



VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Son Thanh La

DEFECTS MANAGEMENT IN EMBEDDED SYSTEMS

Technology and Communication
2017

ABSTRACT

Author	Son Thanh La
Title	Defects management in Embedded Systems
Year	2017
Language	English
Pages	63 + 0 Appendices
Name of Supervisors	Jukka Matila Heidi-Maria Kallio

The thesis is done for Wärtsilä Finland Oy, Marine Solutions, Engine Performance and Control department. At the moment, the department is using Mantis as bug tracking tool and planning to migrate to Polarion ALM for the completeness of application lifecycle management. Different approaches are analyzed and a decision is made by the project team and the department's management team before an actual implementation can be done.

Defects management contributes a significant part throughout the lifecycle of any product. It is an important part of the ALM chain, from requirement down to source and without traceability of defect, a broken chain cannot optimize its strengths and opportunities. The proper configuration can reduce the occurrence and consequences of a nonconformity and even the cost in product development by prevention at the beginning. The management system is supposed to be continuously improved from time to time with the involvement of maturity and capability level of an organization

The thesis covers requirement development for the defects management system, analysis of the current system, approaches to having defects in Polarion ALM and the implementation of system. Due to the huge impacts on the current way of working, the major part of the thesis focuses on the pre-study phase which results in the selected direction of the department.

CONTENTS

TIIVISTELMÄ

ABSTRACT

1	INTRODUCTION	9
1.1	Wärtsilä Oy	9
1.1.1	Engine Performance and Control	10
1.2	Objectives	11
1.3	Scope and limitation	11
2	BACKGROUND INFORMATION	13
2.1	Application Lifecycle Management (ALM)	13
2.1.1	Different views of ALM.....	14
2.1.2	ALM pillars	15
2.2	Defect Management	16
2.2.1	Defect characteristics	16
2.2.2	Defect Management Systems.....	19
3	REVIEW OF CURRENT SYSTEMS.....	24
3.1	Review current system.....	24
3.1.1	Nonconformity handling processes.....	24
3.1.2	Requirements management	25
3.1.3	Test management	26
3.1.4	Mantis BT	32
3.1.5	Polarion ALM	40
3.2	Analysis.....	45
3.2.1	Processes	45
3.2.2	Tool implementation	46
4	PRE-STUDY PHASE	47
4.1	Requirements development.....	47
4.1.1	Stakeholders identification.....	47
4.1.2	Requirement elicitation results and analysis	48
4.2	Alternative approaches for defects and implementation plans	49
4.2.1	Polarion and Mantis together	55
a.	Decision making by management team	57

5	IMPLEMENTATION	58
5.1	Defect item model.....	58
5.2	Defect workflow	59
5.3	Defect's link role.....	60
5.4	Next steps.....	61
6	DICUSSION AND CONCLUSION	62
7	REFERENCES	63

LIST OF ABBREVIATIONS

ALM	Application Lifecycle Management
EPC	Engine Performance & Control department
T&V	Testing and Validation department
TS	Technical Services department
RFC	Request for change work item in Polarion
CMM	Capability and maturity model
HW	Hardware
SW	Software
WoW	Way of working
OSLC	Open Services for Lifecycle Collaboration

LIST OF FIGURES

Figure 1. Engine Performance & Control organization.....	10
Figure 2. ALM circle (Polarion Overview, 2017)	11
Figure 3. Illustration of ALM (Joachim; Redler Rossberg (Rickard), 2014)	13
Figure 4. Software Development Life Cycle (Software development Lifecycle, 2017) 14	
Figure 5. PMI view of ALM (Joachim; Redler Rossberg (Rickard), 2014).....	15
Figure 6 Microsoft TFS Bug workflow lifecycle (Process Guidance and Process Templates for Team Foundation Server, 2017).....	18
Figure 7. Defect management process.....	19
Figure 8. Structure of Polarion requirements	26
Figure 9. Requirement workflow.....	27
Figure 10. Example of test case in Polarion	28
Figure 11. Workflow of test case.....	29
Figure 12. Example of test case state transition	29
Figure 13. Test case model	30
Figure 14. Example of Polarion test run.....	31
Figure 15. Test run workflow	31
Figure 16. Querying test run.....	32
Figure 17. Mantis issue reporting interface	35
Figure 18. Mantis bug table in MySQL.....	37
Figure 19. Raw measurement of concurrent users.....	38
Figure 20. Concurrent user measurement (@Wärtsilä)	39
Figure 21. Concurrent user measurement (!@Wärtsilä).....	39
Figure 22. Polarion license type features (Polarion Overview, 2017).....	41
Figure 23. EPC's Polarion project structure.....	42
Figure 24. Custom field configuration	42
Figure 25. Polarion work items linking	43
Figure 26. Polarion extensions list	45
Figure 27. Polarion single system option	49
Figure 28. Polarion traceability matrix	50
Figure 29. Summary of traceability	50
Figure 30. Data migration options comparison	52

Figure 31. Feature comparison	53
Figure 32. Confidence interval for licenses	54
Figure 33. Cost calculation	55
Figure 34. Agosense architecture (agosense.instruments, 2017).....	57
Figure 35. Defect from test execution	59
Figure 36. Defect's workflow	60
Figure 37. Required field and function configuration	60
Figure 38. Defect-requirement Link role.....	61

LIST OF TABLES

Table 1. Definitions of error, fault (defect) and failure	16
Table 2. Defect's attribute description.....	17
Table 3. Mantis roles (Österback, 2015).....	33
Table 4. Component's defect work item.....	58

LIST OF APPENDICES

1 INTRODUCTION

1.1 Wärtsilä Oy

“Wärtsilä is a global leader in advanced technologies and complete lifecycle solutions for the marine and energy markets. By emphasizing sustainable innovation and total efficiency, Wärtsilä maximizes the environmental and economic performance of the vessels and power plants of its customers.” (About Wärtsilä , 2017). Setting a vision to be the customer’s most valued business partner, the values of the company are defined as:

- Energy: Capture opportunities and make things happen.
- Excellence: Do things better than anyone else in the industry.
- Excitement: Foster openness, respect and trust to create excitement.

In 2016, Wärtsilä's net sales totaled 4.8 billion EUR. With approximately 18,000 employees, the company has operations in over 200 locations in more than 70 countries around the world. Wärtsilä is listed on Nasdaq Helsinki. (About Wärtsilä , 2017)

Wärtsilä organization can be divided in three main areas: Marine Solutions, Energy Solutions and Services.

Wärtsilä Marine Solutions supplies innovative and integrated solutions to marine and oil & gas industry customers. The products, systems and services are developed based on customers’ needs, using advanced technology, experience and know-how of Wartsila personnel. The wide product portfolio includes automation, navigation & dynamic positioning, ballast water management, compressors, electrical & automation, engines & generating sets, exhaust gas cleaning, gas solutions, inert gas systems, integrated solutions, propulsors & gears, pumps & valves, seals & bearings, ship design, sonars & hydroacoustic systems, waste, oil & fresh water systems. (Wärtsilä, Marine Solutions, 2017)

Wärtsilä Energy Solutions is a world leader in power plan for decentralized power generation market. Offering the products for baseload, peaking and industrial self-generation purposes, Wärtsilä power plants are flexible, efficient and low emission levels. (Wärtsilä, Energy Solutions, 2017)

Wärtsilä Services is responsible for customer support throughout the product installation lifecycle for both marine and energy. They provide a comprehensive portfolio of services,

together with the commitment on quality, expertise and availability. (Wärtsilä, Services, 2017)

1.1.1 Engine Performance and Control

The department is part of Marine Solutions, Engines and Technology organization of Wärtsilä. EPC is responsible for development of engine process performance and functionality. The defined mission is to convert customers' requirements and future needs to optimize performance solutions and to provide engine process and controls expertise to ensure the competitive level of quality, performance and cost throughout the lifecycle of product. (Vuollet, 2017)

EPC consists of seven teams and each team responsible for one area in engine automation system development. (Figure 1)

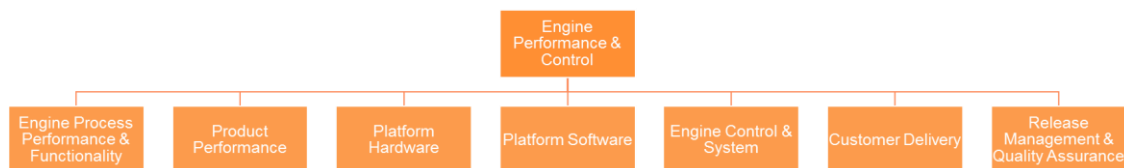


Figure 1. Engine Performance & Control organization

Related to Polarion ALM tool development inside EPC, Polar Bear is created as a virtual team, including 10 members from almost all EPC's teams. Different sections in the ALM circle (**Figure 2**) which are supported by Polarion ALM have been successfully planned and rolled-out by the team in the last several years. In 2017, two topics that relate to the scope of the thesis are Manual Test Management and Defects Management.



Figure 2. ALM circle (Polarion Overview, 2017)

1.2 Objectives

The major target is to expose the most appropriate way to get Wärtsilä's engine automations reported defects visible in Polarion ALM, where other configuration items are stored and managed. The solution needs to clearly prove its advantages going with reasonable costs or drawbacks if there are any. Requirements from stakeholders are collected to avoid major change resistances. Before and after the implementation, all related people will be informed, and given a clear picture about what to expect and how the transition will occur.

1.3 Scope and limitation

The scope of the thesis is to study different alternatives to have defect artifacts in Polarion, which are located in Mantis BT at the moment. The types of defect consist of those related to department's deliverables, no matter whether they are found before or after releases. Different approaches are proposed at the beginning of the projects, including:

- Integration of Mantis BT and Polarion
- Migrating to Polarion from Mantis gradually
- Migrating to Polarion from Mantis immediately
- Other feasible options

The result of the pre-study phase result will be used for Polar Bear and EPC's management team decision making. With the selected option, a corresponding implementation plan will be coordinated and rolled-out together with other projects' team members, which range from specification, in-house configuration to development coordination with the supplier.

Although the thesis' title mentions the complete defects management system, the author and also the project team are not authorized for process updates but only proposals and tool-related supports. Therefore, process improvement will not be discussed in the scope of the thesis, except for personal theoretical research, and the development will mainly follow the current processes and procedures.

2 BACKGROUND INFORMATION

2.1 Application Lifecycle Management (ALM)

The rapid oscillation of industrial environment provides both opportunities and challenges to each individual organization. At the same time, the increases in size and complexity, in terms of project or product portfolio, makes the communication between different parts even more difficult, creating a hole in internal supply-demand chain. In addition to that, the control and management of project, product can be even worse with poorly defined processes or low capability and maturity levels (CMM).

Among those challenges, ALM has immediately raised community's interests as a feasible answer. The ALM process covers the time from the birth of an application from a business need until becoming obsolete (Figure 3).

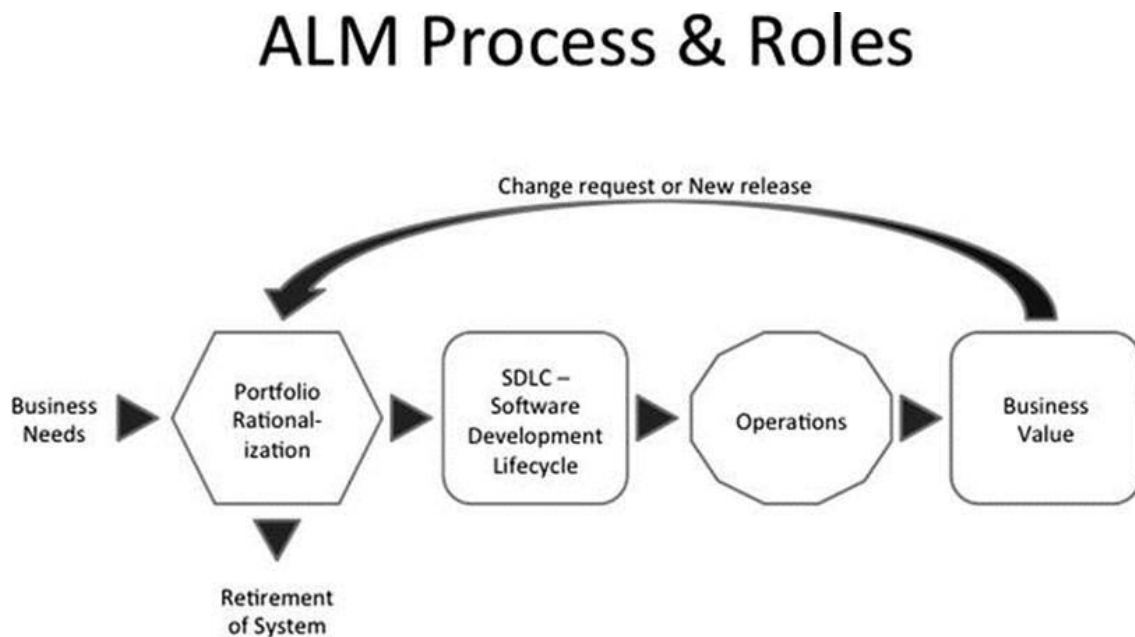


Figure 3. Illustration of ALM (Joachim; Redler Rossberg, 2014)

From the initial thought of having a new product to the current portfolio, some rationalization activities are normally done to verify the actual need and existing solutions' capability. In case the reusability is impossible and new product must be developed, a software development lifecycle (SDLC) is entered, starting from project initiation to maintenance phase (Figure 4). As a result, a new product is added to the organization's portfolio which

fulfils the initial business needs. The values generated from the product, as planned in the beginning, shall come until its retirement.

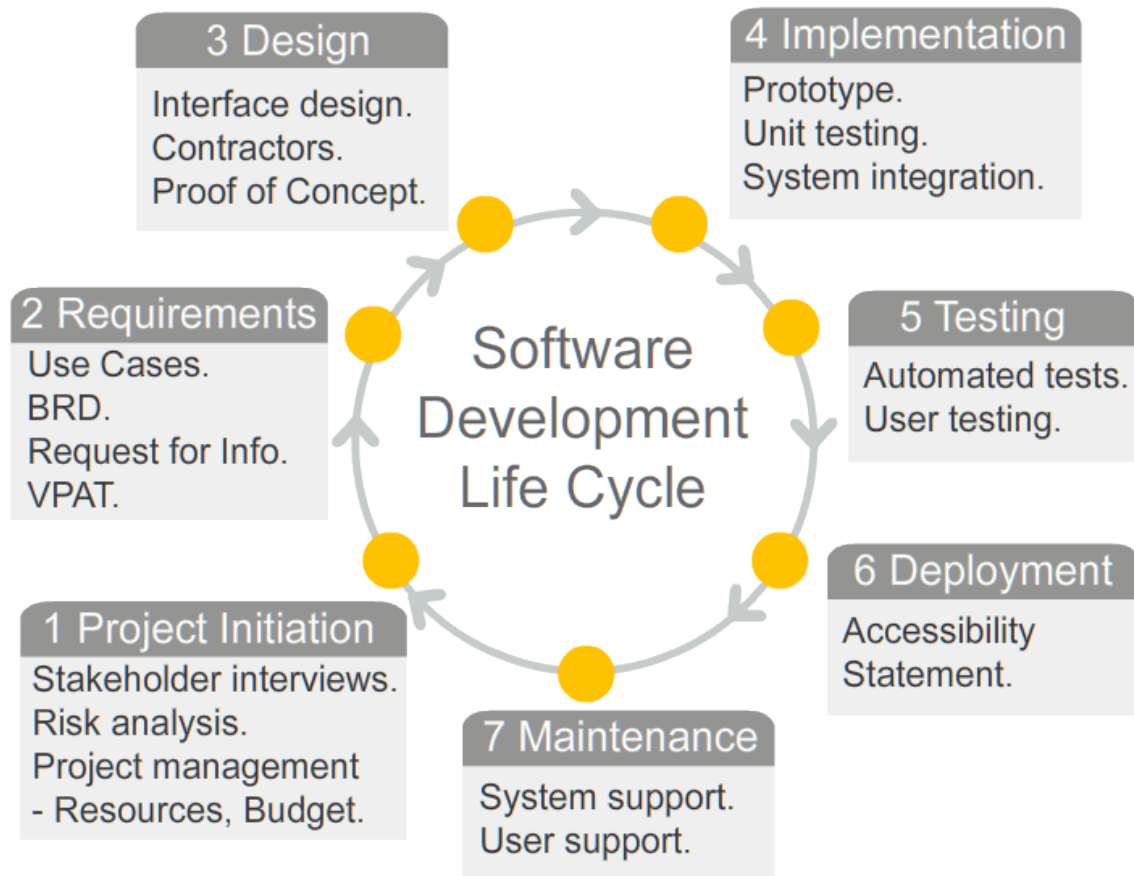


Figure 4. Software Development Life Cycle (Software development Lifecycle, 2017)

2.1.1 Different views of ALM

According to Joachim and Rickard (Joachim; Redler Rossberg, 2014), whenever referring about ALM, there are normally four common views on the concept:

- The SDLC view: The complete ALM concept is narrowed down to SDLC only and the phenomenon is relatively popular in organizations where the technical part has taken the majority, compared to business.
- The service management or operations view: From this point of view, ALM is divided into Development and Operations and obviously, one cannot exist without the other. The main difference compared to the first view is about the starting

point of product lifecycle where for SDLC, it is project initiation and for the later ones, it is the deployment into the production environment.

- Application Portfolio Management (APM) view: Being as part of Project Portfolio Management (PPM), APM has its own view about ALM which starts the lifecycle with a business plan of a new product or an update and close when the product goes to operation. (Figure 5)

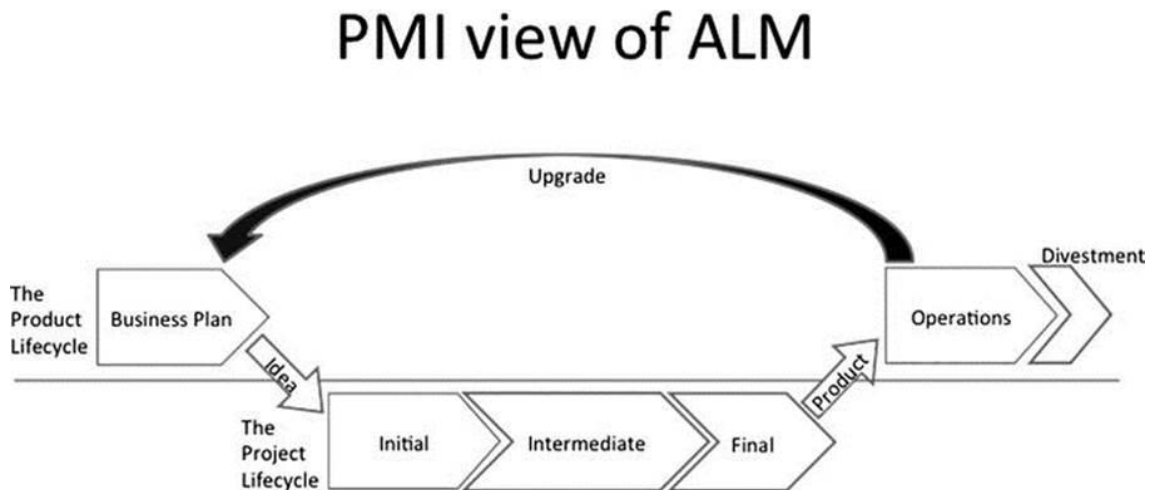


Figure 5. PMI view of ALM (Joachim; Redler Rossberg, 2014)

- Unified view: as a combination of the previous three, the unified view focus on the entire ALM process and this is stated as the only way to take control and optimize the concept of ALM.

2.1.2 ALM pillars

Being firstly introduced by Forrester Research (Dave West with Jeffrey S. Hammond, Mary Gerush, Sander Rose, 2017), ALM can be seen as a combination of traceability, processes automation and visibility, and is independent of the views.

Traceability

When the size and complexity of product or project increase, keeping track of different artifacts (in software development context) is a huge barrier for management and quality achievement. In case traceability cannot be done right from the beginning, the cost for any organization is enormous, which comes from unmanageable products. Therefore,

traceability can be seen in different standards and regulatory, from requirement down to its implementation. For example, ISO 9001:2015 (SFS, Finnish Standards Association, 2015) requires an organization to have traceability in resource management (7.1.5.2), product or service provision (8.5) and releases (8.6) to prove its conformity.

Processes automation

In software engineering, development processes such as Water Fall, Scrum and Extreme Programming (XP) are highly important. Large-size companies also need procurement, sales, and so many other processes for their daily activities. In some situations, a process can be complicated, having numerous steps to follow and involvements from different parties. Therefore, having an automated process can reduce the time and cost, and can improve the effectiveness and accuracy of the process.

Visibility

Visibility is the ability to measure progress or status of goal achievement. From an operational point of view, limited visibility can be an obstacle to achieve efficiency, which requires insignificant (automated) to huge (manual) effort from each member in work reporting or reports generation.

2.2 Defect Management

2.2.1 Defect characteristics

Definition

Ilene Burnstein in Practical Software Testing (Burnstein, 2006) has given the definitions of some of the terms based on the description in the IEEE Standard Collection for Software Engineering Terminology (including IEEE Standard Glossary of Software Engineering Terminology). The definitions for error, faults (defects) and failures are shown in table 1

Table 1. Definitions of error, fault (defect) and failure

Software quality problem	Definition
--------------------------	------------

Error	An error is a mistake, misconception, or misunderstanding on the part of a software developer
Fault (Defect)	A fault (defect) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.
Failure	A failure is the inability of a software system or component to perform its required functions within specified performance requirements.

Although the terms are clearly stated to be different at a certain level, defect, bug and nonconformity are used interchangeably in this context and will be used so in the rest of the thesis. The selected common definition is the deviation of the actual implementation from the initial requirements of the customer.

Attributes

Freimut, in his work for developing and using defect classification schemes, has proposed different attributes, including location, timing, symptom, end result, mechanism, cause (error), severity and cost. (Freimut, 2001). The attributes are explained in table 2.

Table 2. Defect's attribute description

Attribute	Description
Location	The field describes where a defect is found. Some examples are document, requirement, design, source code ...
Timing	The attribute describes the time point when the defect is injected, detected or corrected.
Symptom	Normally when a defect is found, the reporter does not know what the actual root cause is and what he or she observes is only considered as symptom.
End result	The field describes the failures caused by the defect
Mechanism	Similar to timing attribute, the field describes how the defect is injected, detected or corrected.
Cause (error)	The field describes what the culprit is. Example can be misunderstanding of requirement, wrong design and implementation ...
Severity	The attribute describes the seriousness of defect and normally it can be assessed based on impacts, consequences and also estimation, actual resources to resolve.

Cost	The field can be interpreted in term of time, effort or simply money.
------	---

The attributes are appropriate to construct a defect model with all necessary information for management system.

Lifecycle

A defect lifecycle is an ordered set of possible states that a defect goes through during its lifetime. Different research papers and systems have their own interpretations of defect lifecycle, ranging from being simple to extremely complicated and depending on several factors. For example, Microsoft has proposed three different workflows for defect artifact in Process Guidance and Process Templates for Team Foundation Server (Process Guidance and Process Templates for Team Foundation Server, 2017). The workflows are used for SCRUM, Agile, CMMI development processes (Figure 6).

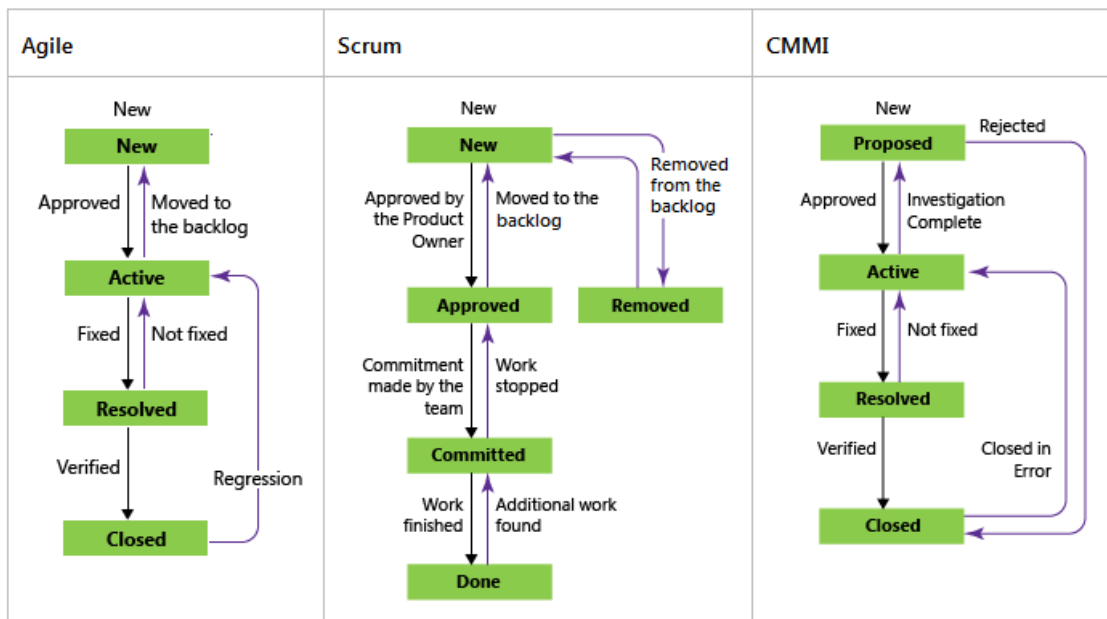


Figure 6 Microsoft TFS Bug workflow lifecycle (Process Guidance and Process Templates for Team Foundation Server, 2017)

However, Timo Koponen (Koponen, 2006), in his paper about lifecycle of defect in open software, concludes that the states of defect can be simplified to only open and closed, which has covered approximately 80% of defects transient pattern for two well-known open source projects: Apache HTTP and Mozilla Firefox, although full workflow is still

implemented in those two. Therefore, there is no particularly ideal model for defect lifecycle and it needs to be adjusted, according to environmental characteristics.

2.2.2 Defect Management Systems

The management system is a combination of defect management processes together with an application for supporting the implementation of the process. A defect reported to the system contains several necessary pieces of information (attributes) described above.

a. Process

Although it is impossible in reality to have zero defect but one of the core philosophies of the process is to prevent defects before their occurrence. Other principles for defect management process include:

- Risk-driven process, i.e. common goal is to reduce risk.
- Integration with development processes
- Automated processes (2.1.2.2)
- Improvement (MOSAIC, INC, 2017)

The major components of the process contain: prevention, baseline, discovery, resolution, improvement and reporting. Figure 7 describes how the blocks can be connected to form the complete process.

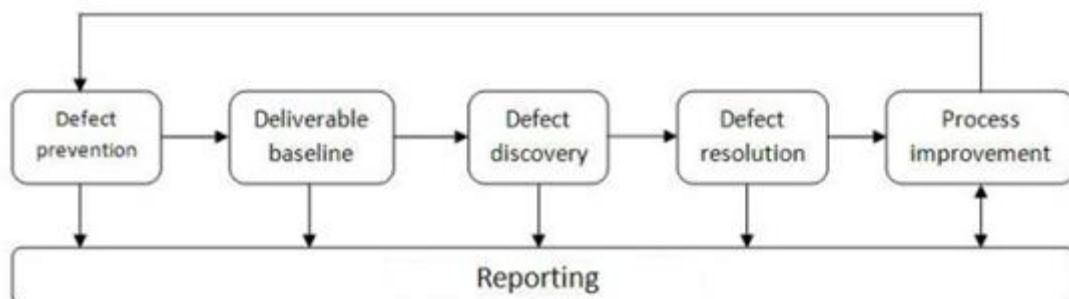


Figure 7. Defect management process

Defect prevention

To prevent that defects happen, different techniques and methods are implemented. However, they all have to start with a common step which is risk identification. Philip Koopman's works (Koopman, 2010) were taken into study. In his research, 43 risk areas for real products in embedded systems are listed and they can be ordered in eight bins, including: development process, architecture, design, implementation, verification and validation, dependability, project management and people. The notable point is that Koopman has done the analysis to show how weak development process affect the success of embedded systems development (Section 4.4). The examples of that type are the absence of written requirements, Software Quality Assurance (SQA) and defect tracking.

Thorough understanding of different risk categories supports development for risk avoidance. Therefore, the research greatly contributes to embedded systems' defect prevention.

Deliverable baseline

A baselining activity can be done when development work reaches a certain defined milestone. With a baseline point of view, an error is considered to be a defect only if the event happens after baselining, otherwise, it is only considered as a bug.

In CMMI-DEV 1.3 model (CMMI Product Development Team, 2010), deliverable is also considered as a configuration item. The work normally consists of key deliverables identification and standard definition for each deliverable.

Defect discovery

Definitely, the existence of a perfect defect prevention system where there is not any single defect being found after the baseline, is impossible. Hence, a defect discovery process is also necessary in the chain of defect management.

An issue found by the user is only considered to be a defect if and only if

- (1) It is reported to the system
- (2) It is a valid defect

To detect defects in a product, different techniques can be done, consisting of static, dynamic and operational ones. The first method group analyses syntax and semantics of a program statically without any execution so it can find a defect in very earlier stage. Some

examples of data-flow analyses, the static analysis tools are Splint, VisualCodeGrepper, FindBugs, LAPSE+, JSLint, pylint and so on. In contrast, dynamic methods test system components under their execution. For instance, code coverage method is used to analyse the degree of source code and program to be executed, where a high coverage section is suggested to have lower chance of containing defect. Some support tools for dynamic techniques are, for example, Valgrind, DynamoRIO, Pin, gcov, gpof and dmalloc. The last technique, which is operational testing, is one of the most expected not to contain any of defect. The execution is done mainly to check the readiness of a system, simulating the real production environment, and a found defect in this phase means the development process is likely need to be restarted in case of requirement type, for example. Therefore, the first two techniques require an effective defect management system to minimize the later defect's consequences

Defect resolution

After the reported defect is confirmed to be valid, resolution process generally contains 4 steps:

- Prioritization
- Scheduling fix time
- Resolving defect
- Resolution report

The actual detail of significantly varies depending on the root cause of a defect, together with organization maturity.

Process improvement

Improvement is a required activity for organization to meet customer requirements and enhance their satisfactions. As beaning mentioned in section 10.2 of ISO 9001:2015 (SFS, Finnish Standards Association, 2015) for each occurrence of nonconformity, risks and opportunity during the planning phase of the project or product need to be reviewed and quality management systems must be updated if necessary. In most of the cases, a defect, no matter of its severity, means that there is a flaw in the current system to be updated and normally, it is the one which allow the escape to happen.

Escape analysis is the study to make preventive plans for avoidance of future escape in development and testing phase. Mary Ann Vandermark (Vandermark, 2003) published a thorough research in the field to summary the definitions together with several notable examples. The process has several steps, including:

- Planning: The process requires resources, commitment and scope of the analysis.
- Categorizations: The effective and meaningful groups will accelerate further work of determining how defect management system can be improved. Some examples are:
 - o Process steps: Development (Marketing review, concept review, design review, code review, unit test, packaging, information development), test (functional test, system test, translation test, installation test and error-injection test)
 - o Product component: hardware components (such as display and power supply), software (such as kernel and application)
 - o Symptom (impact): hanging, data corruption and low performance
- Analysis-tool: Capable tool for statistical tracking
- Analysis-statistics: Statistical analysis of recorded data with, based on area, specific configuration such as time interval and candidates.
- Analysis - Process change: With statistical trends, areas where improvement needs to be made shall be listed in priority order. For instance, if hardware component of the product is having significant number of defect, the responsible team needs to handle the issue, either by in-house debugging or contacting supplier
- Metrics: measurement on improvement shall be made to follow the process performance itself.

Management reporting

Similar to metrics in defect escape analysis, different tactical information about nonconformity shall be recorded to have a comprehensive perception of performance of the system. Some example of metrics is:

- Defect trend, incoming and resolving rate
- Failure costs trend
- Defect found during development and after release

However, the measured values and their trends might depend on numerous hidden aspects such as natural, resources which requires interpreter have a clear overview before coming to any conclusion and decision.

b. Application

The general idea to have a software for defect management is to support the implementation of process, including to ensure the conformity and to simplify the reporting work. A defect management tool has several similarities with issue tracking tool and some example features to be mentioned include the issue's database, notification, states transition, report and so on.

However, for complete application lifecycle management, the tool is required to be more versatile. The set of basic inevitable features includes test, requirement and release management which basically means that defects can be linked to its failed test case, failed requirements, indicating which releases that defect are found in, affected to and how it will be resolved. Some defect management applications with ALM supported or vice versa (i.e.: ALM tool with defect management supported) are Atlassian JIRA, IBM Rational Solution, Microsoft Team Foundation Server, Polarion ALM, HP Quality and so many others (Joachim; Redler Rossberg, 2014).

3 REVIEW OF CURRENT SYSTEMS

The work is divided into two phases: pre-study and implementation with the ratio 60:40 due to the initial analysis of the involvement and impact of the topic.

Pre-study phase contains reviewing of current defect management in EPC, including processes, tool implementation, and analysis different approaches to the goal. The second part of the thesis is implementation which depended on the selected option of EPC's management team.

3.1 Review current system

3.1.1 Nonconformity handling processes

The classification society requires an organization to have quality plan, which includes processes following system development lifecycle, and commit to it.

EPC's quality plan describes the quality assurance of the department when development engine automation systems. In total of 10 processes, two of them, which are **Software nonconformity handling process** and **Engine Automation Platform Hardware nonconformity handling process**, mainly related the current defect management systems. The summary of two processes are described below.

The processes describe how engine automation system defects found in the releases are handled with the goal of minimum delay time. It requires reporters, line managers and experts' responsibility of following procedure and monitoring the performance.

First of all, all engine automation related issues are reported from outside of EPC will go into Mantis BT's common project. The initial analysis by dedicated support team to ensure that all the reported case is clearly described with correct severity, priority and others. In case of unclear submitted information, the case is set to "Feedback" state and reporter will receive notification for updating the information.

If the severity is high or the root cause relies on different components, product improvement case will be open and the cross-functional team will be in charge of it. Otherwise, the original Mantis case is moved to related subsystems' project to be handled by the

assigned person. Other development processes are followed if there is a need for new a release of the system.

Once corrective actions have been performed, related documents are released and stakeholders are informed. Evidence in term of documentation or test report are recorded by the assignee. All actions are written into Mantis note and the case is set to the resolved state.

Depending on root cause of a defect, actions will be performed to prevent similar event. When only workaround or temporary solution is provided, a request for change or work request is created in Polarion and is linked to the Mantis case. The changes can be for products, processes or WoW improvement. A reported case is closed when there's a confirmation from reporter for the solution.

The processes also mention about several metrics/performance indicators for process improvement:

- Number of days from report to closing of case
- Number of days from report to start of process
- Number of defects corrected in last 12 months
- Number of open cases
- Number of minor/major cases
- Number of improvement cases
- Number of days from report to in progress
- Number of days from report to closed / resolved

The processes were recently introduced in October 2016 after the organizational structure changes.

3.1.2 Requirements management

The current state of requirement management system also needs to be reviewed due to the its dependency on the defect handling processes

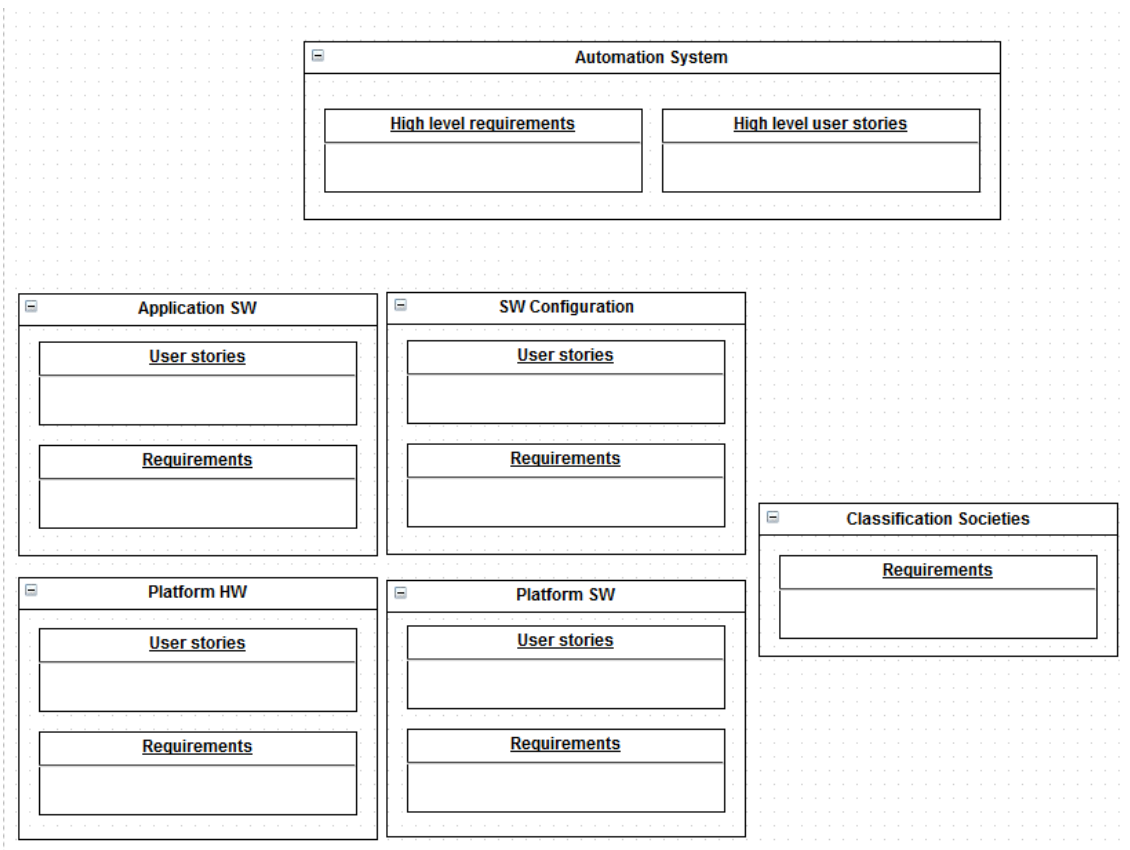


Figure 8. Structure of Polarion requirements

User stories and requirements are stored in Polarion, which is the selected tool for requirements management. High level requirements and user stories are placed in Automation System project while product components (software, hardware, configuration)'s ones are placed in their own projects. Requirements from classification societies are placed in a separate project in Polarion.

The workflow of requirement artifact is described in figure 9.

3.1.3 Test management

Software and hardware test management started to migrate to Polarion at the same time of this thesis started. Because of their close relation, the work has progressed in parallel. The description below is the first baseline of software configuration and hardware testing, which were selected as the piloting areas for Polarion testing.

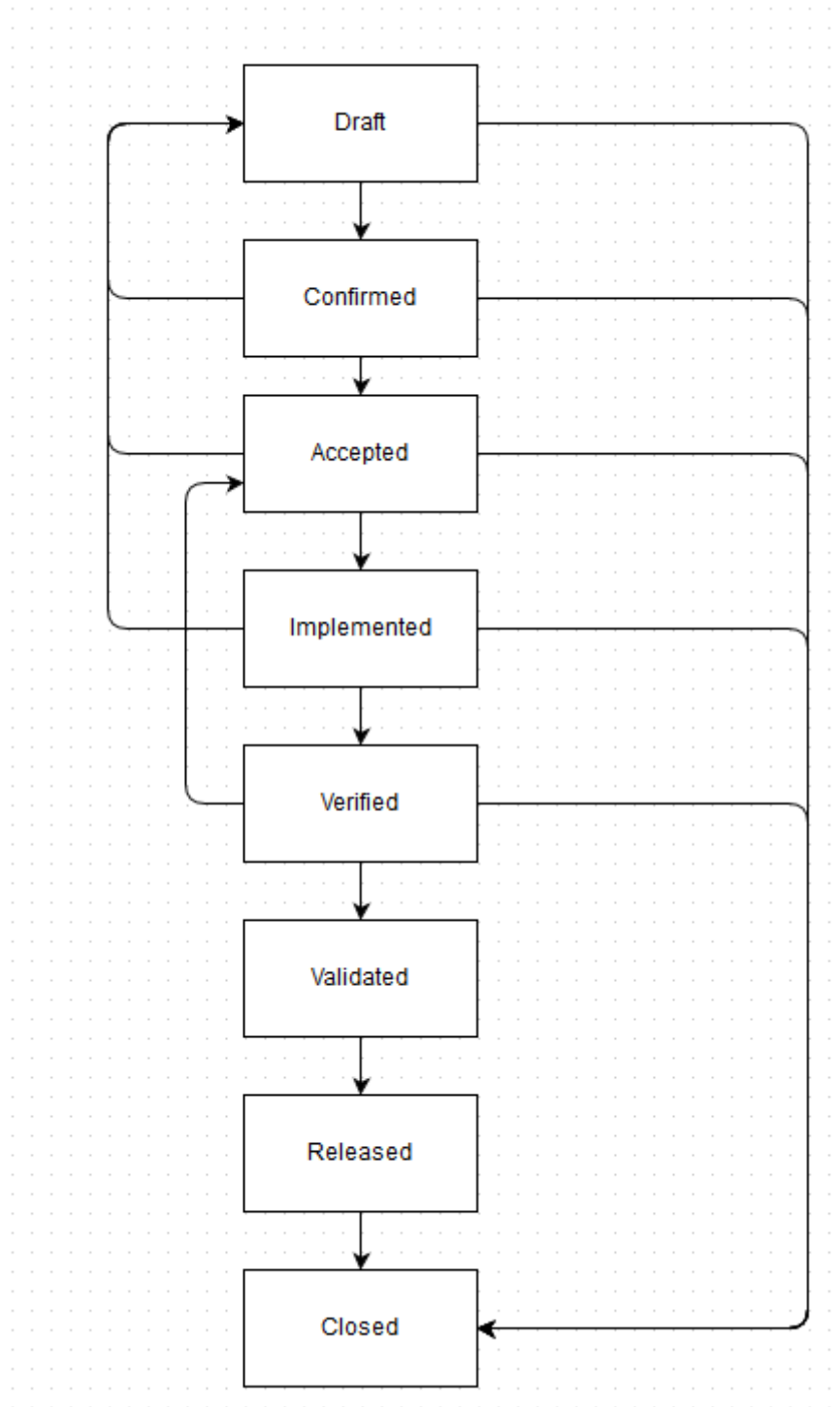


Figure 9. Requirement workflow

Polarion provides both automatic and manual testing features but at this point, only manual testing is in used. Test cases are written by EPC developers or designers and can be either internally or externally executed (by T&V).

a. Test case

Basically, test cases are created and stored in Polarion for verifying and validating requirements that they are connected to. When designing of test case is ready, it's in Active status and can be used to verify/validate its connected requirement(s).



Figure 10. Example of test case in Polarion

The workflow of test case is described below in the figure 11.

When the work item is created, it is in Draft state and the test engineer starts writing the test case description, defining its acceptance criteria goals and test steps. When the test engineer is ready with his/her work, the test case is set to planned state and the approver verifies and gives permissions to the test case to be use by setting its status to Active. The test case is closed when it is obsoleted with the requirement or invalid. The flow also provides certain degrees of flexibility for returning to previous state or skipping to closed state.

The generic entity model of Test case is described in figure 13. The attributes mainly follow default setup of Polarion.

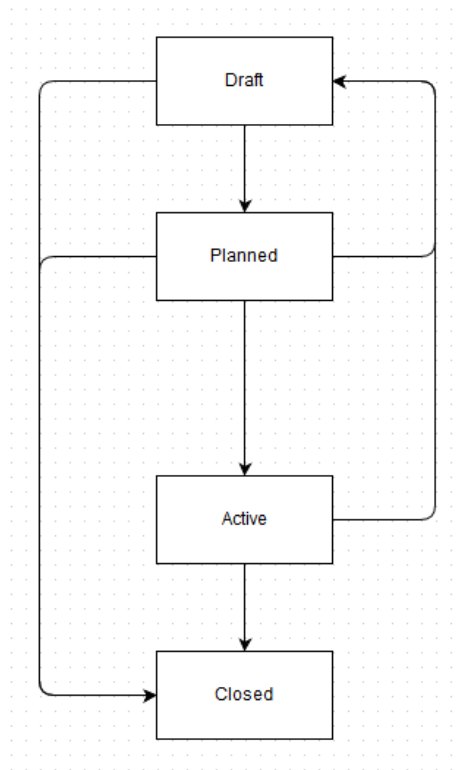


Figure 11. Workflow of test case

SWC-17 +

SWC-13 - Test case 1 in Test Specs

SWC-20 SWC-25 SWC-28 +

Type: Test Case

QA Method: Test

*Status: Active

Resolution: Active

Perform action set to Draft

Perform action set to Closed

Description

Figure 12. Example of test case state transition

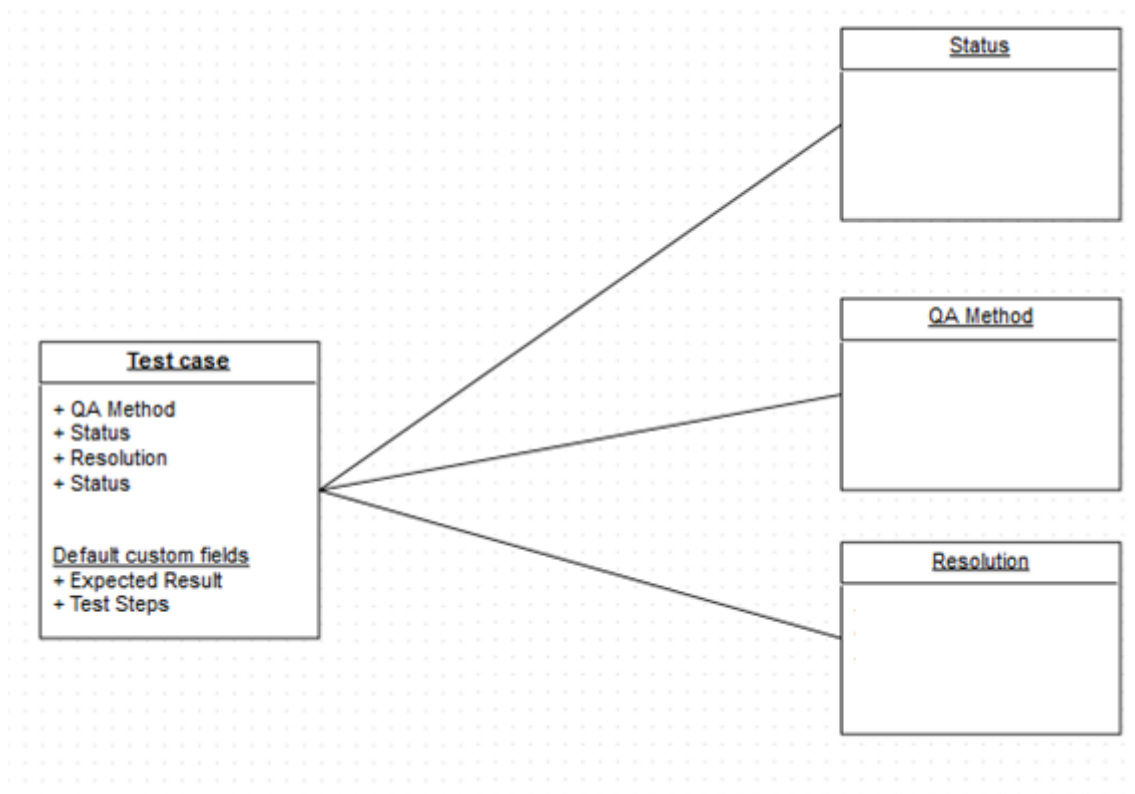


Figure 13. Test case model

During the execution of the test run, if the test case is marked as failed, a defect work item is automatically generated and linked to a specific revision of the test case.

b. Test run

Polarion test run log instances of the execution of a defined set of test cases. When executing a test run, the selected test case can be marked as passed or failed depending on its acceptance criteria. It can also be marked as blocked if the execution cannot be finished and it interrupts the whole execution of the test run.

Figure 14 represents the workflow of test run.

ABC - Standard SG configuration

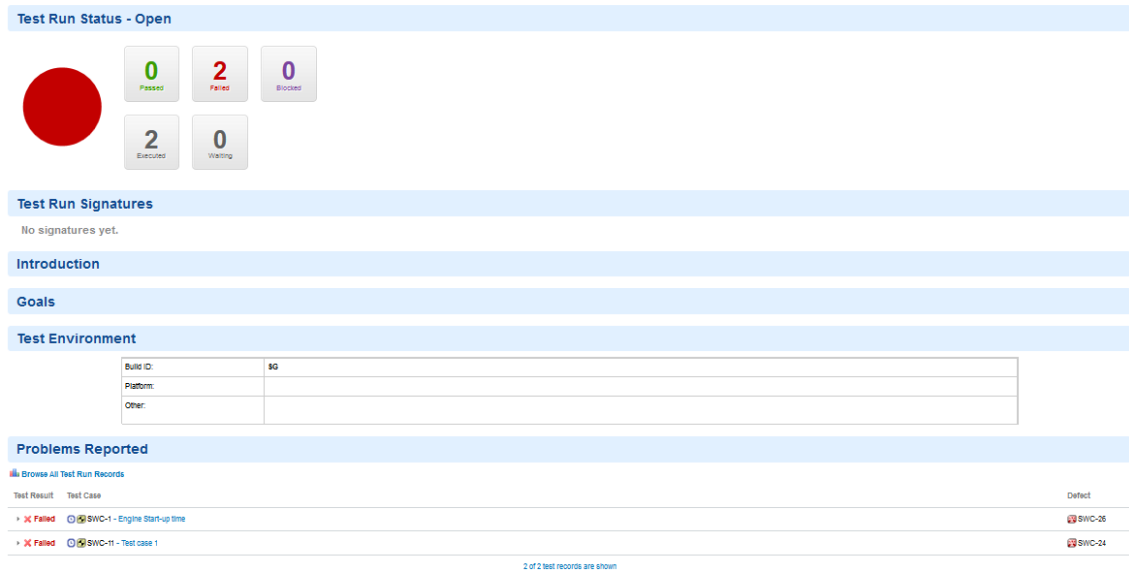


Figure 14. Example of Polarion test run

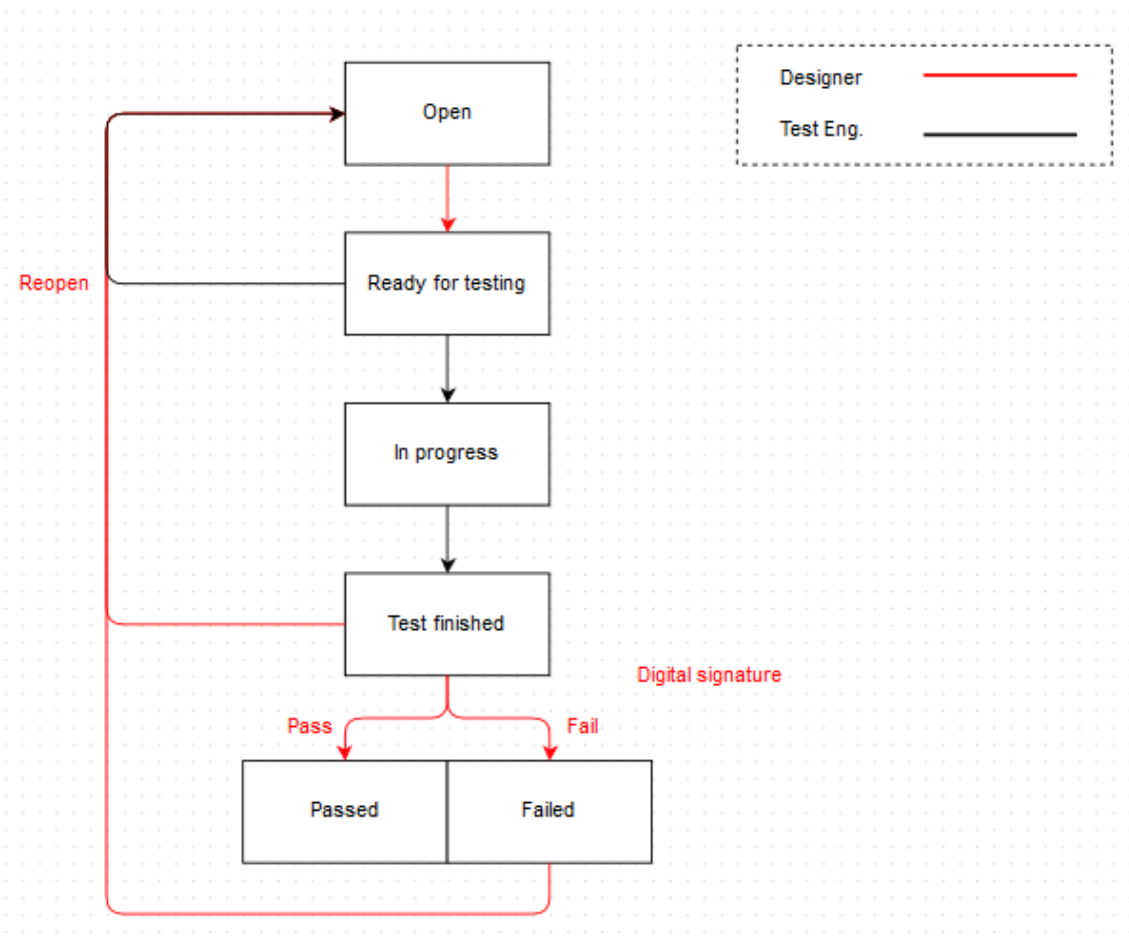


Figure 15. Test run workflow

When a test run is created from the template, the status is open by default. A set of test cases that is preselected by the template are in the waiting list for execution. The test plan designer starts collecting more test cases from Polarion systems before setting the status to Ready for testing and passing it to the test engineer. With a predefined agreement at the beginning of the release, the test engineer with reserved devices executes the test run with In Progress status. Although there is always communication and collaboration during the execution, when it's finished, test engineer sets status to Test finished to automatically notify developers/designers about the completion. The responsible person will mark the whole test run as passed or failed with his/her digital signature, which, most of the time, depends on the results of each test cases Being designed to be reusable, if the test run is marked as failed, it can be reopened and executed again after a corrective action of developers.

The history of execution of the test run is recorded into the system and can always be searched from Polarion query (Figure 16)

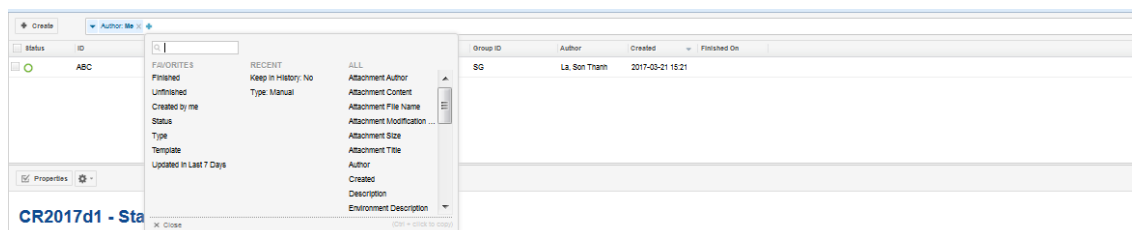


Figure 16. Querying test run

3.1.4 Mantis BT

a. Mantis Bug Tracking

Mantis BT is an open-source web-based web bug tracking tool which was first introduced in 2000. At the moment, the tool supports multiple operating systems, including Windows, Mac OS X, Linux can almost all of the popular web browsers such as Chrome, Firefox, Safari, Opera, IE 7+ and so on. (Mantis, 2017)

Because of its flexibility, the tool normally is configured not only for bugs/issues tracking but also for project or release management (agile Mantis). In EPC, Mantis has been being used to track defects, issues, change requests, tasks and to make releases.

b. Project structure

Mantis system has project structure, including projects and sub-projects. For each project or sub-project, users can have different roles, together with different access levels. Sub-project and project can have different members list. (Mantis Project, 2017)

EPC's Mantis projects can represent components, products or an inbox for change requests. One project normally has managers/responsible persons to assign issues and maintain other project information.

c. Access right control

The tool requires users to manually make a request for an account and access right to specific projects for first time used. There are multiple predefined global roles and with different roles, a user has different access to project's content and configuration.

Table 3. Mantis roles (Österback, 2015)

Viewer	The user can view issues in a project
Reporter	The user can report issues to a project, and comment on issues.
Updater	Role as a “reviewed” or moderator for the project.
Developer	The user can be assigned to an issue, and change the data tied to an issue.
Manager	The user can manage a project.
Administrator	Full rights

d. Issue reporting

Mantis account is compulsory for reporting an issue to the system. Different projects have different reporting forms, depending on defects types. The general compulsory fields for all defects are:

- **Category**
Reported issue can be defect, internal defect or improvement
- **Reproducibility**

The field describes the ability to reproduce the reported defect. The scale of properties is divided into always, sometimes, random, have not tried, unable to reproduce and N/A

➤ **Steps to reproduce**

If the reproducibility of defect is other than N/A, reporter is highly recommended to describe how defect is repeated or triggered.

➤ **Priority**

The priority entered by the reporter shall be low, normal or high. After reported to EPC, the priority is re-evaluated based on its estimated cost over benefit ratio regards to quality and process performance objectives.

➤ **Severity**

The severity of reported defect can be minor or major. The major defects are defined:

- Issue that disturbs or hinders the operation of the engine
- Issue that hinders the creation of a working software package
- Issue that has a big impact on the engine performance
- Issue that breaks redundancy for a single main engine
- Wrong version number on a release, application version, Tool version, HW module version etc.
- Issue that goes against class society rules
- New functionality or functionality change visible for the end customer

Meanwhile, minor defects are defined:

- Issue that is related to tools and does not cause major issues for package creation or engine operation
- Issue that causes annoyance
- Issues with existing workaround
- Typo in a parameter name or missing tooltip or other graphical issues

(Jakobsson, 2016)

➤ **Assign To**

The responsible person who is assigned to the defect and to coordinate the resolving.

➤ **Summary**

The summary of a defect can be considered as the title of the reported issue.

➤ **Description**

Describes the defect including its unexpected behavior and consequences. Also, the environment condition and location should be included for defect handler to fasten the process.

➤ **Additional information**

The field describes the background information, how and where the defect is found. It also includes the acceptance criteria if possible.

➤ **Upload file**

Files and attachment can be included in the reported case.

In some Mantis projects, other extra fields, which might not contain valuable information for reporter but handler, are also shown in the reporting form.

The screenshot displays the 'Enter Report Details' form in Mantis. The form is organized into several sections:

- Category:** A dropdown menu with '(defect)' selected.
- Reproducibility:** A dropdown menu with 'have not tried' selected.
- Severity:** A dropdown menu with 'minor' selected.
- Priority:** A dropdown menu with 'normal' selected.
- Product Version:** A dropdown menu.
- Summary:** A text input field.
- Description:** A large text area.
- Steps To Reproduce:** A text area.
- Additional Information:** A text area.
- Specification status:** A dropdown menu with 'No specification' selected.
- Test case ID:** A text input field.
- Test case review:** A dropdown menu with 'No test case' selected.
- Upload File:** A file upload button with the text '(Browse) No file selected.'
- View Status:** Radio buttons for 'public' (selected) and 'private'.
- Report Style:** A checkbox for 'check to report more issues'.

A 'Submit Report' button is located at the bottom right of the form.

Figure 17. Mantis issue reporting interface

e. Tracking

Mantis tool provides several ways for the reporter to follow the progress without unnecessary interrupt the assignee by calling or email. When a defect is created, the reporter is provided with a hyperlink to track its progress. In addition, whenever there is any update in progress of an issue, reporter receives an email notification including summary of update information. This feature is normally considered to be very convenient but

sometimes, it can become annoying if every single update would trigger an email notification to the user (huge number of emails)

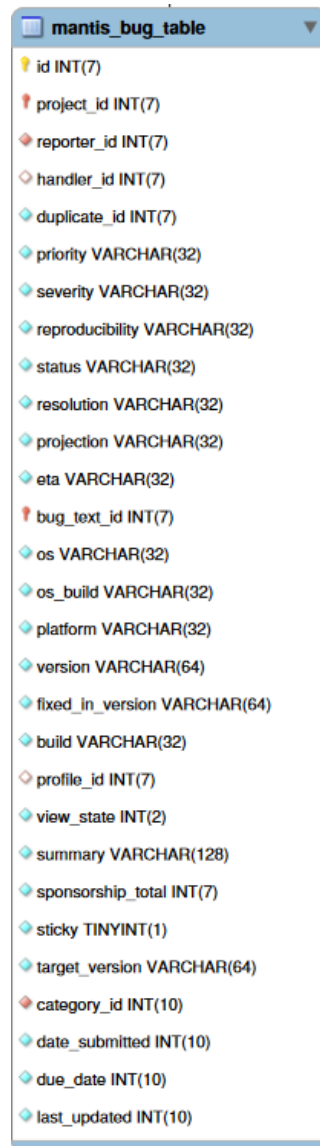
Moreover, “Send a reminder” function provides capability to add user to watches and passively receive emails notification about every update.

Based on the status of an items, user is aware of its progress. The definitions of the items’ status are mentioned in quality plan (Österback, 2015):

<i>New</i>	the case has not been reviewed
<i>Acknowledged</i>	the case has been reviewed, but no decision about implementation is made
<i>Confirmed</i>	the implications of the case has been analysed, the case is approved and will be executed
<i>Assigned</i>	the case is approved and has been assigned to a responsible person
<i>Progress</i>	the requested change is being developed
<i>On hold</i>	the process cannot proceed until other activities are performed
<i>Completed</i>	the development work is done, but all testing is not yet performed
<i>Resolved</i>	the development work is done and tested
<i>Closed</i>	the requested change is released for all concerned systems, i.e. all child cases are <i>Resolved</i> , or, the case is rejected
<i>Feedback</i>	more information needed before the process can proceed

f. Exporting items

Mantis BT has an ability to directly export a list of bugs in different formats such as XML, CSV. Besides, users can always make a query to database (e.g.: MySQL) or using SOAP over HTTP (/mantis/api/soap/mantisconnect.php?wsdl) protocol to get raw data from Mantis server.



Field Name	Data Type
id	INT(7)
project_id	INT(7)
reporter_id	INT(7)
handler_id	INT(7)
duplicate_id	INT(7)
priority	VARCHAR(32)
severity	VARCHAR(32)
reproducibility	VARCHAR(32)
status	VARCHAR(32)
resolution	VARCHAR(32)
projection	VARCHAR(32)
eta	VARCHAR(32)
bug_text_id	INT(7)
os	VARCHAR(32)
os_build	VARCHAR(32)
platform	VARCHAR(32)
version	VARCHAR(64)
fixed_in_version	VARCHAR(64)
build	VARCHAR(32)
profile_id	INT(7)
view_state	INT(2)
summary	VARCHAR(128)
sponsorship_total	INT(7)
sticky	TINYINT(1)
target_version	VARCHAR(64)
category_id	INT(10)
date_submitted	INT(10)
due_date	INT(10)
last_updated	INT(10)

Figure 18. Mantis bug table in MySQL

g. Statistics

Some statistical data are extracted from Mantis database for the decision making process later. For that purpose, a small PHP application was written to connect and execute the query to MySQL database. The PHP program language is selected for several reasons:

- Available together with Mantis
- Ease of use compared to SQL functions for the same purpose.
- Output formatting.
- Visualization
- Personal experience and preference.

The target metrics are:

- Number of projects, active projects
- Number of ongoing, closed cases
- Number of concurrent users (Figure 19)

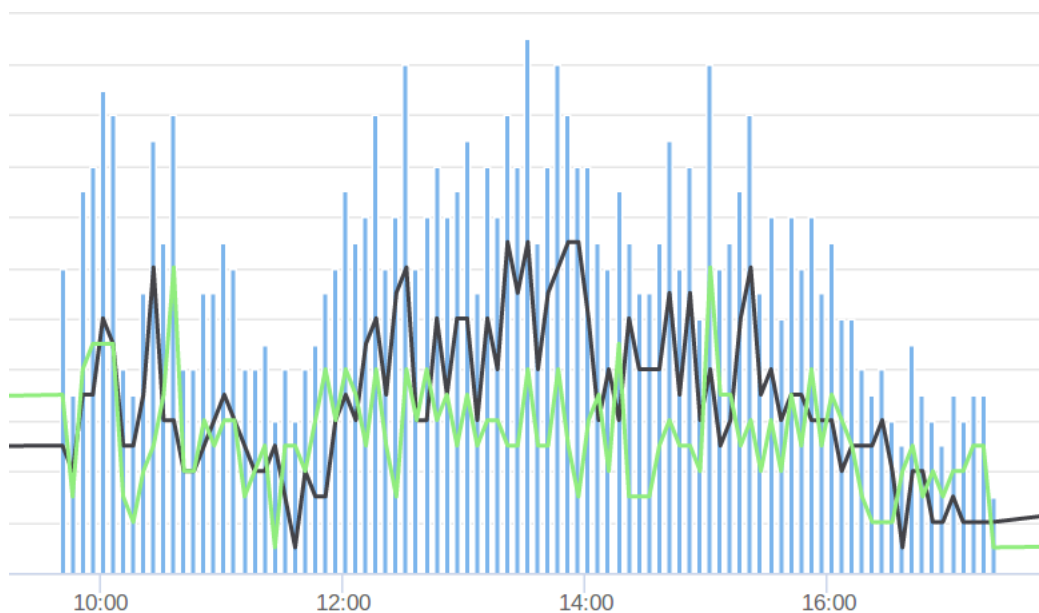


Figure 19. Raw measurement of concurrent users

For simultaneous connections measurement, every 5 minutes, a recorded for number of users interacting with Mantis in the last 5 minutes is recorded and by email address, users are categorized into two groups of users: containing ‘wartsila’ and not containing ‘wartsila’. However, the query cannot analyze if the user is internal EPC or from other Wäertsilä departments. The raw measurement is visualized using Highchart visualization framework for JavaScript for real-time monitoring (Figure 19). The value is also converted into frequency distribution for later analysis.

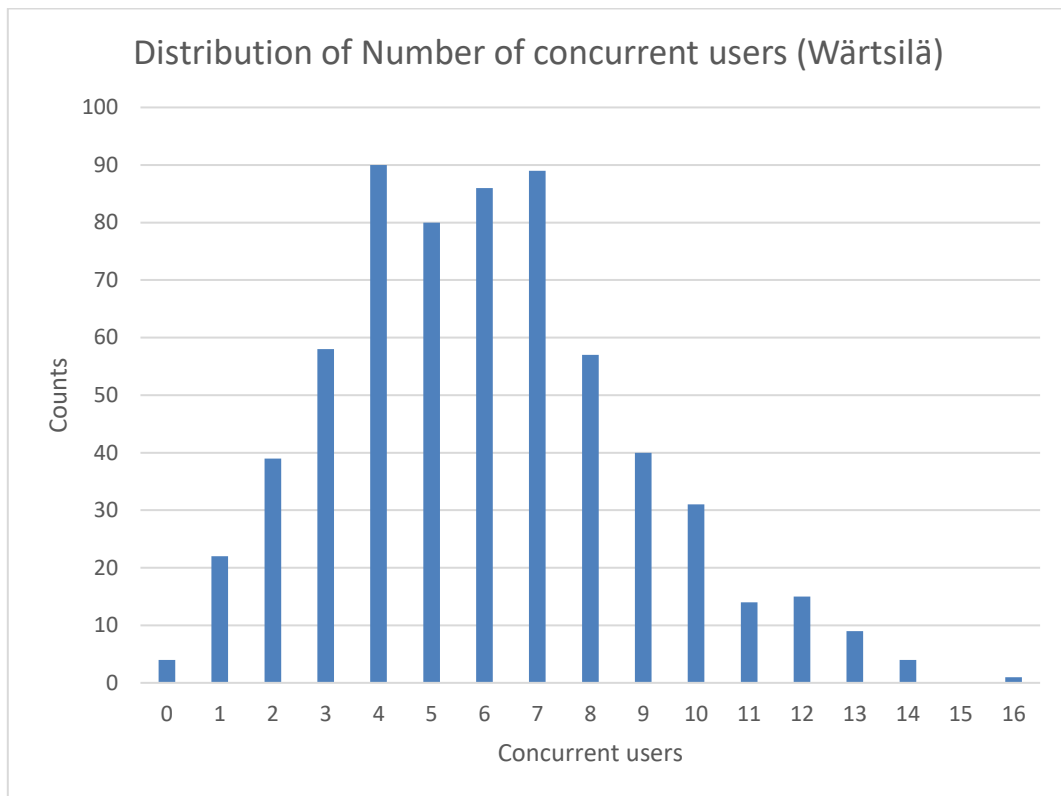


Figure 20. Concurrent user measurement (@Wärtsilä)

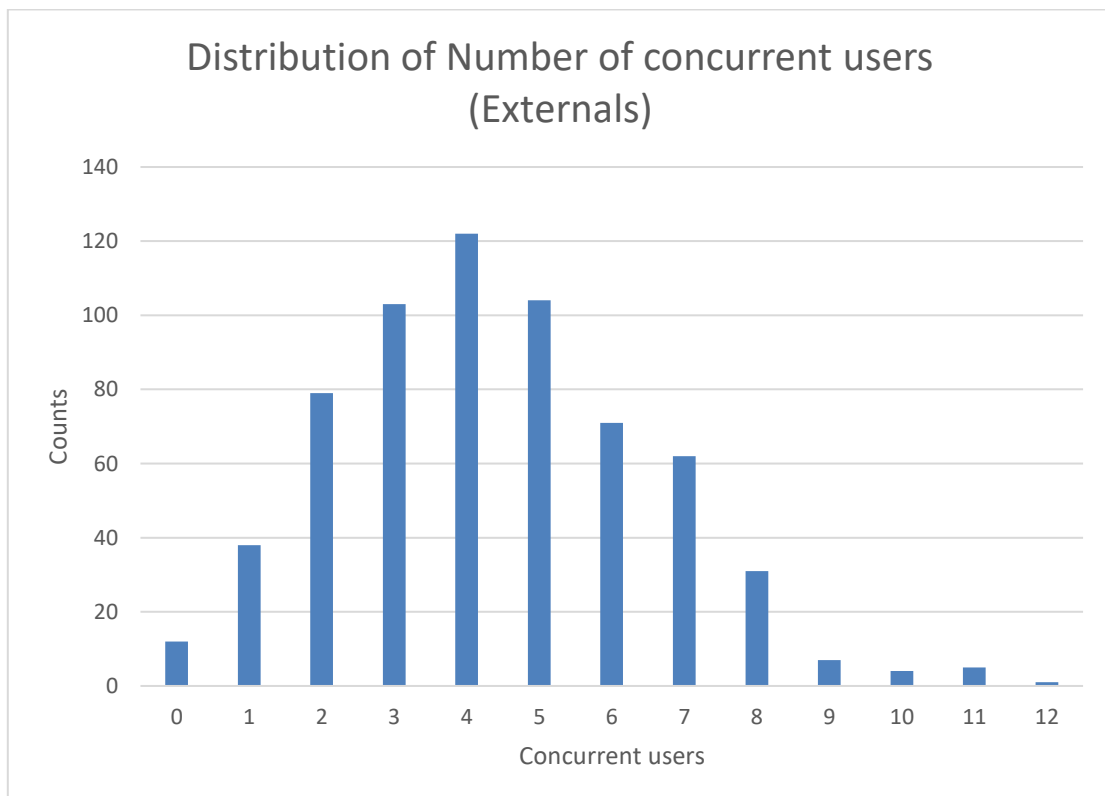


Figure 21. Concurrent user measurement (!@Wärtsilä)

3.1.5 Polarion ALM

a. Polarion

Polarion Software was introduced in 2004 with the mission to advance development, governance and maintenance software by a unified application for requirement, quality and application lifecycle management. The first release of Polarion was in 2005 and nowadays, the application has reached 2.5 millions of users and is part of Siemens PLM Software.

The Polarion unified ecosystem enables organizations, ranging from automotive, medical aerospace and embedded systems industries, improving collaboration, efficiency, productivity, workflow control, integrity, safety, reusability, traceability and in general, quality and reliability while saving time and cost. The application is certified for its compliance with ISO26262/IEC61508 standards (entinn, 2012).

With different customization of features, the tool provides a certain degree of flexibility so users can select the most appropriate one for their needs. The categories consist of Polarion ALM, Polarion QA, Polarion REQUIREMENTS, Polarion PRO, and Polarion REVIEWER. The basics differences in each category can be seen in the picture or via the product features matrix (Polarion Feature Matrix , 2016).

Polarion core functionalities include collaboration, traceability and workflow. Users are able to directly communicate with each other and also to upload attachments. The tool also help auditing, compliance or regulatory inspection easier with traceability of items, together with history of changes (version control). Besides, all configurable items can have their own workflows to control the state transient. Those features are believed and verified by many customers to bring significant benefits to them. (<https://polarion.plm.automation.siemens.com/customer-success-stories>)

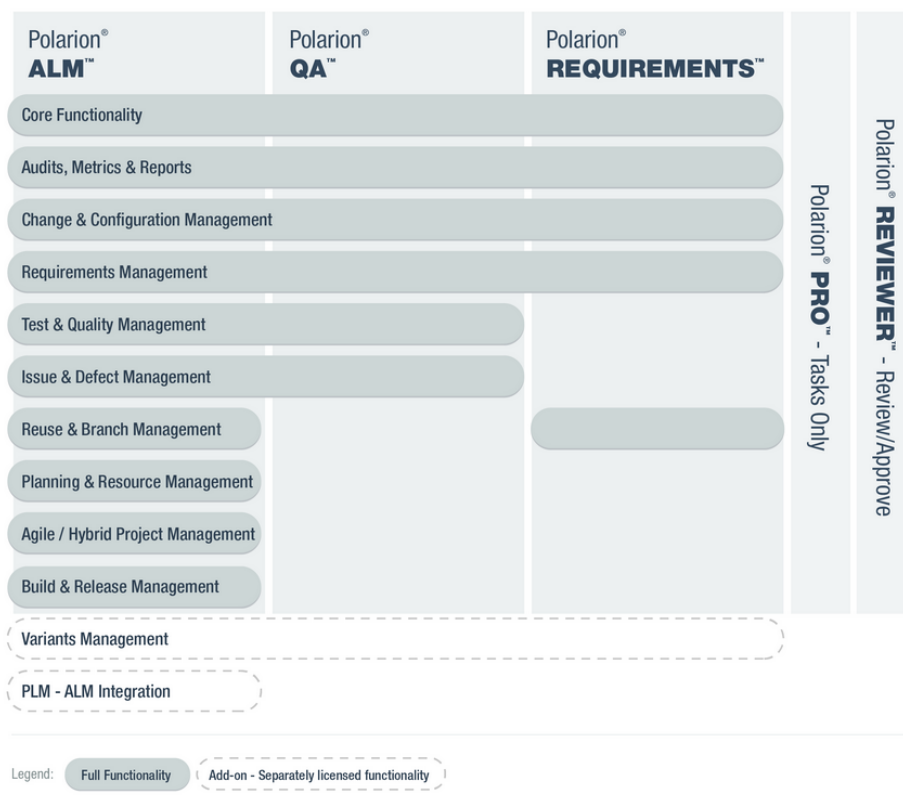


Figure 22. Polarion license type features (Polarion Overview, 2017)

b. Project structure

Similarly, Polarion also has project structures like in Mantis. The configuration of the project can be inherited globally or from a template, or modified based on needs for each projects. As part of the change and configuration management concept, the number of variants is kept as low as possible so in summary, there are four project types/templates that can be used in EPC's setup (Figure 23):

- **Products:** Information about hardware or software platforms and application modules.
- **Projects:** Information about development, research and customer projects.
- **Teams:** Information about resources and tasks management.
- **Systems:** Quality management systems and automation systems.

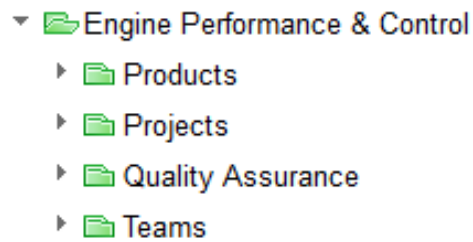


Figure 23. EPC's Polarion project structure

c. Work item

The Polarion work item is an artifact of the development process that is waiting to be implemented, in-progress has already been implemented. In configuration management, Polarion work item can also be considered as a configuration item, which is the selection and specification of the following:

- Products delivered to the customer
- Designated internal work product
- Acquired products
- Tools or capacity assets of the project's work environment
- Other items used in creating and describing these work products

(CMMI Product Development Team, 2010)

Some examples of work items that has already been in used by EPC are Work request, Task, Requirements, User story and Test case.

Each work item type has several default fields and custom fields (Figure 24) that are defined by users. Field types can be string, text, rich text, Boolean, integer, float, currency, date time and enumeration.

task-custom-fields.xml						
ID	Name	Type	Description	Multi	Required	Default Value
COST_CENTER	Cost element	String (sing.)	Cost center, Project or WBS number	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
		String (sing.)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>

Figure 24. Custom field configuration

The items can be linked to each other with predefined links role, which is one of the key features to take advantage of traceability and impact analysis. With appropriate configuration, several benefits from work items linking are:

- Possibility to trace from requirements to the Work Items and source code that implement them
- Possibility to back-trace from some defective code (in a repository revision) to the tasks/change requests that implemented the defective code, and from those to the requirement that called for the functionality.
- Possibility to assess the impact (and by extension, the cost) of a changed requirement by seeing all the artefacts that would be affected by the change.
- Possibility to review and analyse various relationships between Work Items such as dependency, relevance, parent-child, duplication, and follow-up.

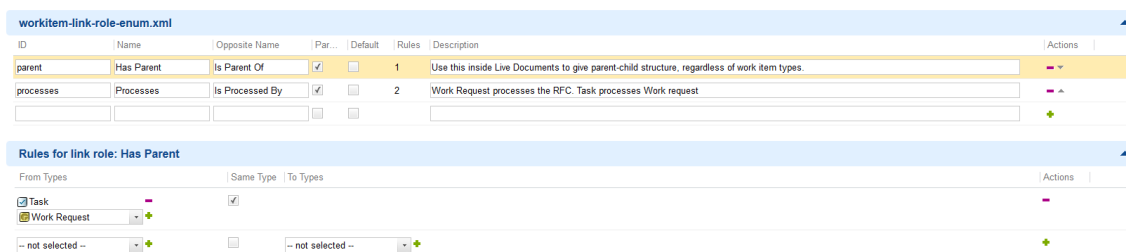


Figure 25. Polarion work items linking

d. Polarion connector

Although Polarion ALM covers all aspects of application lifecycle, it is not always possible for an organization to adapt the complete solution. This case might be true for EPC while Mantis BT has been proved to be a reliable solution for issue tracking.

The Polarion connector is internally implemented Polarion and can be configured in Administrator interface. The feature provides either unidirectional or bidirectional synchronization of data between Polarion and third-party solution. The following section describes how it works:

- 1) Load all items that match a query and other parameters from Polarion ALM.
- 2) Load all items that match query and other parameters from another system.
- 3) Determine what needs to be done with the items.
 - 3.1) Create "pairs" of items that have been synchronized before by looking up the connection information stored in folder [POLARION]/data/synchronizer.

- 3.2) If the "partner item" is found in the queried items, create a pair to be synchronized.
 - 3.3) If an item is connected, but the "partner item" was not found, try to load the partner item by ID to create a pair to be synchronized.
 - 3.4) If loading the item by ID fails, consider the item as deleted.
 - 3.5) Consider all items that are not connected according to information in [POLARION]/data/synchronizer as new items.
- 4) Create new items, synchronize paired items and propagate deletions according to the connector configuration.

The list of currently supported tool does not include Mantis (only Siemens Team Center, Atlassian JIRA, HP Quality Center/ALM, MATLAB Simulink, Microsoft TFS, Perforce, and Salesforce). However, any other applications implementing OSLC (Open Services for Lifecycle Collaboration) can be connected to Polarion OSLC.

Integrations
More than 44 Extensions waiting for your Polaron.

- Show all categories
- Templates
- Agile
- Workflow
- MS Office
- Export & Import
- Integrations**
- Wiki
- Subversion
- Test Automation
- Reports

Featured Extension
salesforce
brings Sales & Support teams in the loop with directional synchronization of Salesforce Cases with Polaron. **BUY for \$9900.00**

All Integrations extensions Free & Paid For All Products

- Integration with VectorCAST/RGW**
Allows requirements tracked in Polaron to be imported into the VectorCAST toolset for mapping to software unit and integration test cases. Test results (pass/fail status of each test) are then
- Polarion Connector™ for Teamcenter™**
Bridge the worlds of ALM and PLM with data federation and process orchestration that have proven to deliver interoperability and end-to-end traceability for complex, multi-system product
- Polarion BIRT Report Viewer**
Integrate BIRT Reports directly to Polaron Web User Interface.
- Role Based Enumeration Filter**
Filter the options available for selection based on the (global or project) roles of the current user.

Figure 26. Polaron extensions list

3.2 Analysis

3.2.1 Processes

The processes have been created and updated for only half of year as a result of organization changes. However, the review on the current processes pointed out one minor problem which was immediately fixed but needs to be mentioned to avoided in the future work.

The **common agreement** for the processes was not examined thoroughly at the beginning by all of concerned stakeholders. Although the role and responsibility of the Support team is clear as the first contact point for engine automation software related nonconformity, the details of the process were not clearly defined. **The existence** of different Mantis projects, which are available to be selected for different defect types or root causes, creates ambiguity for a reporter, who normally cannot analyze completely the problem. On the other hand, the definition of some fields was not commonly defined due to different point of views among departments. For example, severity and priority of a case can be easily mismatched between Services and Technology, when the former analyses situation from customer point of view but Technology also needs to consider pre-agreed roadmap of the Wärtsilä or Marine Solutions, Technology. Therefore, the planning and roll-out need an involvement from all related persons or teams to improve the processes performance.

3.2.2 Tool implementation

a. Mantis

In general, Mantis has been in used for approximately 10 years and is heavily customized. In the tool, issues are linked together with different categories and in different projects, configurations are significantly varied. Therefore, the current implementation in Mantis is assessed to be complicated and cannot be directly transferred into Polarion in any simple ways.

Mantis at the moment have more than 500 registered users and very active used with its configuration to be an agile tool. Although with some features/functions have been implemented or moved to Polarion, the monitored trend of Mantis activities does not show any significant decline.

b. Polarion

In contrast to Mantis, Polarion has been being used for recent years and are still mainly used for project, requirement and task managements. All of the development is well aligned at the beginning, implementing full or simplified version of Polarion default template and good practices. With Polarion flexibility, it is assessed as a potential candidate to be a suitable ALM tool for the department without any detected considerable limitation.

At the moment, Polarion have 433 registered users in the system and is constantly growing in number of active users after requirement, release and test managements implementations.

4 PRE-STUDY PHASE

4.1 Requirements development

4.1.1 Stakeholders identification

By conducting a research on the current status and foreseeable changes in defect management processes, list of involvements has been defined.

Defect reporters and resolvers:

- **Engine Performance & Control (EPC) department**
The department consists of experts in different fields who directly contribute to the common development of Wärtsilä engine automation system. Therefore, it is internal experts who find and later resolve defects related to the system.
- **Testing & Validation (T&V) department**
Being responsible for testing and validating unreleased systems on different engine platforms, T&V reports test results and defects to EPC developers.
- **Technical services (TS) department**
After systems are released to production or field, defects found in the step are reported to EPC by Technical services.
- **Consultants & externals**
Inside EPC, other than internal experts, the development of systems also requires involvement of consultants and suppliers. Therefore, it is also similar that the group is also responsible for detection of nonconformity and corrective actions.

Application administrator

- EPC Polar Bear
The virtual team is responsible for planning and rolling-out of Polarion features, e.g.: defects management in Polarion.

Processes:

- Nonconformity handling processes (HW, SW)
- Development (HW, SW, Control application)

- Release (SW package, standard)

Measurement monitoring:

In general, the department's management team, the project, the product managers are mainly the ones who follow the defect system performance. However, assessments and feedback from every direct user will contribute to the development of the system.

4.1.2 Requirement elicitation results and analysis

In general, the basic features of existed system shall be available to whatever new solutions are provided. Those include:

- Project structure/product categorization for items
 - Reported issues shall be categorized to the appropriate products/components that are impacted
- Notification
 - Users, including reporters and components' owners shall receive notification for reporting, updates, changes in the issues
- Customizable reporting form and workflow
 - Regarding to product types, whether application module, hardware or software, defect items' workflow and model shall be customizable at certain degree of freedom.
- Report items
 - Certain fields of defect items shall be modifiable by reporters.
- Tracking reported items
 - Reporters shall be able to search and follow their reported issues
- KPIs, reports
 - Defined reports and metrics shall be generated from the system
- Unlimited access for certain group of users to the system
 - Group of frequent user group shall have unlimited access to the system
 - License issue shall not prevent users to work with the system
- Users, licenses monitoring
 - System shall have licenses and concurrent users monitoring
- Access right configuration

- Access right to each issue and impacted component information shall be managed

4.2 Alternative approaches for defects and implementation plans

In general, there are two feasible approaches for having defects in Polarion: using Polarion as single system and integration of Polarion and Mantis.

4.2.1 Using Polarion only

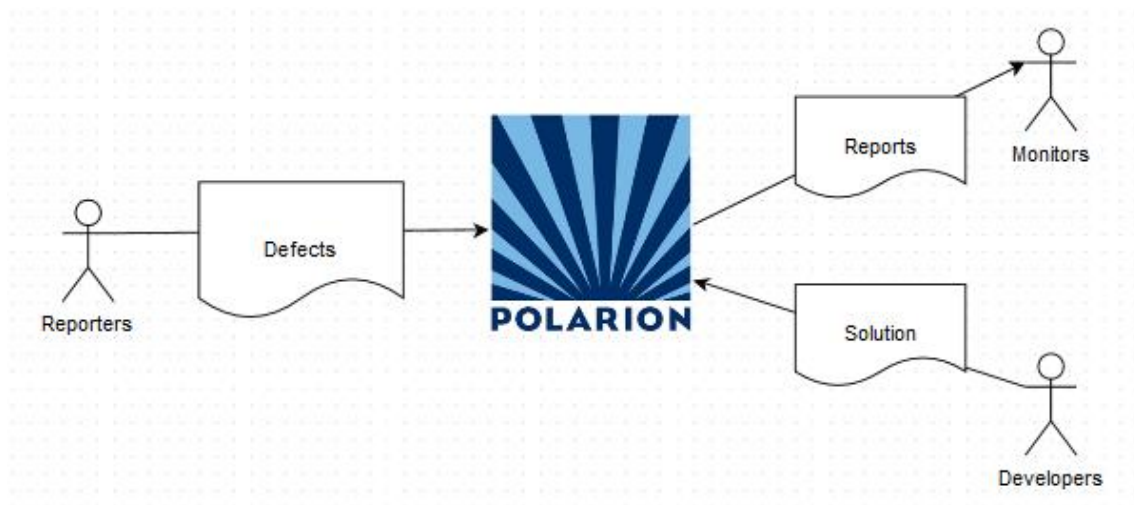


Figure 27. Polarion single system option

As shown in figure 27, reporters will report whatever type of defects directly to Polarion. This also includes cases reported from failed Polarion test run execution, which is automatically generated and linked to test case.

After defects are reported to Polarion, product's owner will receive notification by email and directly work in Polarion's defect case. Impact analysis can start with linking defects to its failed requirements by using traceability matrix feature.

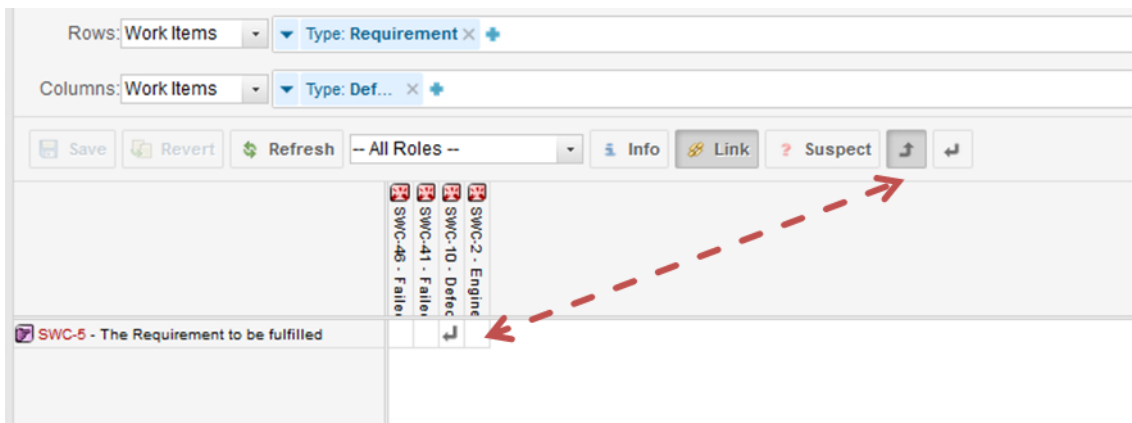


Figure 28. Polarion traceability matrix

Items to be displayed in the matrix can be filtered out in the query bar. As can be seen in figure 28, link role type and direction can be selected in the tab and by clicking to matrix element, two items are linked together with selected relationship. A summary of requirements and linked test cases, defects can be generated automatically by Polarion widget (figure 29).

Requirement that have defects

ID-Title

SWC-5 - The Requirement to be fulfilled

SWC-1 - Engine Start-up time

SWC-47 - Standard CR configuration 2017d1 - EXAMPLE

SWC-10 - Defect with Requirement WI linked

Figure 29. Summary of traceability

a. Data migration

With the option to stop using Mantis and use only Polarion, data migration plan needs to be considered. Basically, there are two approaches (Figure 30):

- Move all Mantis issues to Polarion
- Move selected Mantis issues to Polarion

The “Not moving anything” option is not listed here because with a huge number of cases ongoing in Mantis (different categories), it will be a lifetime of Polarion for closing them.

Moving all Mantis issues to Polarion means all current ongoing cases in Mantis will be closed while its clones are created in Polarion. Closed cases in Mantis will have reference (Hyperlink/Polarion id) to Polarion system. The reference to new cases will guarantee that all linked materials, for instances documents and guidelines, don't need to be updated to Polarion cases. The option will benefits developers so that the transient time, when they need to switch between two applications for the same work, will not suffer. However, with more than 300 active Mantis projects with huge variant in configurations, mapping them to Polarion requires enormous workload that no simple rule/script can handle. Even more important, with a huge number of Mantis cases, Polarion needs to be tweaked in Mantis way, keeping old WoW and reducing the efficiency of initial Polarion idea.

On the other hand, the selected migration resolves the challenges of complexity. The option means that with defined criteria, only certain cases and projects will be moved to Polarion after manual reviewing. If the cases are decided to move to Polarion, they will be closed with two-way reference. As being mentioned above, the option can resolve problem of enormous mapping project-to-project, type-to-type at once but still have disadvantages of undefined transient time due to manual review. Anyway, this disadvantage is a relatively tricky one as it raised from the question whether every single case will be reviewed and until when?

Move <u>ALL</u>	Move <u>SELECTED</u>
- All <u>ongoing</u> (even <u>closed</u>) cases to Polarion	- Manual selection of necessary projects & cases (E.g.: Release in Polarion)
- Close all Mantis case with 2-way ref.	- Close Mantis case with 2-way ref.
(+) No transient time for developers	(+) Complete Polarion benefits
(-) Manual mapping (<u>ALL AT ONCE</u>)	(-) Manual mapping (<u>SELECTED</u>)
(-) Mantis <u>WoW</u> -> Polarion <u>WoW</u> ?	(-) Undefined transient time

Figure 30. Data migration options comparison

b. Licenses analysis

Types

First of all, Polarion product feature matrix is considered to narrow down the license types which will be used for Polarion's users (Polarion Feature Matrix , 2016). In figure 32, a list of necessary features is selected to compare PRO, REQ and QA licenses. The ALM license enables all Polarion features while with REVIEWER license, users can only comment, sign, approve and view work items so they can be off the comparison table.

Features\License	PRO	REQ	QA
Live document (MOM)			
Traceability	No traceability matrix		
Requirement management			No live branch
Test management	Only test execution	Only test execution	
Plan (Personal)			
Risk management		No template	

Figure 31. Feature comparison

The most important feature is Plan, which is widely used at the moment for personal, sprint and releases plan by the department, is not available for QA, REQ and PRO license types so none of them are feasible for EPC internals. However, PRO license with ability to create, update work items, is enough for the suppliers' needs and scope of their work. Therefore, ALM license needs to be allocated to EPC internals while PRO license can be used for others.

Number of licenses

The number of licenses needs to be estimated as accurately as possible. The two methods used for calculation are applying probability model and Erlang calculation. From the frequency distribution of Mantis concurrent users table in section 3.1.4.7, two statistic models are applied to calculate cumulative confidence interval for number of license required.

Internals			Suppliers		
No of license	% Out of license (1)	% Out of license (2)	No of license	% Out of license (1)	% Out of license (2)
8	23,83%	15,08%	6	22,32%	15,24%
9	14,36%	8,26%	7	10,97%	7,61%
10	7,82%	4,19%	8	4,51%	3,44%
11	3,84%	1,97%	9	1,54%	1,42%
12	1,69%	0,86%	10	0,43%	0,54%
13	0,67%	0,35%	11	0,10%	0,19%
14	0,23%	0,14%	12	0,02%	0,06%
15	0,07%	0,05%			
16	0,02%	0,02%			

Figure 32. Confidence interval for licenses

In figure 32, the (1) is marked for calculations using normal distribution and (2) is marked for calculations using Poisson distribution. The random variable satisfied all the assumption so normal distribution and Poisson distribution are both appropriate models:

- High number of sample size
- The occurrences of measurement result are independent as clearly observed in the graph that between 8:30am and 4:30pm, whatever value can be recorded. (Figure)
- An infinite number of occurrences of measurement result can occur in the interval because there's no limitation for Mantis connection.

On the other hand, Erlang traffic model can be used to calculate the number of needed license. From the measurement result, the highest number (1-hour range) in Wärsilä group is 39 and for the Externals group is 32 (arrival rate). With the assumption that each user when connected to the system will spend 5 minutes in average, 0.5% will face running out of license situation, Erlang B's returned values are 3.25 and 2.678, which also means the numbers of licenses for each group are 9 and 8, respectively.

However, as mentioned in section 3.1.4.7, the number of Wärsilä users cannot be categorized into inside and outside EPC so in reality, the ALM license reserved for internals might not be as high as calculated. Besides, the intersection of Polarion the active users set and the Mantis active users set might need to be considered, at the moment or in the near future when release and task managements are widely rolled out. In addition to that. In addition to that, the current ALM license pool has not reached its maximum capacity

and there is still some free spaces, up to 50% or even higher than that at certain time points during the manual observation. Thus, certain value of number can be taken out for above reasons.

In summary, using historical data might be preferred in this case due to some certain unique characteristics of system type so the first result will be used in the proposal. The final values that are presented to management team are 8 ALM licenses for internals and PRO licenses for externals. The calculation of prices can be seen in figure 33.

<u>Internals</u> with @Wärtsilä (EPC, TS, T&V)	<u>Suppliers</u> not with@Wärtsilä
<p>ALM license (13 – 5) x 9,250 = 74,000\$ ~ 67,900 €</p>	<p>PRO license 10 license x 2,350 = 16,450\$</p>
Total: 90,450\$ - 83,000€	

Figure 33. Cost calculation

c. Transition time

With a wide range of product or component portfolio, corresponding to Mantis projects, a decision about when to move is considered to be dependent. One single proposal for the issue is when Polarion configuration is ready and the last release for each component in Mantis is done. The time point is appropriate that early migration can leads to reverse cloning situation (Polarion -> Mantis), which is completely against the trend, while late migration just extend the transient time of the migration.

4.2.1 Polarion and Mantis together

The second approach provides an opportunity for having both systems running at the same time. The Mantis tool will be the place to receive incoming defects and part of resolvers can directly work in Mantis if it's unnecessary for traceability or releases. On the other hand, Polarion is still mainly used for traceability and other ALM functions so the need to have a connector for two-way synchronization is undeniable. Two options can be selected are developing the connector from scratch or using commercial 3rd party tool.

As being partly discussed in 3.1.4.5, OSLC open standard can be used as ALM tools integration specification. Polarion has fully supported the standard but Mantis has not so the work is actually about to develop a plugin for Mantis for connection with other ALM tools. The self-development provides certain degree of flexibility in customization and later will perfectly match the requirements. However, the work is estimated to be too complicated, in both development, update and maintenance then outsourcing is a must option. With too much uncertainty and unclear ROI, the option is negligible when being compared to the other.

Agosense is a German-based company and the only supplier providing the kind of solution that supports both MantisBT and Polarion ALM (About agosense , 2017). With the assumption that the solution works reliably, it satisfies the requirement to have defects item in Polarion with the least impacts on current way of working. By using centralized common service bus and specific adapter for each ALM tool, the solution also opens an opportunity for further integration with others such as Atlassian Jira, CA Agile and Microsoft Projects.

However, the connector solution itself has several disadvantages when compared to having only single system. First of all, having multiple basically raises the complexity in development, implementation and maintenance. The work is tripled whenever there is a need in reconfiguration due to update, bug and so on. Moreover, Mantis connector is not officially supported or referred by Polarion as its absence in extensions list and from consultant point of view, there is significant work to be done with uncertainty the option cannot be recommended.

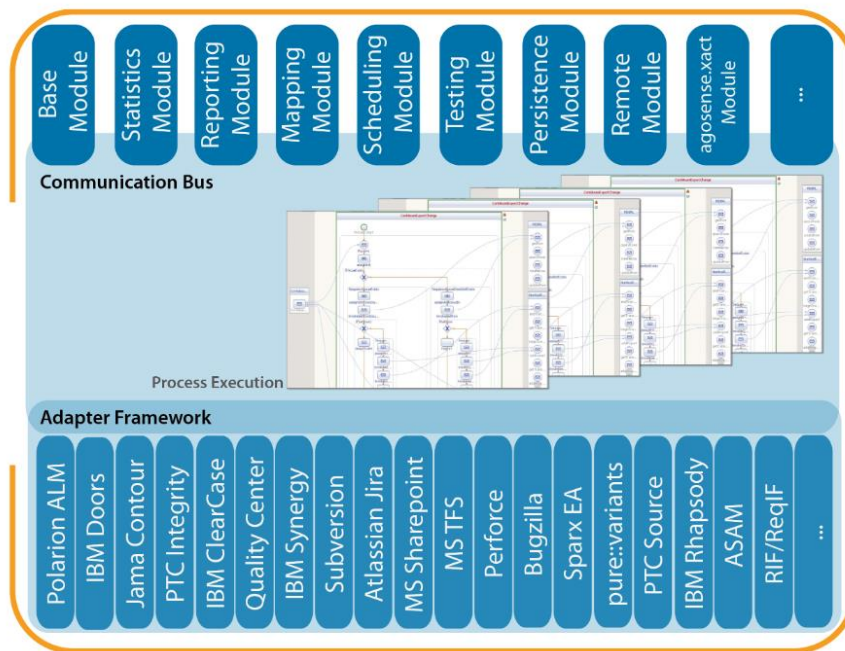


Figure 34. Agosense architecture (agosense.instruments, 2017)

a. Decision making by management team

During the decision-making meeting, Polarion is decided to be a single system for engine automation system's nonconformity management. Besides, the transition will start by closing Mantis's interface to receive issues while, at the same time, redirect user for the new Polarion interface. For existed data in Mantis systems, only selected ones will be moved to Polarion with two-way reference for the needs of traceability.

5 IMPLEMENTATION

Due to the scope and work amount limitations of thesis, only some implementation is done and presented. The work includes configuration of defect artifact in Polarion components' project and preparing the rest of the plan.

5.1 Defect item model

The internal defect model is presented in table 3. Some fields are mandatory for the report creator to fill-in and some are for resolver during defect handling. These include both default (e.g.: title, severity, description, status, resolution, assignee) and custom (e.g.: defect report, engine type, defect type, forward-link to requirement) fields.

Table 4. Component's defect work item

Field	Custom field (X for yes)	Filled by	Data type	Description
Title		Reporter	String	Title or summary of defect information
Defect report	X	Reporter	Rich-text(multi-line)	Written by reporter to describes symptom, reproducibility and so on
Severity		Reporter/ Resolver	Enumeration	Defect's severity
Engine type	X	Reporter	Enumeration	Which engine type that defect was found in
Defect type	X	Reporter	Enumeration	Internal or field defect
Description		Resolver	Rich-text	Written by resolver to clearly describe the reported defect
Analysis		Resolver	Rich-text	Analysis written by resolver to analyze impact, consequences of a defect
Root cause analysis	X	Resolver	Rich-text	Written by resolver to describe root cause of defect

Forward link to requirement	X	Resolver	Link role	Defect is required to link to requirement before going to the next status
Status		Resolver	Enumeration	States which indicates progress of resolving case
Resolution		Resolver	Enumeration	Resolution of a defect
Assignee		Resolver	Enumeration	Responsible person for defect handling.

In case the defect is automatically generated from failed test case (figure 35), defect type's default value is internal and description is auto transferred from test case description. The defect is also auto-assigned to responsible person of component.

Figure 35. Defect from test execution

5.2 Defect workflow

The workflow of the defect is described in figure 36. When being created, the defect item has “To be Analyzed” status and to be able to move to “Analyzed” state, defect is required to be connected to a requirement and the analysis field is filled in (figure 37). If the fix version for the reported defect is agreed, item is moved to “Fix planned” and it is only be closed when the fix version is released. In Polarion configuration, there is certain flexibility for user can turn the case back to “To be Analyzed” or directly to “Closed”.

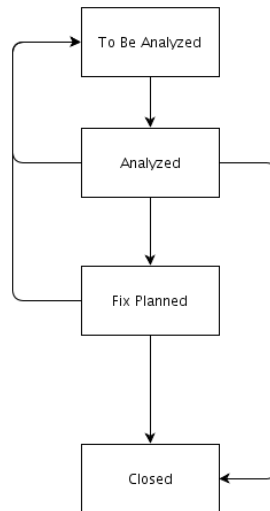


Figure 36. Defect's workflow

Actions						
ID	Name	Required Roles	Required Fields	Cleared Fields	Initial	Requires Signature
toTo-be-analysed	re-analyze		severity,DefectReport	resolution,resolvedOn	<input checked="" type="checkbox"/>	<input type="checkbox"/>
toAnalysed	set to analysed		Analysis		<input type="checkbox"/>	<input type="checkbox"/>
toFix-planned	set to fix planned				<input type="checkbox"/>	<input type="checkbox"/>
toClosed	set to closed		resolution		<input type="checkbox"/>	<input checked="" type="checkbox"/>
					<input type="checkbox"/>	<input type="checkbox"/>

Details for Action: set to analysed ✕

Close

Conditions ▲

Condition Actions

LinkedWorkItem ✖ ✎

+ ✎

Functions ▲

Figure 37. Required field and function configuration

5.3 Defect's link role

For traceability, a link role is required and for different relations, different link roles are made. To be able to connect to requirement and test case, two connections are made: “was found in” and “fails” respectively. Example of configuration for defect-requirement relationship can be seen in figure 38.

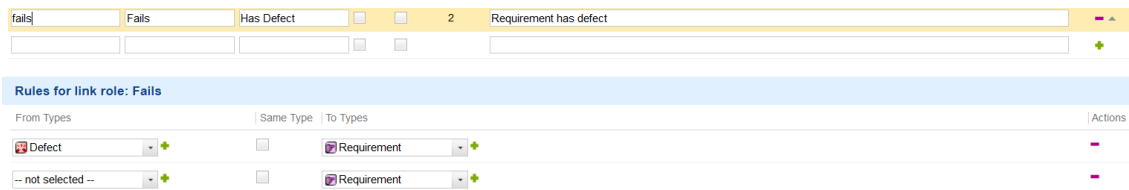


Figure 38. Defect-requirement Link role

5.4 Next steps

The basic configuration is relatively enough for the system to be in piloting phase. However, the work is just partly covers internal defect management only and the rest of the work shall be planned and they include:

- Field defect and its configuration
- Live reports
- Training and guideline for different parties
- Coordination of processes updates
- Concrete planning about migration and piloting projects

6 DISCUSSION AND CONCLUSION

At the end of the thesis work period, it is vital for personal assessment of what has or have not been done what can be improved.

In general, the pre-study phase has been done relatively well with several requirements elicitation events and feasibility study about the initial proposal. With calculated gains/cost and different proposed approaches, a basis decision was made from the management team for further development of the project. In the scope of the thesis, basic defect model and several related configuration has been done and ready for piloting phase.

However, there are several points that can be improved. the pre-study phase could be shortened and it also means that there would be more time for the implementation phase. At the beginning, lack of experience led to missing of some tasks in initial planning and the work breaking down. Besides, personal tasks prioritization was not done properly that some of the thesis work, which should have had the highest priority, were left behind other development. Therefore, these are valuable lessons learned from the thesis work and hence, in later projects, the same problems can be avoided.

In conclusion, the work has built a basic structure for migrating defect management from the old system to Polarion ALM and open an opportunity for further development with automatic defect reporting from engine automation system on the fly.

7 REFERENCES

- About agosense* . (2017, 2). Retrieved from Agosense homepage:
www.agosense.com/english/
- About Wärtsilä* . (2017, 3 1). Retrieved from Wärtsilä homepage:
<http://www.wartsila.com/about>
- agosense.instruments*. (2017, 4 15). Retrieved from agosense:
<http://www.agosense.com/english/products/agosensesymphony/agosenseinstruments>
- Burnstein, I. (2006). *Practical Software Testing: A Process-Oriented Approach*. Springer.
- CMMI Product Development Team. (2010). *CMMI® for Development, Version 1.3*. Software Engineering Institute, Carnegie Mellon University.
- Dave West with Jeffrey S. Hammond, Mary Gerush, Sander Rose. (2017). *The Time Is Right For ALM 2.0+*. Forrester.
- entinn. (2012, 11 27). *A trusted Tool - Polarion qualified for ISO 26262-IEC 61508*. Retrieved from Polarion Blog:
<https://community.plm.automation.siemens.com/t5/Polarion-Blog/A-Trusted-Tool-Polarion-qualified-for-ISO-26262-IEC-61508/ba-p/380729>
- Freimut, B. (2001). *Developing and Using Defect Classification Schemes*. Institute Fraunhofer.
- Jakobsson, J. (2016). *EPC Quality plan*. Retrieved from Wärtsilä IDM.
- Joachim; Redler Rossberg (Rickard). (2014). *Beginning Application Lifecycle Management*. Apress.
- Koopman, P. (2010). Risk areas in embedded software industry projects. *Workshop on Embedded Systems Education*. ACM.

Koponen, T. (2006). Life cycle of Defects in Open Source Software Projects. *IFIP International Federation for Information Processing*. Springer.

Mantis. (2017, 3 9). Retrieved from Mantis homepage: <http://mantisbt.org/>

Mantis Project. (2017, 3 9). Retrieved from Mantis Wiki: <https://www.mantisbt.org/wiki/doku.php/mantisbt:projects>

MOSAIC, INC. (2017, 3 8). *Defect management*. Retrieved from MOSAIC, INC: <http://www.defectmanagement.com/defectmanagement/index.htm>

Österback, P. (2015). *Mantis Guide*. Retrieved from Wärtsilä IDM.

Polarion Feature Matrix . (2016). Retrieved from Polarion homepage: https://polarion.plm.automation.siemens.com/hubfs/Docs/products/Polarion_Feature_Matrix_2016.pdf

Polarion Overview. (2017). Retrieved from Polarion homepage: <https://polarion.plm.automation.siemens.com/products/overview>

Process Guidance and Process Templates for Team Foundation Server. (2017, 6 4). Retrieved from Microsoft developer network: <https://msdn.microsoft.com/library/hh533801%28VS.110%29.aspx>

SFS, Finnish Standards Association. (2015, 9 14). Quality Management systems. Requirements (ISO 9001:2015).

Software development Lifecycle. (2017, 4 15). Retrieved from University of Melbourne: <http://www.unimelb.edu.au/accessibility/users/development>

Vandermark, M. A. (2003). *Defect Escape Analysis: Test Process Improvement*.

Vuollet, T. (2017, 4 1). *Wärtsilä, Engine Performance & Control*. Retrieved from Wärtsilä Engine Performance & Control: https://wartsila.sharepoint.com/sites/compass/Operations/RD_and_design/engine_technology_activities/aut_control/Pages/Deliveries.aspx

APPENDIX 2

Wärtsilä, Energy Solutions. (2017, 3 1). Retrieved from Wärtsilä Energy Solutions:

<http://www.wartsila.com/energy>

Wärtsilä, Marine Solutions. (2017, 3 1). Retrieved from Wärtsilä Marine Solutions:

<http://www.wartsila.com/marine>

Wärtsilä, Services. (2017, 3 1). Retrieved from Wärtsilä Services:

<http://www.wartsila.com/services>