 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

UC 4.4
Medical certification and Requirements management Framework
Tool and methodology report
D404.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Tool and methodology report
Deliverable No.	D404.010
Dissemination Level	CO
Confidentiality	R
Document Version	V1.00
Date	2014-04-30
Contact	Dominique Segers
Organization	Barco
Phone	+32 56 233017
E-Mail	dominique.segers@barco.com

AUTHORS TABLE

Name	Company	E-Mail
Dominique Segers	Barco	Dominique.Segers@barco.com
Frederik Toune	Barco	Frederik.Toune@barco.com
Kurt Pattyn	Barco	Kurt.Pattyn@barco.com
Adriaan Coosemans	Barco	Adriaan.Coosemans@barco.com
Devid Verraes	Barco	Devid.Verraes@barco.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	31 Mar 2014	First version	All
0.02	2 Apr 2014	First Pilot Project input	All
0.03	4 Apr 2014	Update of Engineering Methods	All
0.04	8 Apr 2014	Update of Pilot Project Input	All
0.05	22 Apr 2014	Update after review 1	All
1.00	29 Apr 2014	Update after review 2	All

CONTENT

1 Introduction

1.1 Role of deliverable

This document has the following major purposes:

- Define of the overall use case, including a detailed description of the underlying development processes and the set of involved process activities and engineering methods.
- Provide input to WP601 (IOS Development) required to derive specific IOS-related requirements.
- Provide input to WP602 (Platform Builder) required to derive adequate meta models
- Establish the technology baseline with respect to the use-case, and the expected progress beyond (existing functionalities vs. functionalities that are expected to be developed in CRYSTAL).

1.2 Relationship to other CRYSTAL Documents

This version is the first iteration of this deliverable giving an overview of the desired use case process and the selected engineering methods.

During the course of the CRYSTAL project a total of 3 iterations are foreseen.

New engineering methods and corresponding tool requirements will be added with the following iterations.

1.3 Structure of this document

The Barco CRYSTAL activities focus on setting up the desired process for the Medical certification and Requirements management Framework for the Barco software centric scalable safety critical medical display platform. Chapter 2 provides an overview of the general Barco use case. Chapter 3 gives a more detail view on the involved process activities. Chapter 4 describes the Engineering Methods identified so far. Chapter 5 gives an overview of the pilot project started in Barco and the relation to the envisioned System Engineering Environment. Chapter 6 is giving an overview of the Engineering methods implemented so far in the Pilot Project. Chapter 7 is summarizing the conclusions and indicating the next steps.

2 Use Case Process Description

2.1 Goal: A software centric scalable safety critical medical display platform

There is ever increasing product variability and the need to speed up time-to-market and reduce development time. This requires Barco to change their medical display platform from a hardware centric, custom platform towards a flexible software centric, Commercial-off-the-Shelf (COTS) platform.

2.2 Current practice

Barco's current display platform is based on proprietary hardware using FPGAs. This is cost-effective when there are only a limited number of products to be supported, but it requires a huge R&D effort to develop and maintain when there is an increasing number of product variants. Hardware development typically also requires longer development cycles.

In order to display a medical image correctly, an input signal (typically coming via a DVI input or Display Port input) needs several transformations before it can be shown to a radiologist. This sequence of transformations is the so-called image pipeline.

The output image needs to comply with several FDA regulations (e.g., DICOM GSDF, Grayscale Display Function). Figure 2-1 shows a diagram that gives a high-level overview of a medical display image pipeline.

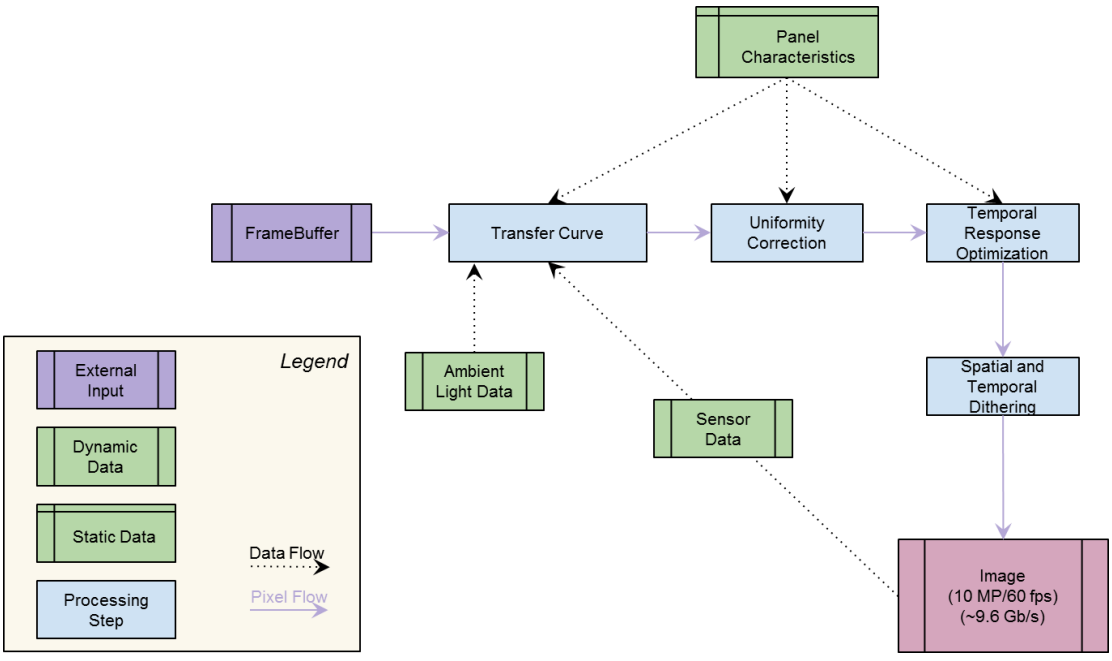


Figure 2-1: Typical Medical Display Image Pipeline.

We would like to replace the proprietary FPGA-based hardware platform by a COTS platform while keeping the same level of safety and performance.

Currently, the image pipeline (as shown in Figure 2-1) is implemented using hardware components (i.e. FPGAs). With this use case, we want to achieve a software-only implementation of the processing steps marked in blue in Figure 2-1, the other steps remain dedicated hardware components. The processing steps from the figure are called Software Imaging Components. In a first step, the components are developed individually on COTS hardware.

By changing the design methodology from hardware-centric to software-centric we make it possible to create product variants by simply changing software configurations rather than writing new code or recompiling the code for new target hardware.

2.3 Introduction to the Barco General PLC Process

In the flow below (Figure 2-2), the generic Barco Product Life Cycle (PLC) process is given. The PLC is the high-level process for defining and developing a product, introducing it into manufacturing and into the market and finally phasing out. The PLC outlines the most comprehensive project-driven routine for handling the design of a new product. The Product Life Cycle (or PLC) design process is organized in different stages. Each stage is concluded with a stage gate review and allows proceeding to the next stage if the review is approved. The generic PLC (marked in yellow on chart below) contains all lifecycle phases of the product. Part of the PLC (marked in green) is phases and gates to actually develop a new product, the so called New Product Introduction (NPI) core process (or design process). The process starts with the Preliminary design phase (after BCR gate, approved Handshake document or Requirement document) and ends with FQR gate.

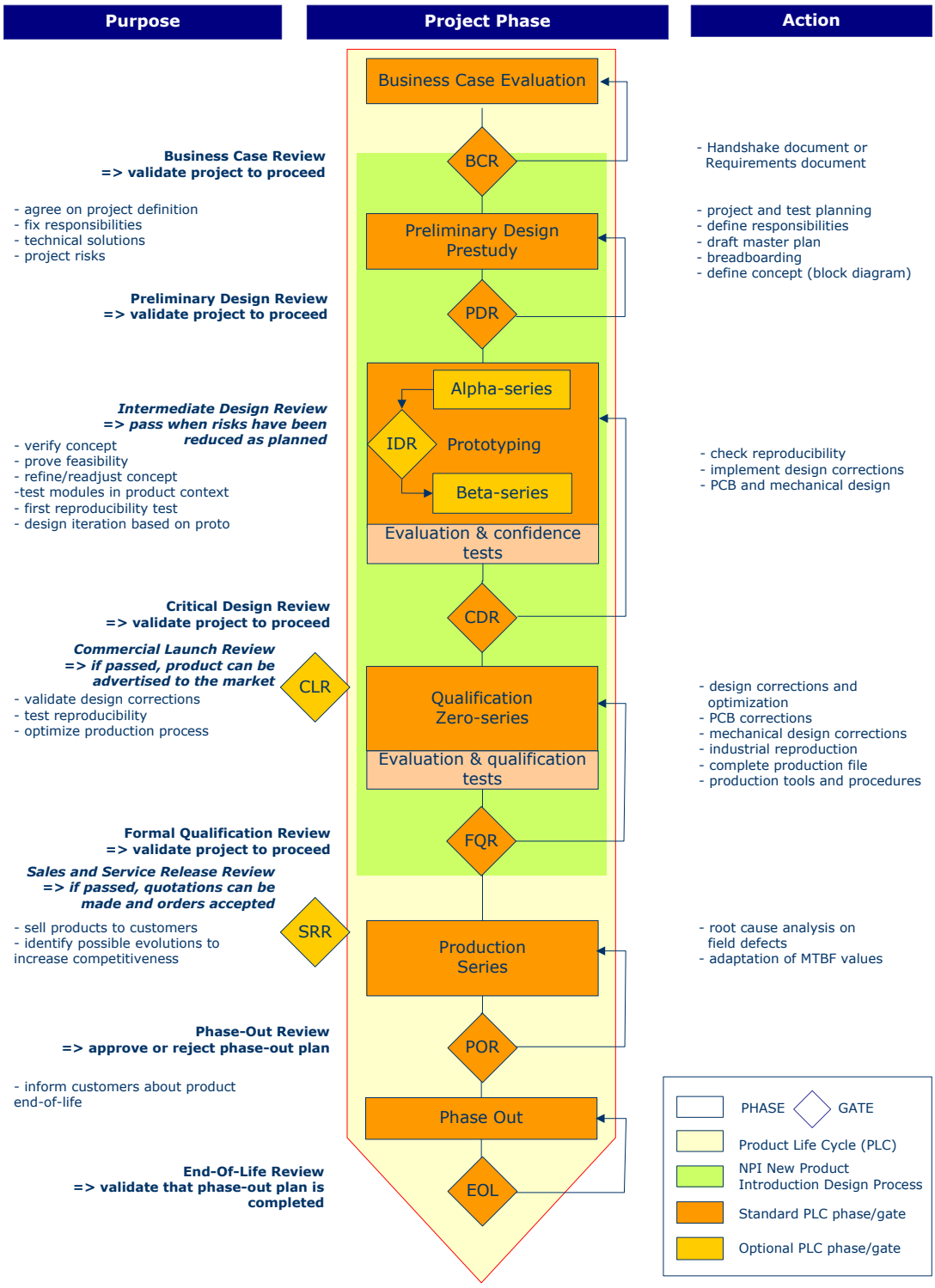


Figure 2-2: Barco Product Life Cycle (PLC) process.

2.4 The Pre-Crystal Software Process

The Pre-CRYSTAL software development process of Barco Healthcare (not taking into account the hardware display development) is shown in Figure 2-3.

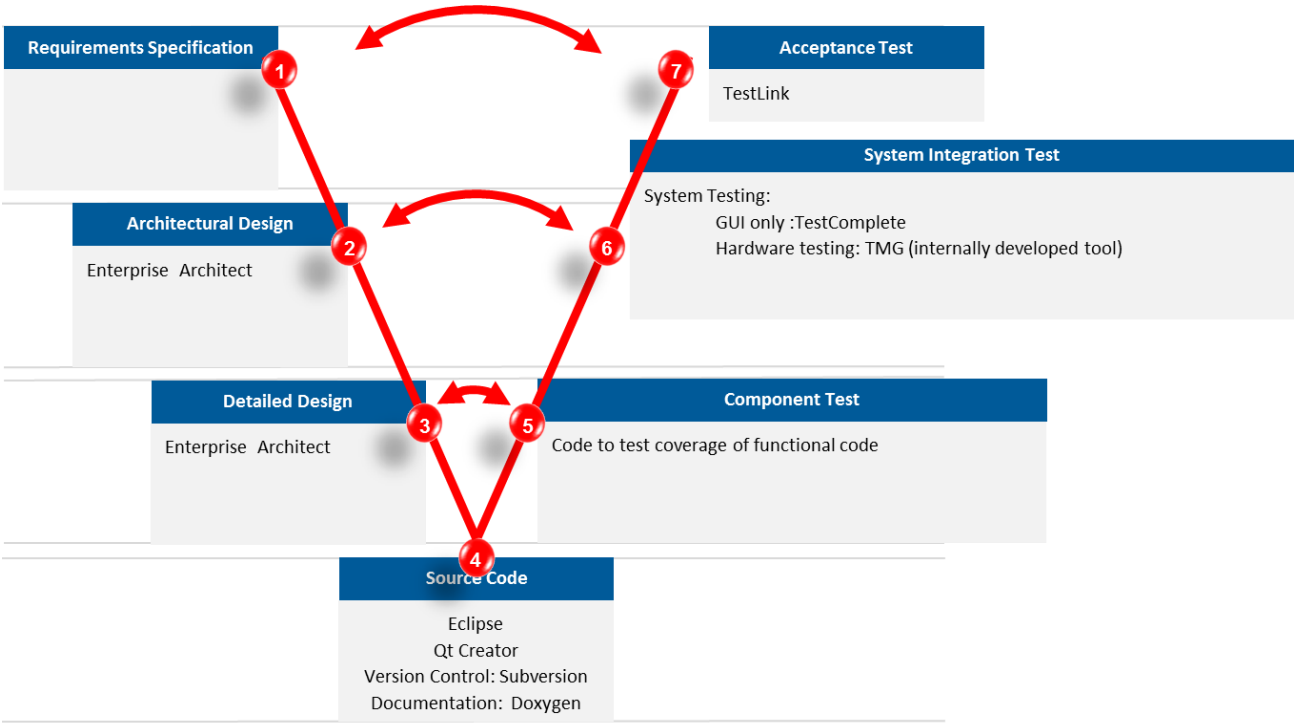


Figure 2-3: Current V model software process.

In the requirements specification step we gather the stakeholder requirements, no dedicated requirement tool is used for this step, this is now done by using Excel and Word files.

Based on the Excel and Word files, the system model and use cases are designed in the Architectural Design & Detailed Design steps, the output is an Enterprise Architect file.

In the current process we are missing traceability to the requirements and we are not supporting the re-use of existing designs or components.

In the Source Code step we implement the functional code without having traces to the model or requirements. Concerning re-use, ad hoc, there is often source level reuse of software components.

Developers will use code to test the coverage of the functional code during Component Test. In the current process we are missing coverage checks on requirements.

QA testers implement a System Integration Test based on use cases; these tests also include reused software components. Also in this step we have no link back to requirements.

In the final step QA testers implement Acceptance Tests based on requirements without having a solid trace to the original requirements.

If we want to integrate the development of a complete display, the current process needs to be extended/enhanced, resulting in a desired process.

2.5 Desired Process Goals

The major drawbacks of the current process are:

- The lack of traceability of requirements.
- No integration of the tools used during development.
- No reuse of models.
- No real reuse of components.
- Integration tests include testing reused software level components.
- Coverage check of unit tests, integration tests and acceptance tests towards requirements.
- No uniform, centralized documentation of the created artefacts.

Goals

Due to the ever increasing regularization within the Healthcare market (FDA, …), documentation, traceability and auditability are becoming more and more important for product development. On the other hand, from an economic perspective, time-to-market requirements are becoming stricter and stricter.

Using **model-based engineering**, we hope to **reduce the costly hardware-development cycles**, so that eventual bottlenecks and/or problems can be detected much earlier and much faster.

Using **tools that '*understand*' each other**, at least we hope to **introduce requirements traceability throughout our development process,** leading to **consistent traceable documentation, as required by IEC 62304**.

Getting approved for FDA, takes a lot of paper work and a lot of time. FDA-approval turnaround time can be reduced by **reusing formerly approved components/**designs into the development process.

2.6 Improvement(s)

We would like to replace the proprietary FPGA-based hardware platform by a COTS platform while keeping the same level of safety and performance.

By changing the design methodology from hardware-centric to software-centric we make it possible to create product variants by simply changing software configurations rather than writing new code or recompiling the code for new target hardware. This will result into a serious reduction in R&D effort spent (30%) on development and maintenance compared to our current hardware-centric platform.

Metrics

To be able to measure improvements, we have defined the following metrics.

1. Traceability of requirements throughout the process:

   For every artefact created during the development process, we can trace back to the corresponding requirement(s).
2. Detection of hardware/software bottlenecks before the design hits the metal:

The new development process includes model based-engineering tools and techniques to simulate and define the optimal architecture avoiding expensive hardware development/test loops.

3. Consistent IEC 62304 documentation:

   When a product is accepted, a simple push on a button should provide an IEC 62304 compliant document. Having this ability during the product development process is an asset, but not a requirement. This implies that the document generator can talk to all relevant tools, repositories, documents, … to gather all required information.

4. Reuse of components and models:

   Tools supporting component-oriented & modular design allowing product variance based on configuration without the need to recompile code or redesign/recertify modules.

5. Binary reuse of previous FDA approved components (e.g. algorithms):

   Automatic generation of product documentation and product specification for the FDA compliance of a specific product variant reusing a previous FDA approved component.

3 Detailed Description of the Use Case Process

3.1 Introduction to Medical Certification and IEC 62304

Regulatory requirements

Figure 3-1 gives an overview of the regulatory requirements involved in the certification activities of Barco software products.

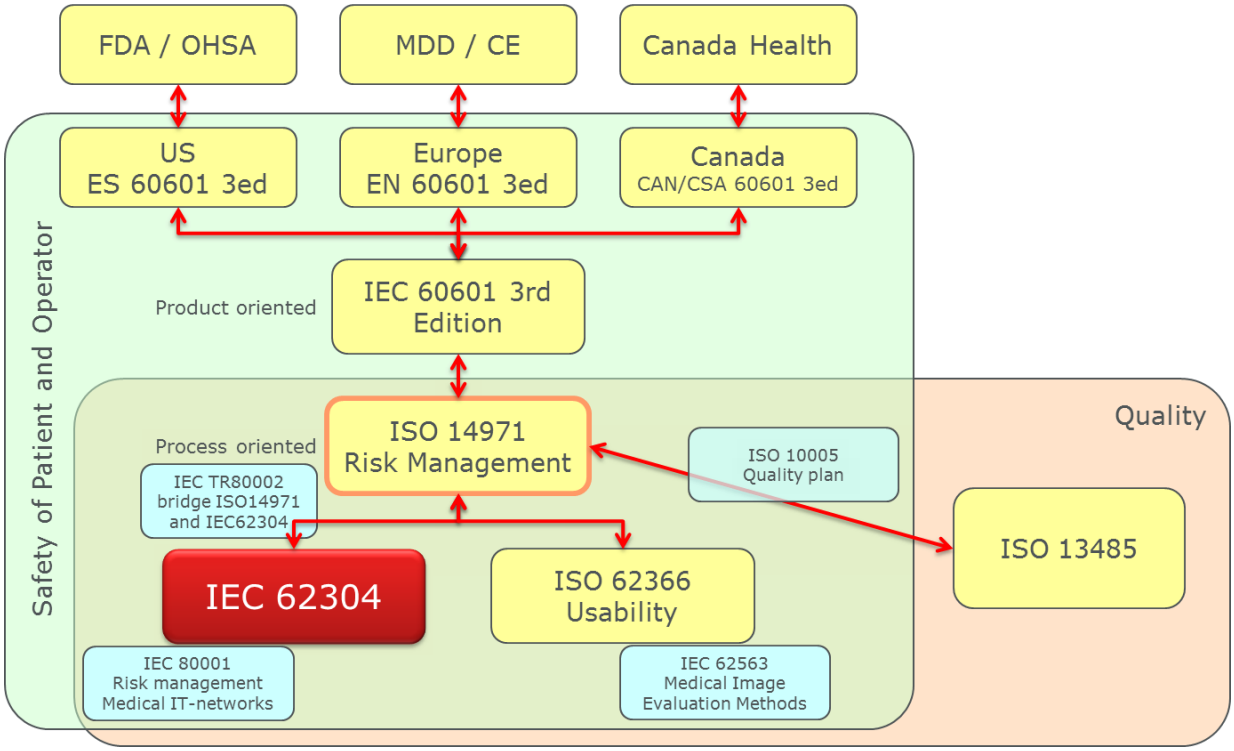


Figure 3-1: Regulatory requirements for Barco Healthcare products.

IEC 62304

"Software is often an integral part of MEDICAL DEVICE technology. Establishing the SAFETY and effectiveness of a MEDICAL DEVICE containing software requires knowledge of what the software is intended to do and demonstration that the use of the software fulfils those intentions without causing any unacceptable RISKS.

This standard provides a framework of life cycle PROCESSES with ACTIVITIES and TASKS necessary for the safe design and maintenance of MEDICAL DEVICE SOFTWARE. This standard provides requirements for each life cycle PROCESS. Each life cycle PROCESS is further divided into a set of ACTIVITIES, with most ACTIVITIES further divided into a set of TASKS."

Note: for more details about these clauses we like to refer to the documentation available on IEC 62304.

IEC 62304 Artefacts

IN CRYSTAL context**, i**n order to consistently produce high quality and safe medical device software conform the ICE 62304 ed1.0 (2006-05) standard, following artefacts are requested:

- Risk Management File (clause 4.2, 7)
- Software Safety Classification (clause 4.3.c)
- Software Development Plan (clause 5.1.1)
- Software System requirements (5.2), including risk control measures (clause 5.2.3)
- Software Architectural Design (clauses 5.3, 5.4)
- Software Test Plan (clauses 5.5, 5.6, 5.7, especially 5.7.1 note 1 and 2)
- Traceability Overview (of test procedures to software requirements) (clause 5.7.4)
- Software Test Report (clause 5.7.5)
- Residual Anomalies (clause 5.8)
- Configuration Management (clauses 5.8.4, 5.8.5, 8)

3.2 The Pre-CRYSTAL Medical Certification and Requirements management Framework Use Case Process

For a description of the software process as used in Barco before the start of CRYSTAL we like to refer to section 2.4 of this document.

Projecting the coverage of the V-model activities over the project timeline resulted in Figure 3-2, here one can see that in the previous process there was a strong focus on starting the work on the source code and there was a fragmented approach on working on e.g. requirements documents just before the planned certification. As mentioned in section 2.4 of this document this was all manual work using mostly Word and Excel without any traceability between the activities. Next to this several activities are missing conform the IEC 62304 standard: Traceability, Risk Management, Maintenance, Classification, ...

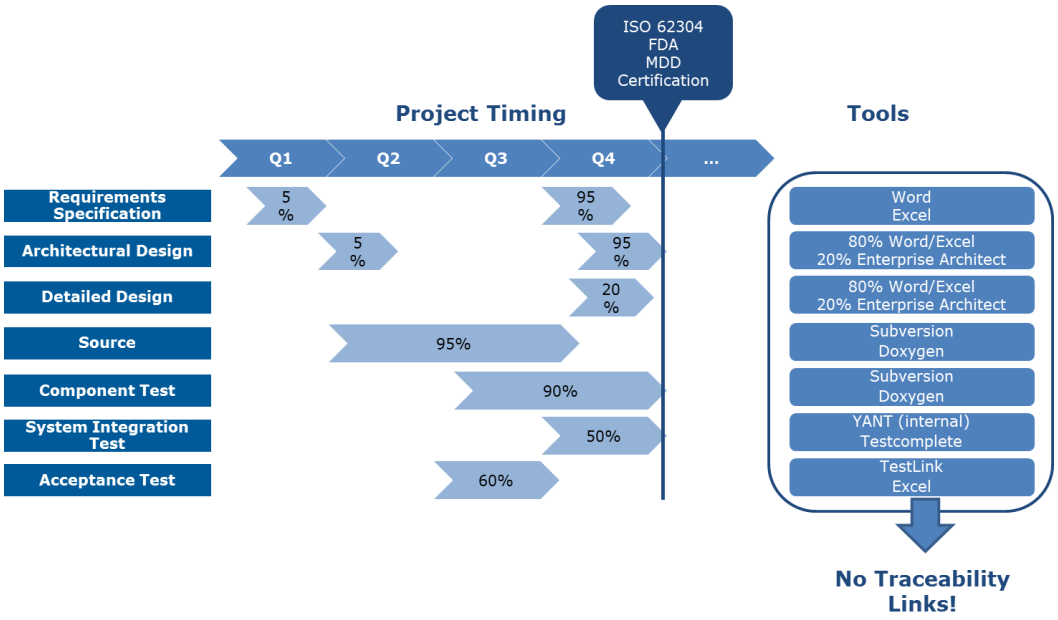


Figure 3-2: Pre-CRYSTAL Medical Certification and Requirements management Framework Use Case Process.

3.3 The Desired Medical Certification and Requirements management Framework Use Case Process

Several new activities need to be taken into account for the Desired Medical Certification and Requirements management Framework Use Case Process, in line with desired goal described in section 2.5 of this document; these new activities have a strong focus in delivering the requested artefacts for the IEC 62304 certification.

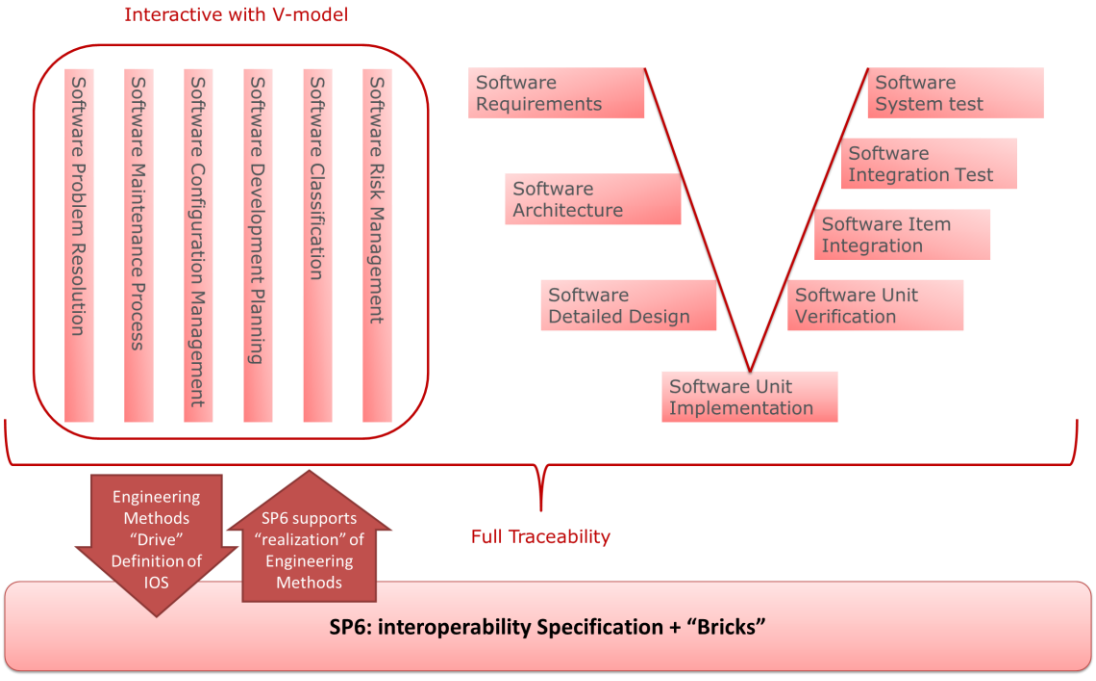


Figure 3-3: Desired Medical Certification and Requirements management Framework Use Case Process.

Next to the focus on producing the required IEC62304 artefacts with full traceability of all process activities, Barco also desires to implement the desired process in an Agile environment, this is also linked with the desired process for the software centric scalable safety critical medical display platform as part of WP405. Below is an illustration of a typical Agile approach.

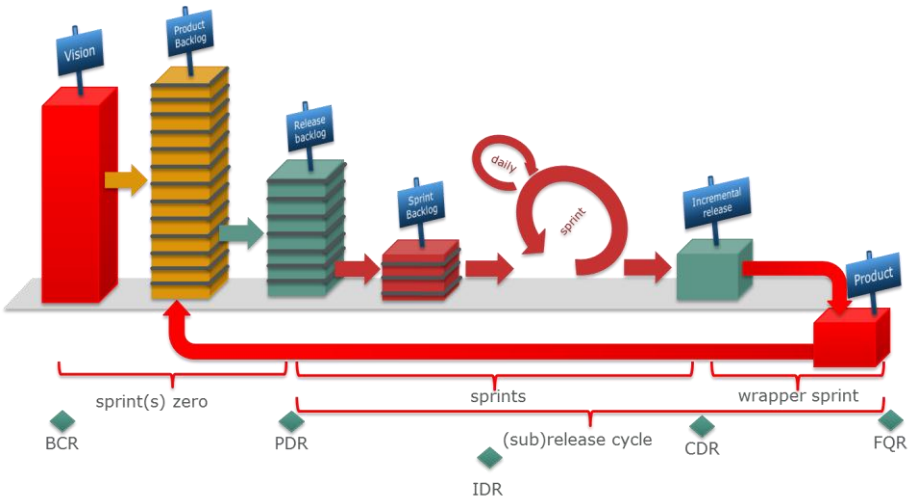


Figure 3-4: Typical Agile approach.

In the desired Agile process, each sprint is incrementally processing a part of the product backlog while coverage all process activities in parallel and this with full traceability, this allows a successful certification for each sub release. Projecting the sprint and according process activities over time results in the desired agile process as illustrated in Figure 3-5: Desired process in Agile environment.

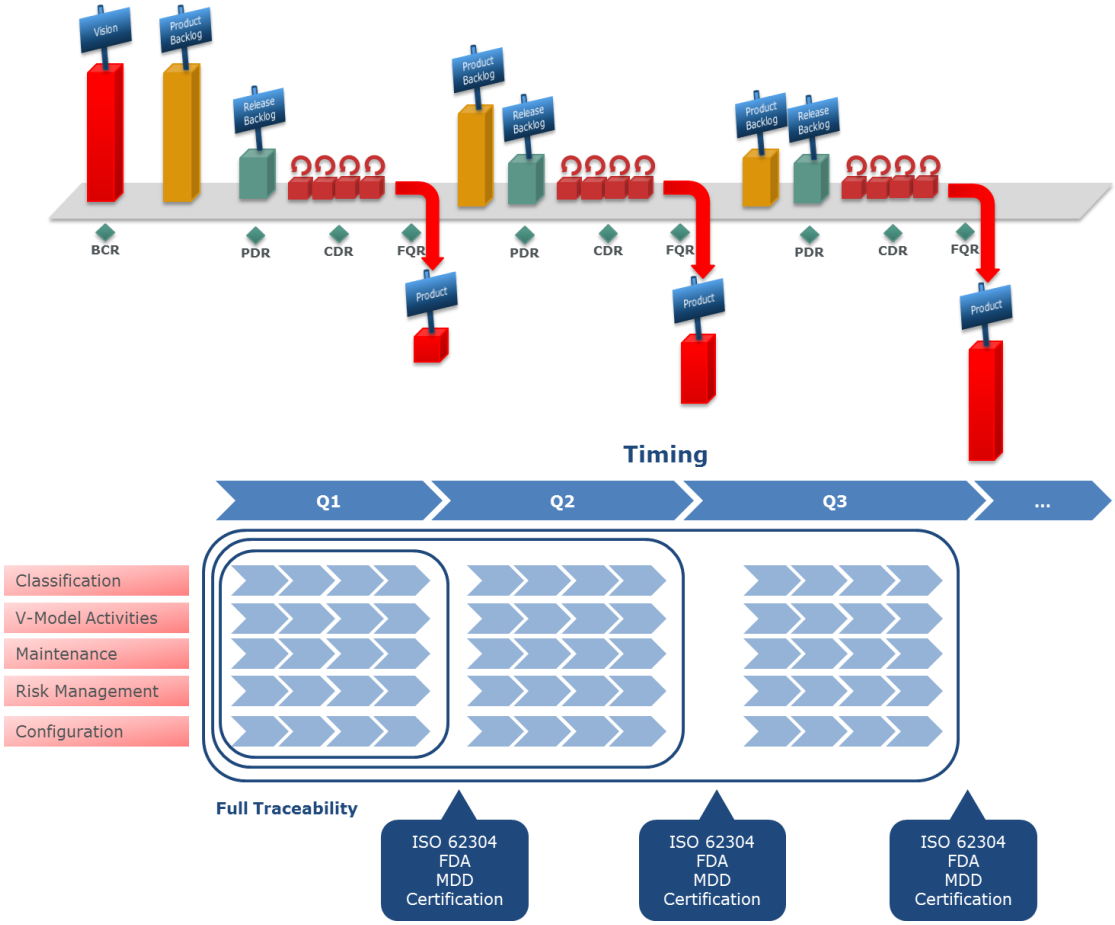


Figure 3-5: Desired process in Agile environment.

.

4 Identification of Engineering Methods

Section 4 gives a detailed overview of all identified Engineering Methods that contribute to the transformation towards the desired design process for the Barco Use Case.

Please note that not all identified Engineering Methods will be developed in detail in the scope of the project.

In this use case the development is focused on those Engineering Methods that are affected by the integration of a new tool or those that provide some interoperability requirements, as well as those contributing to the envisioned engineering process for the medical certification and requirements management framework.

For this Use Case a number of identified engineering methods so far are:

- Software Risk Management.
- Requirements Management.
- Test Management.
- Software Classification.
- Certification Reporting.
- Cross Domain Configuration Management.
- Product Variant Management.

4.1 Software Risk Management

Input

- Risk & Hazard process for medical devices.

Process

- Risk analysis:
  Identify contribution to hazards caused by software items:
  - Incorrect or incomplete specification of functionality.
  - Software defects in the identified software item functionality.
  - Hardware failures or other software defects that could result in unpredictable software operation.
  - Reasonably foreseeable misuse.

- Risk control:
  - Risk control measures implemented in software shall have corresponding software requirements.
  - The risk control measures are implemented and verified following the Software Development Process.
  - Check if the implemented software requirements that are related to risk control measures are verified for correct implementation.
  - Identification of new sequences of events that could result in a hazardous situation, coming from these control measures. Document these in the Risk Management File (RMF).
  - Traceability of software hazards is required:
    - From the hazardous situation to the software item.
    - From the software item to the specific software cause.
    - From the software cause to the risk control measure.
    - From the risk control measure to the verification of the risk control measure.

- Risk management of software changes:
  - Impact of software changes shall be covered and triggers the complete risk management process.

Verification

- At stage gates and also during production and post-production activities.

Output

- Updated Risk Management File RMF and Risk Evaluation Report RER.

4.2 Requirements Management

Inputs

- Product requirements management.
- Software Requirements Analysis.
- Software Development specification.
- Risk Control Measures.

Process

- Software requirements are deducted from the system requirements. If the system requirements are released or updated, the software requirements are re-evaluated for eventual update. System or product requirements management is out of the scope of this procedure but an input for this process step.
- There shall be traceability between system requirements, software requirements, software system test, and risk control measures (see also section 4.4 on Traceability).
- The following type of requirements (but not exclusive) must certainly be considered during this phase:
    - Functional and capability requirements.
    - Software systems inputs and outputs.
    - Interfaces between software systems and other systems.
    - Software-driven alarms, warnings and operator messages.
    - Security requirements.
    - Usability engineering requirements.
    - Data definition and database requirements.
    - Operation and maintenance site or sites.
    - Operational and maintenance requirements.
    - User documentation requirements.
    - User maintenance requirements.
    - Regulatory requirements.
- All of these requirements might not be available at the beginning of the software development.

Verification

The project manager shall at each stage gate and starting from PDR, ensure that following verification activity is performed on the software requirements:

- Implement system requirements including those relating to risk control.
- Do not contradict one another.
- Are unambiguous.
- Are testable (test criteria can be defined and evaluated).
- Are uniquely identified.
- Are traceable to system requirements or other source.

Output

- The software requirements are documented in the Software Requirements Specification (SRS).
- Verification of requirements can be done separately or included in the SRS with additional verification columns.
- The software test report with traces from system test to software requirements is the final verification of testability and implementation of all system requirements.

4.3 Test Management

Input

- Software requirements and/or specifications.
- Software Architectural Design (SAD).
- Software units (source code).
- The Software Test Plan (STP).
- Corrective actions, Preventive actions process.
- Software Testing Process.

Process

- The R&D team shall integrate software units into software items and the software system.
- The R&D team verifies if the software units and software items have been integrated according the STP.
- The R&D team verifies the integration of the software and hardware into the system according to the STP.
- The R&D team executes the integration and system tests according to the STP. Items that have an impact on safety should be subject to more thorough testing. Software integration and system testing can be done on a simulated environment, actual target hardware or the full medical device.

- When changes (design, test plan…) are made during testing:
  - Repeat tests, perform modified tests or perform additional tests, as appropriate, to verify the effectiveness of the change in correcting the problem.
  - Conduct testing appropriate to demonstrate that unintended side effects have not been introduced.
  - Perform relevant risk management activities.
- The R&D team shall create an issue in the defect tracking system for each anomaly found during testing.
- **[Class B]** Regression testing is mandatory for Class B.

Verification

- Stage gate review of test reports.
- **[Class B]** Verify (as part of the maintenance of the Software Development Plan) that software system test procedures trace to software requirements as required in the Software Test Plan.

Output

- Results from integration testing and system testing can be combined into one Software Test Plan and one resulting Software Test Report.
- Software test report STR (first released on CDR) including:
  - Version of software under test, relevant hardware configuration
  - Document of test result (date, pass/fail and a list of anomalies in Defect tracking tool).
  - Retain sufficient records to permit the test to be repeated.
  - Tester identification, relevant test tool identification.
- Software anomalies entered into the software reporting tool.

- **[Class B]** Software risk management process:
  - What: Traceability of software hazards is required:
    - From the hazardous situation to the software item
    - From the software item to the specific software cause
    - From the software cause to the risk control measure
    - From the risk control measure to the verification of the risk control measure
  - How:
    - The requirement, created for the risk control measure, shall uniquely refer to the risk control measure from the Risk Evaluation Report.

        - Tracing of hazardous situations till software root cause.
        - Risk Evaluation Report is tracing between hazardous situation and risk control measure and verification of this measure.

- Software configuration management process:
    - What: Traceability of change request, problem report and approval
    - How:
        - Change request approval via ECP approval of change or problem resolution with link to problem resolution

Verification

- Review of software test plan and software test report at the project stage gates.
- ECP reviewers during ECP approval flow.

Output

- Software Test Plan and Software Test Report with traceability information for R&D traces.
- Risk Evaluation Report for evidence risk control measures.
- Problem database with link to approved ECPs.

4.4 Conclusion: Technical Core Requirements

Technical Core Requirements are the singular documented physical and functional needs that this Use Case and its associated Engineering Methods pose on the tooling Interoperability Specification (IOS) and the Reference Technology Platform (RTP). The requirements are expressed in a technology independent way and drafted by the Use Case Owner in joint consolidation with WP6, e.g. WP601 and WP602.

Over the life time of the Crystal project, the technology provider of a Technology Brick in joint consolidation with WP6 will combine and refine these core requirements into Technical Refined Requirements.

The following Technical Core Requirements are identified as relevant:

ID	Requirement	MoSCoW	Rationale	Related core requirement
TECH_CORE_REQ_0008	Provide IOS interface for requirement management tools.	Must	IOS for the requirements management system.	
TECH_CORE_REQ_0021	Requirements Quality ensure usable export formats.	Should	Ensure that once the Requirements Quality has been achieved that they can be used for any system that the they need to interface with.	
TECH_CORE_REQ_0023	The IOS shall support variability/variant management.	Should	Variability information must be communicated between different tools.	
TECH_CORE_REQ_0040	Semi-automated Requirements mapping.	Should	An automated transfer of data is required to ensure data integrity throughout the requirements engineering flow.	
TECH_CORE_REQ_0050	Automatic generation of safety documentation during development.	Must	Be able to automatically generate safety documentation from development documents.	
TECH_CORE_REQ_0051	Be able to extract relevant incidents from external safety surveillance databases.	Could	Databases like FDA Maude (medical) and NHTSA (automotive) give essential information for incidents reported in the field. It is mandatory to monitor this information and translate this into relevant actions.	
TECH_CORE_REQ_0104	Cross-tool document generation.	Must	Template based generation of certification documentation based on data that is available in different	TECH_CORE_REQ_0050

			tools is important.	

Table 4-1: Tech Core Requirements for WP404.

5 System Engineering Environment

5.1 Pilot Project Pre-Study

A dedicated pre-study exercise was performed to make an analysis of the available platforms and tools.

5.1.1 Study reports

Several study reports were analysed to compare the possible tool candidates.

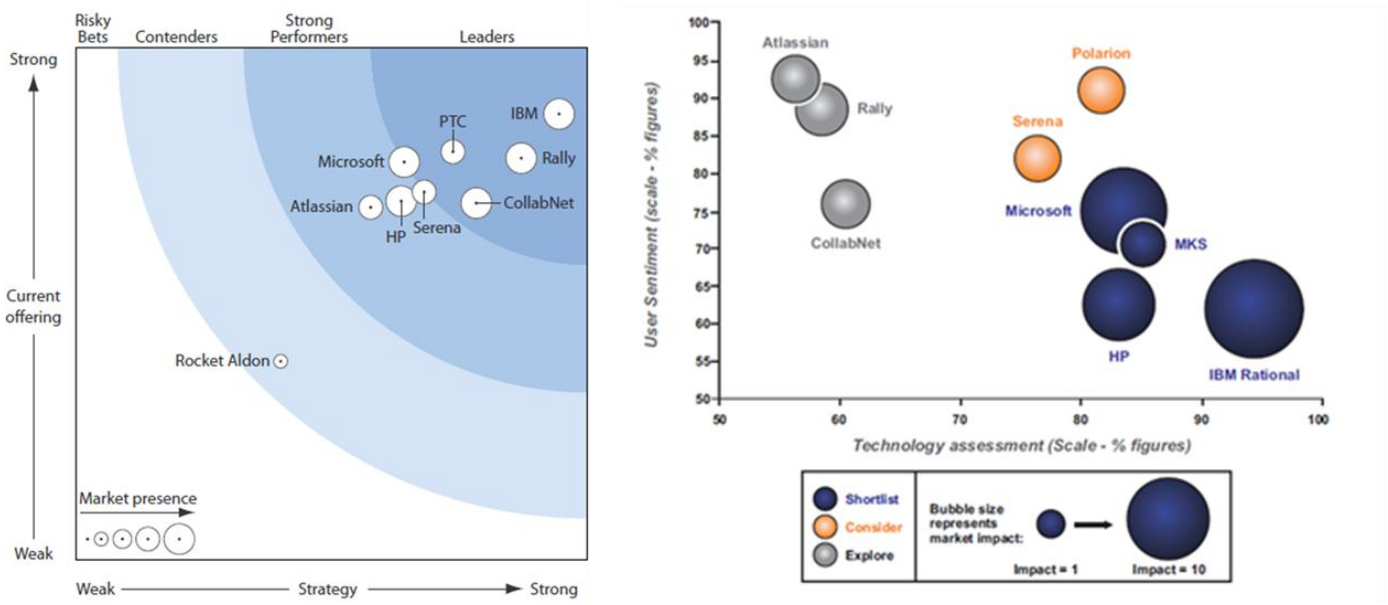


Figure 5-1: Study report results.

5.1.2 Input from Key-Stakeholders

A study was performed to collect the ALM requirements from the key stakeholders at Barco.
For each stakeholder the need for the tool is marked with an 'x' in the table.

Initials	Division	Location	Risk mgmt	Req mgmt	Test mgmt	Pilot	Role
RSW	HC	BE-KND	x			x	Pilot: Risk Manager + STC: risk
FRTO	HC	BE-KND	x	x	x		STC: IEC62304
PIMOU	HC	BE-KND			x	x	Pilot: Test Manager + STC: testing
DRGE	GS/QA	BE-KND	x	x	x		Keep informed
SUSH	GS/SW	IN-NOI		x			Keep informed
HAMR	GS/SW	IN-NOI		x	x		STC: NOI SW lab delegate
JEK	HC	US-BEA		x	x		STC: BEA delegate
JURDE	COL	BE-KUU		x	x		STC: COL delegate
PHMT	PROJ	BE-KUU		x			STC: PRJ delegate (alternating with KC)
KC	PROJ	BE-KUU		x			STC: PRJ delegate (alternating with PHMT)
GEDL	PROJ	BE-KUU			x		STC: PRJ delegate testing
MIHOO	PROJ	BE-LIE			x		STC: PRJ-LIE delegate
MDCR	NETW	BE-KND		x	x		STC: NETW delegate
WOMI	NETW	BE-KND			x		STC: NETW delegate (testing)
JUM	NETW	BE-KND		x	x		STC: NETW delegate
PE	NETW	BE-KND		x			STC: NETW delegate
JRD	NETW	BE-KND	x	x			STC: DATNETW delegate (product owner)
BKB	DAT	BE-KND		x			STC: DAT delegate
HUHE	AVS	DE-KAR		x	x		STC: AVS delegate
KLAB	AVS	DE-KAR		(x)	x		STC: AVS delegate
ROHU	AVS	DE-KAR		x	x		STC: AVS delegate
AX	HC	US-BEA		x		x	Pilot: Product Manager
BJBE	HC	BE-KND		x		x	Pilot: Product Manager
SCOI	HC	BE-KND		x		x	Pilot: Project Manager
JL	HC	BE-KND		x		x	Pilot: Project Manager
ENMA	HC	BE-KND			x	x	Pilot: Test Engineer
JOAK	HC	BE-KND			x	x	Pilot: Test Engineer
LBE	HC	BE-KND		x	x	x	STC: Industrialization Pilot: Specs to testing
EC	HC	BE-KND				x	Pilot: R&D Manager (keep informed)
STEN	HC	BE-KND	x			x	Pilot: Risk Engineer
FDU	HC	BE-KND	x			x	Pilot: Risk (market stage)
LDWA	HC	BE-KND	x			x	Pilot: Risk (market stage)
DEVER	BE-GS/TDM	KND/KU					TDM Manager
ADCOO	BE-GS/TDM	KND/KU					ALM Project Manager
BVA	GS/TDM	BE-KUU					Integrations programmer
WVER	GS/TDM	BE-KUU					Configuration

Table 5-1: ALM requirements by Barco Stakeholders.

5.1.3 Requirements for the Requirement Management Tool

A good requirement management tool presents requirements of a product in the best way for the specific user: in a graphical or textual overview, with links to its parent or child requirement(s). It should support a versioning system so that changes to requirements can be tracked. In the best management system, the requirements are linked with the actions/issues and with the test results. This is not only important to improve the efficiency and consistency of product development, but it is also a thankful tool during audits.

Table 5-2 lists the requirements for a requirement management tool. The following ranking is used:

- **M**: Must have; if it isn't present it isn't acceptable.
- **S**: Should have; it is expected it is in the tool, but if it isn't or if the need can be satisfied in another way it isn't blocking.
- **C**: Could have; nice to have, but not a strict requirement.

Requirement	Description	Importance
Capturing Requirements		
1. Input document enrichment/analysis	Using existing document information (such as glossary, index, etc.), aid the user in requirements analysis, identification of requirements, etc.	S
1.1. Input document change/comparison analysis	The ability to compare/contrast two different versions of a source document	M
2. Automatic parsing of requirements	A mechanism for automatic identification of requirements by key words, structure, unique identifiers, etc. to create requirements from the text	C
3. Interactive/semi-automatic requirement identification	The ability to identify requirements from a text file via interactive means such as mouse highlighting of the requirement text or prompting by the system "is this a requirement?"	C
4. Manual requirement identification	A manual means of identifying or creating requirements	M
5. Batch mode operation	A mechanism for inputting/identifying requirements from outside of the tool	S
5.1. Batch-mode document/source-link update	Does the tool have the ability to update existing linked documents from new/changed versions of the source documents without having to re-establish traceability links	M
6. Requirement classification	Does the tool have the ability to classify/categorize requirements during identification	M
Capturing system element structure	**The tool must capture the elements that build the system so that links/allocations can be made between requirements**	
1. Graphically	Can the tool graphically capture system implementation (such as architecture, functional decomposition, WBS, etc.) and display them graphically such that requirements can be linked to them	S
2. Textual	Can the tool textually capture system implementation (such as architecture, functional decomposition, WBS, etc.) and display them textually such that requirements can be linked to them	M
Requirements flow-down	**Once the requirements have been captured and system architecture captured, requirements are allocated to the various system elements**	
1. Requirements derivation (req. to req, req. to analysis/text)	The ability to derive/create additional requirements and link between them such as requirement to requirement, or requirement to text to derived requirements	M
2. Allocation of performance requirements to system elements (weight, risk, cost, etc.)	The ability to link performance requirements to system elements such as weight, cost, throughput, etc. This also includes the ability to allocate portions of that performance requirement to system elements	M
3. Bi-directional requirement linking to system elements	The linking of requirements to system elements can be accomplished from either end of the link (from the implementation back to the requirement or from the requirement down to the system element)	M
4. Capture of allocation rationale, accountability, test/validation, criticality, issues, etc. - if so how and what mechanism does it use?	Also critical, is the ability to attach rationale, assignments, criticality, test/validation and many other issues to the requirement, allocation, and the system element to which a requirement is linked	M
Traceability analysis	**Once the allocations are complete, the user will want the ability to see the links where they come from, where they go, and why they apply**	
1. Identify inconsistencies (orphans,...)	The tool should allow the user to identify inconsistencies such as unlinked requirements or system elements (orphans)	M
2. Visibility into existing links from source to implementation - i.e. follow the links	With the requirement links in place, the user needs the ability to follow the links to see where they come from and where they go to	M
3. Verification of requirement	Throughout the life of the project, the requirement management tool will be used to verify that the requirements have been met. The tool should provide the ability to document that the requirement was fulfilled, how it was done, and who was responsible	M
4. Requirement performance verification from system elements (roll up of actuals)	Once performance requirements have been allocated to system elements, the requirements management tool should support the verification of those requirements by rolling up actuals and reporting on variances (this is the allocated weight versus the actual weight)	S
5. Change impact analysis	The tool should be able to perform an impact analysis when a change is made to a requirement or to the element structure	S
Configuration mgmt	**Keep track of changes to requirements and provide access control**	
Documents and other output media		
1. Standard specification output	The requirements management tool should output documentation in various military/commercial standard formats (490, 2167, etc.)	S
2. Quality and consistency checking (spell, data dictionary)	The tool should support document quality and consistency checking through spell checking, data dictionaries, acronym tables, etc.	S
3. Presentation output	Once the information is loaded, the requirements management tool should support the generation of presentation quality charts and graphs	M
4. Custom output features and markings (user definable tables, figures, security markings..)	The tool should support the output of documents in finished form including page security markings, graphics/figures, user definable tables, indexes, etc.	S
5. WYSIWYG previewing of finished output	The tool should allow the user to view the document on-screen in finished format	M
6. Status reporting	Tool users need access to status information in the requirements management tool	M
6.1. Technical Performance Measurement status accounting	Status current technical performance of various allocated performance requirements and monitor progress towards goals	S
6.2. Requirement progress/status reporting	Status reporting on current compliance/non-compliance to various requirements	M
6.3. Other ad hoc query's and searches	The requirements management tool should support ad hoc query's and searches per the user's discretion	S

Table 5-2: Requirements for Requirements Management Tool.

5.1.4 Requirements for the Risk Management Tool

The mitigating actions defined during a risk assessment are in fact additional requirements. Therefore they should also be included in the ALM tool so that risk control is optimally executed and can be proven.

Table 5-3lists the requirements for a risk management tool. The following ranking is used:

- **M**: Must have; if it isn't present it isn't acceptable.
- **S**: Should have; it is expected it is in the tool, but if it isn't or if the need can be satisfied in another way it isn't blocking.
- **C**: Could have; nice to have, but not a strict requirement.
- **NR**: Considerd not relevant.

Requirement	Description	Importance
1. Import	Import the result of a risk assessment from another application	C
2. Risk assessment in the tool	The ability to perform the assessment in the tool itself through built-in templates. The tool should allow or even promote the risk assessment process according to certain standards (e.g. ISO 14971 for Healthcare)	M
2.1. Customizable templates	Templates should be customizable or even drafted from scratch	M
2.2. Templates for standard risk analysis techniques	Such as FME(C)A, FTA, HAZOP, …	M
2.3. Automatic risk ranking	Risk ranking bases on Probability and Severity (and practicability and detectability)	M
2.4. Manual risk ranking	Modifying the automatic ranking	NR
2.5. Risk summary	KPI indicator for risk level, reporting towards DPR meeting	M
3. Defining mitigating actions with link to requirement	Both mitigation and effectiveness must be reported	M
4. Export	Export to Excel, Word, pdf... using templates	S
5. Reporting and follow-up	Ability to extract reports/graphs (e.g. on a dashboard) to monitor the evolution of the risk level during the project	S
6. JIRA (iTrack) integration	Have the ability to score itracks in the same way as in the ALM tool, so that the assessment of iTracks is consistent with the risk management	M

Table 5-3: Requirements for the Risk Management Tool.

5.1.5 Requirements for the Test Management Tool

Good test management practice dictates that each time you run the test case you record the exact release, build/iteration and configuration that you are testing against.

Table 5-4 lists the requirements for a test management tool. The following ranking is used:

- **M**: Must have; if it isn't present it isn't acceptable.
- **S**: Should have; it is expected it is in the tool, but if it isn't or if the need can be satisfied in another way it isn't blocking.
- **C**: Could have; nice to have, but not a strict requirement.

Requirement	Description	Importance
Composing a test plan		C
Assigning tests		
1. Assignment at different levels	Assigning a complete test set of test cases to a particular person, per test case or even per test step	M
1.1. Assignment per test set		M
1.1.1. Assignment of a group of test cases on a "test cycle"	Possibility to select only a part of a version test set to be executed in a given context (e.g. scrum: tests of a given sprint, out of all tests)	S
1.2. Assignment per test case		M
1.3. Assignment per test step		M
2. Changes to test configuration during test	Possibility to change the test configuration or the assignee to complete the test at a later time	M
3. Dashboard overview	Allowing overview and follow up of tests in a customized way	S
Test library	**Provide a library of reusable test cases. A good tool has the capability to isolate yet link test cases between the library and the run instances.**	
1. Manage test case in the library	The update must be independent of already run test instances as this might possibly influence the results (branching)	M
2. Create run time instances of the cases	It must be possible to do a one time update that is not pushed to the library (branching)	M
3. Update the cases in the library	Updating the library instance AND propagating that change to the instances already assigned for execution (for small changes such as typos)	M
4. Update the case at run time	Update the run instance AND propagate that change back to the library so that it's incorporated in future runs BUT with a warning message	M
5. Manage the relationship between the run instances and the library instance	Replicating updates across projects where different projects are using the same case	M
Parametrization	**It allows easier management of similar test cases. However, it can significantly increase the complexity to the test process.**	
1. Parametrization inside the tool	This is useful if the amount of parameters is low or if a small change has to be done before execution of the test	S
2. Parametrization outside the tool (import capability at test case level of xls or csv files)	Other tools are likely to be better at making lists	M
3. Ability to assign set of parameters to multiple test cases		S
Reporting test results		
1. Test plan status	Showing scheduled/result status of the test plan	M
1.1. Drill down	Possibility to drill down from the global test plan status to individual test cases	M
1.2. Ability to report test status per product version	Showing which tests (including test case version) are scheduled, not scheduled, successful and failed	M
1.3. Ability to report test status per configuration	Showing which tests (including test case version) are scheduled, not scheduled, successful and failed	M
2. Ability to report on library level	Cross project evaluation of a success rate of a specific test set/case	M
3. Dashboard reporting	Reporting in the tool through charts and graphs of which the content can be chosen by the user	M
4. Exporting the report	Ability to output the generated results to a variety of public and military standard formats	M
Retesting		
1. Linking issues to failed tests	Once a test fails, new issues are created. They need to be linked to the particular failed test so that it's traceable	M
2. Build-up of regression test set	Building up a set of tests that will have to be executed on the next product version	S
2.1. By selecting issues with 'fixed' tag	A quick selection of issues that are fixed but where the fix is not yet verified by a test	C
2.2. By re-assigning test cases or test sets	For defects that have been discovered during previous test runs: automatically filter these items, but manually select them	S
2.3. By assigning new test cases	For defects that have been discovered outside the test process	S
3. Build-up of other types of test sets	Similar to regression tests, but other type: automation, 1h regression, smoke test, ...	S
Configuration mgmt	**Keep track of changes to requirements and provide access control**	
Interaction with other tools	Specifically for test management	
1. API to automatically fill-in results of automated tests	A programmable access so that an automated test can be configured to export its results to the ALM tool	M
1.1. Automatic integration with standard test tools	like Squish, TestComplete	S
2. Starting automated test	Starting automated tests from inside the ALM tool	C
2.1. License mgmt of the automated test tool	Make license available again after the test has finished	M

Table 5-4: Requirements for the Test Management Tool.

5.1.6 Requirements for the Configuration Management Tool

Requirement	Description	Importance
1. Configuration Identification		
1.1. Structure	Determine the hierarchy of the complete system	
1.1.1 Textual	Textual indication of the direct relatives	
1.1.2 Graphical	Graphical overview of the structure	
1.1.3 Drill down	Possibility to navigate along structure	
1.2. Setting baselines	Setting baselines of the complete project/system and for parts of it	
1.3. Export in/to a template	Export baseline report to Excel, Word, ... and fit a user defined template	
1.4. Automatically detect widows and orphans	Alert when widows or orphans exist in the structure	
2. Configuration Control		
2.1. Version control	Versioning all items: requirements, files (source codes, designs, reports, ...), test cases, ...	
2.1.1. Revert	Undo change by reverting to previous version	
2.1.2. Compare	View before/after change	
2.1.3. Automatic logging changes	What was changed by whom and when	
2.1.4. Additional logging possibilities	Why was it changed, how, comments, ...	
2.2. Associate change with issue	Ability to link change between versions to an issue in iTrack	
2.3. Comparing baselines	Comparing and contrast between various baselines of the complete project/system or for parts of it	
2.4. Assigning status to item	An item can be marked as 'draft'/'in development', 'pending approval', 'approved', 'obsolete', ...	
2.4.1. Visual representation of different statuses in structure overview	e.g. a color code, icons, obsolete items as 'strike trough', ...	
2.5. View change statistics	Ability to request statistics of engineering change process	
3. Change impact analysis		
3.1. Automatically marking direct relatives	When an item is changed, all connected items should be marked that they might have to change accordingly	
3.2. Automated notification of owners of impacted items	Change notifications should arrive at owner of the impacted items	
3.2.1. Notification inside the tool		
3.2.2. Option to be notified via e-mail		
3.3. Stop change in progress	Ability for impacted person to raise alarm and stop ongoing/planned change	

Table 5-5: Requirements for the Configuration Management Tool.

5.2 Pilot Project

For WP404, the Medical Certification and Requirements management Framework, Barco started to setup a System Engineering Environment (SEE), this is the Pilot Project running in the Barco Healthcare division.

The goal of this pilot project is to introduce a development framework that allows full traceability between requirements, design, source code and test results according to the V-Model. It also includes risk management and interfaces with Jira/iTrack for issue management and requirement management for Agile development teams.

Tool selection for the Pilot Project

Following tools are selected and used in the context of this pilot project:

Targeted Activity	Pilot Tool
Requirement Management	Integrity
Test Management	Integrity
Defect Management	Jira/iTrack
Risk Management	Integrity

Table 5-6: Tool selection for Pilot.

5.3 High level overview of the SEE for WP404 at M12

Figure 5-2: Positioning the WP404 Pilot project on the high level SEE overview.Figure 5-2 is positioning the Pilot Project activities of WP404 onto the high level overview of the SEE, as mentioned in D404_10 this Pilot is covering the top level of the V-model. The activities of WP405 are covering the IEC 62304 compliancy for the remaining levels of the V-Model. For more details on the WP405 activities we would like to refer to CRYSTAL Deliverable D405_10.

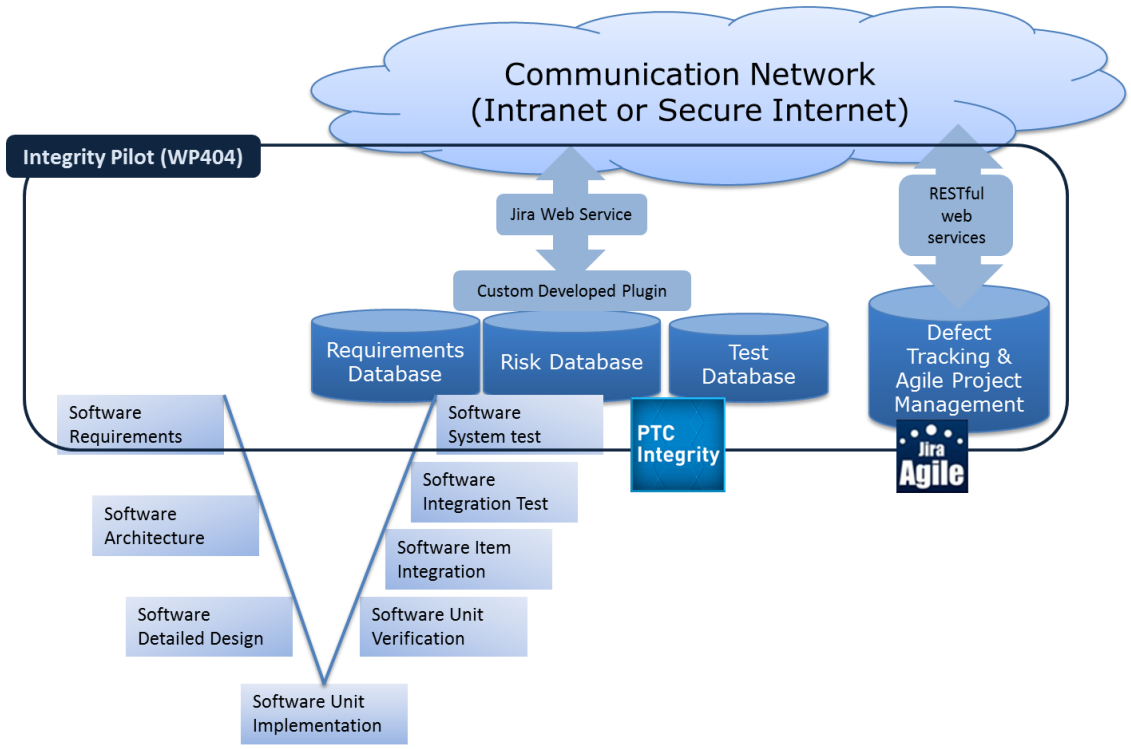


Figure 5-2: Positioning the WP404 Pilot project on the high level SEE overview.

Some of the desired tool connections that are needed for the IEC62304 compliance use case are already being implemented. For the interface connection between PTC Integrity and JIRA we are currently using a combination of a custom developed plugin on the PTC side and RESTfull web services on the JIRA side. The basis for this custom PTC interface was made by PCT, further refinement activities were performed by Barco.

5.4 Conclusion: Envisioned SEE

Based on the identified engineering methods and future planned activities the envisioned SEE is shown in Figure 5-3.

Future integration works include:

- Open Standards interface for Integrity
- Integration with our PLM system (PTC Windchill).
- Integration with Enterprise Architect.
- Integration with Matlab Simulink.
- …

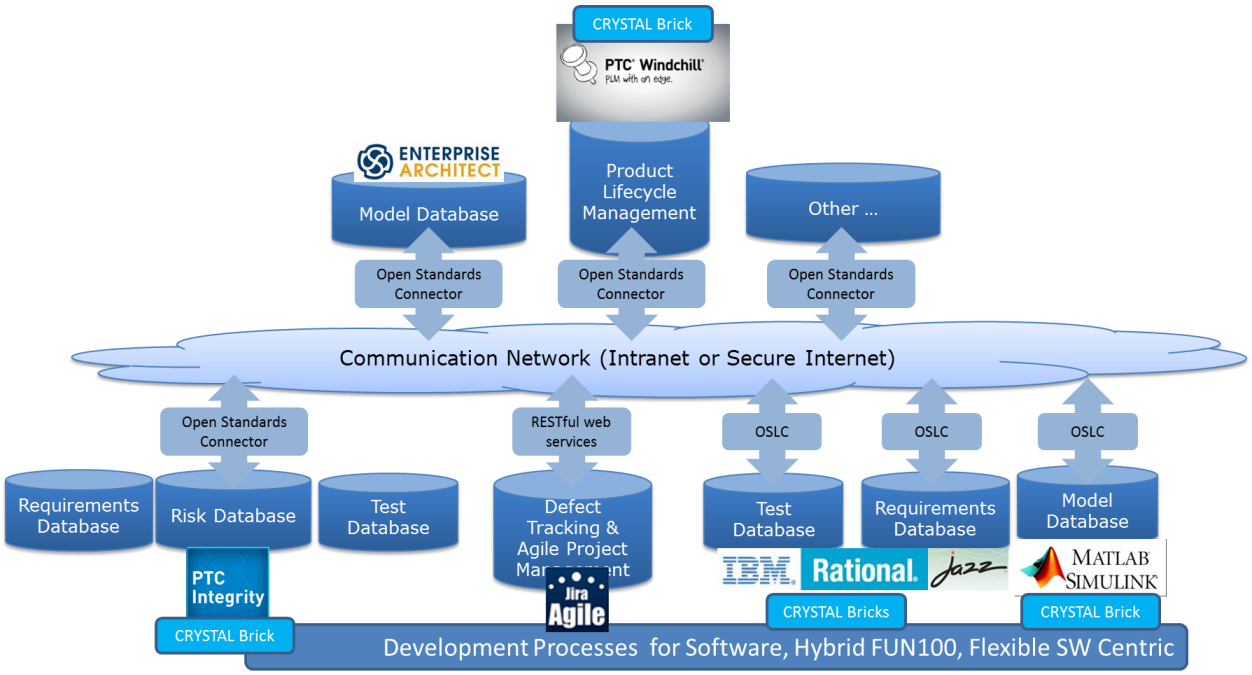


Figure 5-3: Envisioned SEE.

6 Implemented Engineering Methods

6.1 Introduction to the document models are used for the implemented Engineering Methods

General document model configurations

Document Model mapped to the V-model

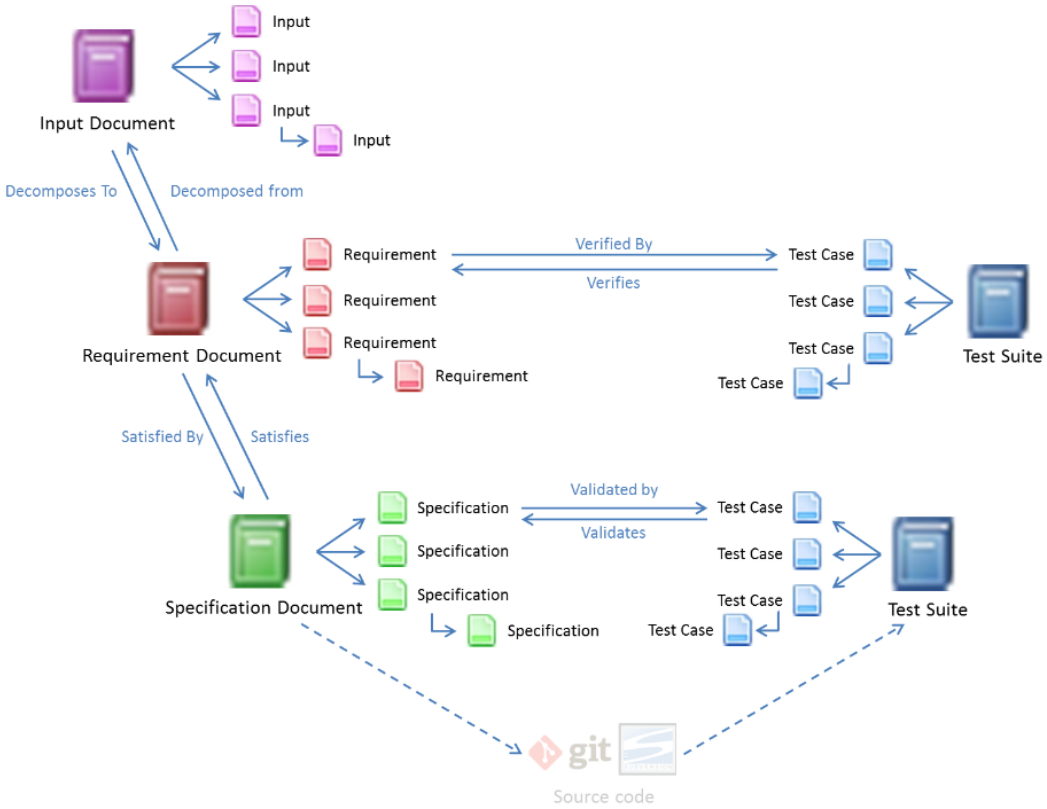


Figure 6-1: Document Model on V-model.

Three levels of item types: Project item, Document items and Content items.

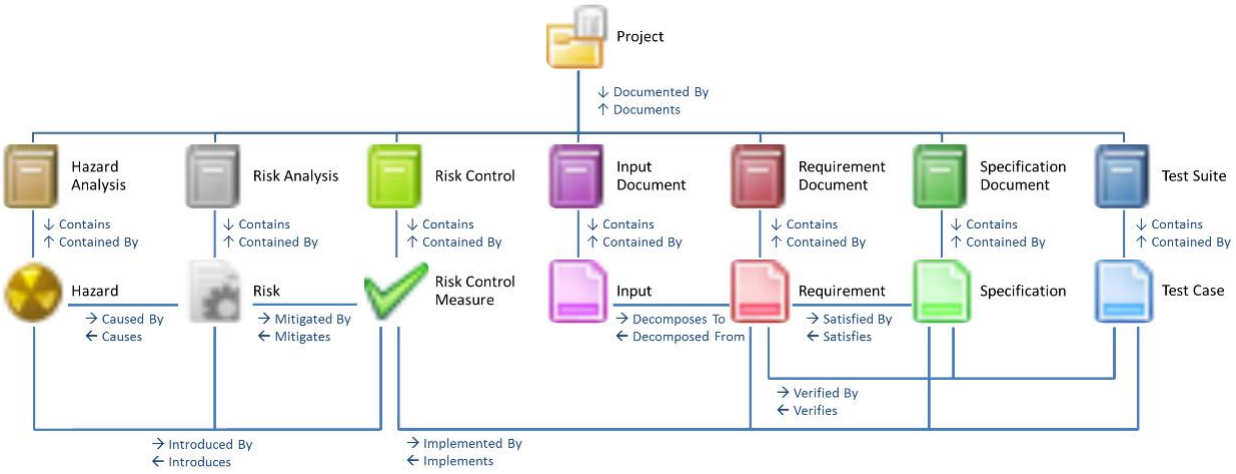


Figure 6-2: Overview of the used items.

6.2 Engineering Method: Software Risk Management

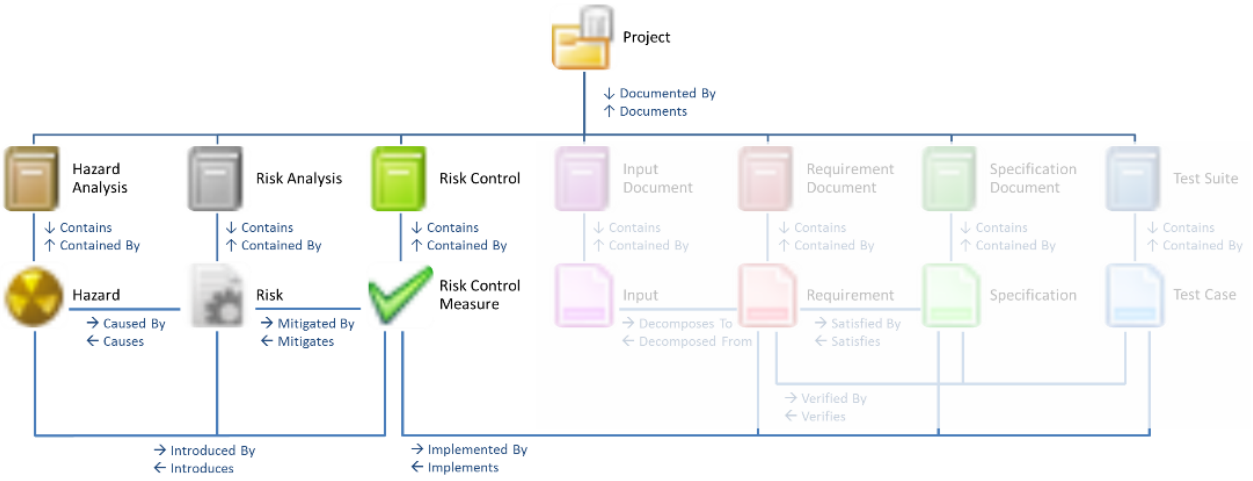


Figure 6-3: Software Risk Management.

6.2.1 Hazard Analysis domain

Purpose

To describe the hazard, a hazard is a threat to life, health, property, or environment. Risks, described in the Risk Analysis phase, describe a likeliness to cause hazards to occur with a certain level. To avoid discussion in that phase, it is good to have an agreement on the definition and the potential danger a hazard poses.

"The terms hazard and risk are often used interchangeably, however, in terms of risk assessment, these are two very distinct terms. As defined above, a hazard is any biological, chemical, mechanical, or physical agent that is reasonably likely to cause harm or damage to humans or the environment with sufficient exposure or dose. Risk is defined as the probability that exposure to a hazard will lead to a negative consequence, or more simply:

Risk = Hazard x Dose (Exposure).

Thus, a hazard poses no risk if there is no exposure to that hazard."

Document Model

Document = Hazard Analysis
Content = Hazard
Content categories = Hazard, Heading, Comment

Significant Edit Fields

Significant fields are fields that, when edited, can trigger a 'Suspect Flag' to related items or can trigger a version increase.

- **Text (Description): i.e. the identification of the hazard.** It is a very short summary of the hazard. Examples include: Overdose, Underdose, Electromagnetic radiation exposure, Allergic reaction, Unsafe Disposal, etc.
- **Foreseeable sequence of events:** A text field which identifies (and enumerates) the foreseeable sequence of events that are required for a hazard to occur. It can be limited to one event, but often there are more.
  For instance, the Hazard description captures in this field that in order to be cut (hazard) there needs to be "1. Sharp edges" and "2. User contact with the surface." If a risk would describe the creation of sharp edges, this wouldn't necessarily be a problem if that particular surface is never exposed to the user.
- **Hazardous Situation:** A text field that describes in more words what the hazard actually is so that everybody understands the same. It can be the same as the first field, but often it is a bit more elaborate.
  For instance: "Overdose: the user receives more drug than required to maintain desirable levels."
- **Harm:** A text field to list all possible harms related to this hazard. Depending on the level of exposure, one can usually list multiple degrees of harm.
  For instance: Overdose can lead to "Temporary unconsciousness, organ damage, coma, death". Electrocution can lead to "burning wounds, heart fibrillation, death".
- **Category**: Hazard, Comment (non-meaningful item) and Heading (non-meaningful item).

Traces

Upstream

Introduced By: Risk Control Measure items

Downstream

Caused By: Risk items

6.2.2 Risk Analysis domain

Risks describe a likeliness to cause hazards to occur with a certain level. In this domain, the risks are described and consequently evaluated against the probability of occurrence, the likelihood of the risk leading to harm and the severity of the harm. Risks are related to hazards in a many to many relationship. Risks are mitigated by Risk Control Measures. The individual and combined effect of these measures is also visible from this domain.

Purpose

Risk Analysis

Describe the risks that might cause hazards to become incidents and indicate what triggers them and how big the effect can be. The cause and the effect are important parts of the description since they are an aide in the risk estimation process to be more consistent.

Risk Estimation

Estimate the probability that the risk occurs (*"Occurrence [P1]"*), the likeliness that the hazard leads to an incident when the risk has occurred (*"Likeliness [P2]"*) and the severity of the harm if the hazard has become an incident triggered by this risk (*"Severity [S]"*).

Occurrence [P1]	Likeliness [P2]	Severity [S]
Improbable	Never	Negligible
Remote	Rarely	Minor
Occasional	Unlikely	Serious
Probable	Likely	Critical
Frequent	Certainly	Catastrophic

The Risk Level is calculated from these values through the Risk Index field.

Risk Evaluation - Before Risk Control Measures

Severity =>	Negligible					Minor					Serious					Critical					Catastrophic				
	o0	o1	o2	o3	o4	o0	o1	o2	o3	o4	o0	o1	o2	o3	o4	o0	o1	o2	o3	o4	o0	o1	o2	o3	o4
Never	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1
Rarely	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	2	1	1	1	1	1	1
Unlikely	2	2	2	2	2	2	2	2	2	1	2	2	2	1	1	2	2	2	1	0	1	1	1	1	0
Likely	2	2	2	2	2	2	2	2	1	1	2	2	1	1	0	2	2	1	0	0	1	1	1	0	0
Certainly	2	2	2	2	1	2	2	1	1	1	2	1	1	0	0	2	1	0	0	0	1	1	0	0	0

- Rows: Likelihood
- Columns: Occurrence (o0=Improbable, ..., o4=Frequent)
- Invalid or undefined value for Severity, Likelihood or Occurence => result: 99

The Risk Level uses the values of the Risk Index. The latter is hidden from the default view. Both are calculated instantaneously when a Risk Estimation is changed.

Output of Risk Index	Risk Level
0	Intolerable
1	Risk Reduction
2	Insignificant
≥3	Unspecified

Link to risk control: Risk Reduction

Gives a quick overview of all mitigating Risk Control Measures for this particular risk. The default columns are: ID, Category, Text, Occurrence Reduction, Likelihood Reduction, Severity Reduction, Assigned User, Relationship Flags. See below for more details about the Risk Control Measures.

Residual Risk - After Risk Control Measures

Shows the calculated fields for the remaining risk values: Residual Occurrence, Residual Likelihood, Residual Severity and Residual Risk Level. The level is calculated in the same way as defined above. The individual values (residual occurrence, likelihood and severity) are calculated from the original estimate minus the reduction value for all related control measures. Obviously, it will not go lower than the lowest allowed value.

Risk / Benefit Analysis

When the Risk Level after measures is still higher than "Insignificant", but the product is still allowed to be released, evidence must be provided. For instance, in an MRI scanner the patient is exposed to significant doses of radiation. But, it is the best available solution, so a Risk / Benefit Analysis will turn out in favor of allowing the device on the market.

The analysis can be attached as prove through a related "Evidence" item. This item is also used to provide requirement verification/validation proof for requirements that are verified/validated by demonstration, inspection or analysis instead of a test case. It's a plain item that allows attachments and an annotation. It is a separate item to enable the suspect link mechanism when things would change. It has a simple workflow (proposed, in review, accepted, rejected; all state transitions are allowed).

Document Model

Document = Risk Analysis

Content = Risk

Content categories = Software Risk, Hardware Risk, Heading, Comment

Significant Edit Fields

- Text (Description): A unique summary of a risk.
- Cause: What can cause the risk to take place?
- Effect: The effect the risk / failure mode has on the product or system. This is not necessarily equal to the hazard, but it can.
- Category: Software Risk, Hardware Risk, Comment (non-meaningful item) and Heading (non-meaningful item).

Traces

Upstream

Causes: Hazard items.
Introduced By: Risk Control Measure items.

Downstream

Mitigated By: Risk Control Measure items (standard risk mitigation process).
Motivated By: Evidence items (Risk / Benefit analysis).
Verified By: Evidence items.

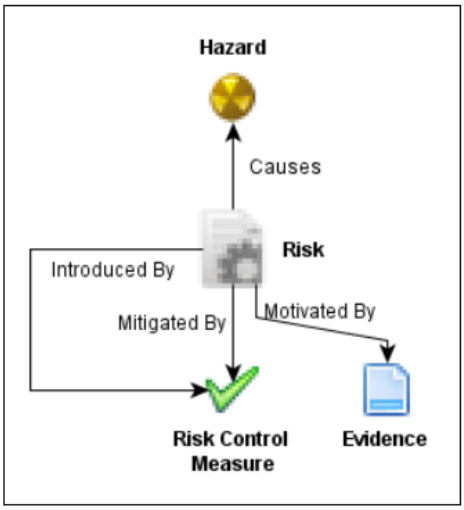


Figure 6-4: Risk Analysis Traces.

6.2.3 Risk Control domain

Purpose

To link risk mitigating measures to Risk items. Each Risk Control Measure has an individual scoring of 'Risk Reduction' for Occurrence, Likelihood and Severity, where "0" means the measure has no effect for that particular scoring and "-4" means that the measure completely eliminates that scoring.

Occurrence Reduction	Likeliness Reduction	Severity Reduction
-0 (No effect)	-0 (No effect)	-0 (No effect)
-1	-1	-1
-2	-2	-2
-3	-3	-3
-4 (Max. reduction)	-4 (Max. reduction)	-4 (Max. reduction)

Working with a risk reduction instead of a new risk value enables the reuse of control measures for different risks. It also makes the scoring more objective since the control measures are rated separately and not with a specific risk level in mind.

Document Model

Document = Risk Control
Content = Risk Control Measure
Content categories = Inherent safety by design, Protective measures, Information for safety, Heading, Comment

Significant Edit Fields

- Text (Description).
- Category.

Traces

Upstream

Mitigates: Risk items

Downstream

Introduces: Risk and Hazard items
Implemented By: Requirement, Specification, Test Case and Change Order items
Verified By: (Verification) Test Case and Evidence items

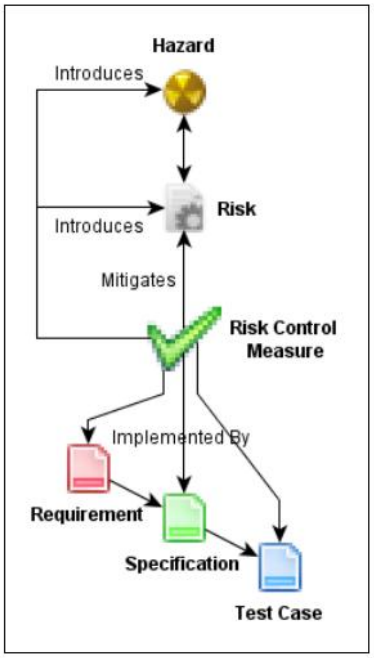


Figure 6-5: Risk Control Traces.

6.3 Engineering Method: Requirements Management

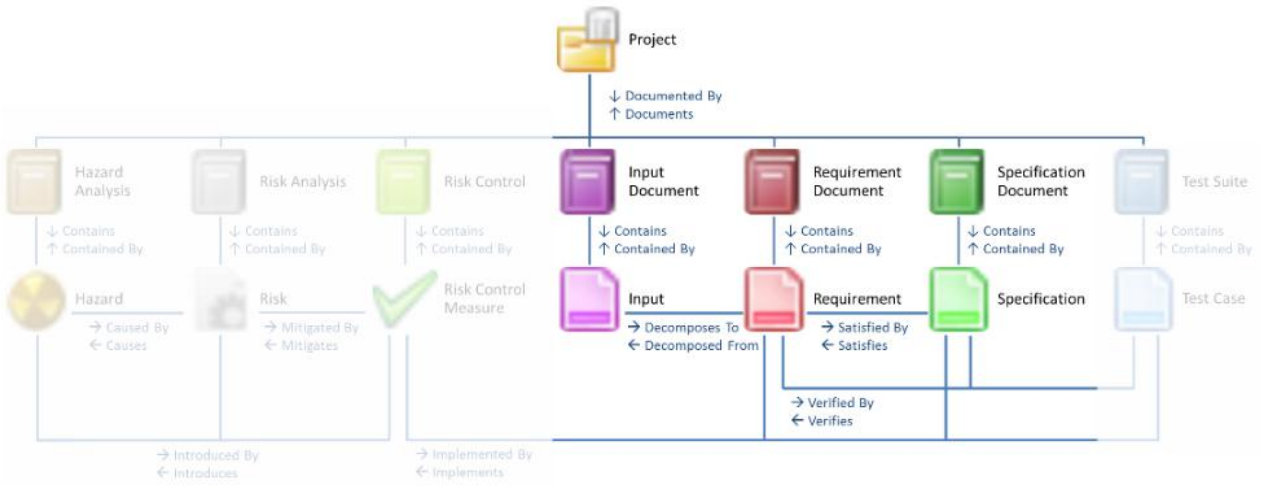


Figure 6-6: Requirements Management.

6.3.1 Input domain

Purpose

The purpose of the *input domain* is to **draft the business case**. It allows to list and argument strategic decisions for the project to be a success. It is typically the domain for the **product manager**. This document outlines customer requests, market needs, industry specifications and other constraints that need to inform the product requirements. They are typically written in the voice of the customer and are often still too vague to be verified/validated. That's why they only trace to *requirements* and not directly to *test cases*.

Examples

- When the business analysis concludes that it is critical that the product must be sold in country xyz, this would be described in the input domain. In the requirement domain this input would be the reason why certain regulatory requirements are indeed required.
- While requirements such as "the user must see the time in an easy to read fashion" are absolutely not acceptable in the requirement domain, they can be in the input domain because this is just the first step towards the actual requirements. A valid requirement, related to this input, can be that "for the date display, the user must be able to enter their preferred view of the date where the relative positions of the day, month and year can vary. For example DD/MM/YYYY."

Recommendations

- Business case report :
  - Project drivers, constraints and use cases.
  - A first set of requirements, if available.
- Wish list.

These documents precede the actual PRD (Product Requirement Document) and its content is not verified/validated by tests.

It is not mandatory to use this domain. A team can also decide to include this type of information in the product requirement document itself. However, it should be noted that in that case

- They also appear in project metrics unless their category is set to 'Comment', but then you lose a level of detail and filtering capability;
- There is no suspect link on changes. The current mechanism only makes a suspect when the upstream trace has changed; there is no suspect notification for downstream traces nor for peer traces. This behaviour is configurable by the Administrator. To circumvent this, we've made it possible to use the "Decomposes To/Decomposed From" trace pair within the domains as well.

When use cases in your project are decomposed to requirements and don't need to be validated in their turn, it is recommended to put them in an input document. Through peer traces it is still possible to relate them to other project inputs.

Document Model

Document = Input Document
Content = Input
Content Categories = Input; Heading; Comment

Significant Edit Fields

Significant fields are fields that, when edited, can trigger a 'Suspect Flag' to related items or can trigger a version increase.

Text: The description field. This field can contain rich content (tables, pictures, OLE objects).

Traces

Upstream

None, this is the highest level.

Downstream:

Decomposes To: Requirement items (and Input items, though not by Alt+Drag&Drop).

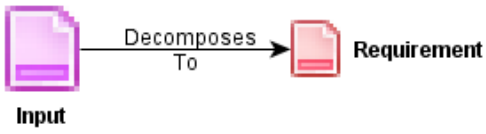


Figure 6-7: Input Domain Traces.

6.3.2 Requirement domain

Purpose

The requirement domain is the core domain where requirement management happens. Typically, this domain is a response to the input domain where the business case was made and marketing requirements were set. Documents in this domain describe the requirements in a technical way in the voice of the manufacturer. **All requirements must be verified/validated.** This can happen via a traced *test case* or via *evidence*, which covers verification by analysis, demonstration and inspection.

Recommendations

A project should contain at least one product requirement document. If you want to include (NPI) process requirements as well, we recommend putting them in (a) **separate document**(s). This enables you to move them through a workflow at a different timing. It's also easier to communicate to management and customers if you keep product requirements separated from additional internal (process) requirements. **In general, when a subset of requirements must/can be delivered sooner/later than others, you put them in a separate document.**

Requirements are typically definitions, an answer to "What is required?". In additional (optional) text fields, **rationale and fit criteria**, a requirement can be more specifically defined. The information in these fields are an answer to "Why is it required?" and "How do we prove that the requirement is fulfilled?". These additional fields are not mandatory since we don't want to create overhead. However, we strongly recommend using them as they help to understand the real purpose of the requirement, specifically at a later stage in the project when they are validated. We provided these fields so that the factual description can remain short, which allows to keep a better overview.

When use cases are detailed enough, have fit criteria and should be validated, it is recommended to include them in the product requirement document. When they don't need to be validated, but you want them in the same document anyway, it is recommended to use the 'Comment' category instead. That way, you can't filter on use cases anymore, but your project metrics (e.g. 'number of requirements without a verification trace', 'verified pass percentage', etc.) will remain correct. Note that it is possible to create peer traces, i.e. from requirement (e.g. a use case or a comment) to requirement (e.g. a functional requirement). However, in the current configuration, peer traces don't become suspect after a change on either side. A suspect flag is only raised when the upstream trace has changed. To circumvent this, we've made it possible to use the "Decomposes To/Decomposed From" trace pair within the domains as well.

Document Model

Document = Requirement Document.
Content = Requirement.
Content Categories = Use Case; Functional Requirement; Non-functional Requirement; Heading; Comment.

Significant Edit Fields

Significant fields are fields that, when edited, can trigger a 'Suspect Flag' to related items or can trigger a version increase.

Text: The main description field. This field can contain rich content (tables, pictures, OLE objects).
Fit Criteria: The description field that is particularly important for the downstream documents.
Category: The requirement type.

Traces

Upstream

Decomposed From: Input items (and Requirement items, though not by Alt+Drag&Drop unless when one of the two is a Use Case where the Use Case is higher in the hierarchy).
Implements: Risk Control Measure items.

Downstream

Decomposes To: User Story items (and Requirement items, though not by Alt+Drag&Drop unless when one of the two is a Use Case where the Use Case is higher in the hierarchy).
Satisfied By: Specification items.
Verified By: Test Case items, Evidence items.

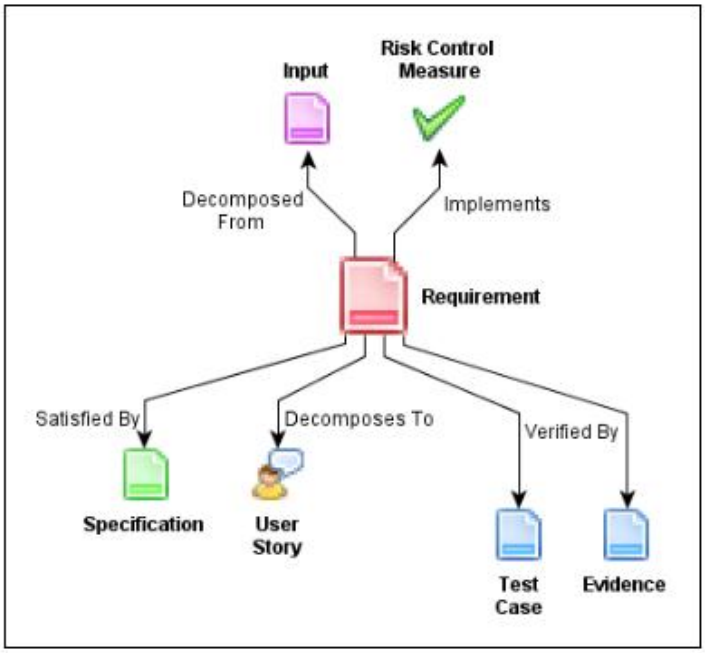


Figure 6-8: Requirement Domain Traces.

View in Pilot Project: single Requirement.

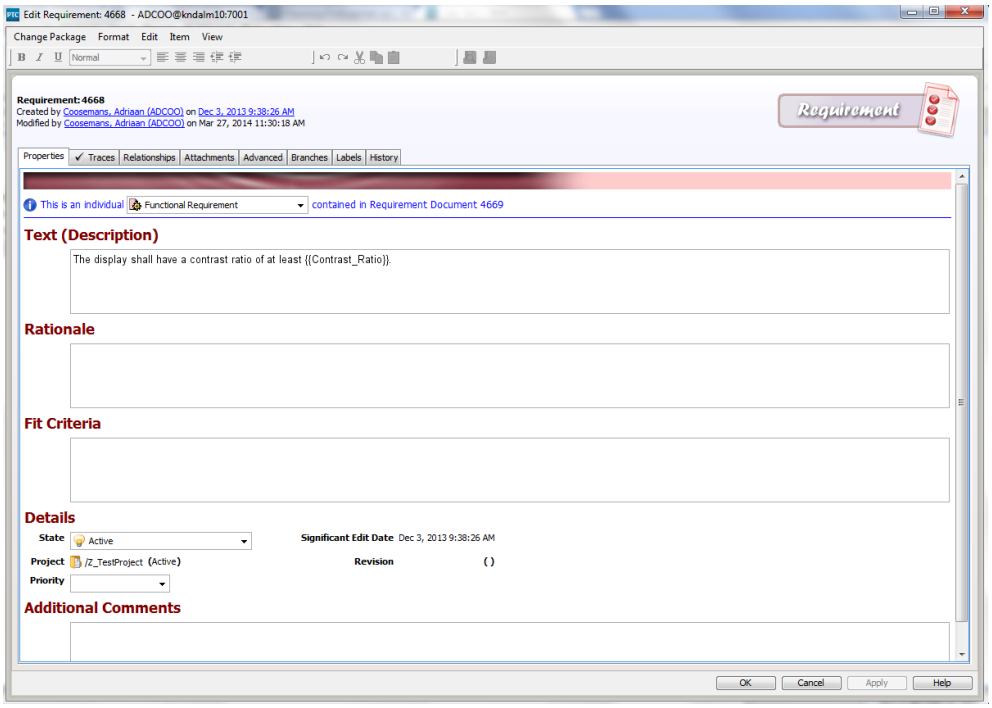


Figure 6-9: Single Requirement in Integrity Pilot.

View in Pilot Project: Requirement document.

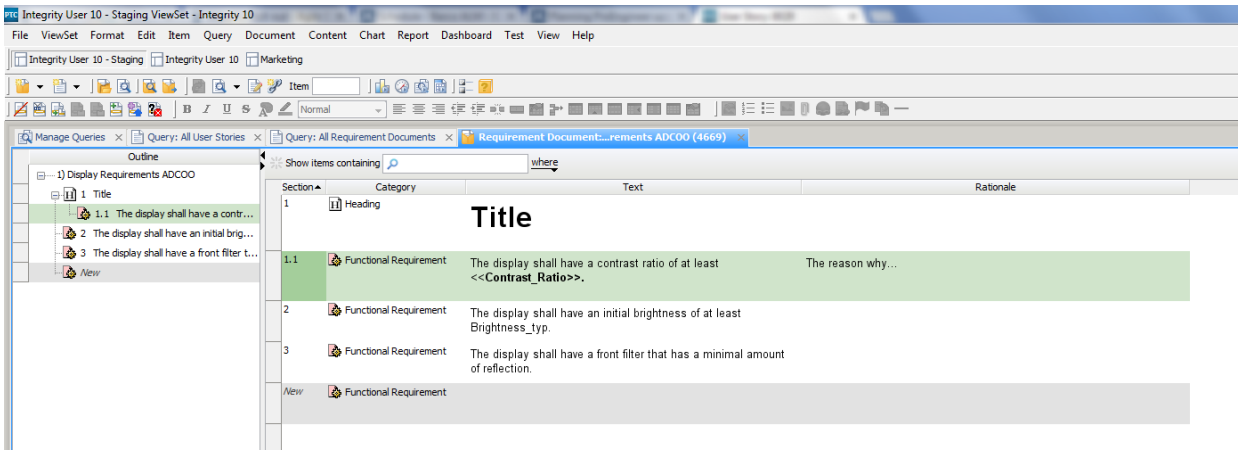


Figure 6-10: Requirement document in Integrity Pilot.

6.3.3 Specification domain

Purpose

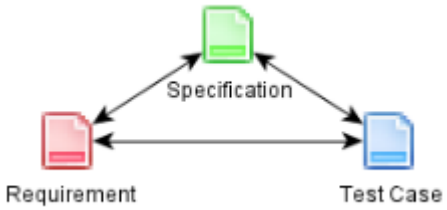
Specifications allow you to provide more detail than you would provide in a requirement document. Typical content describes the actual implementation of the requirement through description, images, design diagrams, etc. They should be verified/validated by a *test case* or *evidence*.

Recommendations

In the current configuration, verification/validation of a specification doesn't replace the verification/validation of the corresponding requirement(s) in metrics and charts such as "requirements not verified". Of course, the traceability exists, but the metrics are not adapted to look further than one trace link. If you don't use the metrics, you don't have a problem. If you do use them, we recommend to establish a direct trace between the requirement and the test case as well, even when you also trace the same test case to a specification that you've traced to a requirement.

Breaking down your requirements to specifications is not mandatory.

If you want to use all the functionality of charts, queries and reports, we recommend tracing your test cases both to the relevant requirement and the specification items.



Don't use specifications if it brings no added value to your project. Trace directly to test cases or evidence items.

If you use specifications and just want to establish traceability, but you don't care about metrics and charts. Just establish the traces you need.

Document Model

Document = Specification Document
Content = Specification
Content Categories = Specification; Heading; Comment

Significant Edit Fields

Significant fields are fields that, when edited, can trigger a 'Suspect Flag' to related items or can trigger a version increase.

Text: The description field. This field can contain rich content (tables, pictures, OLE objects).

Traces

Upstream

Decomposed From: Specification items, though not by Alt+Drag&Drop
Implements: Risk Control Measure items
Satisfies: Requirement items; User Story items

Downstream

Decomposes To: Specification items, though not by Alt+Drag&Drop
Verified By: Test Case items; Evidence items

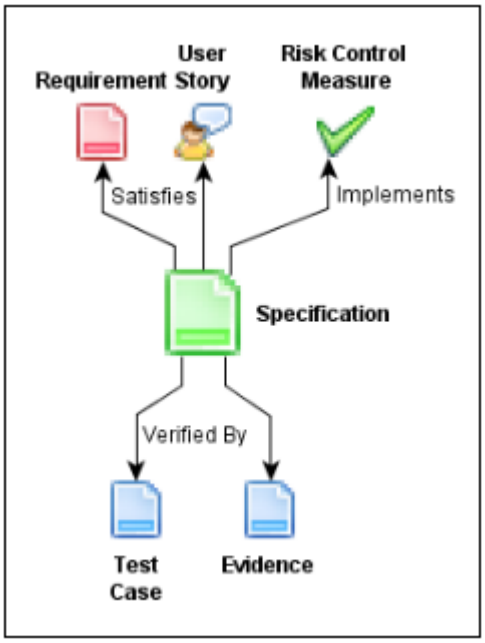


Figure 6-11: Specification Domain Traces.

6.3.4 User Story

Purpose

The *User Story* item type exists for the purpose of the more agile development teams. It is similar to the requirement item in that it has three default text fields to describe it: a 'Summary' field, a 'Description' field and a 'Fit Criteria' (or 'How to Demo') field. The big difference is that it is an item that exists by itself. This means that you can't group User Stories into a document like you do with requirements; you relate them to individual requirements instead. Also the workflow is managed per user story, while for the requirement domain it is managed on the document level for all containing requirements.

View in Pilot Project: User Story.

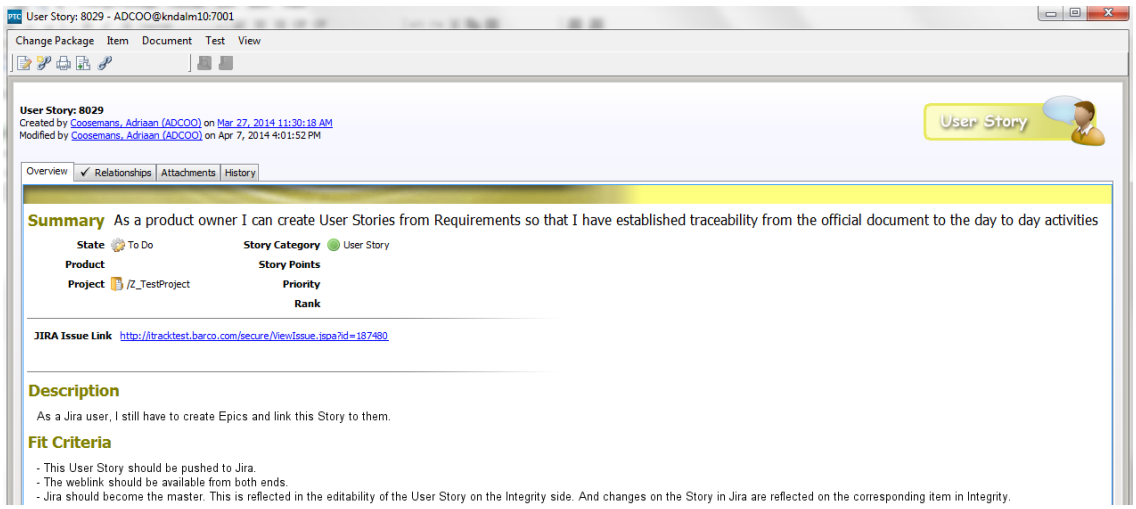


Figure 6-12: User Story in Integrity Pilot.

Use case 1: Stories created in Jira

When? No need to relate them to a requirement document (or feature tree). But you do want to trace them to test cases.

What to do?

1. Make sure you have an Integrity project with a manually established link to Jira;
2. Create your Stories in Jira;
3. Your Stories will be automatically pushed to Integrity and remain synchronized (Jira stays the master);
4. If you need it, you can relate the User Stories in Integrity to Specifications;
5. The test team can relate their test cases to up to date User Stories in Integrity.

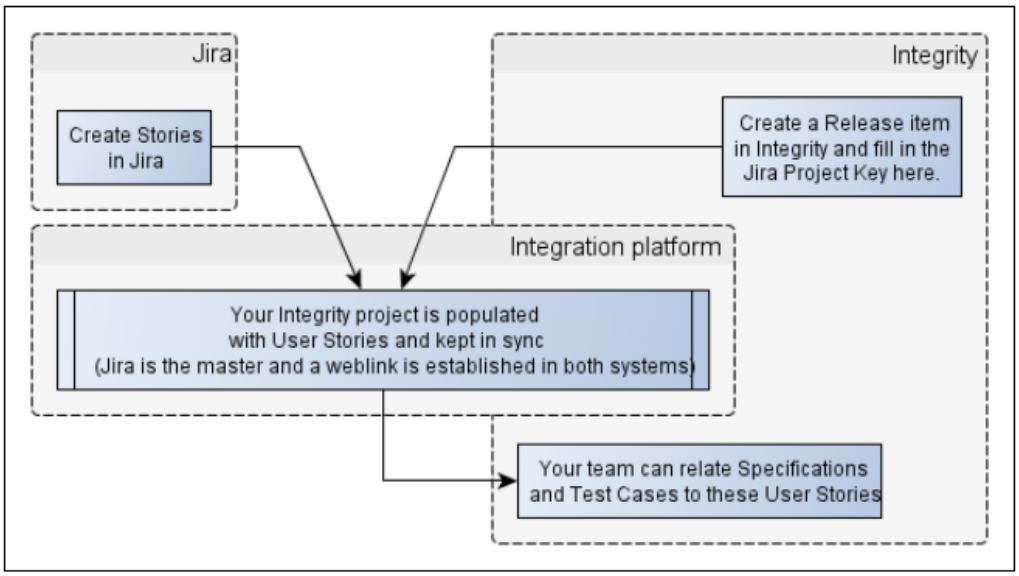


Figure 6-13: User Story concept in Pilot.

View in Pilot Project: User Story in Jira.

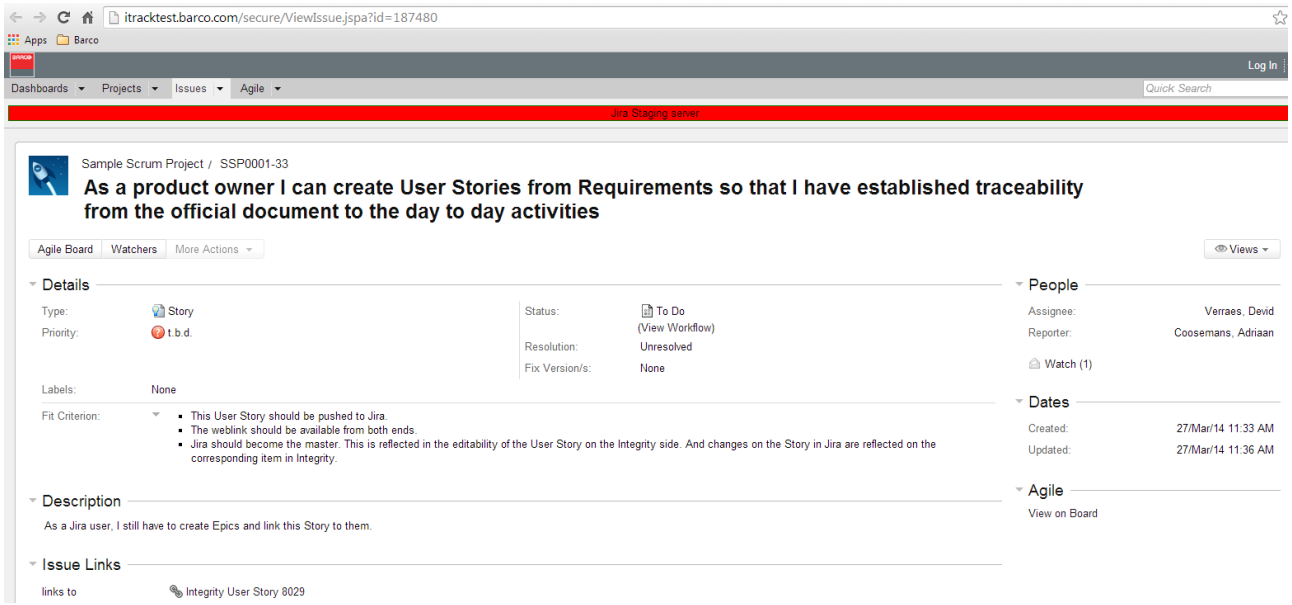


Figure 6-14: User Story in Integrity Pilot.

Use case 2: Stories created in Integrity

When?

- When the inputs of your project originate from Integrity;
- When you need a trace to official requirements or to a feature tree that you document in Integrity;

What to do?

1. Create User Stories in Integrity (related to a *Product* item) -> Draft state.
2. Groom them (define content, size, rank in product backlog) -> To Do state.
3. Add them to a *Release*, which must be linked manually with the corresponding Jira project.
4. From here, your User Stories are automatically pushed to Jira -> Jira becomes the master.
5. Changes in Jira are synchronized to Integrity, not vice versa.
6. If you need it, you can relate the User Stories in Integrity to Specifications.
7. The test team can relate their test cases to up to date User Stories in Integrity.

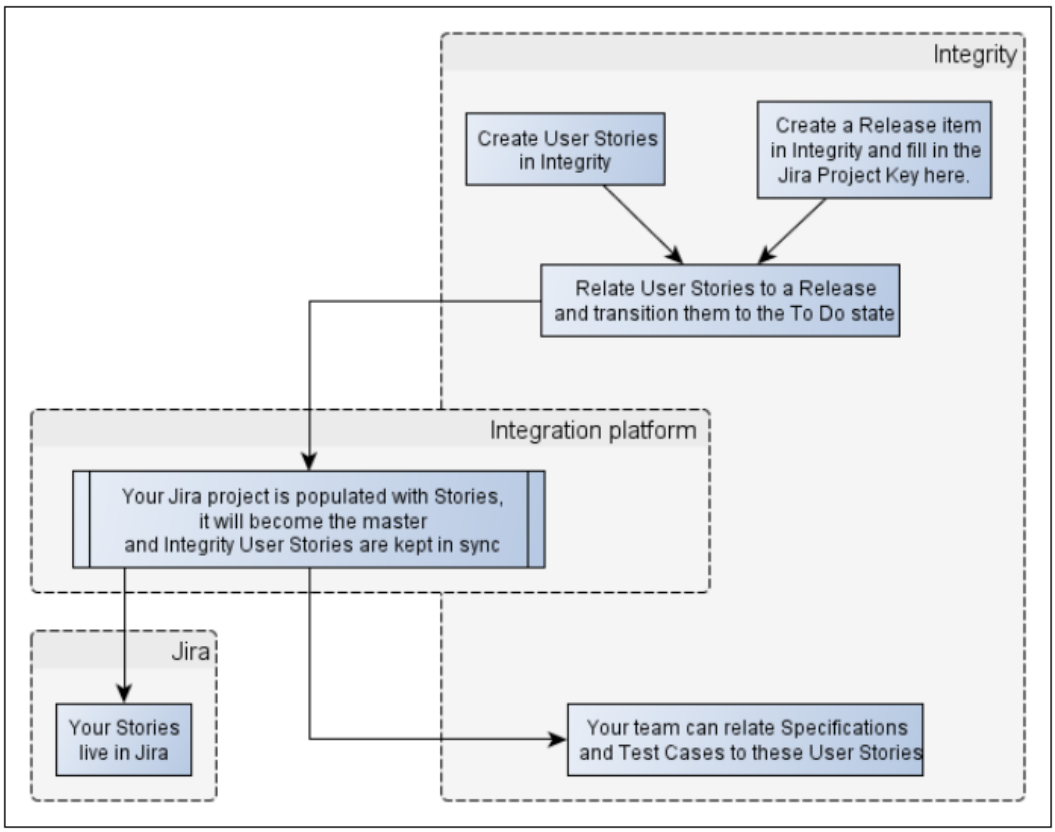


Figure 6-15: Stories created in Jira.

Recommendations

To meet the way of working in agile development teams, the *Product* item has been extended with a 'Product Backlog' which is your collection of User Stories that have no related Release. We recommend to use this for the 'grooming' process.

Each release must be a new project, backed by a *Release item*. This Release item contains all the User Stories that have been groomed and are ready to be implemented in this release.

If your system under development is (or will become) large, it is recommended to keep a Requirement Document in the root folder of your project. This can serve as a 'feature tree'.

Document Model

The User Story item is a separate item. It is not part of a document and thus it has a separate workflow. The workflow is like the Story type in Jira/Greenhopper. The Draft state is an extra for the teams that define their work in Integrity (Use Case 2)

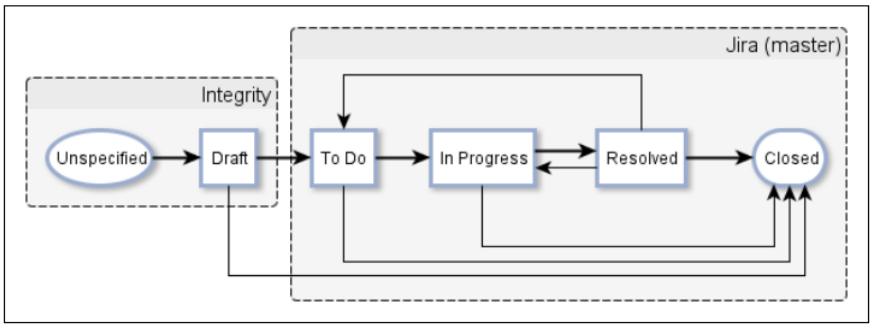


Figure 6-16: Story flow in Pilot.

- Once your Story is created, it is in the Draft state.
- Once it has a Story Size (in Story Points) and Rank in the product backlog, you should move it to the To Do state. User Stories are pushed to Jira Stories when they are in this state and part of a Release. From then on, Jira is the master. All changes in Jira are synchronized back to Integrity. No changes are allowed in Integrity.
- Once the User Story has been assigned to a Sprint, it moves to the In 'Progress' state.
- Once development and testing is complete, a 'Story' moves to the 'Resolved' state. It remains there until that Story's functionality is demonstrated to the Product Owner and he/she accepts it as working software, whereupon it moves to the 'Closed' state. If the Product Owner is not happy with the implementation, it either moves back to In Progress (potentially moving back to 'To Do' if it needs to be continued in another Sprint).

Categories

- User Story

Traces

Upstream

Decomposed From: Requirement items

Implemented In: Release item (and Sprint item: not enabled)

Downstream

Satisfied By: Specification items

Verified By: Test Case items; Evidence items

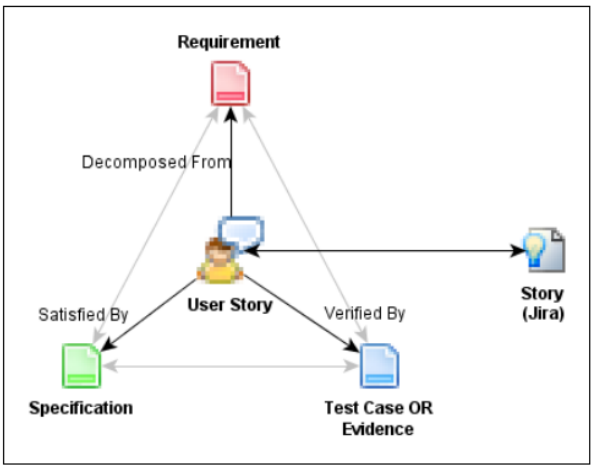


Figure 6-17: User Stories Traces.

View in Pilot Project: Traceability link between Requirement document and Use Story:

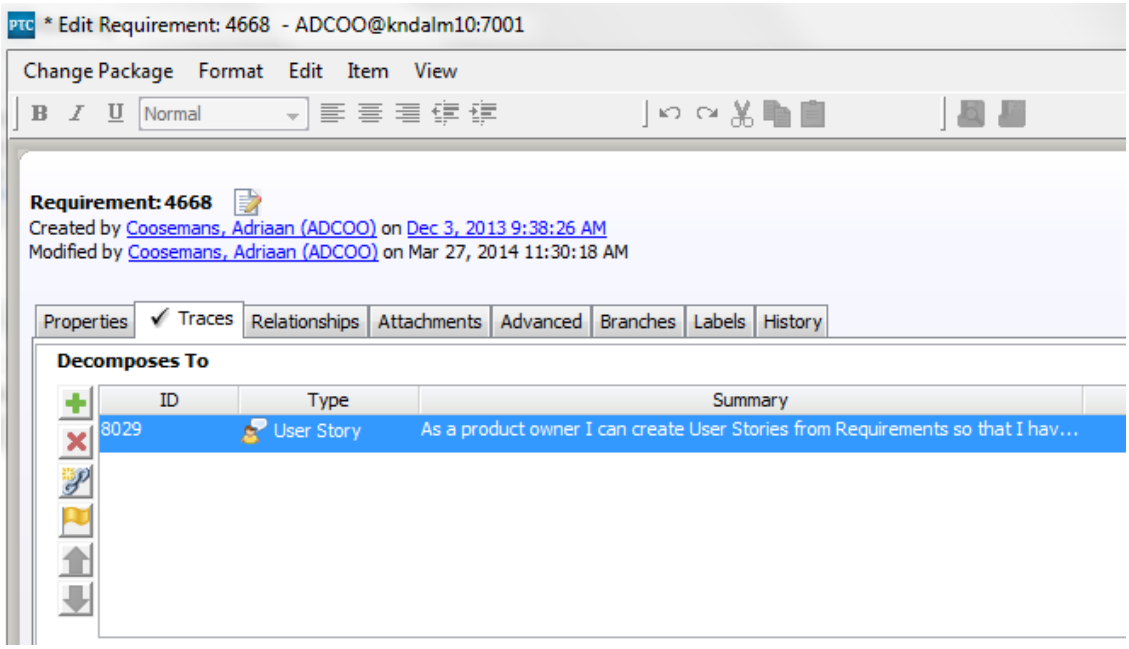


Figure 6-18: Traceability link between Requirement document and Use Story in Pilot.

View in Pilot Project: User Story Web interface.

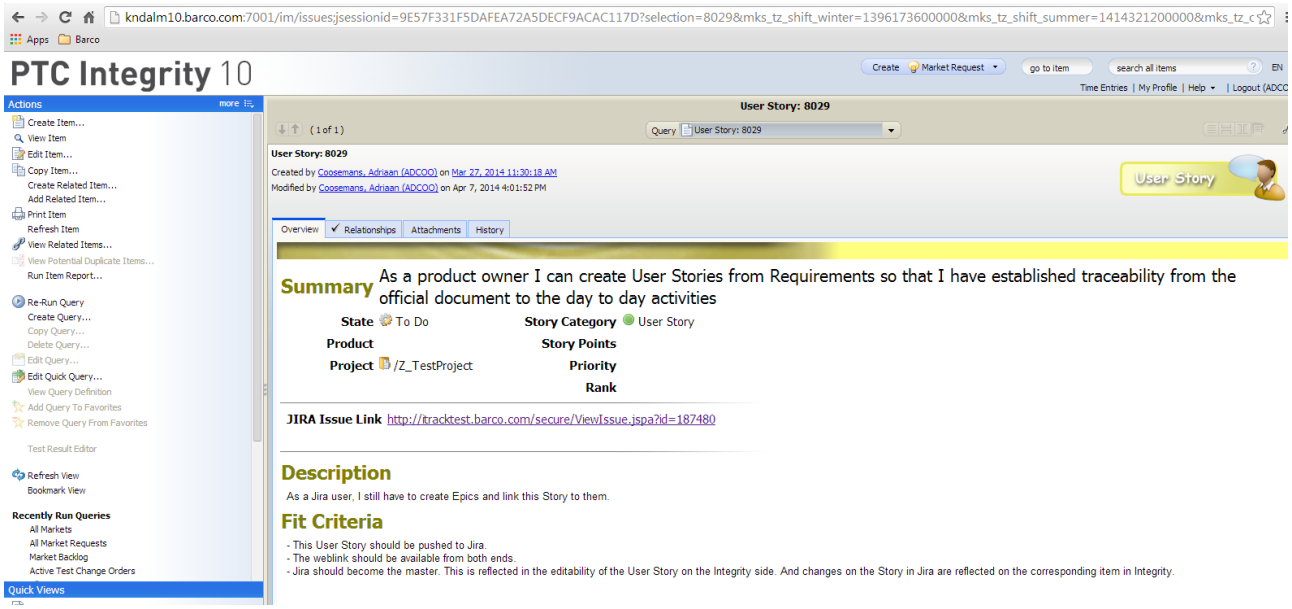


Figure 6-19: User Story Web interface in Pilot.

View in Pilot Project: User Story Traceability Web interface.

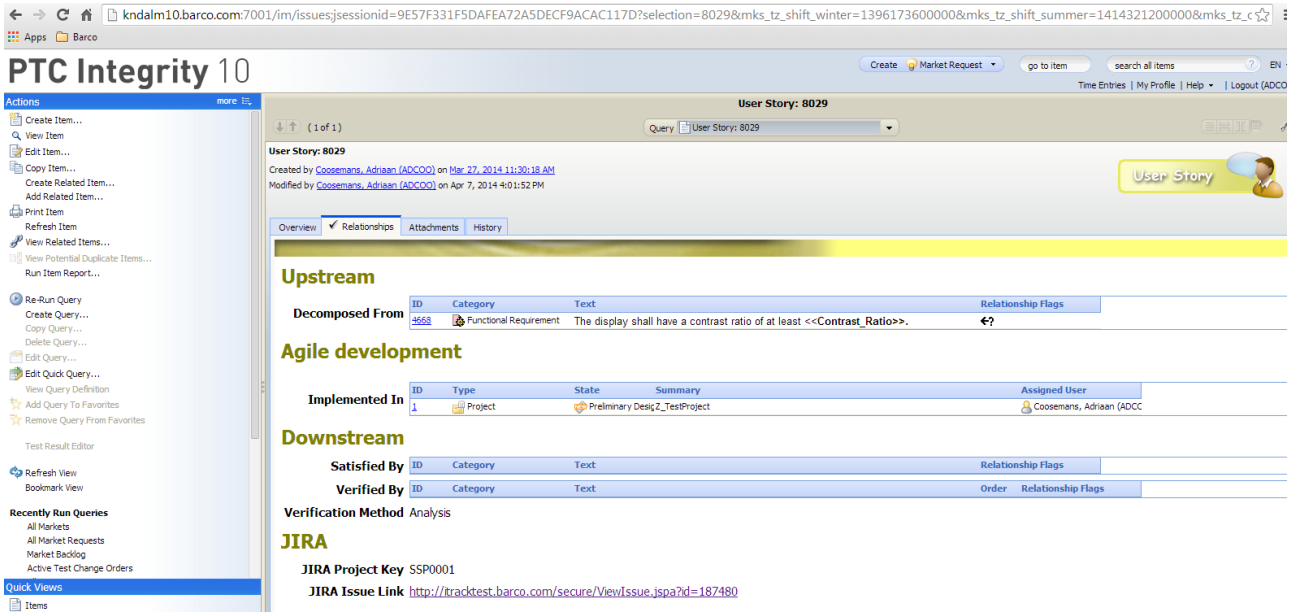


Figure 6-20: User Story Traceability Web interface in Pilot.

6.4 Engineering Method: Test Management

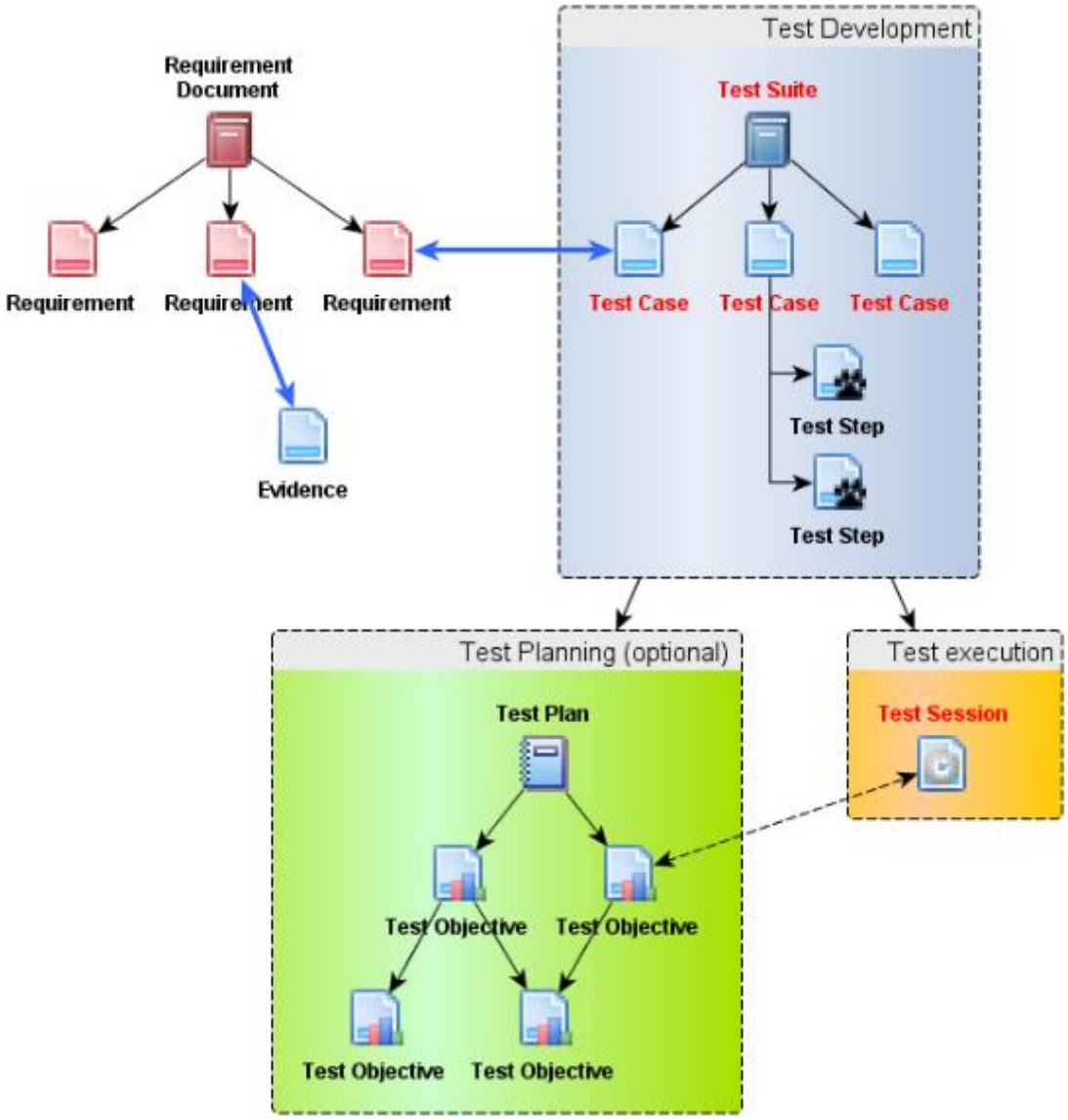


Figure 6-21: Test Management.

Purpose

The purpose of the test domain is to verify/validate product requirements described in the requirement domain. In terms of documents and content it is very easy: just test suites with test cases (and optional test steps). But this domain also allows planning the testing and logging the results. So, there are some additional items to facilitate these processes.

6.4.1 Test Development

It is recommended to use the Test Developer View Set to write your test cases.

Like every other document type, but especially relevant for test suites, it is recommended to write different test suites when they are managed by different persons or when they will flow to their workflow at a different pace.

Document Model

Document = Test Suite

Content = Test Case > Test Steps

Content Categories = TBD; Heading; Comment

Each Test Step is a separate item. This allows logging test results per test step as well. And more advanced, it also allows you to build a library with test steps that can be reused in different test cases.

It is also possible to verify/validate requirements by Analysis, by Demonstration or by Inspection. In that case evidence can be provided through the Evidence item. This method is preferred because it allows to keep all counters and project statistics (such as "requirements not verified") intact. Using an Evidence item spares you some time of going to a test process. The Evidence item has the three alternative verification methods as a Category field. It doesn't fit in a document, it's an item that exists by itself and has its own workflow.

Below the workflow of the Evidence item is illustrated, allowing verifying a requirement without a test.

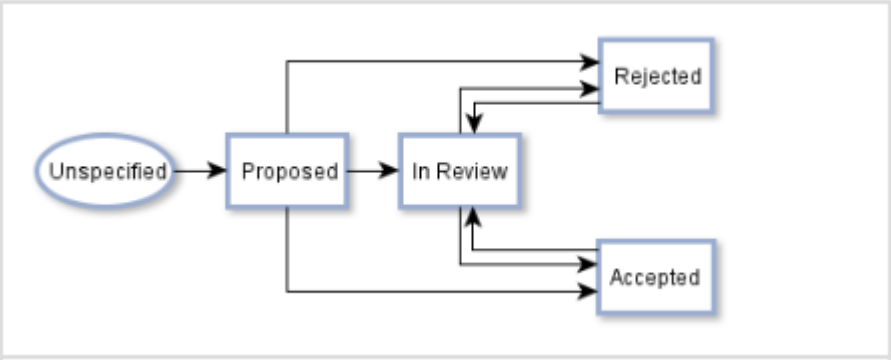


Figure 6-22: Evidence item workflow.

Analogue to Requirement Management and Risk Management, the workflow of test cases and test steps are managed simultaneous per test suite. So, the Evidence item is an exception to this rule.

Significant Edit Fields

Significant fields are fields that, when edited, can trigger a 'Suspect Flag' to related items or can trigger a version increase.

Text: The description field. This field can contain rich content (tables, pictures, OLE objects)
Expected Results: A text field to describe the expected outcome of executing that test case.
Test Steps: A reference field to all related test steps.

Traces

Upstream

Verifies: Requirement items, Specification items, User Story items, Risk Control Measure items.

Downstream

None, this is the lowest level.

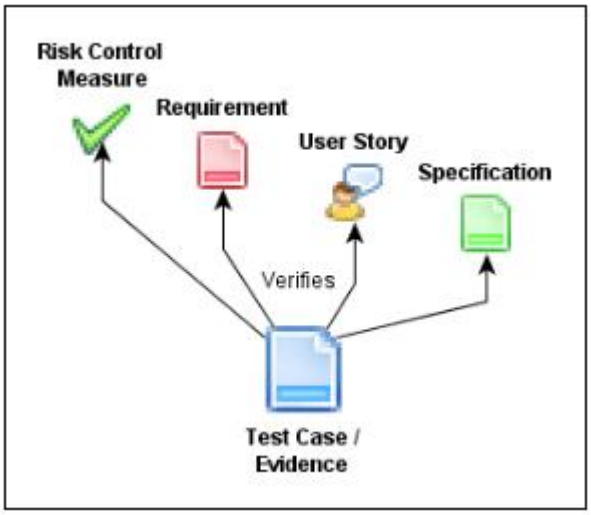


Figure 6-23: Risk Development Traces.

6.4.2 Test Planning

This phase is optional, yet recommended.

It is recommended to use the Test Manager View Set to do the planning and gather metrics of all tests.

Items

- Test Plan: the top item of the planning process. It doesn't contain any tests; it's only the top item of the planning process. You would have only one Test Plan per project. The Test Plan is only relevant when you have multiple Test Objectives in your project and you want to gather the metrics of different Objectives here.
- Test Objective: a process item to group the Test Cases from your Test Suites according to your way of working. It is possible to associate the Test Objective with entire Test Suites and/or a combination of Test Cases from different test suites. You can make a complete work breakdown structure of Test Objectives. The Test Plan would be the top item. Examples include:
  - Confidence Testing and Qualification Testing
  - Platforms or predefined configurations
  - Regression testing, alpha testing, white box testing, black box testing, smoke testing, user acceptance testing, ...

Workflows

Workflow for a Test Plan

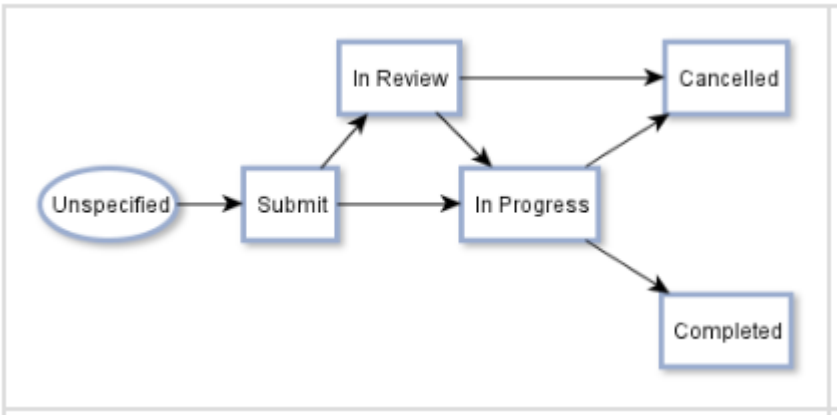


Figure 6-24: Test Plan workflow.

Workflow of a Test Objective

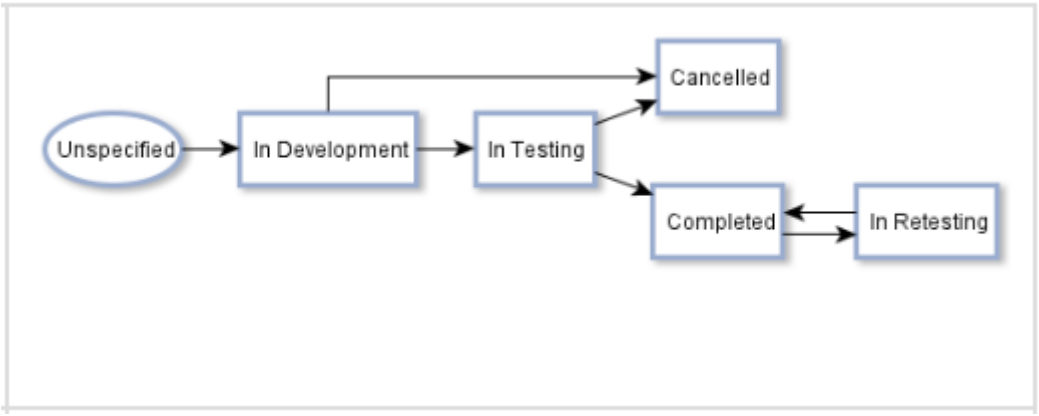


Figure 6-25: Test Objective workflow.

6.4.3 Test Execution

It is recommended to use the **Test Engineer View Set** during test execution to log results.

Items

- Test Session: A unique run of a set of tests. You should log the test results per test session and per test case. Optionally, you can log the result per test step as well. Upon creation, you can relate it to a Test Objective. The system will propose you with the set of test cases/suites that is assigned to that Test Objective. The test engineer is free to select a subsection or add other test cases as well. The stakeholder can also set the parameter values for that specific session. Test session status and results are reflected on the associated test objective items as well. The last test result of a test case is also visible from the related Requirement.
- Defects: When a test case is marked as failed, you can create a related Defect. This Defect will be pushed to iTrack where it is managed. The status is reflected back to Integrity.

Workflow

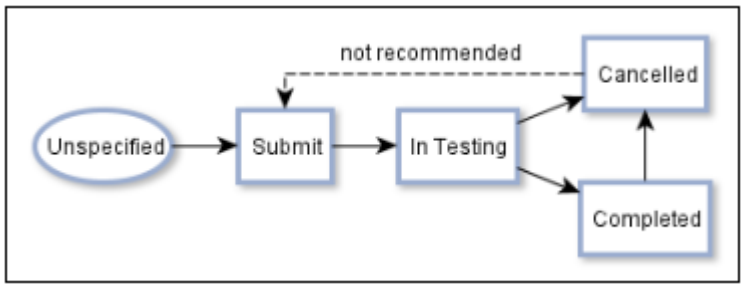


Figure 6-26: Test Execution workflow.

- Once the Test Session is in the "In Testing" state, you can no longer add test cases.
- While you could go all the way back from Cancelled to Submit, it is not recommended unless you've made some mistake and didn't log any test results yet. The purpose of a Test Session is that it is unique. If you already logged results and cancelled the session for some reason. It is better to create a new session if restarting involves going through some tests again.

Automated testing

The automated test execution framework (ATEF) is currently not enabled yet.

7 Conclusions and Way Forward

Conclusions of the Pilot project at M12

We successfully implemented the following Engineering Methods:

- Software Risk Management.
- Requirements Management.
- Test Management.

Resulting in:

- Integrity replacing Excel files and Jira requirements and specifications.
- Integrity interfaces with Jira/Greenhopper on User Stories and Defects. They can be created in both systems, but Jira is always the master.
- Integrity replacing Excel files and TestLink.
- Defects that springs from failed test cases are created from PTC Integrity. They are pushed to, and further managed in Jira.
- Risk Management is tailored for medical devices.

We noticed that requirements in Integrity are too high-level and therefore also too broad to have a one-to-one mapping between a requirement and a user story.

Way forward

Figure 7-1is showing the envisioned goal for the general process at Barco with a connected ALM and OLM process.

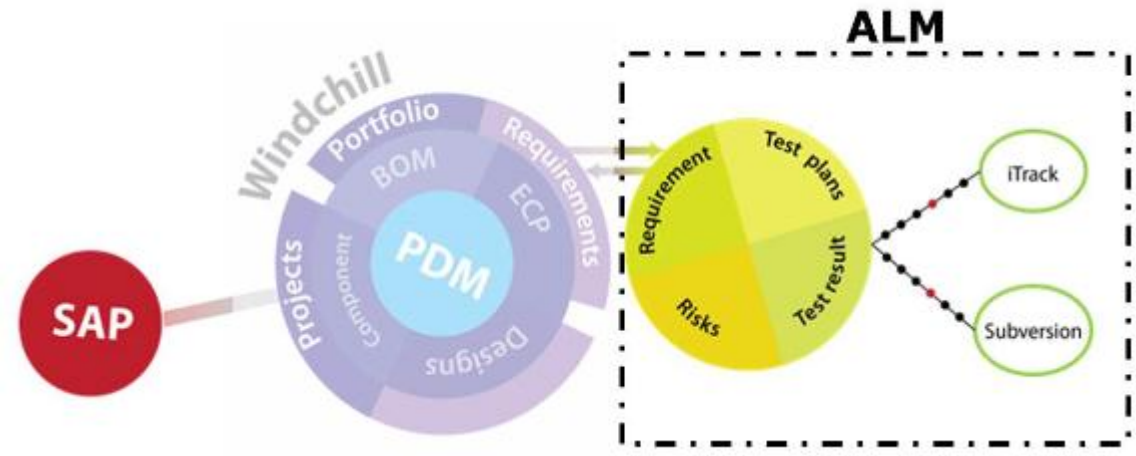


Figure 7-1: Envisioned connected ALM-PLM process.

Following integration and improvement activities are on the roadmap tor year 2 and 3:

- Integration between Integrity and Subversion.
- Integration between Integrity and Git.
- Automatic documentation generation.

- Client UI improvements.
  - Discussions with PTC product management ongoing.
- Support automated testing.
- Introduce a board with drag & drop functionality (desired for Agile teams: story boards).
- Integrate with Subversion.
  - Expectations to be defined/aligned.
- Integrate with Windchill.
  - This implies a thorough expansion of the Integration Platform.
- Integrate with Enterprise Architect.
- Integrate with Matlab/Simulink.
  - This is client to client integration (not using the integration platform).
- Integrate with Green Data Manager.
  - We need the browser edition first.

8 Terms, Abbreviations and Definitions

Table 8-1 provides an overview terms, abbreviations and definitions used in this document.

BCR	Business Case Review
CAPA	Corrective Actions Preventive Actions
CDR	Critical Design review
DICOM	Digital Imaging and Communications in Medicine
DVI	Digital Visual Interface
ECP	Engineering Change Proposal
EOL	End-of-Life Review
FDA	Food and Drug Administration
FMEA	Fault Tree and Effect Analysis
FPGA	Field-programmable gate array
FQR	Formal Qualification Review
FTA	Fault Tree Analysis
GSDF	Grayscale Display Function
HW	Hardware
IOS	Interoperability Specifications
NPI	New Product Introduction
NRE	Non-recurring engineering
PDR	Preliminary Design Review
PLC	Product Life Cycle
POR	Phase Out Review
R&D	Research and Development
RER	Risk Evaluation Report
RMF	Risk Management File
SAD	Software Architectural Design
SRS	Software Requirements Specification
STP	Software Test Plan
SW	Software

Table 8-1: Terms, Abbreviations and Definitions.