 Customer

This work is done for Rocla Oy which is a part of Mitsubishi-Caterpillar Forklift Europe B.V. (MCFE) which in turn is a part of global company Mitsubishi-Nichiyu Forklift CO. LTD. (MN). Rocla Oy designs and manufactures electric warehouse and counterbalance trucks for the European market. Rocla Oy also manufactures automated warehouse trucks which require no driver. These automated trucks are typically sold as a customized solution for specific customer.

Rocla Oy has a long history as a traditional machine manufacturer. Software has not been a major factor in truck design until last few decades. The importance of software first started to rise with the automatic trucks. In them the core technology is software. All new manually operated trucks are designed as fly-by-wire with software controlling all aspects of the truck. With software being a relatively new addition to truck design the processes and conventions with it are not very mature and most modern methodologies have not yet been fully realized with current truck safety standard not yet requiring much formality from the design processes as it does from testing.

1.2 Problem Statement

Currently there is no clearly defined process for software development due to relative newness of the role of the software. Improvements in productivity and quality can be made by defining a process based on proven methods and taking it into use in the multitude of software projects done. This is especially true in the future as the amount of software projects done is steadily increasing.

The old safety standard for battery powered trucks has no strict standards on software development but this is bound to change in a new version which is currently in drafting phase. Current drafts of the new standard implicate the introduction of requirements for software development from EN ISO 13849-1. This imposes severe restrictions on how firmware of the control units of the trucks is developed and how the service tools used to configure, program and diagnose those controllers is developed. (EN 1175-1:1998+A1, 2008. EN ISO 13849-1, 2008)

This work focuses on developing an agile process for development of service tools used in configuration, programming and diagnosing the control units of truck. These tools are typically either desktop software which connects to trucks via some peripheral or handheld devices used to directly connect to truck.

1.3 Project Goals

The goal of this work is to define a process and a set of tools to use in development of desktop software for truck service tools in compliance with relevant parts of EN ISO 13849-1. The process should follow the principles of agile software development for all the applicable parts. Having to comply with safety-critical software development standards imposes a lot of restrictions on the processes and activities done during development when comparing to current trends in agile software development where strict processes, gates, documentation etc. are being done less and less.

Also the suite of tools to use in the development shall be evaluated and initial set of which to use shall be defined in this work. Tool selections shall be done with ideology of ALM with a toolset covering all phases of development with as good integration as possible. Due to resource restrictions this process should also be as lean as possible and has to be adaptable with minimal training or purchasing costs.

2 SOFTWARE DEVELOPMENT MODELS

This chapter introduces a few commonly used software development models and explores their perceived advantages and weaknesses

2.1 Waterfall model

Waterfall model is the traditional method for software development where activities are done in phases from top down typically starting with requirement engineering phase. Before going to next phase there usually are some design criteria which have to be met. These criteria's can include documentation, management approval, test results etc. Waterfall is the first clearly defined software development method to emerge and the principles were first described by Herbert D. Benington in 1956. (Wikipedia, Waterfall model, 2015)

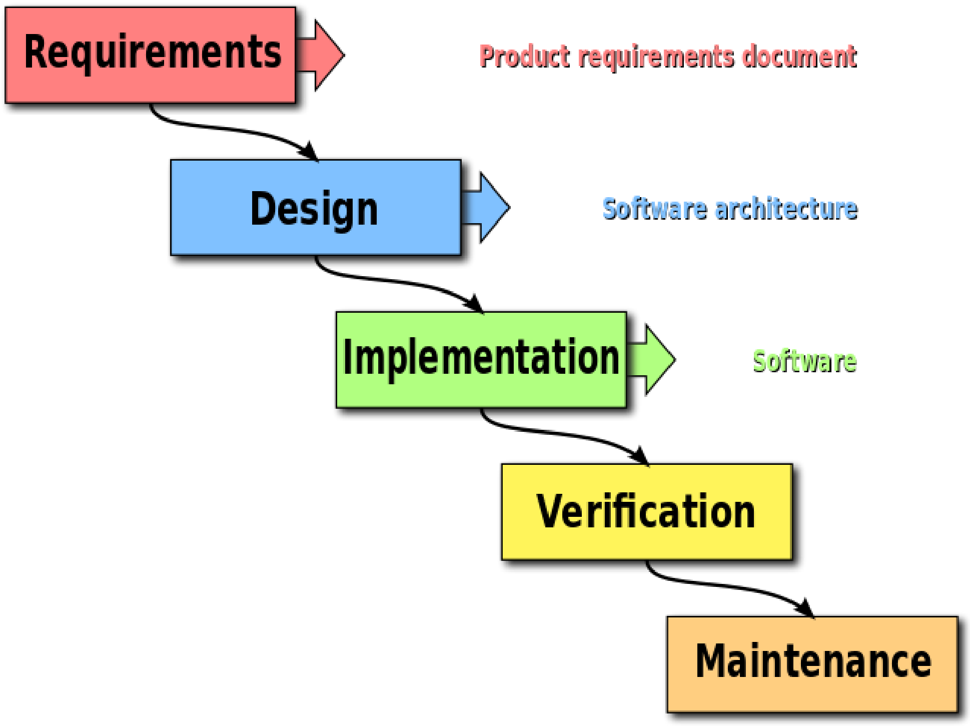


Figure 1. Waterfall model. (Wikipedia, Waterfall model, 2015)

The typical phases of waterfall model include:

- Requirement engineering
- Software architecture and module design

- Implementation
- Module, integration and system testing
- Maintenance

Advocating the use of waterfall model is that it causes development work to be done in a structured order with clear definition of what to be done in which order. Big part of the time used in the project is dedicated to creating requirements and software specifications documents before involving coding. The idea behind this is to do as much design up front as possible to minimize uncertainty and doing wrong things later in the project, mainly coding. Typically the cost of fixing a problem rises the longer it is left unattended. Also the big role of documentation imposed by this model is seen as good thing as typically software developers tend to create inadequate documentation which can lead to severe problems in maintenance phase of projects when the original developers are no longer available. (Wikipedia, Waterfall model, 2015)

Critics of the waterfall model state that due to the sequential nature of the model it cannot adapt to reality of changing requirements fast enough. In waterfall model a lot of work can be invested in implementing features before the end customer sees them. This can lead to a lot of wasted effort in such cases where the customer is not happy with the end product. Critics advocate having a process specifically aiming for constant change instead of rigid structure. (Wikipedia, Waterfall model, 2015)

2.2 The V-model

The V-model is an extension of waterfall model with some support for iterations built in. The V-model approach consists mainly of the same higher level activities as the waterfall model. In V-model each the initial design activities (left side of the V) has a corresponding verification action (right side of the V) with coding being at the lowest level of the V. Each verification phase then has a link back to design process to see that the design requirement has been fulfilled. This is a step towards iterative approach from the traditional waterfall model due to having the idea that some part of initial design will be modified during the development process. (Wikipedia, V-model (software development), 2015)

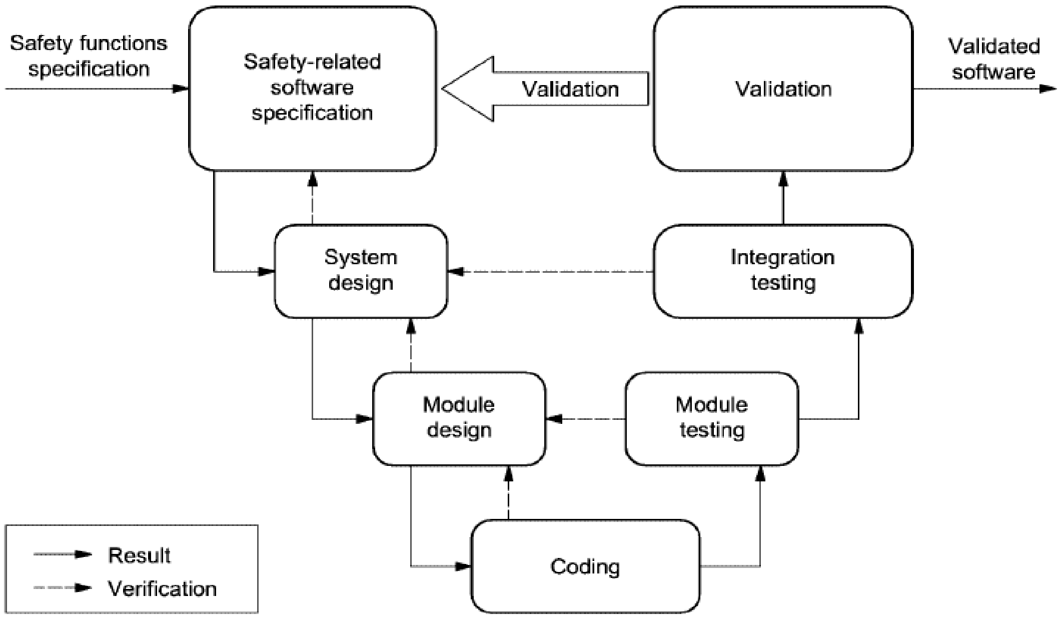


Figure 2. V-model. (EN ISO 13849-1, 2008)

The V-model is heavily in use due to being easy to understand by project management. It is also seen as having all the advantages of sequential and disciplined methodologies of the waterfall model with added agility and response to changing requirements. V-model is also very close to agile methods in the idea that things are iterated over in small blocks. In several cases it can be difficult to differentiate between the two models. (Wikipedia, V-model (software development), 2015)

The model is criticised partly due to the same reasons as waterfall model as being too rigid and still not agile enough to adapting changing requirements. It is seen as too simplified version of agile development with little correlation to how things are done in real life projects. It is also said that although the V-model has iterative methods in it, it fails to address the real problems early on. The first testing phases only include testing of low level software modules which are not yet presentable to end user for feedback. This can lead to it having the same problems as the waterfall model with spending a lot of time on work that can be subject to change once the customer gets to actually use the product and get a good understanding of what works and what does not. This can partly be countered by delivering mock-ups to customer for evaluation before the actual product is implemented. This is however approaching a grey area where it could be argued that it is more of an agile method versus traditional V-model way of doing things. (Wikipedia, V-model (software development), 2015)

2.3 Agile software development

This chapter dives in to the world of agile software development. Agile methodologies are explained and the most common things that are done wrong are explored. Also the status of current adaptation of agile methodologies in the industry is represented.

2.3.1 Overview

Agile software development is not a single model but instead is an ideology of doing software in a way that rapidly delivers working software for customer evaluation and then adapts the design based on the feedback received. Also the up-front design is in some cases minimized to a point where no requirement or specification documentation is done before coding. Documentation is minimized to only what is essential. This leads to customer getting working product in small increments which he can then review and request changes to it. This continues until the product is finished. (Wikipedia, Agile software development, 2015. Manifesto for Agile Software Development, 2015)

Agile development evolved overtime with people growing more and more frustrated with old models of development seeing them taking too much time away from delivering the customers what is important. Big change towards incremental development happened during the 90’s leading to publication of the “Manifesto for Agile Software Development” in 2001. The manifest outlined the ideology and key principles behind agile software development on which the modern variations are still based on. (Wikipedia, Agile software development, 2015. Manifesto for Agile Software Development, 2015)

2.3.2 Iteration and feedback

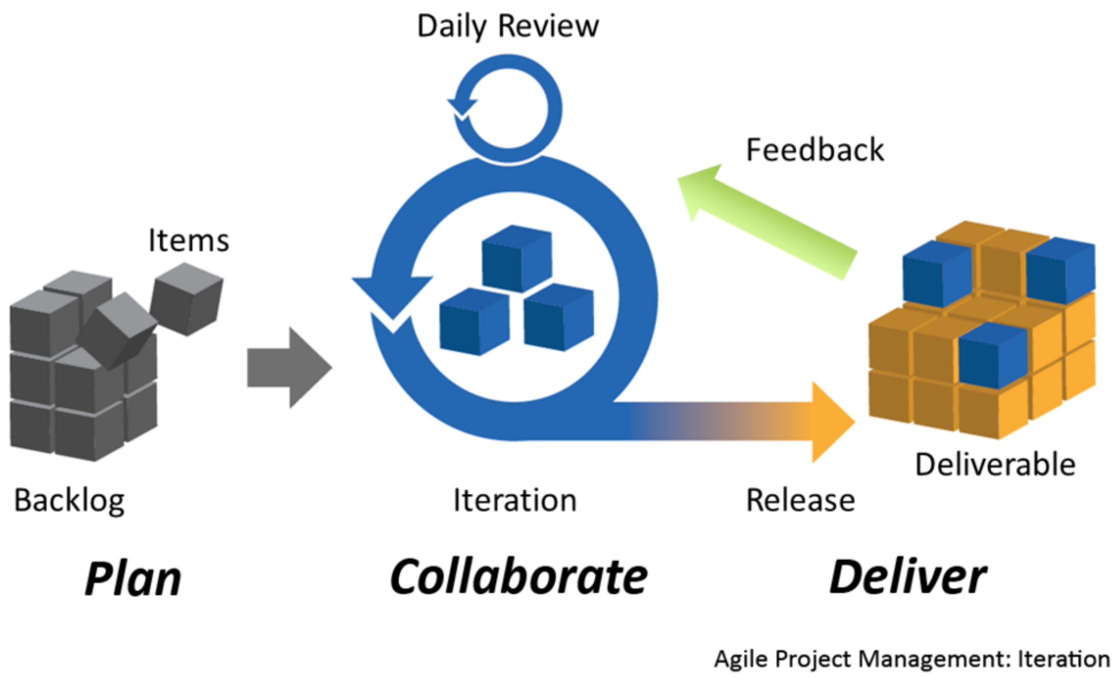


Figure 3. Iterative development model (Wikipedia, Iterative and incremental development, 2015)

In the heart of agile is doing development in iterations. Even though several different methods of doing agile software development exist, what they all have in common is iterative approach to software development. Iterative development is not unique to agile development and has been around longer than agile. Iterative development has been mentioned in literature as early as the 80's pre-dating the emergence of agile methods. (Wikipedia, Iterative and incremental development, 2015)

In practice iterative development means that the end product is defined in small blocks implementable in short amounts of time. After each iteration the software is left in a state where it can be packaged and tested for release to customer or as close to it as possible. Sometimes it can be even released to end users for actual usage. Essentially iteration covers all the stages of traditional waterfall and V-models in a miniature size. After iteration feedback from customer is gathered and required modifications to the product are planned. These iterations are then repeated until the product is ready.
Each of these iterations can be broken down to smaller iterations for development team to handle internally. For example doing a complex UI for customer can be broken down to doing small parts of it at a time even when not releasing to the end customer. (Wik-

ipedia, Agile software development, 2015. Wikipedia, Iterative and incremental development, 2015.)

The critics of agile development say that doing design in these small blocks can cause the big picture to be lost and the architecture of an application to become a complete mess. In some part this is a misconception as even if you do not fully draw up the architecture of an application in a single go you still have to do an architectural design in the beginning even if it is not completely finished in e.g. the first iteration of development. Nevertheless architectural decay can indeed be a big problem when doing agile development but there are several methods that are commonly used to counter these problems. One of the most used ones is a constant refactoring and evaluation of architecture alongside code refactoring at the end of iteration. This activity incorporates the idea of looking at the big picture at the end of iteration and adapting the architecture as needed. If done right this iterative approach and constant architectural evaluation while the application matures can be seen as advantage over traditional methods of architectural design. (El-Khawaga, Galal-Edeen, Riad, 2013)

Big advantage of iterative approach is that the software is in a state of being more or less ready at the end of iteration which allows rapid deployment when necessary. This is in heavy contrast to traditional methods where the product testing and packaging for end user is done in a one big block at the end of development. This allows for rapid deployments in cases of emergency. (Wikipedia, Agile software development, 2015. Wikipedia, Iterative and incremental development, 2015.)

Compared to traditional methods this way of doing things also incorporates a lot more visibility on the state of the project. In traditional models it can be as long as months without developers producing anything concrete on which the status of the project could be measured on. In this respect agile builds on top of iterative methods by removing the traditional documentation and formal processes with direct communication with the stakeholders. (Wikipedia, Agile software development, 2015)

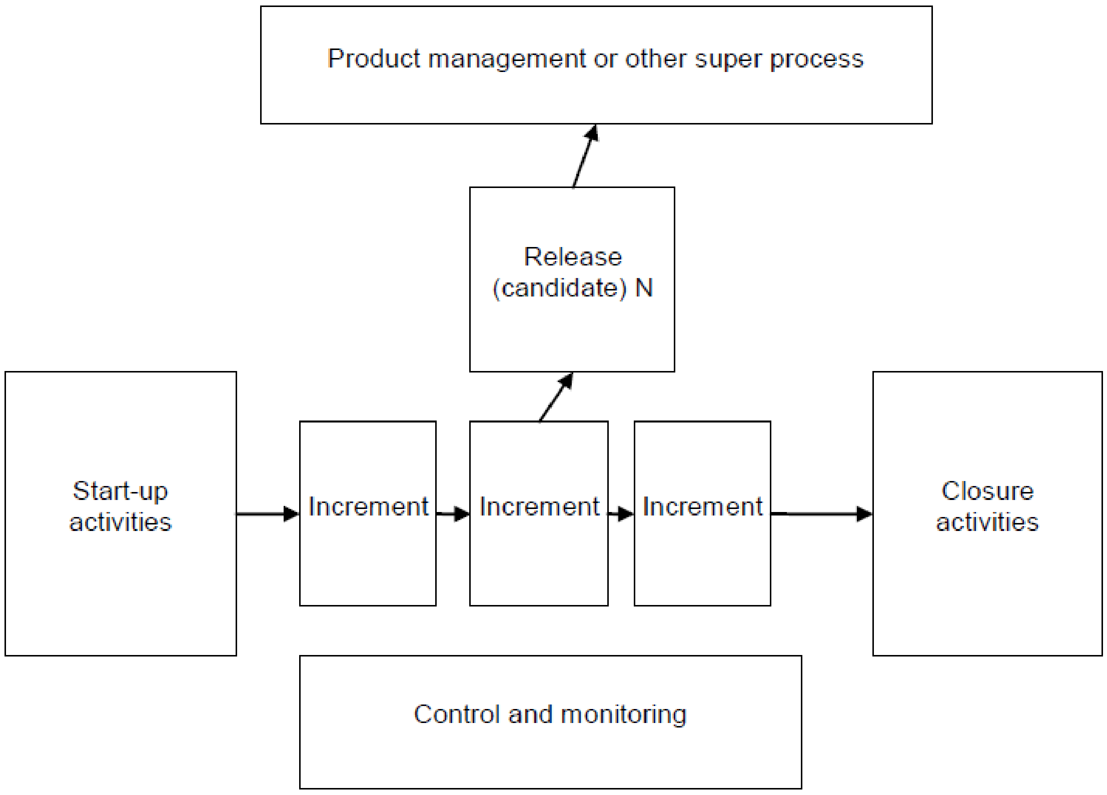
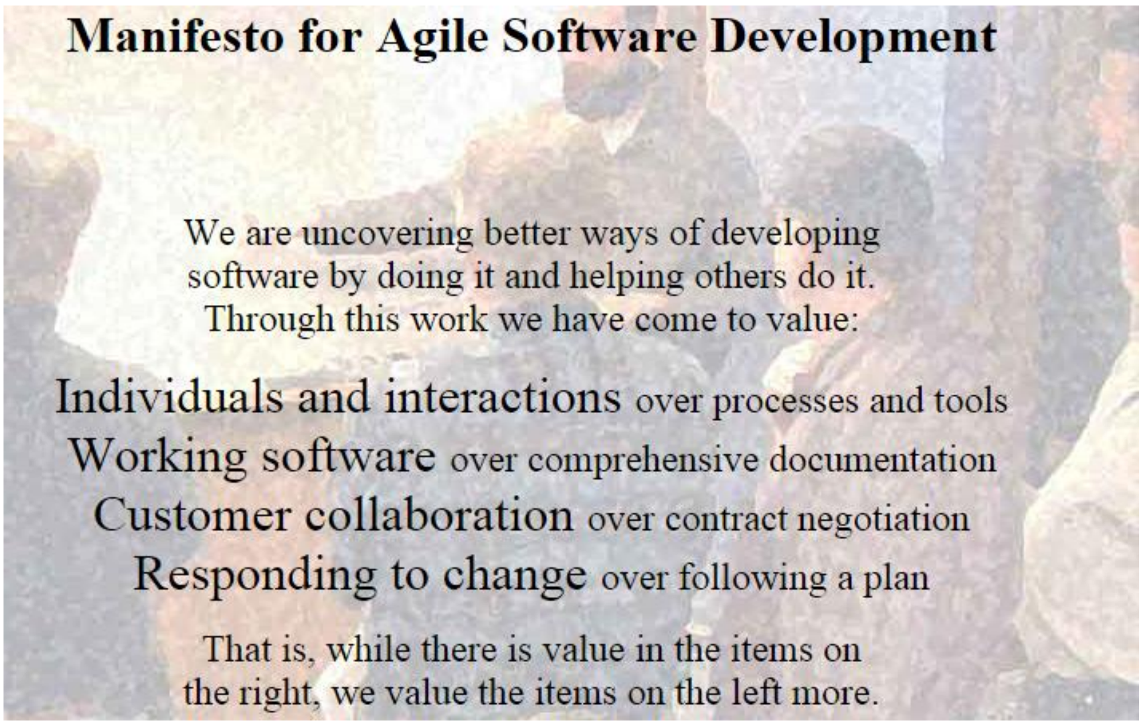


Figure 4. Basic agile process. (Vuori, 2011)

2.3.3 Pitfalls of agile software development



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Picture 1. The agile manifesto. (Manifesto for Agile Software Development, 2015)

Ironically the biggest problems in doing agile development come from the agile manifesto itself. People tend to overemphasize the ideas on the left side and completely eschew the ones on the right even though the manifest itself states they still have value. “Individuals and interactions over processes and tools” does not mean you should have no processes or tools; instead you should have processes and tools to support individuals and interaction. “Working software over comprehensive documentation” does not mean you should not have any documentation at all, instead you should document only what is needed to support the creation of working software. “Customer collaboration over contract negotiation” does not mean you should not negotiate at all before starting development as not negotiating about the price and specifications would lead to customer not knowing what the cost of what they are buying is. Own experience as a customer has shown though that typically doing initial negotiations as light as possible and paying by the hour leads to cheaper overall costs. This does require heavy involvement from the customer in iteration of development. “Responding to change over following a plan” does not mean you should not have a plan at all; instead you should have a plan that is susceptible to change.

A typical situation is where all the discipline is thrown out of the window and coders are left on their own to write code without any requirements or specification documentation or organization of work with only vague requirements given in a meeting. Needless to say this leads to chaos. This approach is completely beside the point of agile. It could be argued that agile methods are a lot stricter on the organization of work than traditional methods. In waterfall model a developer could be left to work on a feature alone for several months with no collaboration or testing enforced. In agile development this is exactly the opposite when the developers have to produce something that works constantly with constant monitoring and measurement of the results of work.

2.3.4 Agile adaptation

Agile software development has been accepted as de-facto standard of software development and almost every company that does software development has adopted some form of agile development. Although several established ways of doing agile development have emerged (e.g. Scrum or Extreme Programming) a typical situation is adapting a hybrid model which takes the parts of different agile development methods deemed most useful for the company. The successful usage of agile methods in software development has also lead to adaptation of agile methods into other fields of engineering. (Wikipedia, Agile software development, 2015)

3 APPLICATION LIFECYCLE MANAGEMENT

This chapter explains the concept of application lifecycle management and takes a look at the key concepts it envelopes.

3.1 Overview

In short, application lifecycle management (ALM) means governing all the activities during the lifecycle of a software product from the initial planning to the end of usage. It is similar to product lifecycle management concept but the term is meant to mean specifically software applications. ALM covers monitoring and reporting all the design artefacts during the process including (but not limited to) requirements, specifications, source code, bug reports and test results. ALM as a concept emerged mainly from different software vendors offering tool packages for doing all these activities and so ALM system is used as a term to describe a suite of tools for this purpose. Atlassians JIRA and Microsofts Team Foundation Server are some of the most known commercial ALM solutions. (Wikipedia, Application Lifecycle Management, 2015)

3.2 Key components

As the concept of ALM is rather new and the most profound advocates for it are also selling commercial solutions to the problem, a neutral evaluation of tools and solution capabilities can cause problems. Commercial vendors tend to overemphasize the parts their solutions shines in and disregard the ones they are not so good at. For evaluation of ALM solutions there have been several studies but perhaps the most interesting of them is the “Towards an Application Lifecycle Management Framework” dissertation work by Jukka Kääriäinen in VTT which describes means and methods for neutral evaluation of ALM solutions in a form of ALM framework. (Kääriäinen, 2011)

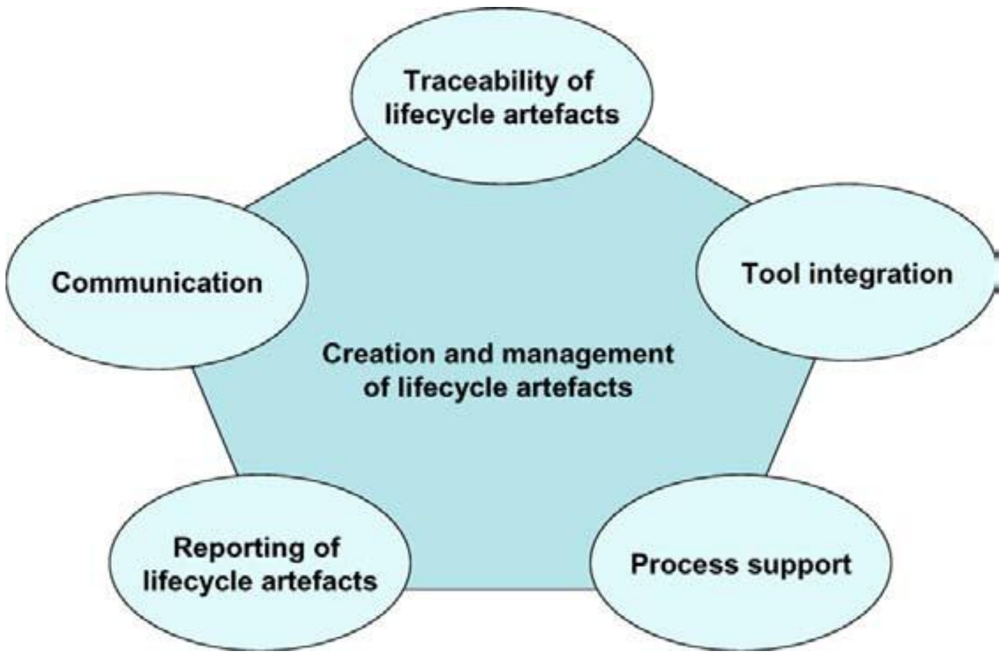

Traceability of lifecycle artefacts
Communication
Tool integration
Creation and management of lifecycle artefacts
Reporting of lifecycle artefacts
Process support

Figure 5. Key functions of ALM system. (Kääriäinen, 2011)

Kääriäinen, 2011, describes the key components of an ALM system as:

- **Creation and management of lifecycle artefacts**
  ALM wraps around the concept of lifecycle artefacts. A lifecycle artefact can be for example a requirement, design specification, test case or test result. As a minimum requirement and the most important thing, an ALM system must support creation of these artefacts. (Kääriäinen, 2011)

- **Traceability of lifecycle artefacts**
  In order to trace which requirement is satisfied by which design specification and which test in turn verifies it an ALM system needs to have a way to link design artefacts to each other. These links typically include linking requirements to design specifications all the way to test results and releases. Dates, authors and status changes of all these artefacts need to be recorded in ordered to see the chronological progress of each feature. (Kääriäinen, 2011)

- **Reporting of lifecycle artefacts**
  Just having a way to link and trace design artefacts to each other is not enough, you also need to be able to create human-understandable presentations of the statuses and relationships of the lifecycle artefacts. The creation of these reports should be automated. These reports can include visual or textual representations.

Typically different kinds of reports are offered for different kinds of needs. E.g. QA manager might need list of test results for a given release. These reports can be printable documents but more typically they could be e.g. web pages with the statuses of items in current development iterations. (Kääriäinen, 2011)

- **Communication**
  Good ALM solution should facilitate communication in the team. For real time communication this could mean having integrated real time chat or video conference possibilities inside the tools. Perhaps more importantly an ALM should facilitate the sharing of knowledge between team members in written form. One of the most popular medias for this is having an integrated wiki inside ALM on which all team members can document e.g. some difficult development task, development process conventions, creation of release packages etc. Also having the possibility to add informal comments to lifecycle artefacts, for instance commenting why test result failed or why some feature is not implemented. This kind of documentation especially follows the principles of agile development as all the formality in it is removed and only the actual value adding part of it is done. (Kääriäinen, 2011)

- **Process support**
  Tools should not dictate how a development team functions but instead should support the way the development team functions. This means that a good ALM solution should adapt to different ways or development methodologies some teams like to use. (Kääriäinen, 2011)

- **Tool integration**
  Good integration of tools is essential for having an ALM solution that is easy to use and to minimize wasted effort by doing as much automatically behind the scenes as possible. In software development one of the most important elements is the integration of the source code version control system, requirements management system and testing and release systems. This holds especially true for agile software development where changes to requirements and releases are constantly being done so if the integration between systems are bad it will lead to either a lot of wasted effort on manual work or people stopping to follow the development processes the ALM is supposed to facilitate. (Kääriäinen, 2011)

Different vendors have varying support for each of these features and in some cases some of them can be missing completely. ALM is more of a general name for the umbrella concept rather than strict format on how these activities should be setup. In ALM ideology all these aspects should integrate together though. Different emphasis on the different aspects of solution in selection of tools should be considered depending on the scope and quality of the development. A big factor is also the size and locations of the team. Same solution cannot fit a 5 person team located in the same office versus 100 person team distributed among different time zones. The larger the team the more emphasis on the quality of solution as the management of such work gets exponentially more difficult. (Kääriäinen, 2011)

4 DEVELOPMENT OF SAFETY-CRITICAL SOFTWARE

This chapter explores the restrictions and requirements imposed by different safety standards for the development of safety-critical software development. The viewpoint of this chapter is especially the development of safety-critical software for electric forklift trucks.

4.1 Overview

The term “safety-critical software” means software which controls or monitors functions in which a failure can cause harm to human beings, damage to equipment/property or harm to environment. There are various different fields where software is considered to be safety-critical, for example medical or automotive applications. The way software is done in these fields is typically highly regulated and has to fulfil requirements of safety standards. Regular auditing by governmental bodies for the companies working in safety related application fields are common. (Vuori, 2011. Wikipedia, Life-critical system, 2015)

Safety standards vary greatly depending on the field of application and can be dramatically different. For example, safety of industrial assembly machine is an order of magnitude of less importance than safety of nuclear plant or military weapons system. The key principle and starting point of development should therefore be the assessment of the risks in the application in question. This risk analysis should be done and documented according to the standard in question. The risks are typically categorized by severity and probability of occurrence. The formulas and categories used are dependent on the field of application (the standard used for the requirements). For each risk identified there then has to be some sort of mitigative measure identified. This can be for instance a certain technical requirement for the application or indication or plan how training or some other action is enough for mitigation of the risk and no design actions are required. The severity of the measure should directly correlate with the severity of the risk in question. (Vuori, 2011. EN ISO 13849-1, 2008)

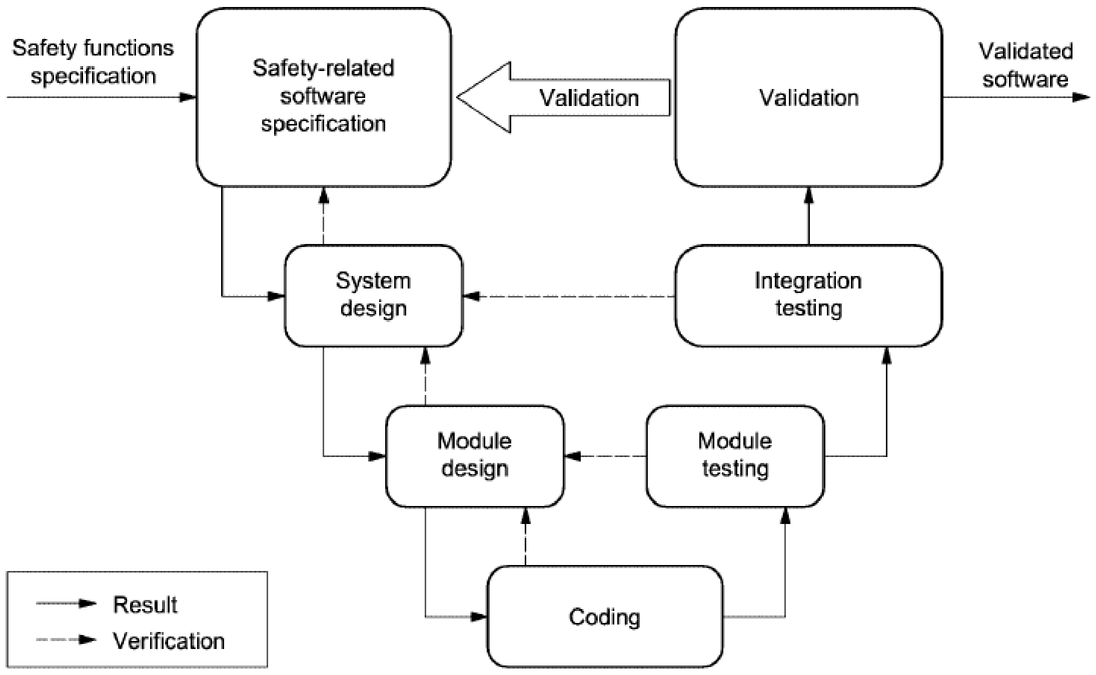


Figure 6. Overview of the required development process from EN ISO 13849-1. (EN ISO 13849-1, 2008)

After the initial risk analysis the risks that are identified to be taken into account in the design process are then translated into safety requirements. These safety requirements are then either taken into account in specification of features of the application or in some cases they have to be made into features themselves with no other function in the application than ensuring safety. All of these safety related items in the design have to be tracked and regularly analysed during the development of the product. This imposes severe restrictions on the documentation and monitoring of the development process and traditionally the development processes for safety-critical applications have been older and tested ones. In several modern safety standards the older V-model is still the recommended model of development. (Vuori, 2011. EN ISO 13849-1, 2008)

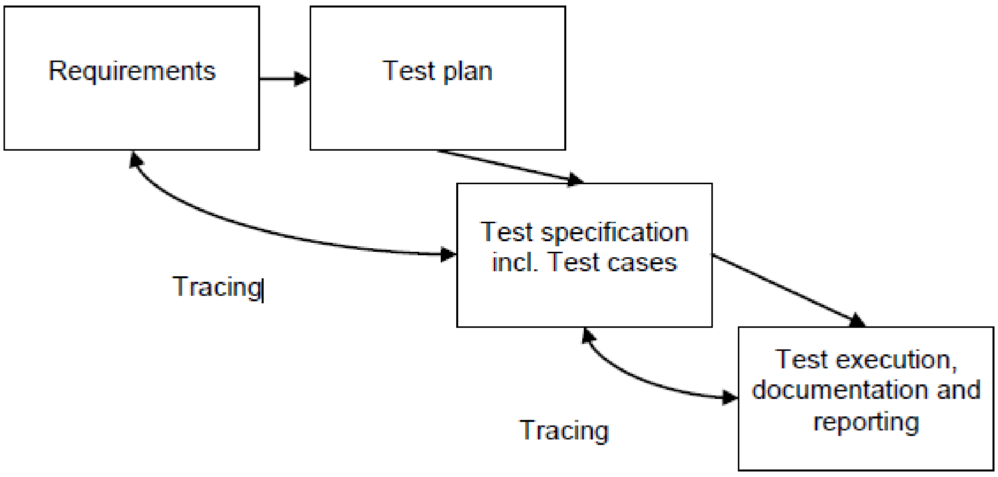

Requirements
Test plan
Test specification incl. Test cases
Tracing
Test execution, documentation and reporting
Tracing

Figure 7. Tracing tests to requirements. (Vuori, 2011)

In addition to translating risks into safety requirement and taking them into account during the design process there also needs to be formal testing of the features. This means having test cases that can be linked to specifications and requirements as seen in figure 7. Usually in the definition of a test case there is a mention of the requirement or design specification in question. Typically the standards for safety-critical software development also inflict restrictions on the development process itself. The V-model shown in figure 6 is typically required to be followed in development of software. Normally also following of some certified quality standard (e.g. ISO 9001) in project management and quality management systems is required. This causes some restrictions on how the development, especially in the documentation processes, is done. (EN ISO 13849-1, 2008)

4.2 SFS-EN ISO 13849-1 requirements for software development process

Forklift truck safety is regulated under EN1175 standard which in turn is under SFS-EN ISO 13849-1, "Safety of industrial trucks – Electrical requirements – Part 1: General requirements for battery powered trucks". Currently applying EN1175 does not impose the requirements of SFS-EN ISO 13849-1 on truck software development. New version of EN1175 is currently in draft state. Rocla Oy is participating in its creation. In current draft full requirements for embedded and application software and parameter modifying tools from SFS-EN ISO 13849-1 are going to be in effect. Current estimation for new version of EN1175 to be in effect is in year 2017. This gives a reasonable transition period in which to make sure all the requirements are fulfilled. (EN 1175-1:1998+A1, 2010. EN ISO 13849-1, 2008. EN 1175-1padova, 2014)

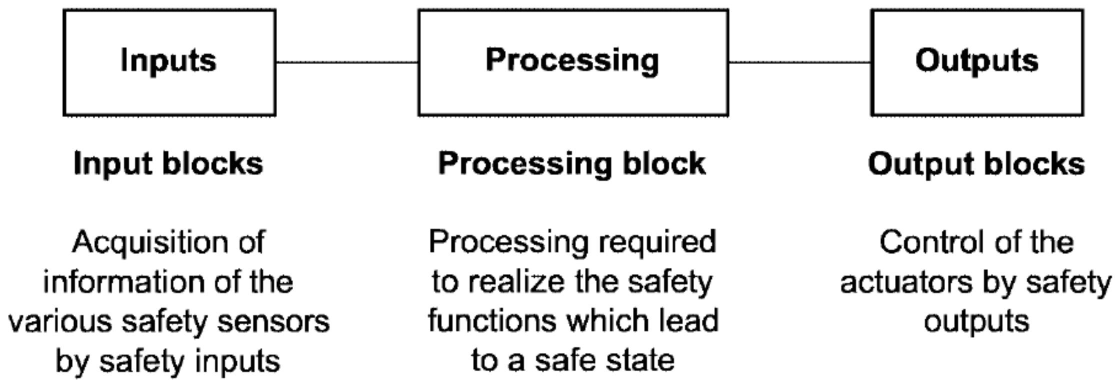
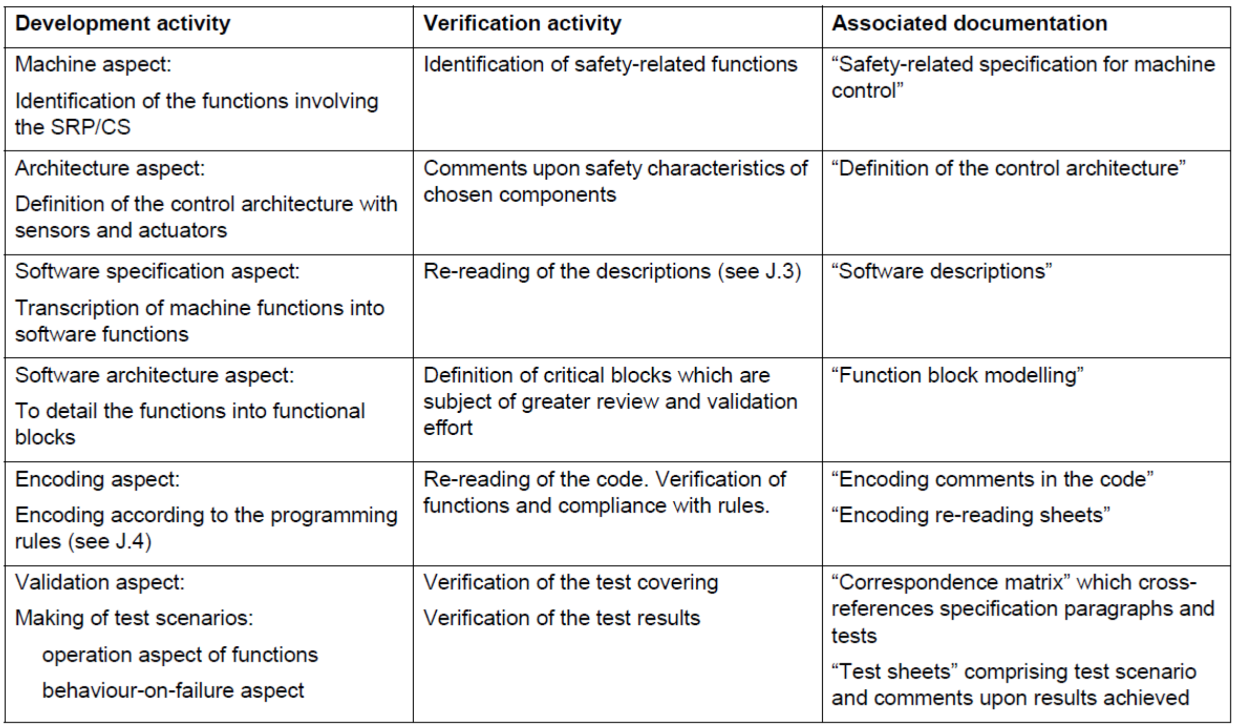

Inputs
Processing
Outputs
Input blocks
Processing block
Output blocks
Acquisition of information of the various safety sensors by safety inputs
Processing required to realize the safety functions which lead to a safe state
Control of the actuators by safety outputs

Figure 8. General model for software architecture in safety-related software according to SFS-EN ISO 13849-1. (EN ISO 13849-1, 2008)

SFS-EN ISO 13849-1 categorises safety-related functions of software according to performance levels (PL) with required performance level (PLr) meaning a safety that has to be reached by certain functions of the system (e.g. control of steering). For software this imposes a lot of restrictions including (but not limited to) using a coding standard, general architecture model usage and usage of semi-formal (e.g. diagrams) methods to clarify the workflow of the software. Code simulation and static analysis tool usage are also required depending on the required safety level of the function. (Vuori, 2011. EN ISO 13849-1, 2008. Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)

Restrictions on which programming languages can be used for development can be imposed depending on the PLr of the application. The highest safety levels can require the usage of limited variability language (LVL) in which what is doable with the language can be limited (as opposed to using full variable language (FVL)). LVL languages are typically either graphical block diagram languages or written languages with very limited instruction sets. Perhaps the most typical LVLs are IEC 61131-3 conforming languages for programmable logic controllers (PLCs). FVLs mean the most common programming languages, e.g. Assembler or C/C++. When C or C++ is used in safety-critical applications, typically heavy code analysis or simulation is required due to very low restrictions on what can be done with the language. Most common languages fall into category of FVLs by default unless a specific, verified version of them is created and verified to cope with LVL requirements. (Vuori, 2011. EN ISO 13849-1, 2008.

Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)



Development activity	Verification activity	Associated documentation
Machine aspect: Identification of the functions involving the SRP/CS	Identification of safety-related functions	“Safety-related specification for machine control”
Architecture aspect: Definition of the control architecture with sensors and actuators	Comments upon safety characteristics of chosen components	“Definition of the control architecture”
Software specification aspect: Transcription of machine functions into software functions	Re-reading of the descriptions (see J.3)	“Software descriptions”
Software architecture aspect: To detail the functions into functional blocks	Definition of critical blocks which are subject of greater review and validation effort	“Function block modelling”
Encoding aspect: Encoding according to the programming rules (see J.4)	Re-reading of the code. Verification of functions and compliance with rules.	“Encoding comments in the code” “Encoding re-reading sheets”
Validation aspect: Making of test scenarios: operation aspect of functions behaviour-on-failure aspect	Verification of the test covering Verification of the test results	“Correspondence matrix” which cross-references specification paragraphs and tests “Test sheets” comprising test scenario and comments upon results achieved

Figure 9. List of required development, verification and documentation actions in software development process according to SFS-EN ISO 13849-1. (EN ISO 13849-1, 2008)

Perhaps even higher impact on the software development as a whole is caused by the requirements for the development process. The V-model shown in figure 6 is also required by SFS-EN ISO 13849-1. This is the typical high level structuring of software development activities required by safety standards. Figure 9 lists the activities required to be done during the development process in addition to technical restrictions on the coding and architectural design of application itself. These are required from all safety levels. Other things required from all safety levels include having a functional testing of the software and all the necessary lifecycle actions for each of the releases. Lifecycle activities mean having appropriate specifications and most importantly documented test plans and test results with appropriate testing process. (Vuori, 2011. EN ISO 13849-1, 2008. Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)

For higher PL levels certified quality and project management systems are required. This is a companywide requirement which means that the standard of whole organization has to be of certain level before being allowed to create high risk applications. Other requirements from the higher safety levels include structured documentation of risk analysis, requirements, safety requirements, specifications, test plans and test results with full traceability. Also more rigorous testing is required compared to lower safety levels. One of the most important requirements is also doing of impact analysis for each new release of applicable software. This means analysing which parts of the system the change affects and which parts of the system the change does not affect with attached reasoning for each implication. For all affect parts then changes to risk analysis, requirements and specification will then have to be considered. The minimum for all affect parts is to re-test them and verify functionality. (Vuori, 2011. EN ISO 13849-1, 2008. Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)

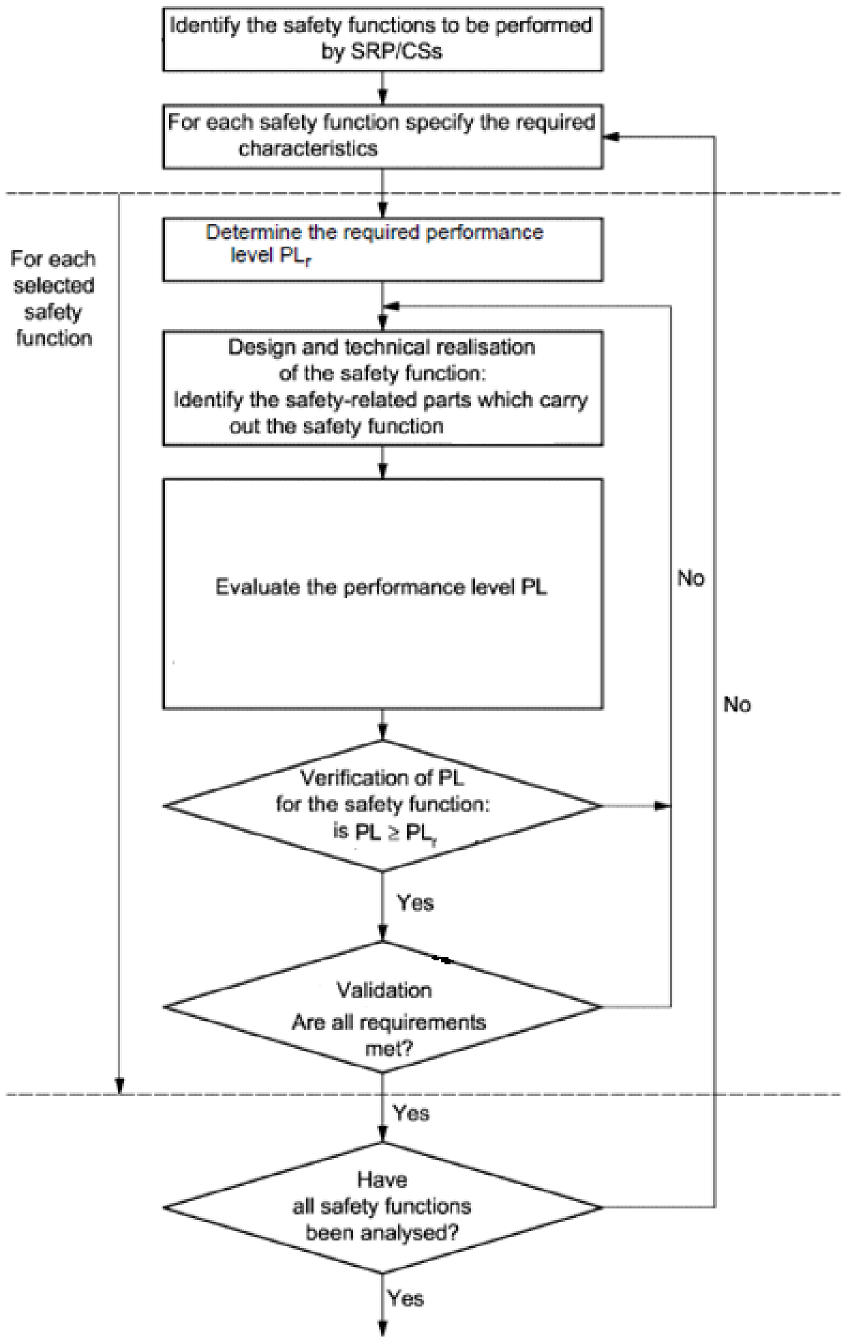

Identify the safety functions to be performed by SRP/CSs
For each safety function specify the required characteristics
For each selected safety function
Determine the required performance level PL_r
Design and technical realisation of the safety function: Identify the safety-related parts which carry out the safety function
Evaluate the performance level PL
No
No
Verification of PL for the safety function: is $PL \geq PL_r$
Yes
Validation Are all requirements met?
Yes
Have all safety functions been analysed?
Yes

Figure 10. Iterative process for design of safety-related parts of system. Also relevant for designing safety-related software features. (EN ISO 13849-1, 2008)

Figure 10 shows the iterative process to be used in designing safety related functions of the truck. This same iterative process applies also to software design. All these activities should also be documented appropriately with traceability and impact analysis if they are done for software that has already been released to the field. This incremental design method is very closely related to agile software development model. (Vuori, 2011. EN ISO 13849-1, 2008)

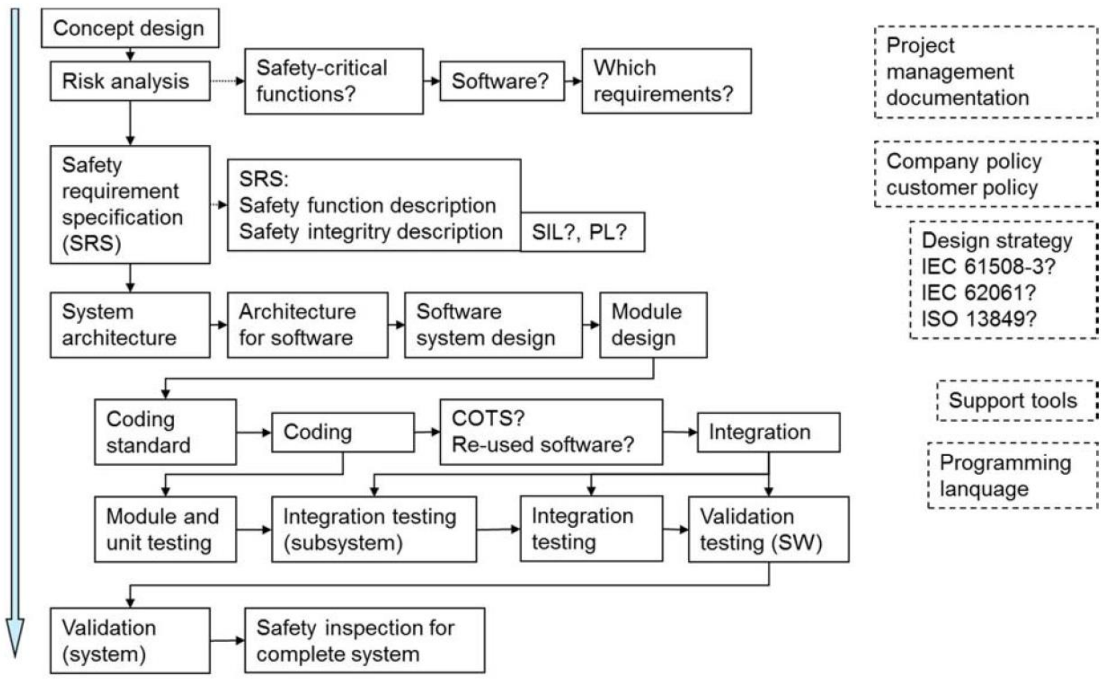

Concept design
Risk analysis
Safety-critical functions?
Software?
Which requirements?
Safety requirement specification (SRS)
SRS: Safety function description Safety integritry description
SIL?, PL?
System architecture
Architecture for software
Software system design
Module design
Coding standard
Coding
COTS? Re-used software?
Integration
Module and unit testing
Integration testing (subsystem)
Integration testing
Validation testing (SW)
Validation (system)
Safety inspection for complete system
Project management documentation
Company policy customer policy
Design strategy IEC 61508-3? IEC 62061? ISO 13849?
Support tools
Programming lanquage

Figure 11. Activities and considerations that need to be done during design process of safety-critical software. The right side boxes show some things to consider during some phases of this process. (Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)

Designing software according to all requirements by standard with all the design efforts and testing done correctly alone is not enough, the designing entity also has to be able to prove the steps have been taken if an audit is done. Figure 11 shows the usual activities that need to be done when designing safety-critical software in machine application. Each of these steps has to be auditable afterwards meaning some form of documentation has to be done from each step. Traditionally this has led to either intangible mass of word documents with references to other documents or the usage of software tools with integrated tracing between documents and design artefacts. This leads to safety-critical software being extremely expensive to develop when compared to normal development but these activities cannot be skipped in order to get a permit to sell these kinds of products. (Vuori, 2011. EN ISO 13849-1, 2008. Malm, Vuori, Rauhamäki, Vepsäläinen, Koskinen, Seppälä, Virtanen, Hietikko, Katara, 2015)

4.3 Requirements for software-based parameterisation of truck

The controllers and embedded software used in forklift trucks are typically generic to the point that the same controller and embedded software can be used in several different truck models for several different manufactures and brands. To allow the extent of customization required by each manufacturer, a large part of the functionality of the software is defined by parameterization. When these parameters modify the safety-critical aspects of the embedded software then naturally the requirements for safety-critical software development also apply to the tools used for setting these parameters. The new draft of EN1175 lists the requirements for parameterization along the same lines as SFS-EN ISO 13849-1 so the assumption that the requirements from SFS-EN ISO 13849-1 apply also for desktop tool for parameterization. A good example of safety-critical parameter modified would be the maximum speed of the truck. In electric trucks this is typically controlled by setting the maximum Hz of the motor. Needless to say the value of this parameter has a very high impact on the safety of the truck and setting it without any limitations could lead to catastrophic results. (EN ISO 13849-1, 2008. EN 1175-1padova, 2014)

SFS-EN ISO 13849-1 lists specific requirements for the tool used for modifying safety related parameters:

- Control the range of valid inputs.
- Control data corruption before transmission.
- Control the effects of errors from parameter transmission process.
- Control the effects of incomplete parameter transmission
- Control the effects of faults and failures of hardware and software of the tool used for parameterization.

SFS-EN ISO 13849-1 also states that either all the same requirements as for SRP/CS shall apply for parameterization tool or a special procedure must be used. This procedure must either retransmit the modified parameters back to tool for verification or use “other suitable means of confirming the integrity of the parameters”. Using this alternate method for confirming the parameters is highly preferred to implementing the safety requirements of SFS-EN ISO 13849-1. (EN ISO 13849-1, 2008)

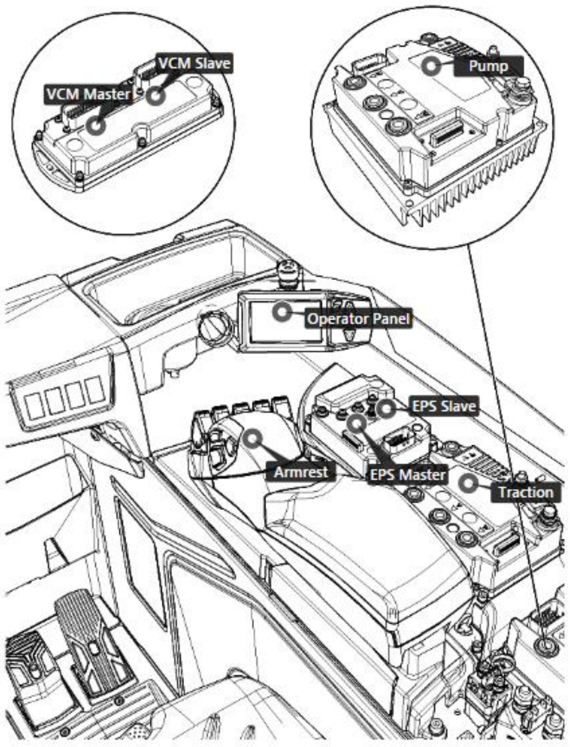
Basically this means that for the parameterization the requirements boil down to having specifications and tests for implementation of the features listed above. In the scope of this work this would mean having formal requirements, specifications and test results for the features with traceability. This leads to implementing a lightweight version of system that would be required from SRP/CS.

5 IMPLEMENTATION

This chapter introduces the software development that is done at Rocla Oy. Also the current status of software development processes and methods in Rocla Oy is studied and analysed and a possible solution to fulfil standard requirements and improve the effectiveness of development work is represented. A look is taken into different possible application lifecycle management tool sets and a selection of an ALM solution for pilot use is made.

5.1 Overview

Currently there are 2 categories of trucks produced at Rocla Oy: manual and automatic. Manual trucks mean the traditional human operated trucks and automatic mean trucks that are completely controlled by computer logic with no human guidance. Automatic truck development is left out of the scope of this thesis.

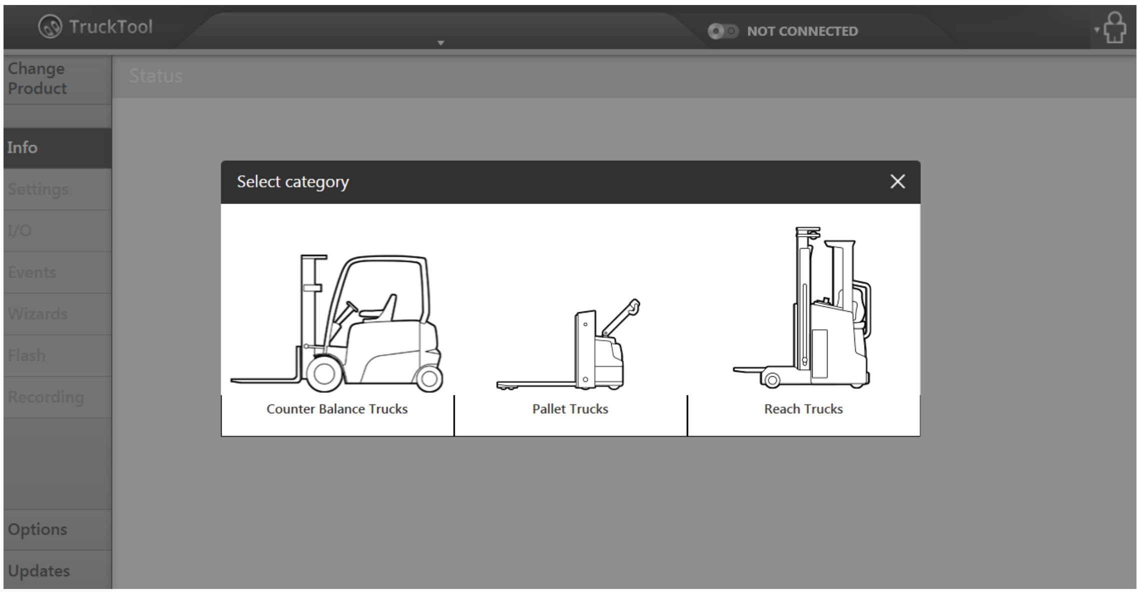


Picture 2. Example of different controllers on a reach truck. Label represents a processor in a controller. There are 1 or 2 processors per controller (master and slave where applicable).

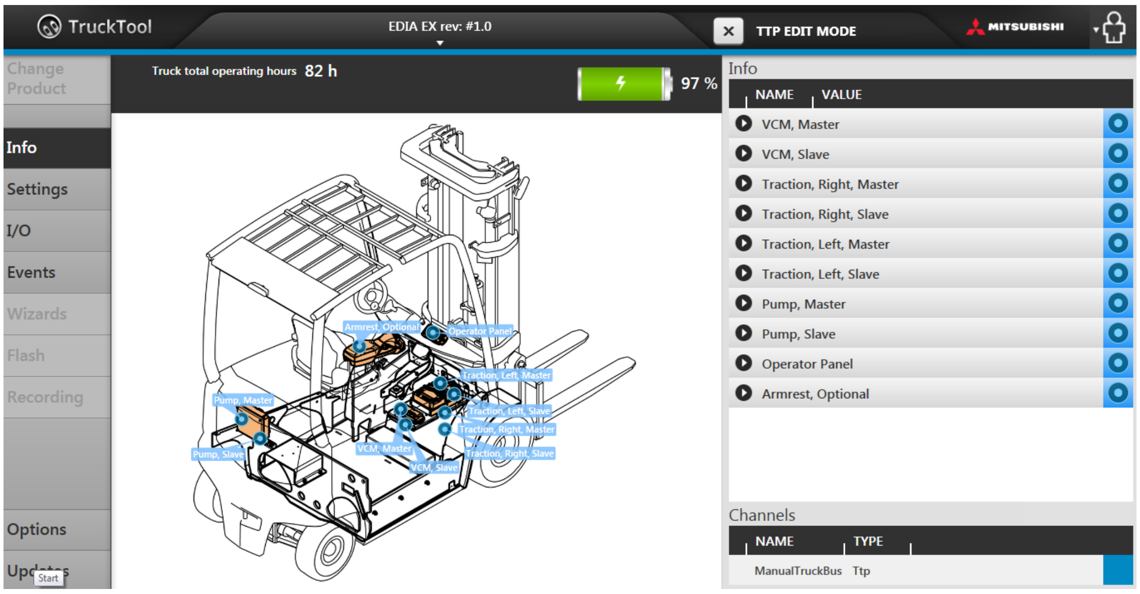
A typical truck consists of several different controller units that each controls a different aspect of the truck. Some of these controllers have dual processors for safety reasons: same functions are done by 2 different processors and the results are evaluated. If the results do not match the truck is set to error mode and normal function is stopped. These controllers are usually motor controllers (e.g. pump or traction), input units (e.g. finger-tip switch unit), output units (e.g. display) or vehicle controller master unit which controls the operation of the whole truck. The controller hardware is bought from outside suppliers and custom software for Rocla is created.

The development of the customized software is typically outsourced to the manufacturer of the controller unit. Software requirements specification, some levels of design specifications and all integration testing of the different controllers is done at Rocla.

Service tool for programming these controllers and setting different parameters on them is called TruckTool. Development of TruckTool is done in-house at Rocla Oy with full control on development and testing at our own hands.



Picture 3. Truck model selection at TruckTool start up.



Info

NAME	VALUE
VCM, Master	
VCM, Slave	
Traction, Right, Master	
Traction, Right, Slave	
Traction, Left, Master	
Traction, Left, Slave	
Pump, Master	
Pump, Slave	
Operator Panel	
Armrest, Optional	

Channels

NAME	TYPE
ManualTruckBus	Ttp

Picture 4. TruckTool info-view with a visual representation of a truck and its status.

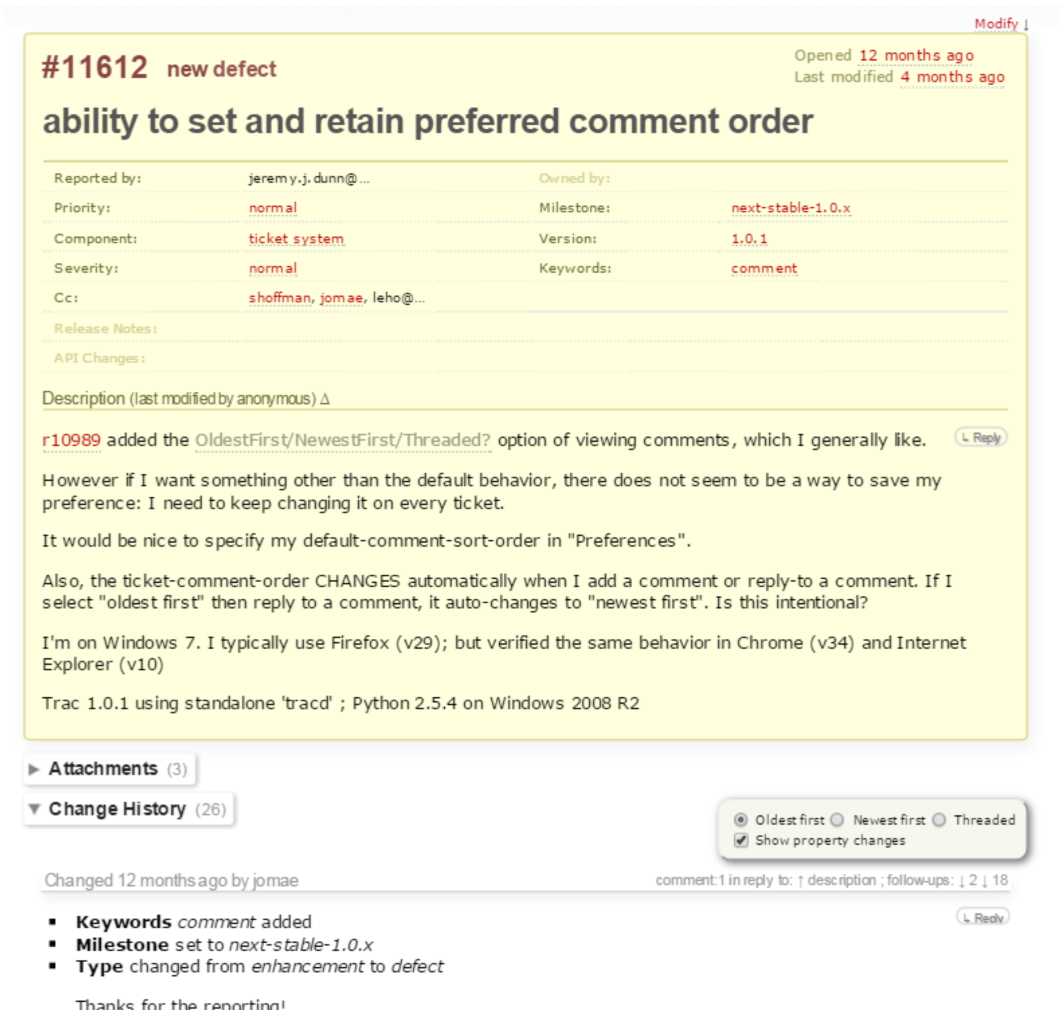
TruckTool supports a wide variety of the current trucks in production in Finland with some exceptions on the older models. Support for more models is currently under development and TruckTool aims to unify service of all trucks under one tool. TruckTool is a desktop application and it connects to trucks with USB adapters via service connectors on truck. Access to TruckTool is controlled by login with individual user credentials. For this purpose TruckTool also has an associated website with password creation, installer and instruction distribution.

5.2 Current status

Current status of development can be categorized to 2 different categories: software projects which include desktop and server development and truck embedded software projects. This chapter will take a look at the status of both of them with emphasis on the software projects and specifically the development of service tool.

5.2.1 Software projects

Software projects currently follow most principles of agile software development. Documentation and initial planning is reduced to minimum. Trac is used for project management and issue tracking. Trac is a web-based open source tool for software project management and it has an integrated wiki and a basic task tracking and management system. Trac also shows commits from underlying source control system and supports setting links to tasks form in commit messages. Most of the documentation is done to wiki integrated inside Trac or Tracs issue management system. Teams working in these projects are typically distributed to several locations. This causes a heavy emphasis on effective communication. This communication is facilitated by having daily meetings every morning, having open chat channels to everyone. Source control is done with Git which supports this kind of distributed development well due to its decentralized system.



Modify ↓

#11612 new defect

Opened 12 months ago
Last modified 4 months ago

ability to set and retain preferred comment order

Reported by:	jeremy.j.dunn@…	Owned by:	
Priority:	normal	Milestone:	next-stable-1.0.x
Component:	ticket system	Version:	1.0.1
Severity:	normal	Keywords:	comment
Cc:	shoffman, jomae, leho@…		
Release Notes:			
API Changes:			

Description (last modified by anonymous) Δ

r10989 added the OldestFirst/NewestFirst/Threaded? option of viewing comments, which I generally like.

However if I want something other than the default behavior, there does not seem to be a way to save my preference: I need to keep changing it on every ticket.

It would be nice to specify my default-comment-sort-order in "Preferences".

Also, the ticket-comment-order CHANGES automatically when I add a comment or reply-to a comment. If I select "oldest first" then reply to a comment, it auto-changes to "newest first". Is this intentional?

I'm on Windows 7. I typically use Firefox (v29); but verified the same behavior in Chrome (v34) and Internet Explorer (v10)

Trac 1.0.1 using standalone 'tracd' ; Python 2.5.4 on Windows 2008 R2

▸ Attachments (3)

▾ Change History (26)

[✓ Oldest first] Newest first Threaded
[✓ Show property changes]

Changed 12 months ago by jomae

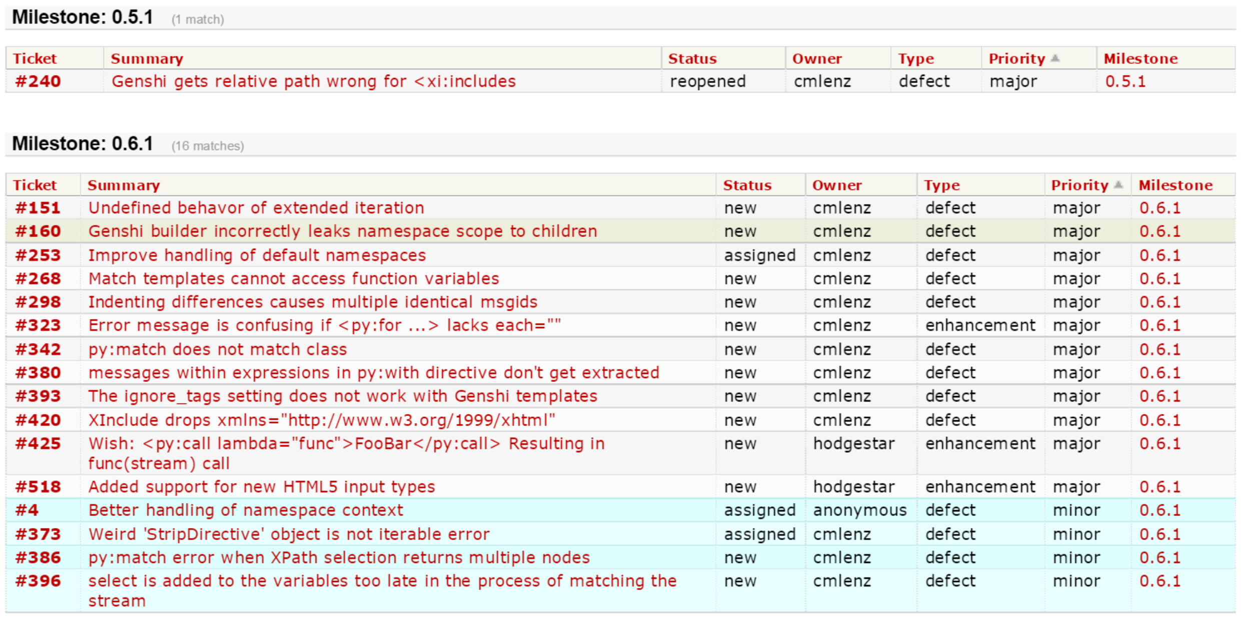
comment:1 in reply to: ↑ description ; follow-ups: ↓ 2 ↓ 18

- **Keywords** *comment* added
- **Milestone** set to *next-stable-1.0.x*
- **Type** changed from *enhancement* to *defect*

Thanks for the reporting!

Picture 5. An example of a ticket inside Trac system (Trac, 2015)

Daily work is separated into tasks. Inside Trac these items are called “tickets”. These tasks then in turn are distributed into sprints that typically last 2 weeks. There is some variance in the length of sprint depending on the current resourcing in the project. During the execution of each task the developers are required to add information about the solution they are creating for the task to project management task tracking system. Before a developer can deem a work finished, he also has to create a test plan for the new feature he has implemented and execute those tests. These tests also include writing automated unit tests alongside code and executing them before finishing the task. Before finishing all the steps required for each task the developer is not allowed to commit his changes into common source control repository. In every commit message there also has to be a link to a task in question in order to check the specifications and tests associated to the changes in code. In the spirit of agile, these tickets are very informal in nature with free communication allowed in form of comments and constant evolution of the specification inside the ticket. Editing all the fields of the ticket is open for all developers in the spirit of wiki editing.



Milestone: 0.5.1 (1 match)

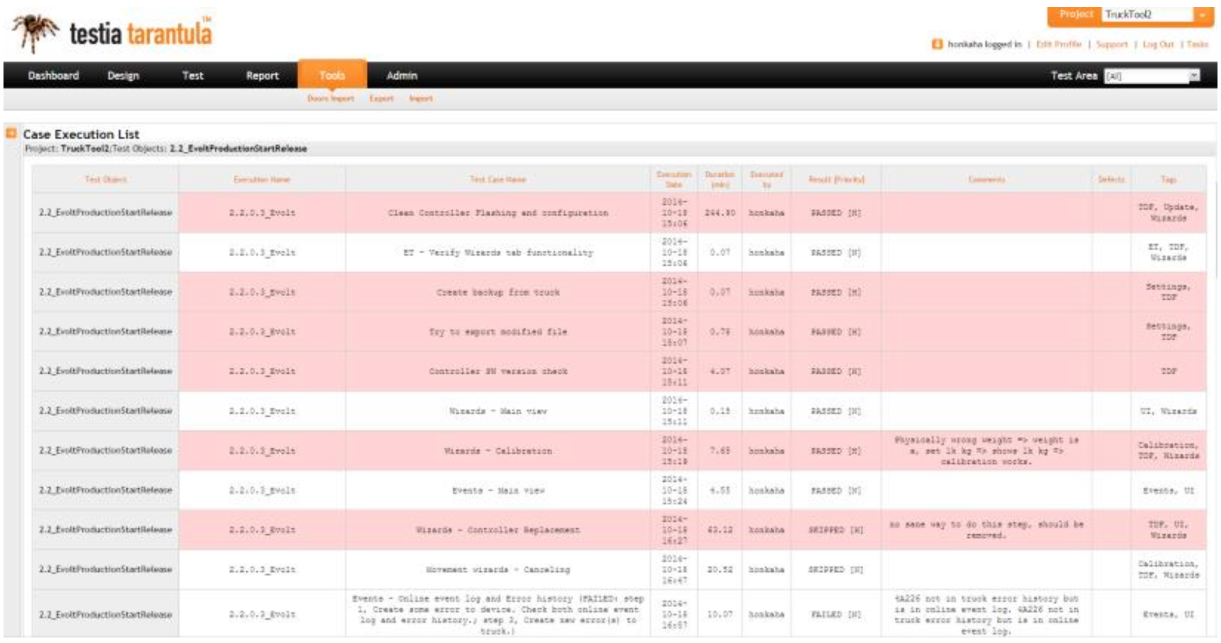
Ticket	Summary	Status	Owner	Type	Priority ▲	Milestone
#240	Genshi gets relative path wrong for <xi:includes	reopened	cmlenz	defect	major	0.5.1

Milestone: 0.6.1 (16 matches)

Ticket	Summary	Status	Owner	Type	Priority ▲	Milestone
#151	Undefined behavor of extended iteration	new	cmlenz	defect	major	0.6.1
#160	Genshi builder incorrectly leaks namespace scope to children	new	cmlenz	defect	major	0.6.1
#253	Improve handling of default namespaces	assigned	cmlenz	defect	major	0.6.1
#268	Match templates cannot access function variables	new	cmlenz	defect	major	0.6.1
#298	Indenting differences causes multiple identical msgids	new	cmlenz	defect	major	0.6.1
#323	Error message is confusing if <py:for ...> lacks each=""	new	cmlenz	enhancement	major	0.6.1
#342	py:match does not match class	new	cmlenz	defect	major	0.6.1
#380	messages within expressions in py:with directive don't get extracted	new	cmlenz	defect	major	0.6.1
#393	The ignore_tags setting does not work with Genshi templates	new	cmlenz	defect	major	0.6.1
#420	XInclude drops xmlns="http://www.w3.org/1999/xhtml"	new	cmlenz	defect	major	0.6.1
#425	Wish: <py:call lambda="func">FooBar</py:call> Resulting in func(stream) call	new	hodgestar	enhancement	major	0.6.1
#518	Added support for new HTML5 input types	new	hodgestar	enhancement	major	0.6.1
#4	Better handling of namespace context	assigned	anonymous	defect	minor	0.6.1
#373	Weird 'StripDirective' object is not iterable error	assigned	cmlenz	defect	minor	0.6.1
#386	py:match error when XPath selection returns multiple nodes	new	cmlenz	defect	minor	0.6.1
#396	select is added to the variables too late in the process of matching the stream	new	cmlenz	defect	minor	0.6.1

Picture 6. Trac task listing board. Tasks are group by sprints or milestones. Inside a group the tasks are then grouped by status and priority. Generally tasks waiting to be done are at top, tasks under execution or testing in the middle and tasks done and tested at the bottom. Colours are also used for showing status of task. This example is taken from public Trac website. ((Trac, 2015)

The execution of these tasks by developers is monitored via task list report inside Trac as seen in picture 6. This board is a variation of Kanban and Scrum boards. Perhaps the biggest visual difference that in typical agile boards the tasks are presented sideways where tasks waiting to be done are on the left and tasks already done on the right.



Case Execution List

Project: TruckTool2/Test Objects: 2.2_EvoltProductionStartRelease

Test Object	Execution Name	Test Case Name	Execution Date	Duration (min)	Executed by	Result (Priority)	Comments	Defects	Tags
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Clean Controller Flashing and configuration	2014-10-18 15:06	244.90	honkaha	PASSED [H]			TDF, Update, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	ET - Verify Wizards tab functionality	2014-10-18 15:06	0.07	honkaha	PASSED [N]			ET, TDF, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Create backup from truck	2014-10-18 15:06	0.07	honkaha	PASSED [H]			Settings, TDF
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Try to export modified file	2014-10-18 15:07	0.78	honkaha	PASSED [H]			Settings, TDF
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Controller SW version check	2014-10-18 15:11	4.07	honkaha	PASSED [N]			TDF
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Wizards - Main view	2014-10-18 15:11	0.15	honkaha	PASSED [N]			UI, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Wizards - Calibration	2014-10-18 15:19	7.65	honkaha	PASSED [H]	Physically wrong weight => weight is x, set 1k kg => shows 1k kg => calibration works.		Calibration, TDF, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Events - Main view	2014-10-18 15:24	4.55	honkaha	PASSED [N]			Events, UI
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Wizards - Controller Replacement	2014-10-18 16:27	63.12	honkaha	SKIPPED [H]	no sane way to do this step, should be removed.		TDF, UI, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Movement wizards - Canceling	2014-10-18 16:47	20.52	honkaha	SKIPPED [N]			Calibration, TDF, Wizards
2.2_EvoltProductionStartRelease	2.2.0.3_Evolt	Events - Online event log and Error history (FAILED: step 1, Create some error to device. Check both online event log and error history.; step 3, Create new error(s) to truck.)	2014-10-18 16:57	10.07	honkaha	FAILED [N]	4A226 not in truck error history but is in online event log. 4A226 not in truck error history but is in online event log.		Events, UI

Picture 7. Tarantula, a web based test case, set and execution management tool.

Currently there is no unified tool for manual testing used for software projects but in TruckTool project we are using Tarantula. Tarantula is a web-based tool for creating test cases and sets of cases. These cases or sets can then be formulated for test execution plans linked to specific software objects (e.g. releases). These test executions are also done inside Tarantula with an intuitive UI where the tester sets the result of the test and optionally some comments about the execution. Results of these sets automatic date and time information are stored inside Tarantula but can also be exported as PDFs or Excel spread sheets.

5.2.2 Truck embedded software projects

Truck embedded software is typically done by the manufacturers of the embedded controllers or some other vendor with ties to the controller manufacturer. The methods and processes used in different embedded projects vary greatly but what most of them have in common are the creation of initial monolithic specification document and defect and some part of task tracking inside Trac. Also the integration testing in actual truck is done by Rocla.

Typically these projects start with the creation of a design documentation which encompasses all the functions required form the embedded software. This document is typically very detailed with a lot of charts etc. to give detailed specifications of the functions. This document is then presented to the software vendor after which they go to work. At this point the visibility of the development status becomes obscured until after some time the first version of the software is delivered to Rocla for testing. After the initial delivery all the defects and feature wishes are recorded as tasks in Trac and a similar monitoring of their statuses is then done as in software projects. In this phase a lot of communication with the vendor is required and several new releases with bug fixes are typically delivered in short time window. Most of the communication in this phase is done by email with occasional visits either to Rocla by the subcontractors or Rocla personnel going to subcontractor premises. During this development testing phase also automatic testing of the new software is done with a hardware-in-the-loop automated tester.

After the development work is done to the point that the truck itself is beginning to be ready for delivery, a formal manual testing phase for the software starts. This testing is done according to current standards (EN1175) and as usually is the case with safety-critical applications tends to take a long time. Currently there are no specific software testing tools used for this but planning and reporting is done mostly in word documents and Excel spread sheets. All the resulting documentation is stored and version controlled in Aton PDM.

5.3 Problems

This chapter analyses the perceived problems in current processes and the challenges that we are facing with the coming of new version of safety standard with considerably more strict requirements for software development. This chapter deals only with service tool development from software project side leaving other pure software projects out of scope as they are not affected by EN1175 changes. Embedded software projects are considered in general.

5.3.1 New version of EN1175 and the adoption of EN ISO 13849-1 development standards for software development

In current version of EN1175 the requirements specifically for software are limited to having certain features with no limitations on processes, methods or documentation done in development of software and focuses on the features of truck functioning in a certain way and testing the truck as whole system with no notion of testing the software specifically. There is one particular requirement for software which is that the operator of the truck cannot be able to modify any safety-related parameters on truck in such way that it can cause danger to the operator or environment. (EN 1175-1:1998+A1, 2010)

The current draft of the new version of EN1175 implies using EN ISO 13849-1 requirements for all software development activities with forklift trucks. This is brings a very large change to software development processes and methodologies compared to the current situation. Software has to now be done in a way that it can be proven to fulfil the standards. (EN ISO 13849-1, 2008. EN 1175-1padova, 2014)

5.3.2 Service tool development

Based on requirements for software based parameterization for truck service tools from SFS-EN ISO 13849-1 the requirements for desktop software are to have a verification process for the parameters written to device (See chapter 4.3). On top of this there should also be a formal documentation for auditability.

The minimum documents should include:

- Requirements
- Design specifications
- Test case specification
- Test results
- Release notes

Doing this kind of documentation also tends to cause improvements in the quality of the software when done correctly and at correct time in development project.
The biggest problems in the current process when viewed from the point of fulfilling EN ISO 13849-1 requirements are not having any formal requirements, limited design specifications only in the form of tasks in Trac and very informal release notes. Big problem is also the traceability of all the documentation to each other. Trac and Tarantula do not integrate to each thus leading to having no direct link between specifications and formal testing done before each release.

5.3.3 Truck embedded software development

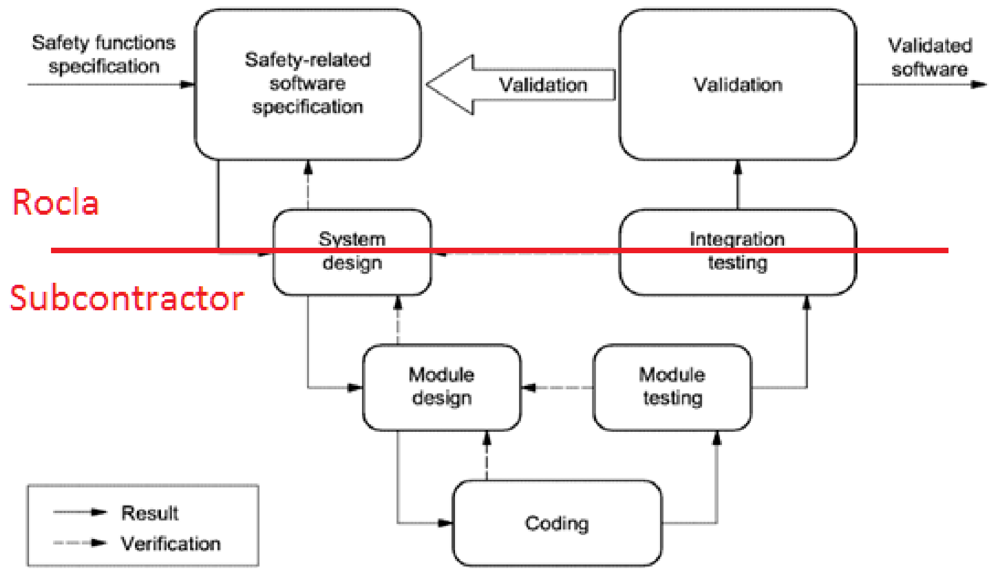


Figure 12. Separation of development activities illustrated in software development V-model between Rocla and subcontractors. (EN ISO 13849-1)

Problems in embedded projects can be separated to two levels from which the higher level is handled by Rocla and lower by subcontractor. Figure 12 illustrates this in relation to the V-model used for determining different levels of activities in context of whole project.

The current specification document used in truck embedded software development is a mix of requirements and design specifications with some test cases added alongside. In the V-model this document roughly fits somewhere in between system design and safety-related software specification. There is no clear separation which part are requirement specifications and which is design documentation. Parts of the “System design” box are also done as Trac tickets.

Current formal testing before releases is done not specifically for the software but the truck as whole and is not linked to the software specifications. In the V-model this can be seen as fulfilling “Validation”-box on the right side. Integration testing is currently done very informally against the Trac tasks to verify the finished status of each task.

Perhaps the biggest problem on Rocla side of the development is having traceability between each of these tasks. For each specification there should a corresponding test that clearly verifies that the requirements are met. In documentation this would mean having a way to determine from the result. Also the traceability of test result to software versions could be improved. This problem is also magnified by not having enough co-operation in between testing and design departments until very late stages of the project which causes the test specifications to be done in haste when the product is already functional.

For the subcontractor part there currently is no visibility to the development processes for Rocla and we cannot verify if the subcontractors are fulfilling the requirements imposed by standards. The biggest problem for Rocla in this respect is the need to get the documentation and test reports about the software development done by subcontractor for filing and auditability. The processes and methods for handling this are currently either missing, not implemented on day to day basis or are very cumbersome to implement. One example of this is the current design document. Due to its large size there is a perceivable threshold in updating it on every minor design change leading on it to be outdated or lacking details compared to current actual design. Also on the occasions it is updated it tends to not be used as a method of communication due to it being easier to communicate the changed specifications by some other means e.g. Trac tasks or by email messages. This leads to writing so called “dead documentation” which fails to fulfil its duty to as a means to communicate the design requirements to the subcontractors.

5.4 Improvements

In this chapter several different ways for improving the processes and methods used in service tool and truck embedded software development are analysed. The biggest points of improvement are identified and the way to proceed is selected.

5.4.1 Service tool development

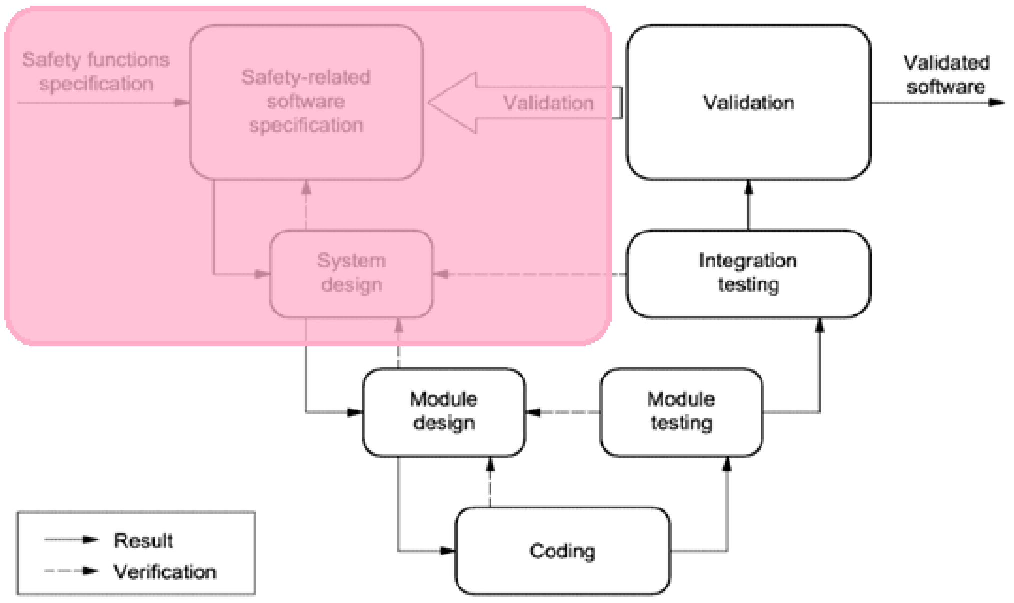


Figure 13. Part of the V-model currently found lacking in service tool development. (EN ISO 13849-1)

The biggest improvements to be made in service tool development are the documentation of requirements and formal designs. Currently neither of these activities is done in a way that would fulfil the requirements for safety-critical software development processes. Formal system design is currently on the level of having a rough UML diagram of the architecture.

To improve this there should be documented requirements filtered from the standards to apply specifically for this application. Designs to fulfil those requirements should then be created and linked to these requirements. This documentation would also include having a way to link the current tasks and test specifications and results to these requirements. Currently the systems used for development (Trac and Tarantula) have no way to trace tasks and tasks to each other so a new way to do this has to be found.

5.4.2 Truck embedded software development

The current design documentation for trucks should be broken down to 2 different documents with 1 holding the requirements and 1 holding the system level designs to fulfil these requirements. This would improve the ability to understand what the standards require form truck and how these requirements are specifically going be met. Also having clearly separated requirements and design specifications would improve the ability to create tests more neutral to the technical solutions and focusing on proving the requirements to be fulfilled.

Test specification work should also start at the same time as truck specifications are made so that the test should be ready for execution once the development is finished. This would balance out the workload of testing throughout the whole project and lessen the bottleneck effect it currently creates at a critical phase of the project. Also having a larger crowd participating in the initial phases of project can lead to finding design defects before they have been implemented. Better documentation for the integration phase of testing is also required. Test specifications have to be documented clearly enough so that the test can be easily recreated for future test rounds and the exact software versions the test have been executed with have to be recorded alongside test results.

Separating the requirements and design specifications form each other would also lead to lighter and less monolithic design documents. This would improve the ability to use these formal design documents in addition to fulfilling the documentation needs imposed by standards but also as a vessel of communication between Rocla and the subcontractors. This would to more efficiency in day to day work by eliminating duplicate work.

Currently major part of the documentation in truck embedded software projects is done by using word documents and excels spread sheets. As the complexity of documentation increases the less effective this becomes. Especially traceability between design artefacts becomes exponentially more problematic when the amount of documentation rises. Also the revision control and change tracking is difficult and relies on manually documenting the changes in between revisions when compared to systems specifically created for this kind of work.

A process to integrate formal design and testing documents from subcontractor to Rocla system has to be implemented. This process needs to be continuous so that up-to-date versions of these documents are always delivered alongside new software binaries so that these documents can be filed whenever a release to field is made.

5.4.3 Conclusion

When looking at areas needing improvement for service tool and truck embedded software development it can be seen they have a lot of common problems. Perhaps the biggest area of improvement in both them is the need to improve documentation. Especially the traceability of data in different documents needs improvement. A clear solution for addressing most of these problems is the introduction of comprehensive application lifecycle management toolset for management of all the activities described in the V-model for software development.

Toolset alone is not enough but also the processes and methods on how to use this kind of toolset in the most effective and lean possible way have to be studied and defined. This evaluation work should be done as cooperation between different departments involving personnel form all aspects of projects for the best possible outcome and selection of most usable and acceptable solution. These processes and methods also should not be fixed but instead should be in constant state of evolution with adaptations made according to each projects special features and the working habits and even the preferences of the involved personnel. For the scope of this thesis the tool selection should be made. The work on defining processes should also be started and visualizing the future steps outside the scope of this thesis should be done.

5.5 Options for application lifecycle management solution

There are multiple tool vendors providing either complete suites of tools for application lifecycle management or tools concentrating on certain aspect of the lifecycle. Especially for test case, result and execution plan management there are several different kinds of tools available. There are also solutions available that are specifically tailored for safety-critical product development.

Some of the biggest selection criteria were easy adaptability to our heterogeneous project needs and the ease of deployment and usage as this would not be done as separate project but as part of two on-going projects, on-the-fly as one could say. One big selection criteria was also the cost of the solution for approximately 10 concurrent users. To ease the initial threshold of adaptation also the preference for cloud-based with a subscription based payment model was preferred over having to install the system on premises.

One requirement also is that some way to function with Aton PDM has to be realized. Currently all the product data in Rocla Oy is stored inside Aton PDM and this should be the case also for software projects. Aton has an inbuilt change management system with some issue tracking features but they are not on par with modern tools associated with software development and so they are not considered as an option for ALM solution. (Aton, 2015)

5.5.1 Modifying Trac and Tarantula to fulfil new requirements.

First and most logical option for process was to explore the possibility of using or improving current Trac and Tarantula systems for fulfilling the new requirements. There is also a Trac plugin called “Test Manager” for creating test cases, execution plans and tests reports. Tarantula has in addition to its test management features also requirements management with good report exporting features.

Analysis of how requirement management and traceability could be done with Trac led to the conclusion that there is no good inbuilt or plugin-provided way to document requirements or design documents. Also the Test Manager plugin was found to be difficult and cumbersome to use. Traceability between artefacts was also missing and would have required developing custom made plugin. This was seen as too expensive and time consuming solution.

5.5.2 IBM Rational Doors

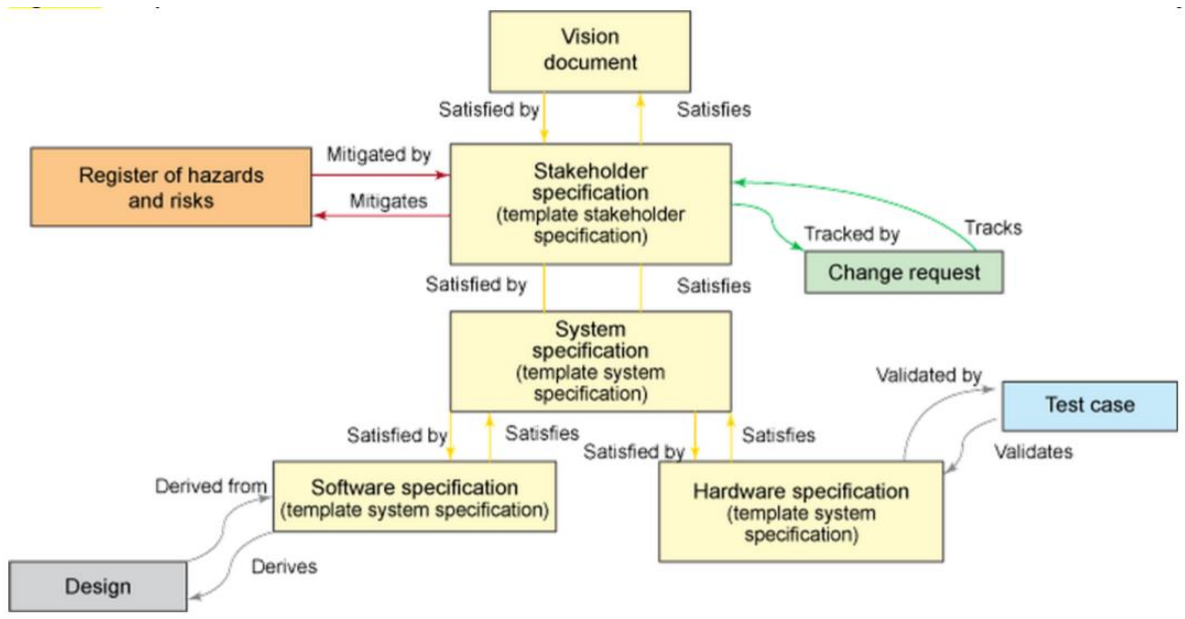

Vision document
Satisfied by
Satisfies
Register of hazards and risks
Mitigated by
Mitigates
Stakeholder specification (template stakeholder specification)
Tracked by
Tracks
Change request
Satisfied by
Satisfies
System specification (template system specification)
Validated by
Test case
Validates
Satisfied by
Satisfies
Satisfied by
Satisfies
Derived from
Software specification (template system specification)
Hardware specification (template system specification)
Design
Derives

Figure 13. Links between documents and design artefacts in Rational Doors. (IBM, 2015)

One of the most famous and commonly used application lifecycle management solutions for development of safety-critical products is IBM Rational DOORS. It has many features including full linking and traceability of design documents as illustrated in figure 13. IBM Rational DOORS also have a cloud based solution available called “Next Generation”. (IBM Rational DOORS Next Generation datasheet, 2015)

The DOORS products seem to emphasize the requirements engineering and standard compliance aspects of the project and are no tailored for software development specifically. They also do not provide task tracking and test management at the same levels as some other solutions. If DOORS were chosen it probably could not alone fulfil all the needs of application lifecycle management and would need additional solutions to be taken into use alongside.

One big factor to take in account with DOORS is also the price which is for the “Next Generation”-cloud in the ballpark of tens of thousands of euros per year. This combined with the fact that DOORS alone is not enough speaks against choosing DOORS.

5.5.3 Polarion

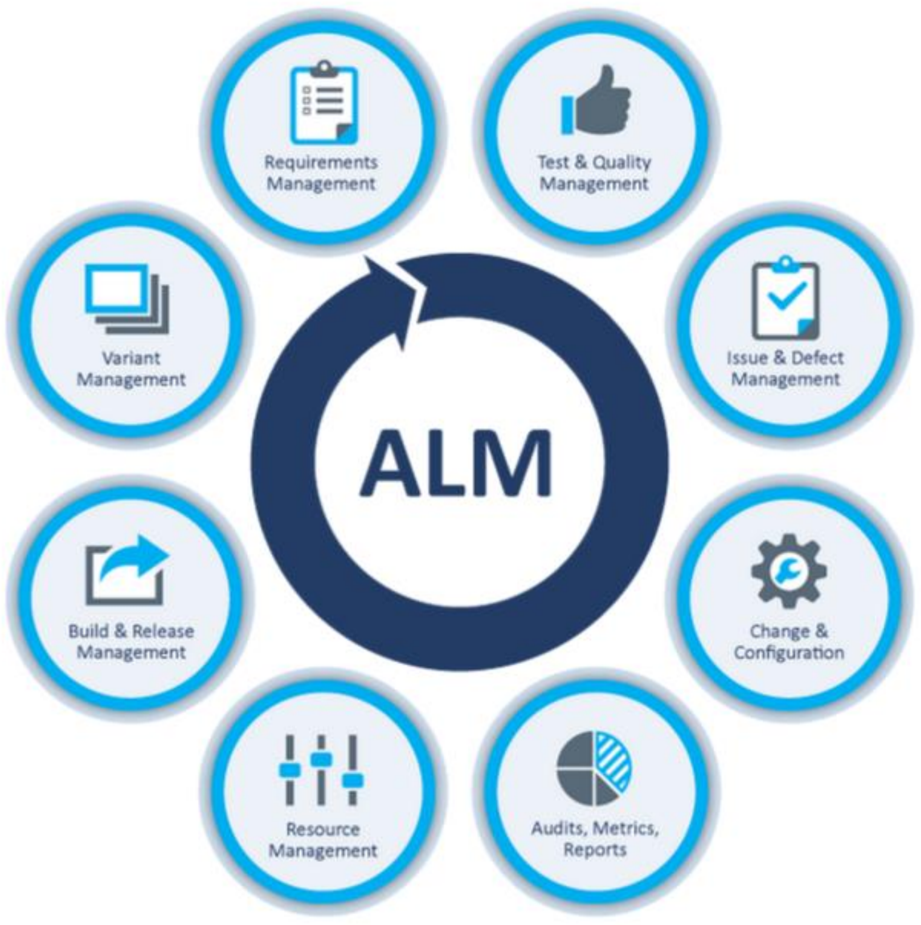


Figure 14. Polarion features overview. (Polarion ALM, 2015)

Polarion ALM is product that integrates all the aspects of safety-critical software development under one solution. Its features include requirements management, test management, source control integration, task and issue management and reporting tools. It is also recognized by TÜV NORD as ISO 26262 and IEC61508 compliant out of the box. Polarion ALM is available either as on premise installable version or as SaaS-solution in web. (Polarion ALM, 2015)

From feature point of view Polarion ALM looks promising and looks like it would fulfil all the requirements for ALM we have. Polarion ALM also has plethora of plugins available and would integrate well with our existing systems. Again as a main problem rises the price. For both on premise and SaaS-versions the initial or per year price tags rise to the ballpark of tens of thousands

5.5.4 Redmine

While not an actual ALM solution a look was taken into Redmine due to one of our biggest subcontractor being considering its adaptation for their development work and having unified systems would have some benefits. Redmine provides the same basic features as Trac including issue management, integrated wiki and issue reporting views. Redmine is free and open-source. Redmine does not seem to offer anything over our current Trac solution. There are possibilities to include files and wiki pages as there is in Trac but no easy solution for requirements management or traceability is provided. (Redmine, 2015)

5.5.5 Microsoft Team Foundation Server

Microsoft Team Foundation Server has established itself as one of the leading ALM solutions. It offers requirements management, task tracking, source repository integration and test management features for manual and automated testing. It also integrates well with Visual Studio and is provided along MSDN subscriptions that developers using Microsoft technologies regularly have. Source control integration is provided either via Team foundation Version Control or Git. Team foundation server can be used with different development environments but provides most support when using either Visual Studio or Eclipse. TFS also provides Release Management tools for automated deployment of releases to dev, test and production environments. (Team foundation server, Wikipedia, 2015)

If development was done only using Microsoft technologies with developers already having MSDN subscriptions then TFS would be a natural choice due to its maturity and completeness of features. In our case roughly half of our development is done using Microsoft technologies with a lot of variance on the other half. In several projects also the source code itself is not visible to us so some of the features in TFS are not useful. We currently also do not have any active MSDN subscriptions. TFS Express edition is free for 5 or less users, otherwise the minimum costs would be about 500$ for the server and then 500$ per user. (Rehnstrom, 2015)

5.5.6 JIRA, Confluence and other products in Atlassian ALM tool family

JIRA is a tool made by Atlassian. JIRA is a project management and issue tracking software with a web UI as is usual for this kind applications. Although JIRA alone does not provide all the features of an ALM solution it is commonly used as a name for the product family that includes other Atlassian tools including Confluence, Stash, FishEye and a plethora of commercial or free plugins sold in Atlassian marketplace. JIRA has an advanced issue tracking system with customizable workflow, agile development workflow features, task tracking boards and multi-tiered issue types. (JIRA, Wikipedia, 2015)

Confluence is a sister product of JIRA. Confluence provides a wiki solution with easy to use editing capabilities in contrast to traditional way of editing Wikipedia through special "wiki-formatting" style which incorporates programming languages features in text editing. Editing capabilities are close to those of Microsoft Word with capability to create tables and include pictures in the documents. Features also include version control of documents. Due to good document editing features, good search features and high user count, Confluence is sometimes valued as the best commercially available wiki solution. Confluence integrates seamlessly with JIRA allowing creation and linking to JIRA issues directly from the wiki pages. (Confluence, Wikipedia 2015.

To get full features of ALM with source code and test management capabilities, one has to select several different modules form Atlassians product family. For our case these modules are:

- JIRA for project management and issue tracking.
- Confluence for requirements and design document management.
- Stash for source code integration.
- Zephyr or some other JIRA plugin for test management integration.

Atlassian and its plugin vendors typically start the pricing for each module from 10€ for 10 concurrent users. If user amount is increased to maximum 25 users the prices typically rise to around 1000€ per module. These prices mean that we could start with around 50€ investment and if our pilot adaptation is successful the next step would cost less than 10000€. (JIRA licensing and pricing, 2015)

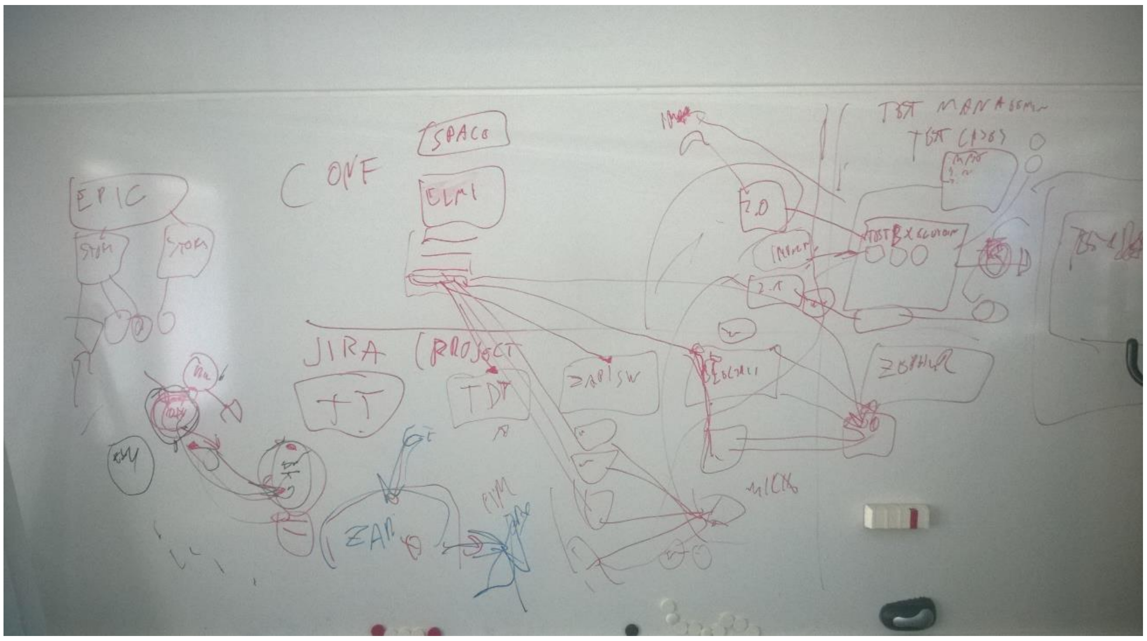
5.5.7 Tool selection

ALM Solution	Notes	Cost (for 10 users)
Modified Trac and Tarantula	On premises. Features developed by ourselves.	Several man-months' worth of development. Extensive hidden costs.
IBM Rational Doors Next Generation	SaaS. Full features.	20 000 €/year. Minimum contract 3 years / 60 000€. (Ballpark figures estimated in initial quote from IBM.)
Polarion ALM	SaaS or on premises. Full features.	SaaS: $17880/year On premises: $24900
Redmine	On premises. Lacking features or developed by ourselves.	Several man-months' worth of development. Extensive hidden costs.
Microsoft Team Foundation Server	On premises Full features but only for software projects.	Estimated $5500. Around $500 per user + $500 for server license.
JIRA	SaaS or on premises. Full features if several modules bought.	SaaS: 10€/module/year. Max 100€/year in our use. On premises: Max 100€ depending on bought modules.

Table 1. ALM solution feature and price comparison table.

During the evaluation of different options for ALM solution it quickly became evident that there are several different options and it would be impossible to evaluate all of them so evaluation was limited to some of the most popular or otherwise relevant to us. It quickly became evident that open source solutions (Trac & Tarantula) would not be able to provide a comprehensive package.

When evaluating commercial solutions the price tag started to play a big role. Due to us being a small company with limited resources the most expensive options were quickly eliminated simply on the basis of their price tags without much effort going into their actual evaluation. See table 1 for comparison of pricing and features. With price playing a big role JIRA was quickly selected for further evaluation. The fact that Atlassian is considered one of the three leading ALM providers alongside Microsoft and IBM (Gartner, 2015) also fortified the decision to select it for detailed analysis.



Picture 8. Sketch made during JIRA evaluation workshop on how V-model could be implemented in our project environment using JIRA, Confluence and Zephyr (test management plugin for JIRA).

Demo installations of JIRA and Confluence were made and evaluation of them was conducted. This testing was done by first letting personnel from both software projects and truck embedded software projects to familiarize themselves with the UI and workflow. After this an evaluation of how using JIRA and its sister products was conducted in a form workshop. During this workshop the features of JIRA, Confluence and Zephyr were tested and an idea of how to use them for our development work was created. A rough sketch shown in picture 15 was created of how our current development activities and project relationships could be realized using Atlassian ALM product family. General consensus was that this toolsets would perceivably suite us well. Decision was made to start piloting JIRA usage in two projects. One project being a pure software project done in-house and another truck embedded software project with low level design done by subcontractor. Having the ALM solution piloted by both types of software related projects we are currently conducting shall give quick feedback on the suitability of JIRA for being our ALM solution choice. Direct integration to Aton is not available but using either inbuilt export features or specific exporting plugins all the data in JIRA can be exported to a form which can be brought into Aton for archiving. As this would be required to be done only on a release of the software the workload is not extensive even if done manually.

5.6 Development process with Atlassian ALM solution

In this chapter the features of different modules in Atlassian ALM solution are explored and an idea of how to use each of them as building block for a solution to realise V-model process is formed. This process can be broken down to 3 major parts: requirement and design document management, task, issue and bug tracking and test management. These tasks are realised with different modules and plugins. One important aspect to explore is also how these different modules link to each other. When comparing ALM solutions to doing documentation on standalone documents this is perhaps the biggest advantage.

5.6.1 Using Confluence for requirements and design documents

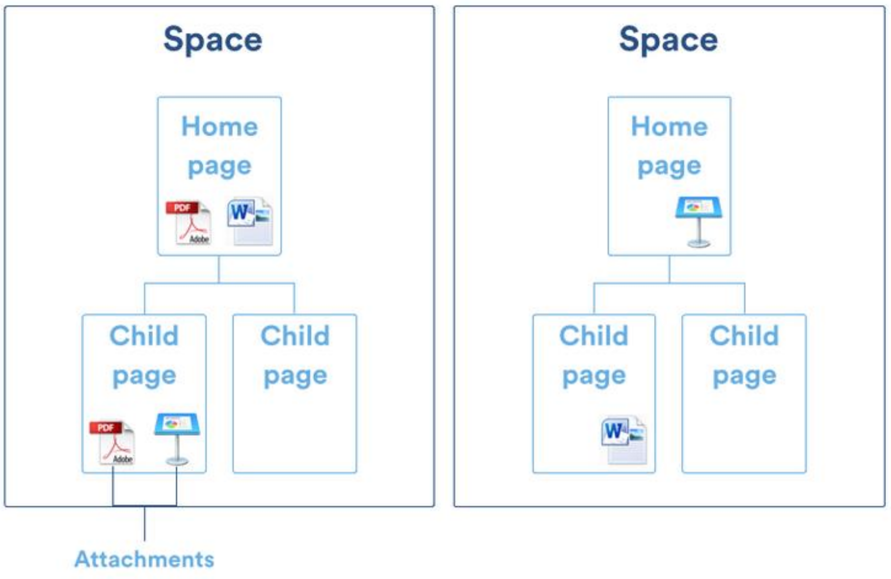
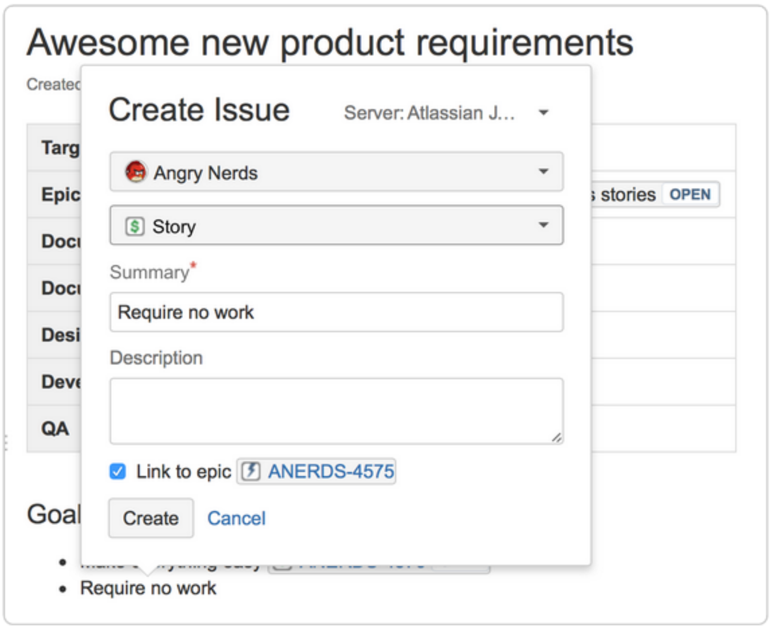


Figure 15. Space and page organization as a tree inside Confluence. (Confluence User's Guide, 2015)

Confluence organises its data in modules called "spaces". Each space can contain several pages in a tree-like organization presented in in figure 15. Each page can contain one or more attachments, e.g. pdf documents and excel spread sheets. These spaces can be public or private. One could use a private space to store personal notes. Each site has an admin which is typically the person who created the space. This admin then controls access and modifying rights for that space or page. (Confluence User's Guide, 2015)


Awesome new product requirements

Create Issue

Server: Atlassian J...

Angry Nerds

Story

Summary*

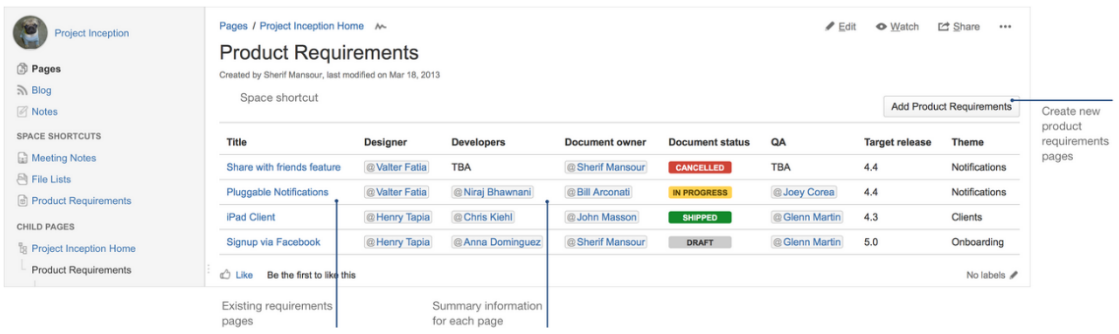
Require no work

Description

Link to epic ANERDS-4575

Picture 9. Illustration of how JIRA issues can be created by highlighting text inside Confluence. (Confluence User's Guide, 2015)

Confluence supports creating JIRA issues by highlighting text inside a page. A popup as seen in picture 9 with issue creation controls is then shown to user for filing in all the relevant data. After issue has been created and popup closed, Confluence creates a link next to text used for issue creation providing easy and direct access to that issue inside JIRA instance that has been linked to Confluence.



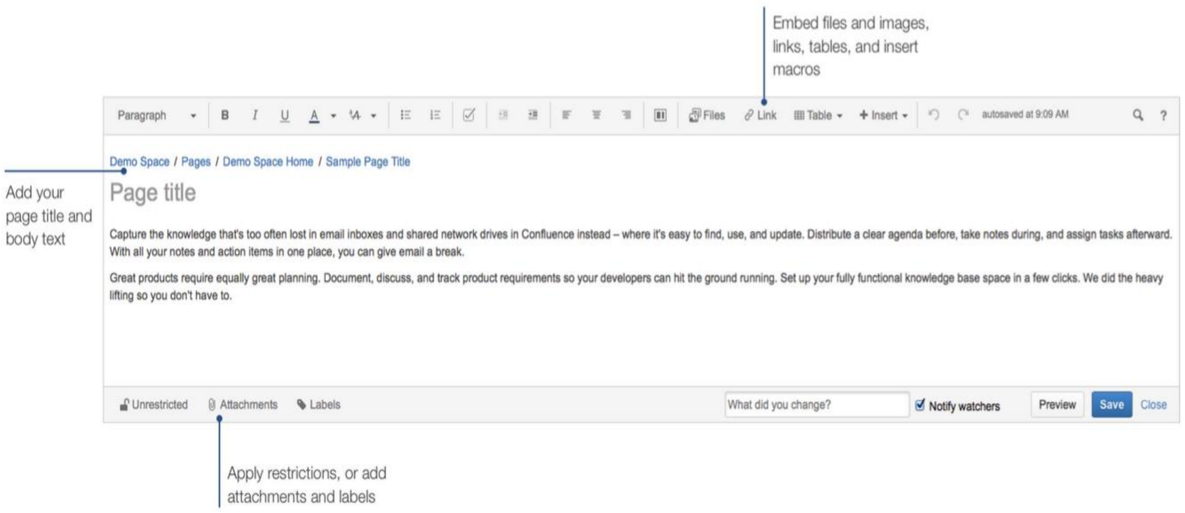
Product Requirements

Created by Sherif Mansour, last modified on Mar 18, 2013

Title	Designer	Developers	Document owner	Document status	QA	Target release	Theme
Share with friends feature	@Valter Fatia	TBA	@Sherif Mansour	CANCELLED	TBA	4.4	Notifications
Pluggable Notifications	@Valter Fatia	@Niraj Bhawnani	@Bill Arconati	IN PROGRESS	@Joey Corea	4.4	Notifications
iPad Client	@Henry Tapia	@Chris Kiehl	@John Masson	SHIPPED	@Glenn Martin	4.3	Clients
Signup via Facebook	@Henry Tapia	@Anna Dominguez	@Sherif Mansour	DRAFT	@Glenn Martin	5.0	Onboarding

Picture 10. Example of a page created from “Product Requirements Blueprint”. (Confluence User’s Guide, 2015)

There are several special page templates or blueprints available for using. Good example of blueprint is “Product Requirements Blueprint” which provides readymade functionality to link requirement documentation residing inside Confluence to issues inside JIRA. This page then shows the status of the realisation of those issues directly inside the Confluence page. (Confluence User’s Guide, 2015)


Embed files and images, links, tables, and insert macros
Paragraph
Files
Link
Table
Insert
autosaved at 9:09 AM
Demo Space / Pages / Demo Space Home / Sample Page Title
Add your page title and body text
Page title
Unrestricted
Attachments
Labels
What did you change?
Notify watchers
Preview
Save
Close
Apply restrictions, or add attachments and labels

Capture the knowledge that's too often lost in email inboxes and shared network drives in Confluence instead – where it's easy to find, use, and update. Distribute a clear agenda before, take notes during, and assign tasks afterward. With all your notes and action items in one place, you can give email a break.

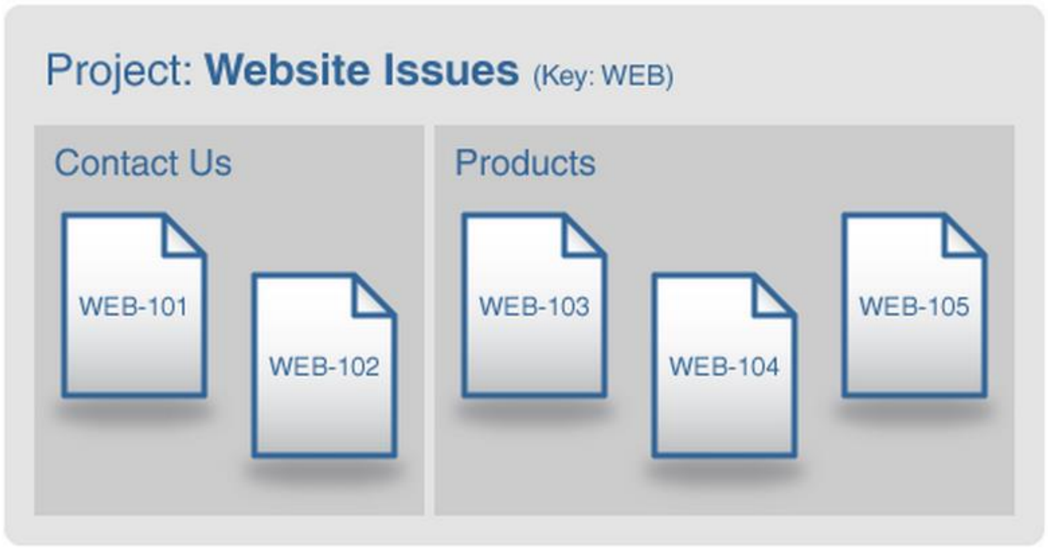
Great products require equally great planning. Document, discuss, and track product requirements so your developers can hit the ground running. Set up your fully functional knowledge base space in a few clicks. We did the heavy lifting so you don't have to.

Picture 11. The Confluence Editor used in editing pages inside Confluence. (Confluence User’s Guide, 2015)

The editing method for pages used Confluence is very similar to that of editing word documents with Microsoft Office. This can be seen as major advantage as most people working on our projects are very familiar and proficient Microsoft Office tools but not so much when it comes to traditional wiki-formatting. Having this kind of editor also provides quick way to migrate existing design documentation done as word documents chapter by chapter as separate pages inside Confluence by means of simple copy pasting. With wiki-formatting this would not be possible and all the used formatting would be lost and would have to recreated. (Confluence User's Guide, 2015)

5.6.2 JIRA for issue tracking

JIRA and JIRA Agile, a module of JIRA specifically tailored for agile development with integrated Kanban and Scrum boards, are tools for tracking tasks and issues. JIRA agile provides special tools associated with agile software development including scrum and Kanban boards and special workflows tailored to be used with agile development processes. (JIRA User's Guide, 2015. JIRA Agile documentation, 2015)



Picture 12. Example of JIRA project with issues grouped under components. (JIRA User's Guide, 2015)

The main concepts in JIRA are project, issue and workflow. Project is a collection of issues for a specific purpose. A typical example of a project would be software application. Each issue belongs to a project. Issues belonging to a project can be grouped under components as can be seen in picture 12. Inside projects there is also the concept of version. Issues can be linked against a specific version. This allows visibility and tracking of which issue is done in which version of the project. Inside a JIRA instance there can be several projects. Also user access can be differentiated between projects allowing customization of who can see which project. (JIRA User's Guide, 2015)

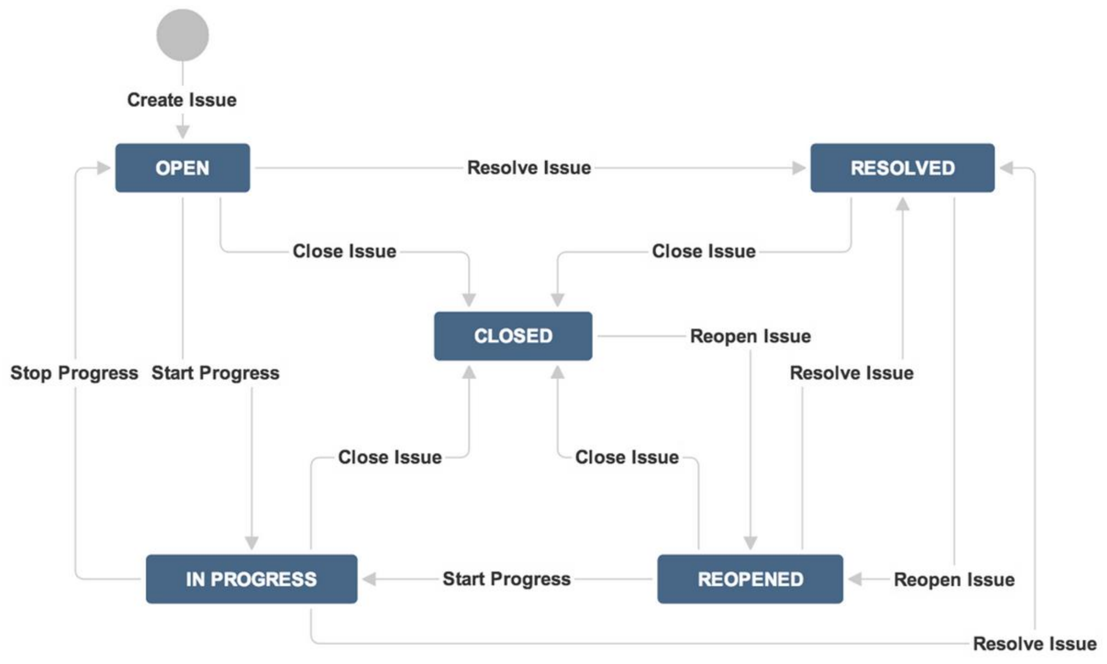


Figure 16. JIRA default workflow for issues. (JIRA User's Guide, 2015)

Issues are the basic artefacts handled inside JIRA. Typically an issue is a logical piece of work that is done by developer. Users can link files, comments, versions and project components and much more to issues. The data included in an issue can be highly customized inside JIRA. There can be several different kinds of issues with different data and workflow. The concept of workflow means that each issue always has a status it currently is in. These statuses can be for example "open" or "in progress" and this status should reflect on what is currently happening with the issue. These statuses normally have certain rules on how the users can transition from one status to another. For example from "closed" status user could only change the status to "reopened" if the JIRA default shown in figure 16 were used. These workflows can be customized to specifically suite the individual project needs. For this purpose a visual workflow editor is provided. There can be different workflows also depending on different types of issues. The contents of issues and projects can also be heavily customized. There are also several plugins available that can extend or modify the basic functionality of JIRA. (JIRA User's Guide, 2015)

5.6.3 Test management with Zephyr plugin

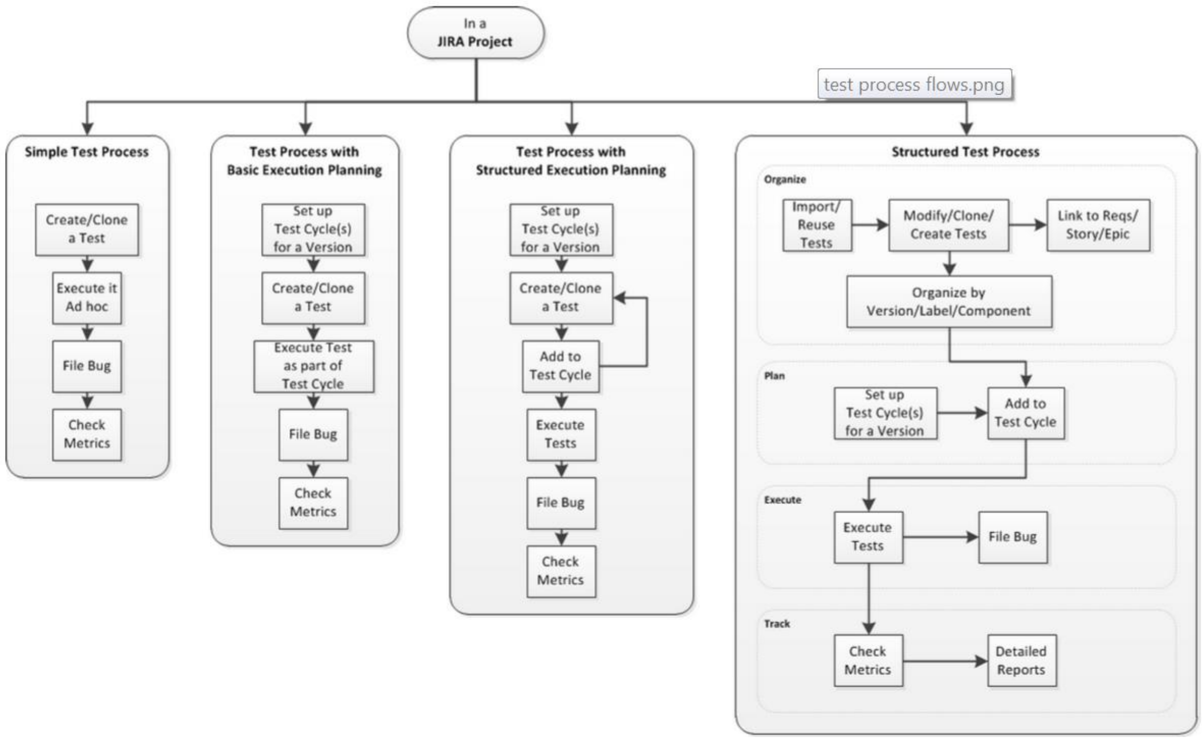
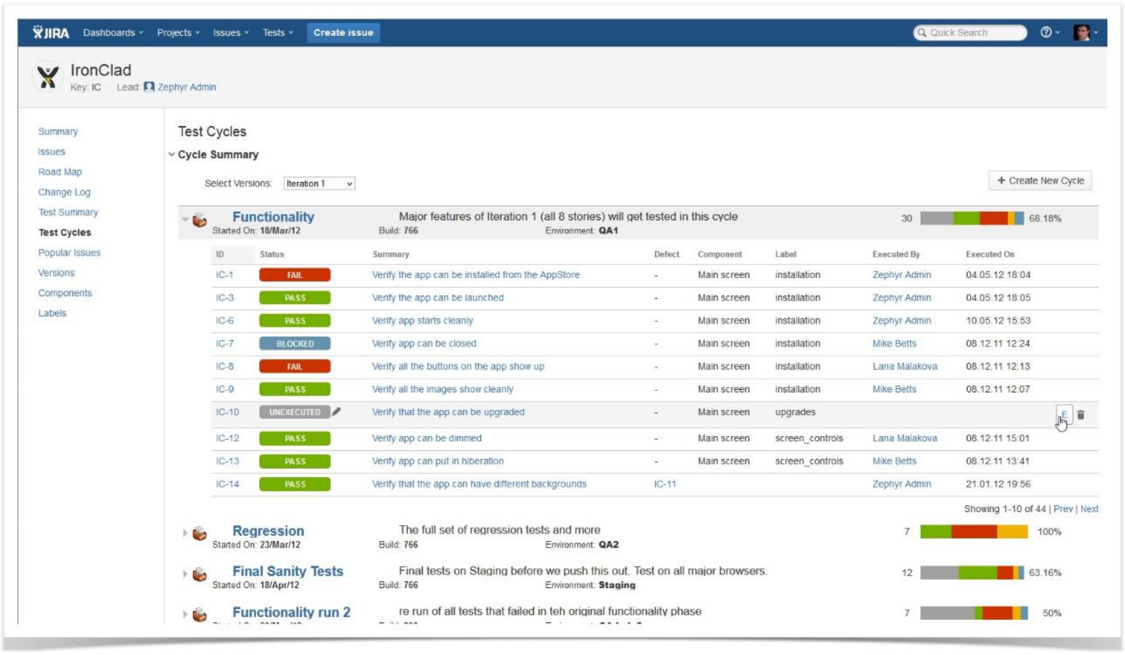


Figure 17. Different test process types usable with Zephyr test management plugin. (Zephyr for JIRA documentation home, 2015)

Zephyr is a plugin (or add-on) for JIRA which is sold separately. With Zephyr user gets special issue types that represents tests. In these issues the user can define different test steps with actions the tester has to take in order to execute the test and then set the result of the test. These tests can be grouped in execution plans and these execution plans can then be grouped under a version of a project. This provides the basic functionality of testing a software release and having test results for it. Some options for different types of processes that can be used in testing with Zephyr are illustrated in figure 17. (Zephyr for JIRA documentation home, 2015)



IronClad
Key: IC Lead: Zephyr Admin

Test Cycles

Cycle Summary

Select Versions: Iteration 1

Functionality Started On: 18/Mar/12 — Major features of Iteration 1 (all 8 stories) will get tested in this cycle — Build: 766 — Environment: QA1 — 30 — 68.18%

ID	Status	Summary	Defect	Component	Label	Executed By	Executed On
IC-1	FAIL	Verify the app can be installed from the AppStore	-	Main screen	installation	Zephyr Admin	04.05.12 18:04
IC-3	PASS	Verify the app can be launched	-	Main screen	installation	Zephyr Admin	04.05.12 18:05
IC-6	PASS	Verify app starts cleanly	-	Main screen	installation	Zephyr Admin	10.05.12 15:53
IC-7	BLOCKED	Verify app can be closed	-	Main screen	installation	Mike Betts	08.12.11 12:24
IC-8	FAIL	Verify all the buttons on the app show up	-	Main screen	installation	Lana Malakova	08.12.11 12:13
IC-9	PASS	Verify all the images show cleanly	-	Main screen	installation	Mike Betts	08.12.11 12:07
IC-10	UNEXECUTED	Verify that the app can be upgraded	-	Main screen	upgrades		
IC-12	PASS	Verify app can be dimmed	-	Main screen	screen_controls	Lana Malakova	08.12.11 15:01
IC-13	PASS	Verify app can put in hiberation	-	Main screen	screen_controls	Mike Betts	08.12.11 13:41
IC-14	PASS	Verify that the app can have different backgrounds	IC-11			Zephyr Admin	21.01.12 19:56

Showing 1-10 of 44 | Prev | Next

Regression Started On: 23/Mar/12 — The full set of regression tests and more — Build: 766 — Environment: QA2 — 7 — 100%

Final Sanity Tests Started On: 18/Apr/12 — Final tests on Staging before we push this out. Test on all major browsers. — Build: 766 — Environment: Staging — 12 — 63.16%

Functionality run 2 — re run of all tests that failed in teh original functionality phase — 7 — 50%

Picture 13. Zephyr UI for visualization of test plans and execution status. (Zephyr for JIRA documentation home, 2015)

Zephyr also has views and easy-to-use UI components for creating test plans and viewing test statuses. These UIs allow for pre-planned test cycles (tests plans) to be created and executed against a certain version of the project. Executing tests "ad hoc" is also facilitated allowing users to run single test cases without associating them to particular test cycles. Example of test cycle execution status view can be seen in picture 13. With Zephyr installation user also gets a top bar menu for quick access Zephyr features. What is missing in Zephyr that could have an impact on its selection as a test management tool is an easy ability to create and export test reports for each test run. Also linking of Confluence documents to test cycles is missing. For truck embedded software projects there is a special requirement of storing the setup information about the truck itself and its settings used while conducting tests is required. (Zephyr for JIRA documentation home, 2015)

5.6.4 Links between design artefacts and the big picture

Having tools for requirement and design management, issue and project progress tracking and test management is not enough. All these activities have to be linked to each other and with these links the lifecycle of each requirement from idea to design to implementation to test case and result has to be traceable. With Confluence, JIRA and JIRA plugins for test management these links are possible to create with a certain level of automation as all these products belong to same family and are linked to each other. (Confluence user's guide, 2015. JIRA user's guide, 2015)

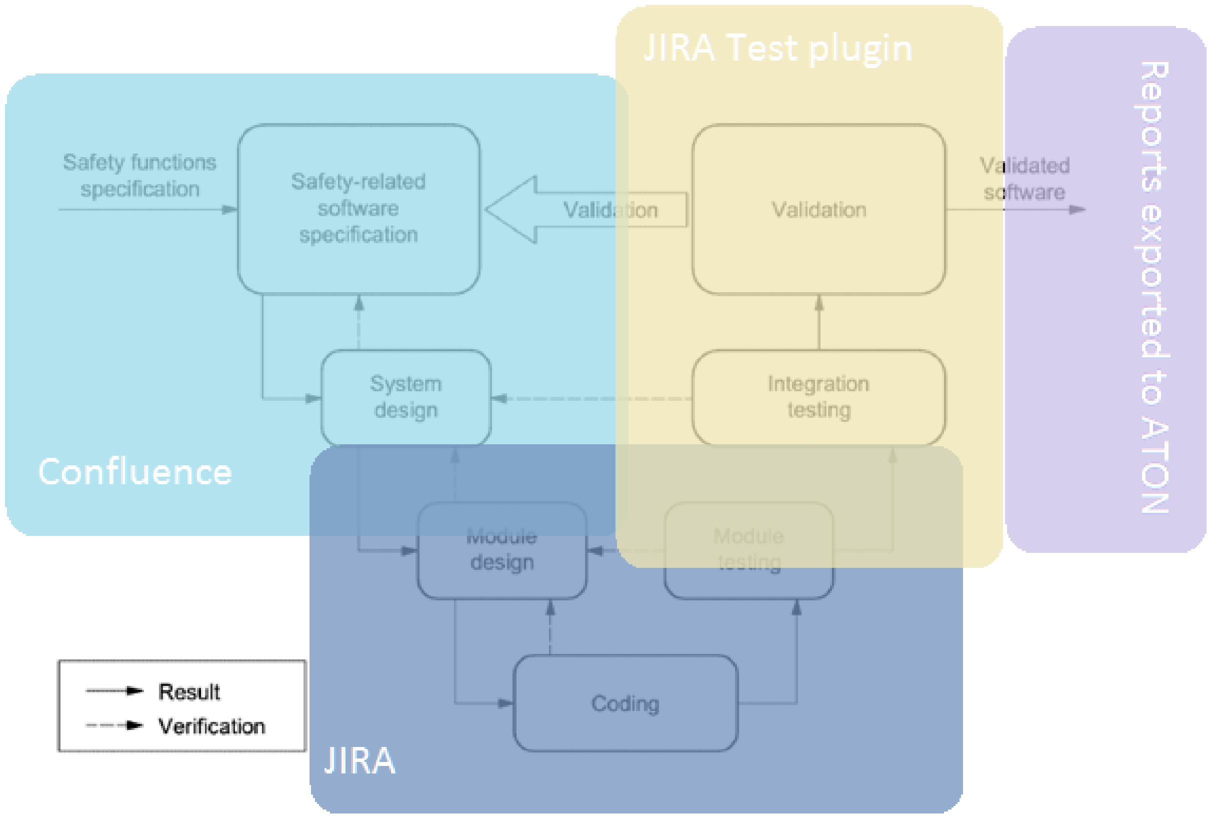


Figure 18. Big picture of how Atlassian products can be used in Rocla software development for compliance with EN ISO 13849-1 V-model process requirement. (EN ISO 13849-1, 2008)

Figure 18 shows the idea of how to use each part of Atlassian ALM solution for different parts of the V-model. Confluence is heavily featured on the specification and design phases and would replace current word and excel documents. JIRA would take the current position of Trac as an issue and project management tool. Zephyr or some other test plugin would handle part of test management currently done either with Tarantula or Excel spread sheets. At the end of the V-model process all the documentation involved for the software to be released then should be exported and archived into Aton PDM for official storage and possible change management processes required by e.g. controller manufacturer. Detailed descriptions of the processes to use should be created during and after the piloting projects where the best ways to use these tools to support our current ways of doing things are found. Current processes are also very informal with little detailed requirements so it is possible that some completely new process features and conventions shall be created during the piloting.

A typical lifecycle for a design artefact would go as follows:

1. A requirement is created on a Confluence page
2. A design fulfilling the requirement is created on a Confluence page and linked to the requirement.
3. Test case is created to confirm the requirement and design and is linked to both of them, either directly or through one or other.
4. Issue is opened against the design. A link to the issue is in place on Confluence page.
5. Issue is given to developer for implementation
6. Issue is finished.
7. Test case confirming the implementation against the design and requirement is executed.
8. Test result is recorded.

Some open questions on which kind of Confluence pages and templates or blueprints should be used for requirements and design documents. Also the question if design documents should be in Confluence or should they be done as Epics (high level issues with several levels of sub-issues linked in them). Overall though it looks like the V-model is implementable and taking JIRA in use instead of current multitude of systems would allow us to implement features not possible earlier and would reduce the overall workload required to realise all the required activities as opposed to doing all the documentation as separate documents with manual traceability matrixes.

6 CONCLUSIONS AND FUTURE WORK

This chapter contains information about the final status of the project at the time of writing this thesis. Also notions on what is still missing and needs more work are made. Finally the results of this work and its impact on the future of software development at Rocla Oy are analysed and reflected on.

During the writing of this thesis the acquisition of JIRA and Confluence instances for pilot projects has been done and the migration issues for current milestone of service tool development to JIRA is about to start. Some plugin selections for JIRA are still open and a real example of having whole project documentation in Confluence has not yet been evaluated. Although JIRA is currently perceived to be a good choice for us its real usefulness is yet to be seen once it is taken into full use and experiences for using it with collaboration with different subcontractors have been evaluated.

6.1 Finalization of the selection of test management plugin

Perhaps the biggest issue still open is the selection of test case management plugin for JIRA. So far only Zephyr has been evaluated in depth and it was found lacking some features considered very important for truck embedded software projects such as revisioning of test cases and the possibility to easily link the data of truck hardware and configuration used in testing. Work on evaluating other possible plugins and selecting one that best suits our needs is a first priority in the JIRA adaptation project as piloting it for truck embedded software projects cannot really start before the plugin is chosen. Also the possibility to modify could be explored but it should consider after all other possibilities have already been deemed unusable.

6.2 Automated report generation and storage in Aton

The process of exporting all the documents related to certain software release needs further study. Manually finding each document that has to be stored in Aton and exporting them from Confluence and JIRA could lead major amount of work to be done in each release. Possible plugins handling this kind of activities have to be explored and selection has to be made on which one of them to use, if any, has to be made. One possible course of action is also creating own plugin or some other automated way of programmatically getting all the documents linked to a software release version out of JIRA. Also a possibility to link JIRA with Aton PDM should be explored.

6.3 Using JIRA as general documentation tool and communication platform

Using ordered wiki like documentation with possibility to link at pages and modify on the fly could be useful outside of software development scope. Currently some software projects use Tracs integrated wiki for all unofficial documentation. This has shown to be a very agile way of doing documentation. Also the availability of documentation and the reliability of not losing any documentation has been noted to be a large improvement over the current method of sharing project documentation via networked hard drives. The possibility to use Confluence as a de facto documentation and information sharing platform should be explored.

Atlassian products also include HipChat which is an online private and group chatting application with features like video conferences and real time file sharing. Team Calendars is an application which features personnel and project tracking. These also integrate well with other Atlassien products. Usability of these in Rocla software development could also be explored. Especially HipChat looks promising if used for communication with the multitude of subcontractors.

6.4 Process and ALM re-evaluation after final version of EN1175 is published

The process for creating a new version of EN1175 is not nearly finished. Currently the new version of the standard is in non-public drafting stage and the first official draft of the new standard is expected to be released later this year. Being in such early development means that final version of the standard can be dramatically different compared to current draft that has been used as an assumption of what the future process requirements. As newer versions of the standard drafts are being released also the process implementation has to be re-evaluated and adjustments to processes and ways of doing things have to be done.

6.5 Final word

The overall success of this project and the impact it will have on software development in Rocla Oy is difficult to estimate as piloting projects are just starting during the writing of this thesis. If everything goes well and the Atlassian ALM solution is adapted it will have a major impact on the quality and agility of the development process. If it is considered only for doing the current activities and not only as “extra work” to fulfil new EN1175 requirements it will have a positive effect on lessening the workload of doing specifications and test management activities manually with Office tools.

Even if piloting of JIRA shows that it is not suitable for us and the search for some other solution is started this project has given a good view for the people involved in it to the possibilities of doing these kind of process activities inside dedicated tools. This has perhaps been the most important outcome of this project so far as equipping people with the knowledge that things can be done differently in equal or in some cases lesser amount of work while providing more value will enable them to seek more information in the future. This can have major effect in the future development of software development processes and the ideas will possibly spread also to other departments of design and while probably not usable as such could spring some new ideas for further improving and consolidating our position as the most innovative truck development unit in the world.

REFERENCES

Wikipedia, Agile software development, http://en.wikipedia.org/wiki/Agile_software_development. Referenced on 21.2.2015

Kääriäinen, Jukka, 2011. ”Towards an Application Lifecycle Management Framework”, VTT. http://www.vtt.fi/inf/pdf/publications/2011/P759.pdf. Referenced on 21.2.2015.

Vuori, Matti 2011. Agile Development of Safety-Critical Software. Tampere University of Technology, Department of Software Systems. Report 14. Tampere, Tampere University of Technology. 95. ISBN (PDF): 978-952-15-2595-7 URN:http://URN.fi/URN:NBN:fi:tty-2011061414702 Referenced on 21.2.2015

European committee for standardization, EN 1175-1:1998+A1, Safety of industrial trucks – Electrical requirements – Part 1: General requirements for battery powered trucks. 23.11.2010

European committee for standardization, EN ISO 13849-1, Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design. 18.5.2008
Wikipedia, Waterfall model, http://en.wikipedia.org/wiki/Waterfall_model. Referenced on 21.2.2015

Wikipedia, V-model(software development), http://en.wikipedia.org/wiki/V-Model_%28software_development%29. Referenced on 21.2.2015

El-Khawaga, G. H.; Galal-Edeen ,Galal Hassan; Riad, A.M.
Architecting in the context of agile software development: fragility versus flexibility, International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.3, No.4, August 2013,
http://airccse.org/journal/ijcsea/papers/3413ijcsea03.pdf Referenced on 15.5.2015

Manifesto for Agile Software Development, http://www.agilemanifesto.org/. Referenced on 21.2.2015

Wikipedia, Iterative and incremental development.
http://en.wikipedia.org/wiki/Iterative_and_incremental_development. Referenced on 21.2.2015 Referenced on 21.2.2015

Wikipedia, Application Lifecycle Management,
http://en.wikipedia.org/wiki/Application_lifecycle_management. Referenced on 21.2.2015. Referenced on 22.2.2015

Wikipedia, Life-critical system. http://en.wikipedia.org/wiki/Life-critical_system Referenced on 24.2.2015

Malm, Timo; Vuori, Matti; Rauhamäki, Jari; Vepsäläinen, Timo; Koskinen, Johannes; Seppälä, Jari; Virtanen, Heikki; Hietikko, Marita; & Katara, Mika. Safety-critical software in machinery applications. http://www.vtt.fi/inf/pdf/tiedotteet/2011/T2601.pdf
Referenced on 22.2.2015

CEN/TC 150/WG 10 N102 20141017 EN 1175-1padova, Safety of industrial trucks – Electrical requirements, working draft. 17.10.2014

Trac open source project, Edgewall. http://trac.edgewall.org/ Referenced on 16.5.2015

Aton, Tuotetiedot tehokäyttöön, Modultek.
http://www.modultek.com/upload/esitteet/aton-esite-fi.pdf Referenced on 17.5.2015

Getting started with IBM Rational DOORS Next Generation, V5.0.
http://www.ibm.com/developerworks/rational/library/rational-doors-next-generation-getting-started/tutorial/index.html
Referenced on 16.5.2015

IBM Rational DOORS Next Generation datasheet.
http://public.dhe.ibm.com/common/ssi/ecm/ra/en/rad14128usen/RAD14128USEN.PDF
Referenced on 16.5.2015

Polarion ALM. https://www.polarion.com/products/alm/index.php
Referenced on 16.5.2015

Redmine. http://www.redmine.org/ Referenced on 16.5.2015

Team foundation server, Wikipedia.
http://en.wikipedia.org/wiki/Team_Foundation_Server
Referenced on 17.4.2015

Understanding Team Foundation Server(TFS) pricing, Rehnstrom, Doug. Learning Tree International, 15.7.2014.
http://blog.learningtree.com/understanding-team-foundation-server-tfs-pricing/
Referenced on 17.5.2015

JIRA, Wikipedia. http://en.wikipedia.org/wiki/JIRA Referenced on 17.5.2015

Confluence(software), Wikipedia. http://en.wikipedia.org/wiki/Confluence_(software)
Referenced on 17.5.2015

JIRA licensing and pricing. https://www.atlassian.com/licensing/jira#serverlicenses-1
Referenced on 17.5.2015

Magic quadrant for application development lifecycle management, Gartner, 9.2.2015.
http://www.gartner.com/technology/reprints.do?id=1-2A61Y68&ct=150218&st=sb
Referenced on 17.5.2015

Confluence User's Guide, Atlassian.
https://confluence.atlassian.com/display/DOC/Confluence+User%27s+Guide Referenced on 17.5.2015

JIRA User's Guide, Atlassian.
https://confluence.atlassian.com/display/JIRA/JIRA+User%27s+Guide Referenced on 17.5.2015

JIRA Agile documentation, Atlassian. https://confluence.atlassian.com/display/AGILE/JIRA+Agile+Documentation;jsessionid=7FD4181B01231E75B48D83F6D7E98109.node1 Referenced on 17.5.2015

Zephyr for JIRA documentation home, Atlassian. https://zephyrdocs.atlassian.net/wiki/display/ZTD/Zephyr+for+JIRA+Documentation+Home Referenced on 17.5.2015