

PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE CRYSTAL CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE CESAR CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT. 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.



CRritical **SY**STem Engineering **Acce**Leration

UC 4.5

Software centric scalable safety critical medical display platform

Tool and methodology report

D405.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Tool and methodology report
Deliverable No.	D405.010
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
Kurt Pattyn	Barco	Kurt.Pattyn@barco.com
Frederik Toune	Barco	Frederik.Toune@barco.com
Reginald Swaenepoel	Barco	Reginald.Swaenepoel@barco.com
Stephane Deltour	Barco	Stephane.Deltour@barco.com
Bart Diricx	Barco	Bart.Diricx@barco.com
Ronny Van Belle	Barco	Ronny.Vanbelle@barco.com
Adriaan Coosemans	Barco	Adriaan.Coosemans@barco.com
Arjen van de Wetering	IBM	Arjen.van.de.Wetering@nl.ibm.com
Sytze S.H. Kalisvaart	TNO	Sytze.Kalisvaart@tno.nl
Bardo B.J.H. Bakker	TNO	Bardo.Bakker@tno.nl
Martijn M.H.P. van den Heuvel	TU/e	m.m.h.p.v.d.heuvel@tue.nl

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	31 Mar 2014	Initial version	All
0.02	08 Apr 2014	General update	All
0.03	14 Apr 2014	Update Engineering Methods for New SW Design Process	All
0.04	16 Apr 2014	Final Update FUN100 Platform	All
0.05	21 Apr 2014	Update of general structure	All
0.06	28 Apr 2014	Update after review 1	All
1.00	29 Apr 2014	Update after review 2	All

CONTENT

D405.010	I
1 INTRODUCTION.....	6
1.1 ROLE OF DELIVERABLE	6
1.2 RELATIONSHIP TO OTHER CRYSTAL DOCUMENTS	6
1.3 STRUCTURE OF THIS DOCUMENT	6
2 USE CASE PROCESS DESCRIPTION	7
3 DETAILED DESCRIPTION OF DESIRED DESIGN PROCESS.....	8
3.1 HIGH LEVEL OVERVIEW OF THE DESIRED DESIGN PROCESS	8
3.2 DETAILED ACTIVITY DESCRIPTION	11
3.2.1 Activity: Impact Analysis.....	11
3.2.2 Activity: Requirements.....	12
3.2.3 Activity: Components Engineering.....	13
3.2.4 Activity: System Engineering.....	15
3.3 CONCLUSION	17
4 IDENTIFICATION OF ENGINEERING METHODS.....	18
4.0 BARCO ROADMAP AND THE SELECTED CRYSTAL ACTIVITIES	18
4.1 TRACK 1: NEW SOFTWARE DESIGN PROCESS	22
4.1.1 Introduction.....	22
4.1.2 Engineering Method: Requirements Gathering	23
4.1.3 Engineering Method: Requirements Traceability	23
4.1.4 Engineering Method: Iterative Development.....	23
4.1.5 Engineering Method: Process Automation.....	24
4.1.6 Engineering Method: Key Quality Metrics	24
4.2 TRACK 2: FUN100 DESIGN PROCESS.....	24
4.2.1 Introduction to FUN100	24
4.2.2 IEC-62304 compliant Test Framework for FUN100.....	26
4.2.3 Engineering Method: Software Unit Testing for FUN100.....	28
4.2.4 Engineering Method: Component Testing for FUN100.....	28
4.2.5 Engineering Method: Architectural Design for FUN100	29
4.2.6 Engineering Method: Software Engineering for FUN100.....	30
4.3 TRACK 3: FLEXIBLE SW CENTRIC DISPLAY DESIGN PROCESS	35
4.3.1 Introduction.....	35
4.3.2 Engineering Method: Functional Modeling.....	36
4.3.3 Engineering Method: Performance Simulation	37
4.3.4 Engineering Method: Combining Functional Modeling & Performance Simulation.....	37
4.3.5 Engineering Method: Modular Software Components for QT and gStreamer (Work in Progress)	38
4.4 CONCLUSION: TECHNICAL CORE REQUIREMENTS	39
5 SYSTEM ENGINEERING ENVIRONMENT	44
5.0 INTRODUCTION: HIGH LEVEL OVERVIEW OF THE WP404-405 SEE AT M12.....	44
5.1 TRACK 1: SEE FOR THE NEW SOFTWARE DESIGN PROCESS AT M12.....	45
5.2 TRACK 2: SEE FOR FUN100 AT M12	47
5.2.1 Test Framework for FUN100.....	47
5.2.2 Architectural design for FUN100.....	48
5.3 TRACK 3: SEE FOR FLEXIBLE SW CENTRIC DISPLAY AT M12	49
5.4 CONCLUSION: ENVISIONED SEE.....	51
6 IMPLEMENTED ENGINEERING METHODS	52

6.1	TRACK 1: NEW SOFTWARE DESIGN PROCESS.....	52
6.1.1	Engineering Method: Requirements Traceability	52
6.1.2	Engineering Method: Key Quality Metrics	53
6.1.3	Engineering Methods: Iterative Development & Process Automation.....	54
6.2	TRACK 2: FUN100 DESIGN PROCESS.....	55
6.2.1	Engineering Method: Software Unit Testing.....	55
6.2.2	Engineering Method: Component Testing.....	58
6.2.3	Engineering Method: Architectural Design for FUN100	60
6.2.4	Engineering Method: Software Engineering for FUN100.....	62
6.3	TRACK 3: FLEXIBLE SW CENTRIC DISPLAY PLATFORM DESIGN PROCESS	67
6.3.1	Engineering Method: Functional Modeling.....	67
6.3.2	Engineering Method: Performance Simulation.....	67
6.3.3	Engineering Method: Combining Functional Modeling & Performance Simulation.....	69
7	DEMONSTRATOR DESCRIPTION.....	71
8	CONCLUSIONS AND WAY FORWARD	73
8.1	CONCLUSION	73
8.2	INTEROPERABILITY ISSUES	73
8.3	WAY FORWARD	74
9	TERMS, ABBREVIATIONS AND DEFINITIONS	75

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 CRYSTAL activities focus on setting up the desired process for the Barco software centric scalable safety critical medical display platform. Chapter 2 provides an overview of the general Barco use case, for this section we refer to the CRYSTAL Deliverable D_404_010. Chapter 3 describes the envisioned use case design process. Chapter 4 describes all identified CRYSTAL activities and the link with the Barco roadmap. Chapter gives an overview of the SEE at 12. Chapter 6 show the first implementation results in the new design processes. The selected demonstrator description is given in chapter 7. Finally chapter 9 is summarizing the conclusions for year 1 and looks ahead to the future work for this project.

2 Use Case Process Description

For more details about the Barco Use Case Process Description we would like to refer to section 2 of the Use Case Process' of CRYSTAL Deliverable D_404_010.

3 Detailed Description of Desired Design Process

Section 3 is describing a more detailed description of the desired design process for the Flexible SW Centric display platform.

3.1 High level overview of the Desired Design Process

Figure 3-1 describes the desired development process used for a software-centric and cost-effective replacement of the image pipeline. It is represented as series of activities ('activity diagram') that are executed to deliver the product.

Figure 3-2 shows the legend for all symbols that are used to describe this process. The activities are represented by a rectangle. The start of our development process is indicated by an open circle. The process ends with a completed product and is represented by a solid black circle.

The process is repetitive and uses an agile approach to come to the final product in series of refinement steps, the dotted box surrounding step 3 till 8 indicates one sprint cycle in the agile process.

Step 10, the impact analysis, is the activity in which the appropriate action is decided for all setbacks (arrows going to the left and up) or unexpected events (event blocks at left). At the appropriate level of the development team (e.g. software team for software component changes, system architect for changes in functions or decomposition), the implications of the changes are discussed, the consequences for other components are analysed and the proposed actions on the various levels of the development process are determined. This could be for example adding one requirement or changing a set of related requirements; could imply a change of architecture or could imply a reallocation of functions between two GPUs.

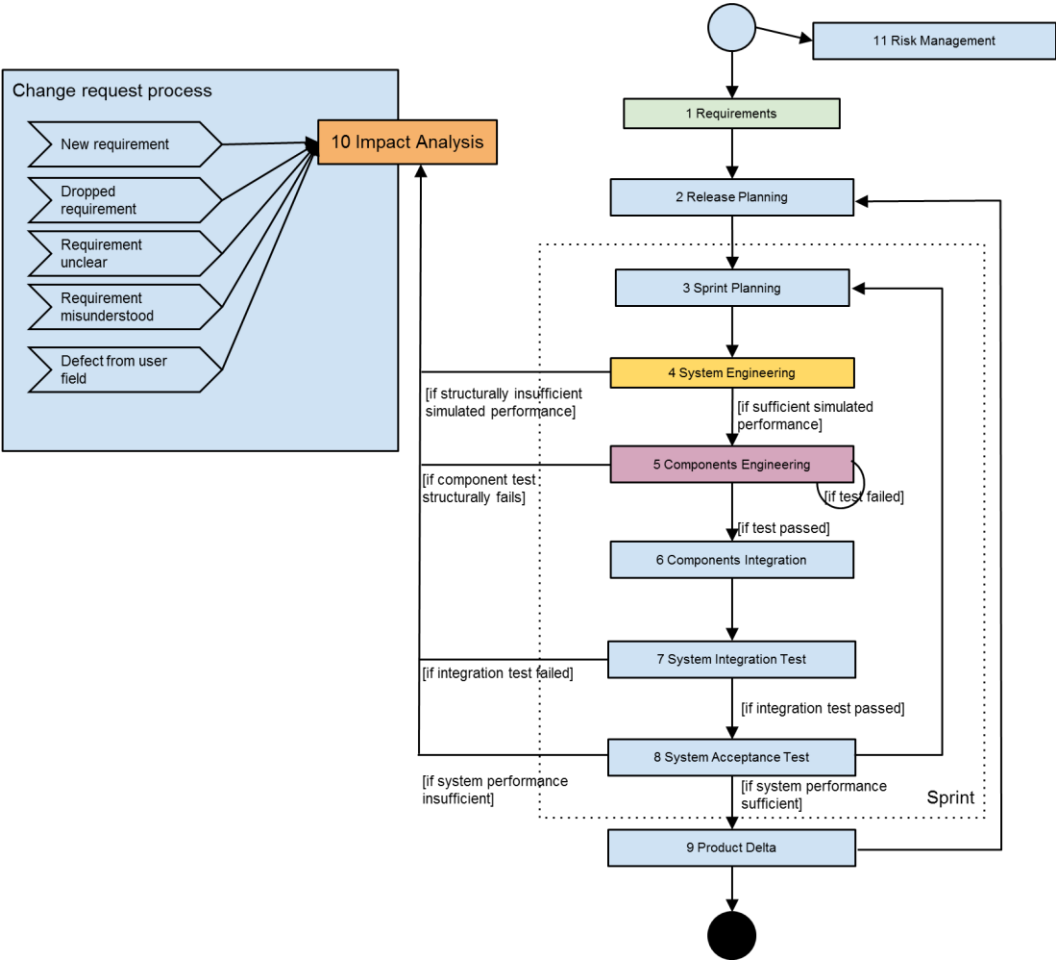


Figure 3-1: Software Display System Process.

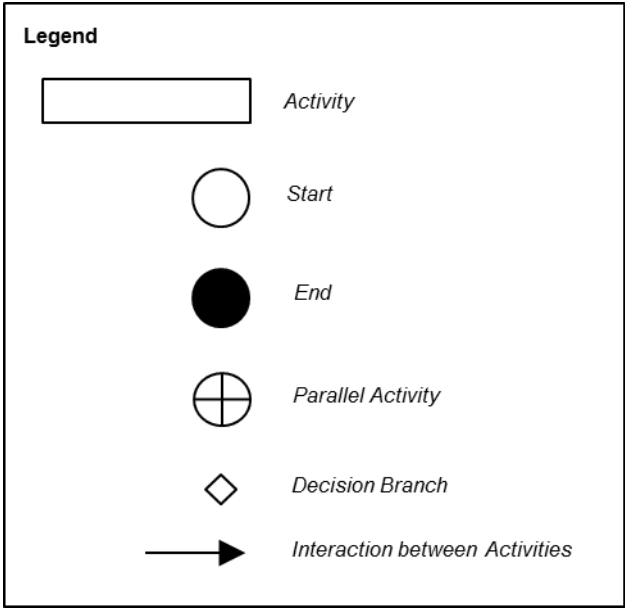


Figure 3-2: Process Activities Legend.

Short description of the activities:

1. Requirements

During this activity the Stakeholder Requirements are captured, maintained and managed.

Input artefacts: Market analysis, stakeholder interviews, legal standards.

Output artefacts: Prioritized Stakeholder Requirements, Product Backlog.

2. Release Planning

Based on the Stakeholder Requirements the product development is planned with respect to releases and their features.

Input artefacts: Prioritized Stakeholder Requirements, Product Backlog.

Output artefacts: Business Case, Release Definition, traceability from Releases to Stakeholder Requirements.

3. Sprint Planning

A Work Breakdown Structure is created for the product release of which the development is started.

Input artefacts: Release Definition, Product Backlog.

Output artefacts: Work Breakdown Structure, Sprint Backlog.

4. System Engineering

During this activity the System Requirements are created and an executable System Architecture is defined and verified by simulation. The System Architecture describes the complete system with all the engineering disciplines such as mechanical, electrical and software.

Input artefacts: Stakeholder Requirements

Output artefacts: System Requirements, Executable System Architecture (System Use Cases, Architectural Decomposition, Component Descriptions, Executable Functional Models), System Requirements to Stakeholder Requirements traceability, System Use Case to System Requirements traceability, Architecture to System Requirements traceability, Trade-off Analysis Report, System Performance Analysis Report.

5. Components Engineering

The architectural components that are defined during System Engineering are handed over to the engineering disciplines where they will be defined, designed, created and tested.

Input artefacts: System Requirements, Executable System Architecture, Component Description.

Output artefacts: Component Requirements, Component Design, Realized Components, traceability from Requirements to System Requirements, traceability from Component Design to Component Requirements.

6. System Integration

The components that make up the system are integrated, thus assembling a part of or the complete system.

Input artefacts: To be decided.

Output artefacts: To be decided.

7. System Integration Test

The system is tested with respect to the system requirements that are created during Systems Engineering.

Input artefacts: To be decided.

Output artefacts: To be decided.

8. System Acceptance Test

The system is validated with respect to the stakeholder requirements that are captured during the Requirements activity.

Input artefacts: To be decided.

Output artefacts: To be decided.

9. Product Delta

To be decided.

Input artefacts: To be decided.

Output artefacts: To be decided.

10. Impact Analysis

Change proposals are evaluated, the impact is determined and the proper actions are defined and planned.

Input artefacts: To be decided.

Output artefacts: To be decided.

11. Risk Management

To be decided.

In the next paragraph these activities that are defined during the Crystal project are described in more detail.

3.2 Detailed Activity Description

From the high level overview described in 3.1 we selected 4 activities, in this section 3.2 these activities are investigated in more detail. Following activities were selected:

- Impact Analysis.
- Requirements.
- Components Engineering.
- System Engineering.

3.2.1 Activity: Impact Analysis

In Figure 3-3, the Impact analysis process is shown. Typically, Impact analysis is started based on a change request, unexpected event or a setback in the intended development approach. The first step is to analyse whether the change has impact on the functions of the system and the corresponding architecture. For this, the current preferred architecture is used as a reference. The impact on function level may be none.

In an advanced approach, the relationships between functions or components are laid down in a design structure matrix which plots functions against functions with an indication of the strength of the interaction between the functions. This can also be done at a component level but will be very elaborate. The design structure matrix can help to quickly oversee impact of changes and is useful for all team members to understand system coherence.

If the complexity and impact of the change is hard to assess and thus unclear, a high level simulation with an updated model can be applied to analyse performance in the new situation and to decide in what way to change functions or preferred architecture. If the complexity is manageable at first analysis, such a simulation is not necessary. The new preferred architecture needs have good traceability to the current version of the requirements.

In a next step, the changes to the preferred architecture and the impact of the change on the various components is analysed based on the component archive of current versions of the components. Needed modifications to the components are described as new sprints.

Finally, the change may also have an impact on the installed base, e.g. if a new failure mechanism is detected. In this step, activities are defined to update the installed base, where necessary.

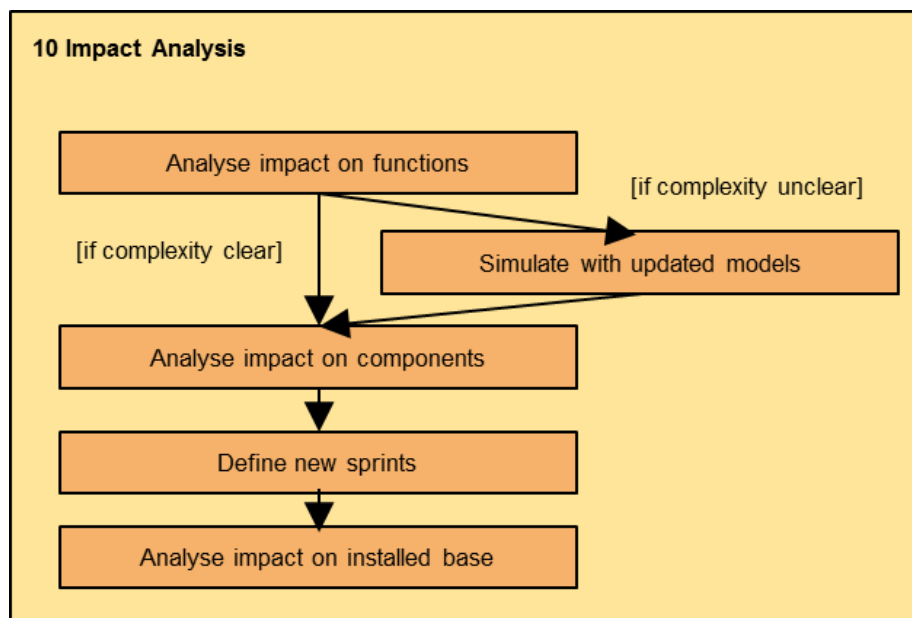


Figure 3-3: Impact Analysis.

3.2.2 Activity: Requirements

Figure 3-4 shows a detail of the Requirements Activity. The Requirements activity is split into 2 parallel activities: Create Legal Requirements and Create Functional/Non-Functional Requirements. The Legal Requirements activity is mainly involved with research into applicable regulations. The Functional/Non-Functional Requirements Activity is a iterative process that involves inputs from customers and market evolutions. As such, they are continuously updated, even during development of the product.

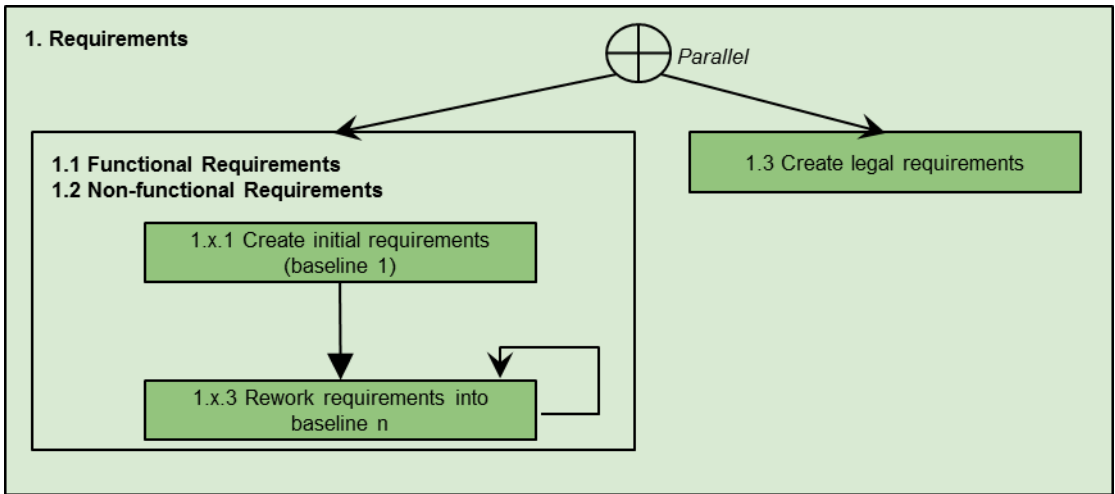


Figure 3-4: Requirements Process Activities.

3.2.3 Activity: Components Engineering

Figure 3-5 shows some details of step 5, i.e., it describes how components are created. The product is divided into constituent components, each with their own requirements. The development process is focused on developing these individual components. Requirements for these components are continuously updated (Backlog refinement), while in parallel, requirements for complete components are being developed (Sprint).

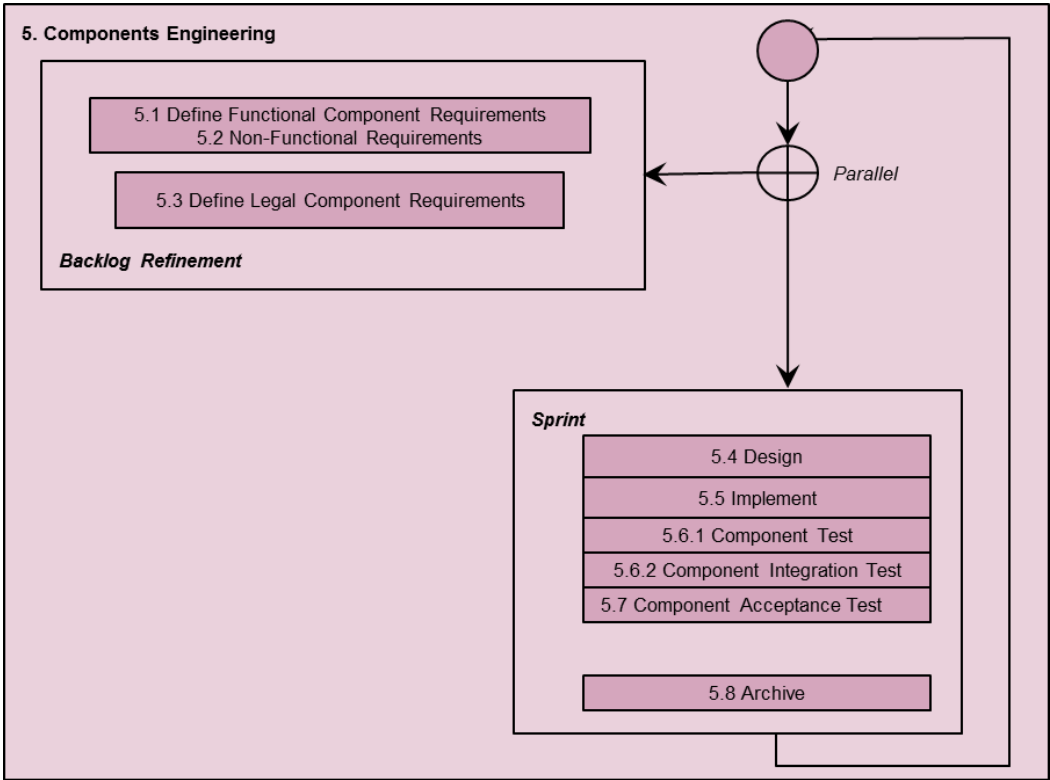


Figure 3-5: Components Engineering Process Activities.

3.2.4 Activity: System Engineering

Figure 3-6 gives an overview of the process flow & iterations for the System Engineering. The goal of this activity is to define the executable system architecture.

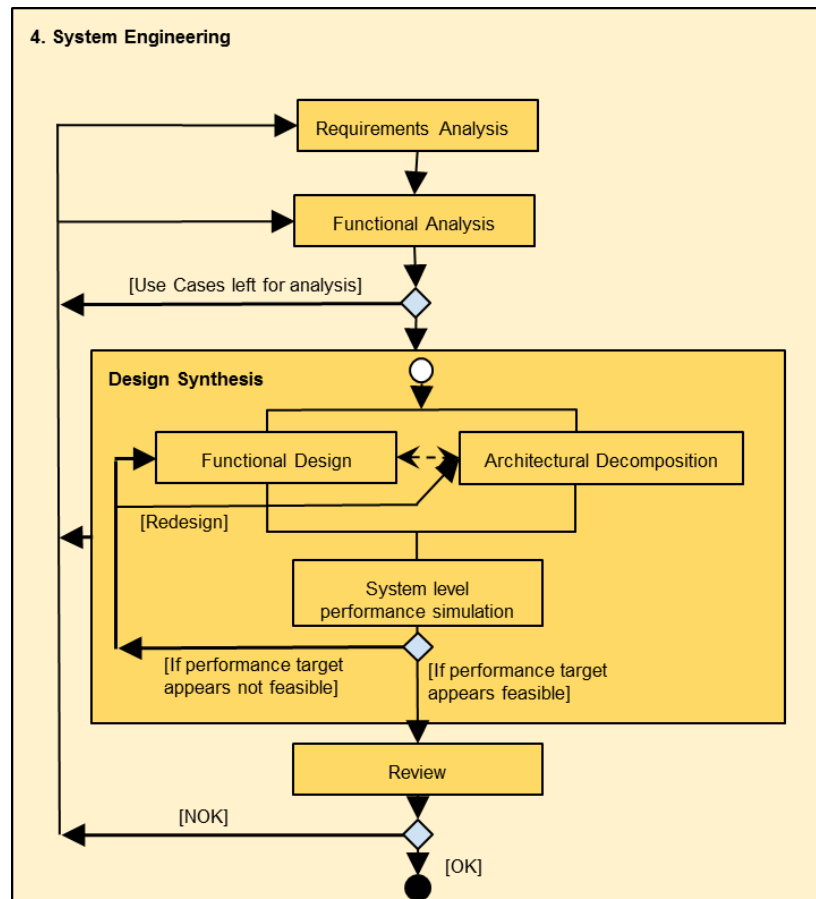


Figure 3-6: System Engineering.

Table 3-1 show an overview of all process steps for the System Engineering Activity and the according input and output.

Activity	Input	Output	Tool Requirements
Requirements Analysis	Stakeholder Requirements	System Requirements System Use Cases System Requirements to Stakeholder Requirements traceability System Use Case to System Requirements traceability	Requirement Management Creating Requirements baselines Use Case Modeling Traceability between Use Cases, System Requirements and Stakeholder Requirements Document Generation

Activity	Input	Output	Tool Requirements
Functional Analysis	System Use Cases	Top Level Executable Functional Model.	Architectural modeling (functional) Architectural simulation (functional, discrete as well as continuous) Traceability between model artefacts to System use Cases Model Formal Verification Model Baselining Document Generation Model reuse Model Consistency Checking
Design Synthesis	System Requirements System Use Cases Top Level Executable Functional Model	Executable System Architecture: <ul style="list-style-type: none"> System Use Cases Architectural Decomposition Component Descriptions Executable Functional Models Architecture to System Requirements traceability	Architectural modeling Architectural simulation (discrete as well as continuous) Traceability between model artefacts and to Requirements Performance Modeling and Simulation Model Formal Verification Architectural Base lining Document Generation Model and component reuse Model Consistency Checking
Functional Design	System Use Cases	Decomposed Executable Functional Model	Model and component reuse Model Consistency Checking
Architectural Decomposition	System Requirements System Use Cases	Architectural Decomposition Component Descriptions	Subsystem Requirements engineering Model and component reuse Model Consistency Checking
System Level Performance Simulation	System Requirements System Use Cases Executable System Architecture	Executable Performance Model Trade off analysis report Performance Analysis Report	Model Consistency Checking

Activity	Input	Output	Tool Requirements
Review	System Requirements Executable System Architecture: <ul style="list-style-type: none">• System Use Cases• Architectural Decomposition• Component Descriptions• Executable Functional Models• Executable Performance Model Architecture to System Requirements traceability	Review comments Review decisions	Review Workflow specification Review commenting on Requirements and Architecture Traceability with specified Architecture Baseline

Table 3-1: Input and output of the System Engineering Activity.

3.3 Conclusion

Based on the Barco Roadmap (section 4.0) three implementation tracks were selected to introduce new engineering methods, tools and methodologies for a more software centric and modular approach to the:

- Track 1: New SW design process.
- Track 2: New Hybrid SW Display Platform (F100) design process.
- Track 3: Flexible SW Centric Design Process.

All details about the 3 implementation tracks and their link to the Barco roadmap are described in section 4.

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.

Our activities are focusing on those Engineering Methods that are affected by the integration of a new tool, those that provide some interoperability requirements, as well as those contributing to the envisioned desired engineering processes supporting a component-oriented & modular design approach and a software centric display platform.

Section 4.0 is giving more background information for the selected engineering methods and methodologies in CRYSTAL and how these are directly linked to the roadmap of the Barco product platforms and the design processes for 3 new platforms. For each new product platform we installed an implementation track to change the design process by introducing, implementing and evaluating new engineering methods, tools and methodologies,

More details about all Tracks and according engineering methods are described in the following sections 4.1, 4.2 and 4.3.

Section 4.4 is giving an overview of the identified Technical Core Requirements as input to the WP6 activities.

4.0 Barco Roadmap and the selected CRYSTAL activities

Barco Roadmap

The Barco roadmap is to move from custom hardware centric display platforms to flexible software centric display platform using commercial-off-the-Shelf (COTS) components in a fully IEC 62304 compliant design environment is directly linked with the Barco goals and Use Case Description in CRYSTAL.

To support this transformation Barco is participating to the CIRRUS and CRYSTAL projects. With the **CIRRUS project** Barco started the research work towards software centric based display platform for diagnostic and clinical review applications, these activities are part of the CIRRUS project, as approved by the Flemish Public Authority IWT (CIRRUS project IWT 120494) and running from 01/07/2012 till 31/12/2014 on national level. The research activities will result in a **first proof-of-concept display platform for diagnostic and review** applications based on software components using COTS hardware. After this first working prototype will be available, a significant amount of **evaluation and post research activities** will be needed before the results of the project can be commercialized. Development effort will be needed on the key components, the integration electronics and mechanics and the EMC validation of the display system. Furthermore extensive validation testing, several pilot series and regulatory approvals will need to be handled as well before commercialization can start of the software centric platform.

As planned on the Barco Roadmap a **first platform using software components running on COTS hardware is planned for the Point-of-care market**. The point-of-care market is a relatively new market for Barco. Initially these products were focused on patient services with a low complexity level, the future trend

however is to offer services for all stakeholders in the patient care chain (patient, nurses, doctors) at a higher complexity level resulting in the need for smarter products.

New engineering methods are first implemented in a pure software design process, this is directly linked to the development of our **next gen QA Web software technologies**, these are quality assurance software services bundled with Barco Displays, this is the first implementation track for the Barco CRYSTAL activities: **Track 1: New Software Design Process**. This track will give further input to start and optimize the design processes for the first hybrid software display platform and the software centric COTS display platforms.

The **FUN100** is the new more software oriented platform using one hybrid software source base. The design process for this new product platform is the second implementation track for our CRYSTAL activities: **Track 2: Design Process for FUN100 Platform**. The results of this track will also be used to start and optimize the design process for the software centric COTS display platforms.

The resulting tools and methodologies used in the new software design process and hybrid software FUN100 process will form the first solid base for **Track 3: The Design process for the flexible software centric display platform**. Barco is collaborating with IBM, TNO and TUE to investigate and evaluate new functional modeling and performance simulation techniques to support the hardware selection and design techniques for the software centric display platform are investigated.

Figure 4-1 below depicts the Barco roadmap and the link between the planned CRYSTAL activities to change the Barco design processes for the new product platforms.

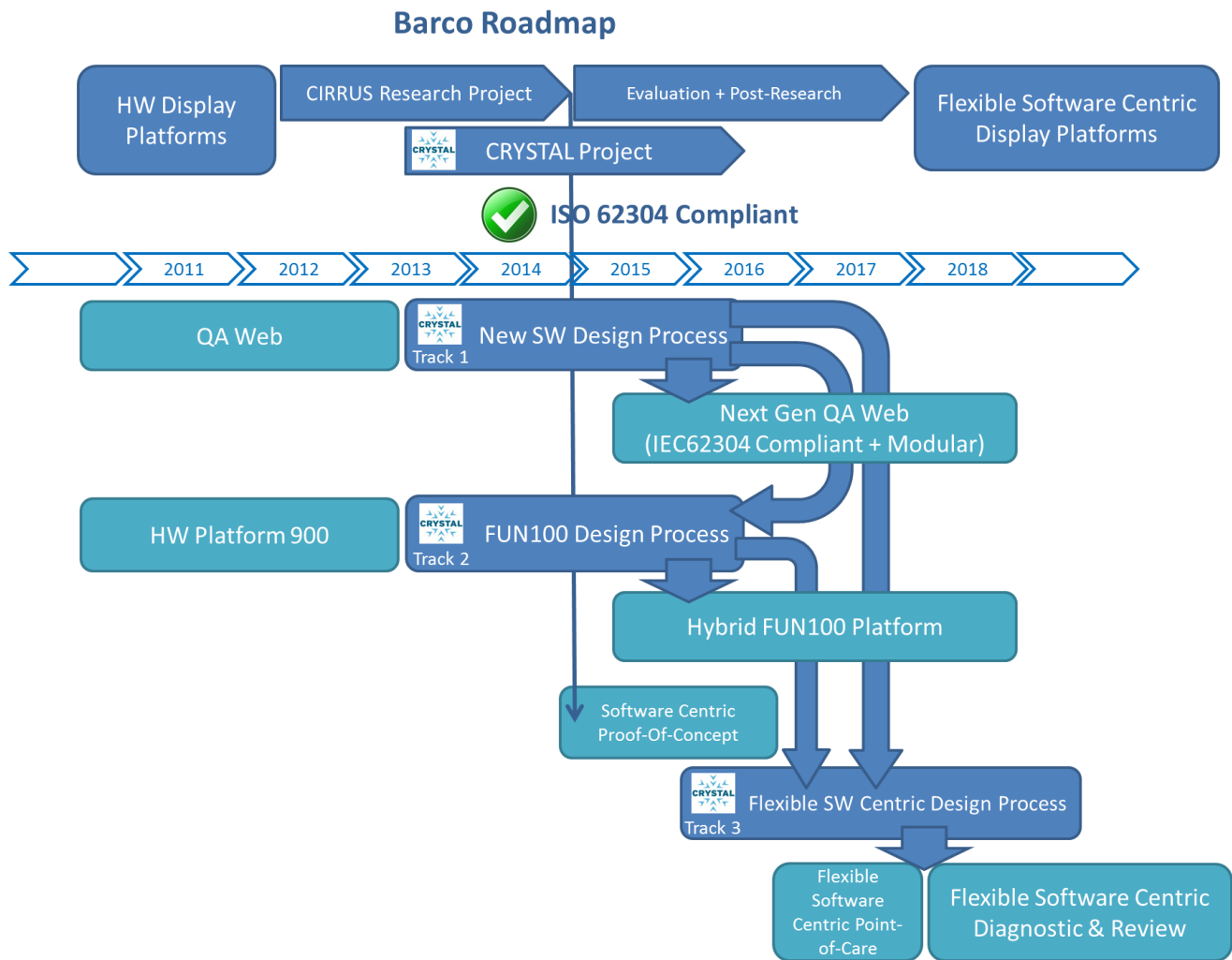


Figure 4-1: Link between Barco roadmap and selected CRYSTAL activities.

Track 1: New Software Design Process

With the introduction of Picture archiving and communication system (PACS) came the need for consistent calibration and quality control for medical displays. Medical QA guidelines like AAPM TG18 and DIN6868-57 were established to quantify and control the quality of medical displays.

The Barco QAWeb service is an all-inclusive secured system is your guarantee for consistent image quality and uptime of all PACS display systems throughout the facility. The system is compatible with Barco diagnostic and clinical displays in addition to non-Barco displays.

The Barco Softcopy QA tools are partly bundled with the display and graphic boards: they allow calibration and control of the graphic board and display so the systems are compliant to DICOM GSDF.

In the context of CRYSTAL Barco started to install a new software design process to ensure the fully IEC 62304 compliancy of the next gen QA Web platform. For this new platform we also want to install a component oriented process, the first concepts have been defined; the more detailed modular design steps will be further worked out in next years of the CRYSTAL project.

More details on the progress of the new software design process are described in section 4.0 of this document.

Track 2: Design Process for FUN100 Platform

For the diagnostic and clinical review applications the **Platform 900** is the hardware based product platform of the current display portfolio for these markets.

A first step towards a more flexible and software oriented approach has been taken with the start of the design process for the FUN100 platform.

The FUN100 platform is a hybrid hardware and software platform designed to drive and control medical displays supporting a component-oriented & modular design approach. The FUN100 Platform defines a family of image processing platforms using one software source base which includes both embedded software and VHDL. The same source based is built into multiple deliverables / install packages which typical depend on the interface board and/or the medical displays it is intended for.

In the context of CRYSTAL Barco worked on the following activities:

- A new methodology for the FUN100 architecture supporting a modular approach.
- Test framework for FUN100 (IEC 62304 compliant) by implementing the new engineering methods Component Integration Testing and Software Unit Testing.

More details on the design process for the Hybrid Modular FUN100 platform are described in section 5.4 of this document.

Track 3: Design process for the Flexible Software Centric Display platform

The tools, methodologies and the according engineering methods as implemented in Track 1 and 2 will form the base of the planned CRYSTAL activities of Track 3, the new design process for the flexible software centric display platform.

To optimize the hardware selection and the modular software design approach for this first software centric point of care medical display platform Barco started collaborating with the CRYSTAL partners IBM, TNO and TUE to investigate and evaluate engineering methods that are not part of the key expertise of Barco.

Together with the research partner TNO and tool partner IBM we are investigating functional modelling, performance simulation and the combination of these 2 engineering methods, these will assist Barco in the support change impact analysis, architectural trade-off analysis and hardware mapping.

The objective of TUE is to predict execution performance of a processing pipeline (gStreamer) more accurately, as part of the intended workflow.

IBM is providing the necessary tool support for these new modeling and simulation techniques.

A pilot project was started at Barco Healthcare with PTC Integrity, other Barco divisions e.g. Barco avionics are using IBM Doors a requirement tool.

As mentioned in the implementation activities of Track 6.1 we noticed that requirements in Integrity are high-level and broad, as part of CRYSTAL we will investigate and evaluate the new IBM tools for the design process on the development side.

More details on the new Functional Modeling & Performance Simulation Framework are described in section 4.3 of this document.

Overview of the Engineering Methods

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	21 of 75

Table 4-1 provides an overview of all engineering methods identified at M12 and their according status. Please note that future planned activities are without engineering method ID and some are not yet assigned to an implementation track.

Track	EM ID	EM Description	Status
Track 1	UC_405_1	Requirements Gathering	First implementation phase
	UC_405_2	Requirements Traceability	First implementation phase
	UC_405_3	Iterative Development	First implementation phase
	UC_405_4	Process Automation	First implementation phase
	UC_405_5	Key Quality Metrics	First implementation phase
	-	Test Automation	Future work
Track 2	UC_405_6	Software unit testing	Implemented for F100
	UC_405_7	Component integration testing	Implemented for F100
	UC_405_8	Architectural Design	Implemented for F100
	UC_405_9	Software Engineering	Implemented for F100
	-	Software unit testing	Future work for U100
	-	Component integration testing	Future work for U100
	-	Architectural Design	Future work for U100
	-	Software Engineering	Future work for U100
Track 3	UC_405_10	Functional modeling	First implementation phase
	UC_405_11	Performance modeling	First implementation phase
	UC_405_12	Functional modeling & Performance modeling	First implementation phase
	UC_405_13	Modular Software Components	Work in progress
-	-	Architectural trade-off analysis	Future work
-	-	Formal Verification	Future work
-	-	Electronics Engineering	Future work
-	-	Mechanical Engineering	Future work
-	-	Automatic Documentation Generation	Future work
-	-	Automatic regulatory compliance testing	Future work
-	-	Not identified Engineering Methods	Future work

Table 4-1: Identified Engineering Methods and Status at t M12.

4.1 Track 1: New Software Design Process

4.1.1 Introduction

At Barco an initiative was started to research and prototype a tool chain for a new software development process with the following challenges.

Challenges

- Formalise and streamline requirements gathering.
- Trace requirements throughout the development process.
- Support iterative development.

- Automate processes as much as possible.
- Measure and visualise key quality metrics.

Following Engineering Methods were identified for Track 1:

- UC_405_1 - Requirements Gathering.
- UC_405_2 - Requirements Traceability.
- UC_405_3 - Iterative Development.
- UC_405_4 - Process Automation.
- UC_405_5 - Key Quality Metrics.

The sections below describe all selected Track 1 Engineering Methods in more detail.

4.1.2 Engineering Method: Requirements Gathering

Up until now requirements were gathered in Microsoft Word and Microsoft Excel documents. Although these documents were based on templates, there was a lot of freedom on how to describe the requirements. Approval of documents was done by emailing the document and pasting an image of a signature into the document. After approval the documents were stored in a central file location.

In order to start agile development, requirements were copied and pasted into the backlog of Atlassian Jira Agile. Changes to requirements involved a labour-intensive manual tracing job.

Although this way of working is fine, it is far from ideal and certainly not easy to fit into IEC 62304 requirements.

It was clear that we needed a more formal way of working and a more 'intelligent' tool than Microsoft Office.

The tool should:

- Be able to import Office documents
- Allow automatic numbering of requirements
- Integrate seamlessly with our agile and testing tools
- Have an integration interface and a customisable workflow.

4.1.3 Engineering Method: Requirements Traceability

One of the requirements of IEC 62304 is that requirements must be traceable throughout the development process. In short, this means that given a requirement it must be possible to indicate which artefacts have been created to implement and to test that requirement.

More specific this means that it must be possible to trace requirements from the requirement tool to the agile backlog, the user stories, the design documents, the source code, the unit tests, the integration tests and finally to the acceptance tests. Moreover, the results of acceptance tests must be visible on the requirement level. Bugs that are detected (either during development or in the field) should, when applicable, be linked to requirements and acceptance tests.

4.1.4 Engineering Method: Iterative Development

The tool chain must support iterative development. This means that requirements do not have to be complete before development can start. However, only approved requirements can be handled in development. Requirements themselves can also evolve over time: they can be fine-tuned, rephrased or even removed.

Changes to requirements should be easily traceable through the whole development chain, which should help us with impact and risk analysis.

4.1.5 Engineering Method: Process Automation

In order for our processes to be repeatable and as error-free as possible and to avoid increased workload and human error, we should strive for as much automation as possible. Results of tests and builds, changes to requirements and lifecycle status of the requirements should be automatically notified to the required tools and stakeholders.

4.1.6 Engineering Method: Key Quality Metrics

According to IEC 62304, Medical devices should be under risk management. Risks should be listed and a mitigation plan should be in place. One of the major risks in product development is bugs. Bugs come in many flavours: memory leaks, performance drops, uninitialized parameters, dead- and live-locks, overflow errors, out-of-memory or disk space errors, logic errors, and so on.

Therefore, a number of metrics will be defined that will:

- Measure the product quality.
- Fail a build when certain criteria are not met.

These metrics should be made highly visible through a dashboard and should continuously indicate the health of the product under development.

4.2 Track 2: FUN100 Design Process

4.2.1 Introduction to FUN100

The FUN100 platform is a hardware and software platform designed to drive and control medical displays supporting a component-oriented & modular design approach.

The FUN100 Platform defines a family of image processing platforms. Each platform is targeted for a given range of Healthcare imaging displays. The platforms share a common architecture, but the implementation is scaled to fit the corresponding display range requirements:

- Platform F100 targets our dual-head high-end displays: Fusion.
- Platform U100 targets our new high-end displays: Unity.
- Platform N100 targets our low-cost single-head displays: Nio.

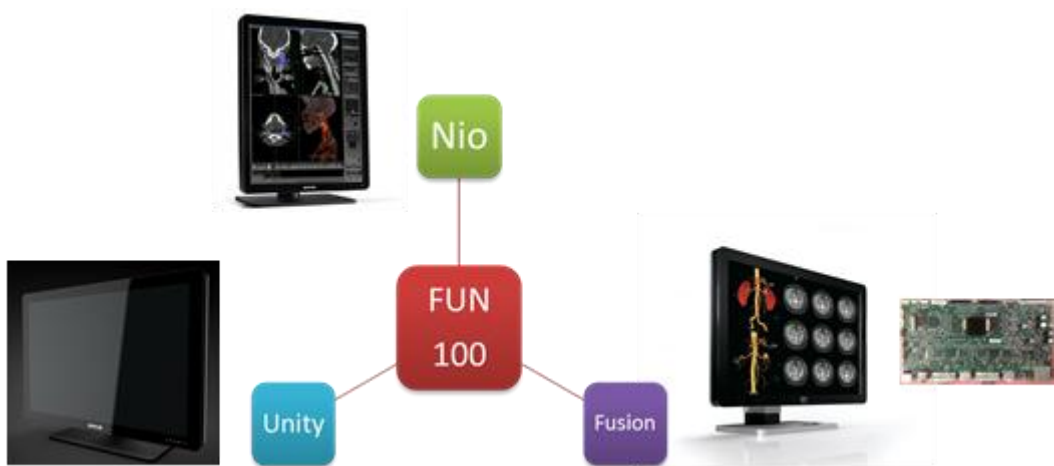


Figure 4-2: Modular FUN100 Platform.

The FUN100 platform is using a common hardware design implemented into different electronic interface boards intended to control medical displays. Currently there are two electronic interface boards defined:

- The F100 interface board, designed for the Fusion type displays.
- The U100 interface board, designed for the Unity type displays.

The FUN100 platform is using one Hybrid software source base which includes both embedded software and VHDL. The same source based is built into multiple deliverables / install packages which typical depend on the interface board and/or the medical displays it is intended for.

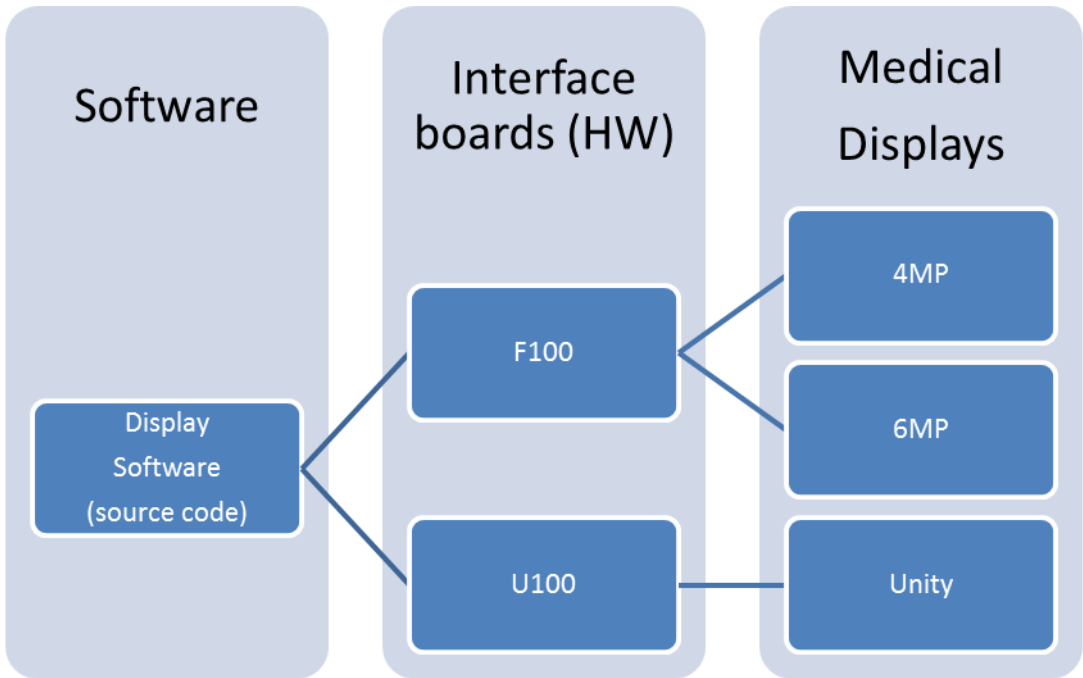


Figure 4-3: One Hybrid software base for FUN100.

Challenges

- Low cost.
- A very tight time to market schedule needs to be met.

- High data throughput.
 - Each of the displays pushes the actual data throughput to the limit of the platform.
 - Biggest data throughput is required on the U100 platform.
- We have to comply with the new standard regarding software development: IEC-62304.
- IEC-62304 compliant Test Framework.
- Hybrid SW development.
- Modular HW architecture design

Identified Engineering Methods for Track 2

Following Engineering Methods were identified for Track 2:

- UC_405_6 – Software Unit Testing for FUN100.
- UC_405_7 – Component Testing for FUN100.
- UC_405_8 – Architectural Design for FUN100.
- UC_405_9 – Software Engineering for FUN100.

We sections below describe all these selected Track 2 Engineering Methods in more detail.

4.2.2 IEC-62304 compliant Test Framework for FUN100

Rationale

1. Required for medical applications: IEC-63204.
2. Growing need due to increase of complexity.
3. To find bugs earlier, shorter time to market.

Goal

In the FUN100 Testing Framework software testing will be tackled on three levels:

- Unit testing performed by software development team.
- Component integration testing performed by software development team.
- System testing by the validation team.

Desired Test Framework

Figure 4-4 below is giving an overview of the desired test for the FUN100 platform, compliant with IEC-62304.

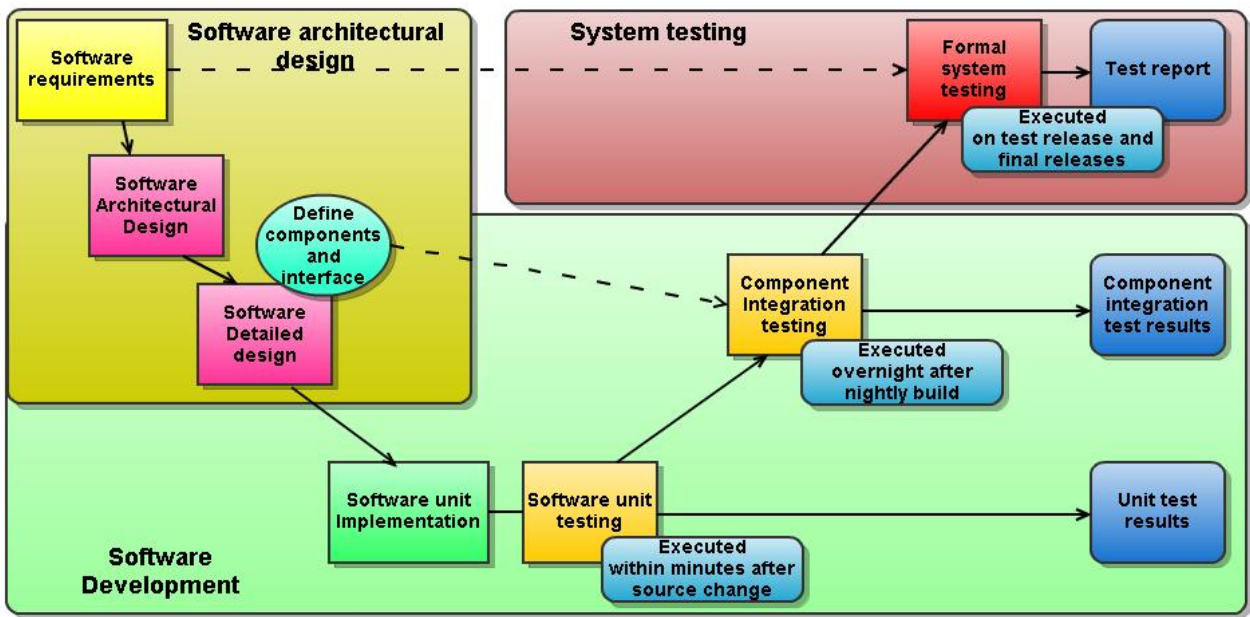


Figure 4-4: Desired FUN100 Test Framework.

Challenge

The most challenging of these three is the component integration testing which exists at a level between unit testing and system validation but has certain overlap with both areas. Where unit testing is typically white-box and system testing is typically black box the component integration is something which lives between both.

Requirements related to IEC62304:

- Trace requirements into architectural design and component test.
- Testing of SOUP (Software Of Unknown Provenance).
- Testing at unit and component level.

Requirements related to obtain a shorter development cycle from software implementation to software release:

- Automation of testing
- Earlier detection of issues:
 - Do earlier testing by having unit and component level testing.
 - Monitoring system parameters to be able to trace when a system parameter started deviating.
- Continuous integration: Make implemented code immediately available and have it tested as soon as possible

Requirements for the (component) testing framework

- The framework should be (at least partially) reusable for other means (system validation).
- Testing should be as automated as possible reducing regression/validation testing cycle.
- It should allow testing interfaces of components.
- As everything it should be lightweight, easy to master, extensible with a minimum of overhead.
- All testing should be integrated, and the execution of tests (and thus also the gathering of reports) should be triggered from a single place.
- Scalability: It should be possible to run multiple instances in parallel (e.g. multiple test stations).

- It should also be possible to run individual tests/test cases.

We started implementing 2 engineering methods to implement the new Test Framework:

- Software Unit Testing for FUN100
- Component Testing for FUN100

4.2.3 Engineering Method: Software Unit Testing for FUN100

Goal

Unit testing is done at source code level. It tests whether the source code is implemented as is expected. Unit testing is the type of testing that is closest to the actual implementation (coding) of the software and has therefore the potential to do the most detailed form of testing.

Characteristics

- Automated: Automatically ran by Jenkins after a source change.
- Fast response: Executed within minutes after a change.
- Non-embedded: Runs on the build host or developers workstation, using hardware abstraction and mocks to simulate the hardware and other components.
- Test a class/source of an embedded application.

4.2.4 Engineering Method: Component Testing for FUN100

Goal

The primary goal of the component testing is to test the individual software components of the software and the interfaces between it.

The component tests will test the FUN100 platform software which is running on actual hardware. Since the support has to support multiple interface boards and displays the component tests will need to be executed against multiple hardware configurations.

The component tests serve three purposes:

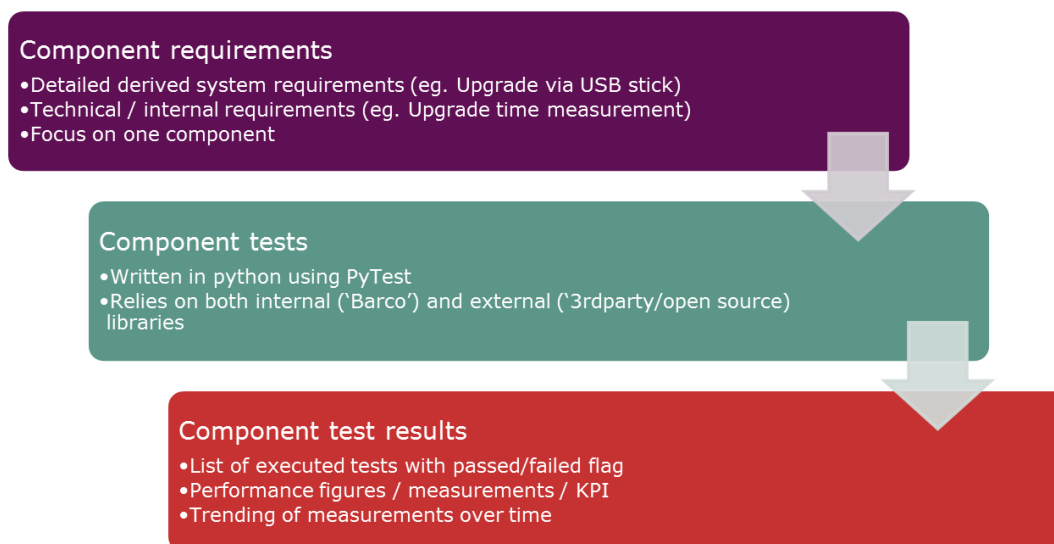
1. Maintaining stability during software development: For this the component tests are run automatically every night on several units upgraded to the latest software. The goal here is to detect defects as soon as possible after they were introduced due to software development
2. Serve as a stage gate towards the system testing. Component tests are to be run on every test release before these are delivered to the system test team. This is done on a representative hardware configuration set.
3. Monitor important system parameters during software development. The component tests keep track of important system parameters across multiple tests runs.

Characteristics

- Automated: Automatically ran by Jenkins every night.
- Medium response: Executed within hours after a change.
- Embedded: Run on the actual hardware.
- Tests at component level:
 - Embedded applications, FPGA, Linux kernel.
 - Interaction between components.

- Grouped per component, test can be rerun for a single component.
- Configurable:
 - Fast / Slow tests.
 - Remote / Local tests.
 - Different types of hardware.

Workflow



Workflow Component Testing workflow for FUN100.

The SEE for this new Test Framework is shown in section 5.2.1, the first implementation result can be found in section 6.2 and 6.2.2.

4.2.5 Engineering Method: Architectural Design for FUN100

This section describes in detail the new methodology for the FUN100 architecture supporting a hybrid software base.

Rationale (Pre-CRYSTAL status)

At the end of the development of platform 900, we found that the complexity had increased to a level that made it extremely difficult to maintain. Each problem, in production or in the field, resulted in a complicated investigation during which engineers had to step through the complete design to find the location of the problem. The solution usually resulted in a complicated series of changes, which impacted the total design. In the end, much of the testing and validation had to be redone, because no-one was sure which parts were affected. This was an expensive situation.

The complexity was created by adding functionality on previous platforms without sufficiently questioning the architecture. That resulted in a design which could not be broken down in different parts to isolate problems. Also dedicated software was made for each hardware implementation, so a rather large matrix of versions began to exist.

Both in hardware and software solutions were added which did not really integrate in the architecture, with a lot of overhead as a result. The advantage of this way of working was that a solution was ready for the market in a relatively short time. However, both the direct cost and the maintenance cost increased.

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	29 of 75

There was clearly a need to take the next step: the development of a derived platform, which would basically support the same functionality. But this time with a completely new architecture based on modularity and reuse. It was also the intention that next generation displays could be derived without a lot of effort, and integrating new functionality would be straightforward.

FUN100 Concept

The concept for the new platform was to have no more hardware on the board than strictly needed for the product. This means that more hardware versions would be needed, but each at an optimized size and cost.

The drawback would be that this means more development time. However the modular design meant that a new hardware version was easily derived, and the software was almost instantly ready due to the parameterized modules.

Strong focus on Testability

For this new platform the decision was taken to implement automatic unit testing and component testing, so that debugging in a later stage could be done more efficiently. Tests are available in software, but once the hardware was available, the same tests were also run in the hardware environment. This creates the possibility to isolate bug solving to the module level, and due to the automatic testing, any change is immediately evaluated and ready to be released.

Conclusion

The SEE for this new FUN Architecture methodology is shown in section 5.2.2, the first implementation result can be found in section 6.2.3.

4.2.6 Engineering Method: Software Engineering for FUN100

This section describes in detail the new methodologies and techniques that were started in Barco to implement a modular approach for the software engineering process for the FUN100.

Portability

A model driven design approach

The rationale of a model driven design approach is the reuse of IP (Image Processing) in all healthcare displays, regardless of the megapixel count, FPGA device family and other product specific parameters. At the moment of this writing Barco Healthcare has implemented many real time image processing applications often with low latency requirements, although the investigated high level platform approach is not restricted to this real time aspect and could, for instance, eventually be applied for video file format conversions in storage devices or for pseudo real time quality of service analysis.

Applications include any adaptations or transforms required for any type of video sources to any information carrier as well as conversions required by the destination, for instance the diagnostic display. Examples of video sources include digital endoscopy cameras, graphical cards, streaming video over a networks, video or image file formats. Typical examples of adaptations for such a sources include (de)interlacing, resizing, (de)compression, Chroma up sampling, noise reduction, motion compensation, edge enhancement, sharpening, histogram equalization, automatic contrast control, ...

In diagnostic displays typical examples are colour space conversions, (de)gammatization, sRGB colour profiles, picture uniformity correction, x-talk compensation, frame rate conversion, temporal display response optimization, grey tracking (according to DICOM), perceptual matching, graphical overlay, colour gamut mapping, electro-optical transfer functions, dithering... The information carrier can be any type of memory or storage device such as DDR memory chips or a flash hard disk, any (local) standard bus such as PCIe, a global or local network. Eventually an image processing application prepares and optimizes the video data

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	30 of 75

for a specific destination or sink, which can be a display device, a new IP video stream on a network, a new file...

Barco currently has an elaborated set of image processing modules. Many of the image processing applications such as 3D colour profiles or dithering have been developed for different implementations in different hardware and/or software environments. In order to reuse the existing IP within all implementations based on the FUN platform, the Fusion, Unity and Nio displays, the processing chain needs to be re-shaped according to a common model, with elements implementing a common interface. High level source code should enable the reuse of existing IP without considerable modifications to the source code. The embedded software which communicates with the IP blocks therefore will "see" a common interface for all displays.

Interoperability – need for standard bus interface

The second reason to model the processing chain is the increasing need for interoperability between various product families and thus the increasing need for adherence to standards. A standard bus interface protocol must be used which can be implemented in both Xilinx and Altera devices and communicate with the corresponding "native" IP libraries, such as memory controllers. Therefore the flow control bus protocol must unify the streaming AXI interfaces, common in Xilinx devices, and the streaming Avalon busses (including the burst mode transfers).

The healthcare displays products evolve from simple boxes towards more complex distributed and integrated systems, with an increasing use of off-the-shelf components, usually in software, licensed with commercial licenses or as open source software. The technological convergence and increasing price pressure drives the re-use of components across multiple products, even if they are deployed in different markets with different end-user requirements. This enforces the R&D departments to design their components to be more flexible and generic.

IP Reuse

Barco has multiple proprietary colour processing implementations based on different FPGA families (such as Altera Stratix or Xilinx Virtex) and different CPU based software implementations. The company's subdivision Silex even developed multiple ASIC devices dedicated to colour processing applications, for instance to calibrate the grey tracking of a display (the LUT ASIC). This means many different types of colour processing architectures exist simply because they use different implementation devices. Within the scope of this platform an inventory is created of the existing IP applications. Based on that we can identify what caused the different implementations of the same functionalities to coexist. Identifying the root causes for this coexistence will be a key element to determine the required capabilities of any future high level code to be developed, regardless of the language to be used.

Related to the above study is the identification of the root causes why proprietary applications or processing blocks were developed instead of COTS solutions. If COTS solutions were used, it is equally important to determine what made them "good enough" for a given application. The high level image processing chain should be capable to absorb the above mentioned functionalities, as well as to interface with them by using interfacing methods and architectural description methods to be determined within the current research.

Scalability

Data throughput and latency requirements

When one wants to process large amounts of data efficiently (for instance to handle 12 megapixels in real time with high accuracy of about 64 bit per pixel) and with low latency, one needs to take into account the locality of the images on the host device. In a high demanding application a lot of time is wasted in copying data around. The latency and throughput of the memory transfers become the bottleneck of the application.

To make things even worse the optimal architecture for high end applications with a high throughput demand not only depends on its host device but also for instance on the type of connected DDR devices (DDR2,

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	31 of 75

DDR3, RLDRAM, QDRAM...) as many image processing applications such as resizing require storing the images in some sort of frame memory.

An abstraction layer, build in to the bus communication protocol of the image processing, should accommodate for a generic external image frame store configuration, depending on the latency requirements. Different approaches exist to interconnect the image processing application with the dedicated memory devices. So far, often the image processing application had to be adapted for a specific memory controller type, which makes the usage of a COTS solution (for instance a memory controller offered by the FPGA vendor) difficult.

The current research attempts to resolve this type of issues by describing the image processing application in a generic way that includes a more abstract definition of for instance the memory transfer operations. A simple high level abstract description of the system or the host device environment would enable the reuse of a high level described image processing feature without the usual considerable modifications to the source code.

The framework must support measuring, configuring and controlling the data throughput typically expressed as image frame-rate and at the same time it must support measuring, configuring and controlling the timing behaviour or latency of individual elements and the complete pipeline.

Parallelisms within the IP chain

The architecture must also support the concepts of parallel and serial processing. Further investigation must define how this can be modelled and to what extend the processing can be predicted and what accuracy can be achieved.

An important aspect is the reuse of embedded software drivers for multiple architectures. The software should not "know" the difference between for example a single processing engine running at 800 MHz or four combined processing engines parallelized and running at 200 MHz. The FPGA architecture, and thus all image processing blocks included, must be foreseen for parallel architectures. The goal of the current coding methodology for this platform is to achieve the ultimate scalable solution. In short term, for instance, the IP framework should offer a fairly optimized solution for 3 megapixel displays (Nio), 6 megapixels (Fusion) and 12 megapixels (Unity).

Model the interconnections between IP blocks

A standard image processing block interconnection method has to be developed which has the flexibility to adapt to the above mentioned combination of bottom-up capabilities propagation through the hierarchy and top-down generic configuration of the global application.

Ideally 1 virtual interconnection between any 2 image processing blocks within the chain should be sufficient. Such a "bus" should be a subset of a framework to connect the image processing application with other functionalities, for instance PCIe controller or a LAN interface. Preferably the application's boundaries should interface with other (COTS) blocks in the system by using a COTS protocol such as AXI4 or Streaming Avalon or the high level framework should provide functions to adapt its native interfacing format to any of such a required standard interfaces. Such a conversion feature should also be developed for the internal block interconnections within the application for debugging and validation purposes as this allows the usage of COTS validation tools to be used. We must further investigate how it needs to be modelled, and what is needed to reach the predictability and QoS.

Runtime support for quality of service

A quality of service analyser should be included based on the current environment, input and output format, viewing conditions, quality metrics specific for the application and other aspects, for instance to guarantee DICOM compliancy.

In the example of a colour transform a QoS analyser could report the J-index variation per quantizing interval based on:

- The current image content (multiple aspects).

- The display type (colour gamut, resolution, size,...).
- The QoS could produce a warning message when for instance the image is not guaranteed to be diagnostically lossless in a critical healthcare application (for instance caused by a too low resolution display). It should compare the currently provided quality to a pre-defined critical specification.

In the example of an electro-optical transfer compensation block for a display a QoS analyser could report the percentage of image detail loss based on:

- The optical electric transfer function provided by the current source format such as sRGB, DICOM, Rec.709, Adobe RGB, ...
- Ambient light conditions
- Defined quality metrics for the application (for instance DICOM-based just noticeable difference curves as used in healthcare)
- Here again the QoS analyser could trigger a warning message when some quantizing levels become invisible on the display system.

Highly configurable

Risks and challenges related to highly configurable coding techniques

The intended methodology which will be used to synthesize the IP framework in FPGA raises a couple of questions and challenges:

- Feasibility of the current compilers XST, Quartus and ModelSim: how well do these tools handle highly generic code? How “deep” can generics be propagated through VHDL 2008 designs?
- Development of a library for all required mathematical functions such as Sinc, Fourier transform, logarithms, matrix inversions...: how fast can the above tools compile these functions? What is the optimal real tolerance? Can the tolerance aspect be made compatible with OpenCL?
- Image processing chain modelling to optimize the performance <> resources <> quality <> Latency trade off. Resource balancing (DSP, logic, RAM, IO bandwidth...) should also be configurable.
- How to make a module adapt to its environment without making the code too complex (port standards used...). Increasing the generic level should even increase readability compared to base code.
- From a certain integration level, we need to standardize the interfaces: what to use? A dedicated video bus, Axi4, Streaming Avalon, G-streamer like interface...
- Integrate the embedded software model to obtain a diamond model for the image processing chain. Which libraries are optimal to integrate C-based software and VHDL-based hardware? Do we need to write new libraries for this? Do we need system C to bridge?
- Reuse the same code in multiple products based on the FUN –Platform. Do we need any device specific libraries for optimal implementation towards the target FPGA which is different for the Fusion, the Nio and the Unity?
- A Wrapper file must be created per project to align integration methodology for all projects derived from this platform. How can these different configurations be integrated in a regression testing platform? Is synthesizing an array of all known IP chain instances feasible with the current synthesis/simulation tools?
- How to apply the concept of model-driver design for system validation for the different use cases based on the diamond model (embedded software + FPGA hardware)? Can hardware always be modelled bit-accurately in software with standard libraries?
- Resource optimization (by combining multiple image processing aspects such as colour space conversions and electro-optical transforms) when integrated in the product.

High level functionalities

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	33 of 75

The model of the image processing platform must support the following high level functionalities: quality parameters are often market specific as the requirements for the same application can be very different.

Consider for instance temporal response optimization algorithms: they can be optimized differently for different types of moving images. Algorithms which introduce so called overshoot artefacts can be perfectly acceptable in broadcasting applications as they should not be visible on bandwidth limited video sources, but they can be unacceptable for diagnostic applications as they could lead to false diagnosis.

For example, grey level tracking has to be very accurate in many healthcare applications to ensure all quantizing steps are almost equally visible according to the DICOM standard, while in a digital cinema projector the whole colour gamut is of equal importance according to the DCI specifications. This means the configuration of a high level image processing application should depend on the global application and even the market it is used in. In some markets artefacts should always be minimized as much as possible, while other markets, such as most applications within the healthcare market, prefer to have predictable artefacts even if this would reduce the overall image quality.

In addition to image quality, another important aspect is low latency while having a high throughput. Acceptable latencies differ from the application, and even the type of operation. Some health care applications, such as endoscopic surgery, have very strict low latency requirements.

The framework must support the configuration of the strategy to achieve the image quality and latency requirements. This strategy – or policy – defines how conflicting requirements must be resolved and how to degrade gracefully when not all requirements can be met.

Testability

A common way of verifying an FPGA IP is to make use of a simulation tool, such as ModelSim®, to simulate the behaviour of the IP. In this approach a set of stimuli is defined to drive the device under test (DUT). Next the responses of the DUT are evaluated, either manually or by comparing against some kind of golden model responses. This approach is very useful for functional debugging as well as a potential way of performing component regression testing.

If the IP doesn't pass the functional simulation verifications, then you can be quite certain that it won't work on the actual FPGA hardware implementation. However, we have experienced several times that, even though the IP was functionally verified successfully by the simulation tool chain, the IP wasn't functioning properly in the actual FPGA hardware implementation. This can be caused by a bug in the FPGA vendor tool chain, a functional bug for which the specific use case wasn't covered by the simulation, some kind of start-up phenomena...

The FPGA vendors all provide 'a' way of performing simulations of an FPGA IP with the actual implementation results of the vendor FPGA tool chain. This way most of the errors introduced by some kind of bug within the FPGA vendor tool chain can be detected in simulation. The downside of this approach is that these simulations tend to be quite complex to set up and the resulting verification is still vendor specific.

As a step forward towards a fully validated COTS solution we added the ability to verify the image processing chain on the actual FPGA hardware implementation. This approach has several advantages:

- No need for long simulation runs
- Verifications are done on real hardware using the actual IP eco system
- FPGA vendor independent
- Tests can be implemented using a higher level software language. This tends to require less effort to implement the test.
- The same test can be run on different hardware implementations. For instance on a F100 and a U100 platform.

Conclusion

The first evaluation results for the new software engineering process for FUN100 can be found in section 6.2.4.

4.3 Track 3: Flexible SW Centric Display Design Process

As mentioned in section 4.0 a first product using a platform based on software components in combination with COTS hardware is foreseen for the point-of-care market.

This new platform from the base for the data used in this implementation track, as such the requirements input for the functional modelling and performance simulation, as well as the modular design approach for a video pipeline based on software components is based on the first data that we have available at Barco for this new platform.

4.3.1 Introduction

The purpose of a functional model is to specify the system behaviour in one coherent and consistent executable model to support system validation by simulation. Performance simulation is done to support performance based trade-off analysis and architectural decision making.

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	35 of 75

So far functional simulation and performance simulation are executed as separate engineering activities; the relations between these models are often not exposed. Often the functional and performance models can impact each other tremendously. A change of the functional model (e.g. because of a changing requirement) can have a huge impact on the performance of a system, and the other way around, a performance change (e.g. because of changing hardware) can have important consequences on the desired functionality.

Integrating functional and performance models into one co-simulated system model will expose these relations to support change impact analysis, architectural trade-off analysis and hardware mapping.

These techniques are not yet used in the design processes at Barco, however these could be key supporting engineering methods for the identified architecture trade off and architecture design, as identified in section 4), for the new design process for Tack 4: the flexible software centric display platform.

Selected Engineering Methods for Track 3

Following Engineering Methods were identified for Track 3:

- UC_405_10 – Functional Modeling.
- UC_405_11 – Performance Simulation.
- UC_405_12 – Combining Functional Modeling & Performance Simulation.
- UC_40513 – Modular Software Components for Qt and gStreamer.

4.3.2 Engineering Method: Functional Modeling

Goal

The purpose of Functional Modeling is to create a Functional Model of (a part of) the System. The Functional Model specifies the Discrete Behaviour (using SysML) and the Continuous Behaviour (e.g. with MathWorks MATLAB/Simulink), as well as the interaction with the Actors, in one coherent and consistent model. The Functional Model is executable so it is possible to validate its behaviour. The traceability of the model artefacts to the input artefacts is carefully maintained.

Workflow

This Engineering Method can be applied during the activities Functional Analysis and Design Synthesis as follows:

- During Functional Analysis one top level Functional Model is created for the System as a whole. No subsystems are defined yet. This Functional Model describes the behaviour of the system and the interaction of the system with its Actors. This model is used later on as input for Design Synthesis to be able to map the functions to the subsystems and to specify the interfaces of the subsystems. The Functional Model is executable so it is possible to validate its behaviour with respect to the System Use Cases that are specified during Requirements Analysis.
- During Design Synthesis the scope of the Functional Model is a subsystem or component. For each subsystem (component) a Functional Model is created, specifying the behaviour of the subsystem and the interaction with other subsystems and with system level Actors. The models are used later on by the different engineering disciplines (electronics, mechanics and software) to design that part of the system.

Tool requirements

- Architectural modeling (functional).
- Architectural Simulation (functional, discrete as well as continuous).
- Traceability between model artefacts and to Requirements.
- Model base lining.
- Architectural Base lining.
- Document Generation.

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	36 of 75

- Model and component reuse.
- Model Consistency Checking.

4.3.3 Engineering Method: Performance Simulation

Goal

The purpose of performance simulation is to verify performance requirements of the functional model in combination with the hardware model. The behaviour of the algorithms and the hardware are modelled. The performance simulation gives an early stage insight on the performance of the total system.

Workflow

The functional model defines the order of execution of functional block. The functional blocks representing algorithms describe the data or signal flow of the system. The system architect describes the hardware of the system and makes (possibly multiple) mappings between algorithms and hardware components. One could imagine a system with a quad core CPU and two GPUs. If two in sequence algorithms are used in the system, the system architect can decide where to execute each algorithm.

To simulate the performance two types of models have to be made: a model for the algorithms, and a model for the hardware. In the first simulation model iteration the level of detail of modelling should be low. If models are successfully verified or are within sufficient trust, one could increase the level of detail.

With the models specified, the performance simulation is performed and the performance requirements are checked. The results of the performance simulation are also checked with real life experience of the system engineer. During implementation of the system the models are verified by performing comparable scenarios on the hardware. The similitude of hardware and algorithm models increase iteratively.

Tool Requirements

The performance simulation is performed by an analytical model of the algorithm and the hardware. The models are implemented in the Simulink and Matlab toolset. A detailed description of the intended algorithms as well as the hardware is required.

- Architectural trade-off analysis

Software Engineering

4.3.4 Engineering Method: Combining Functional Modeling & Performance Simulation

Goal

Integrated functional and performance modelling gives rapid feedback on functional architecture decisions and propagates design and performance changes to functional consequences. Usually, integrated functional and performance is performed on a small number of key performance indicators.

This integrated approach can serve as a tool for architectural trade-off analysis and architectural design but may also be used purely to evaluate functional models.

The interaction between the tools is as follows:

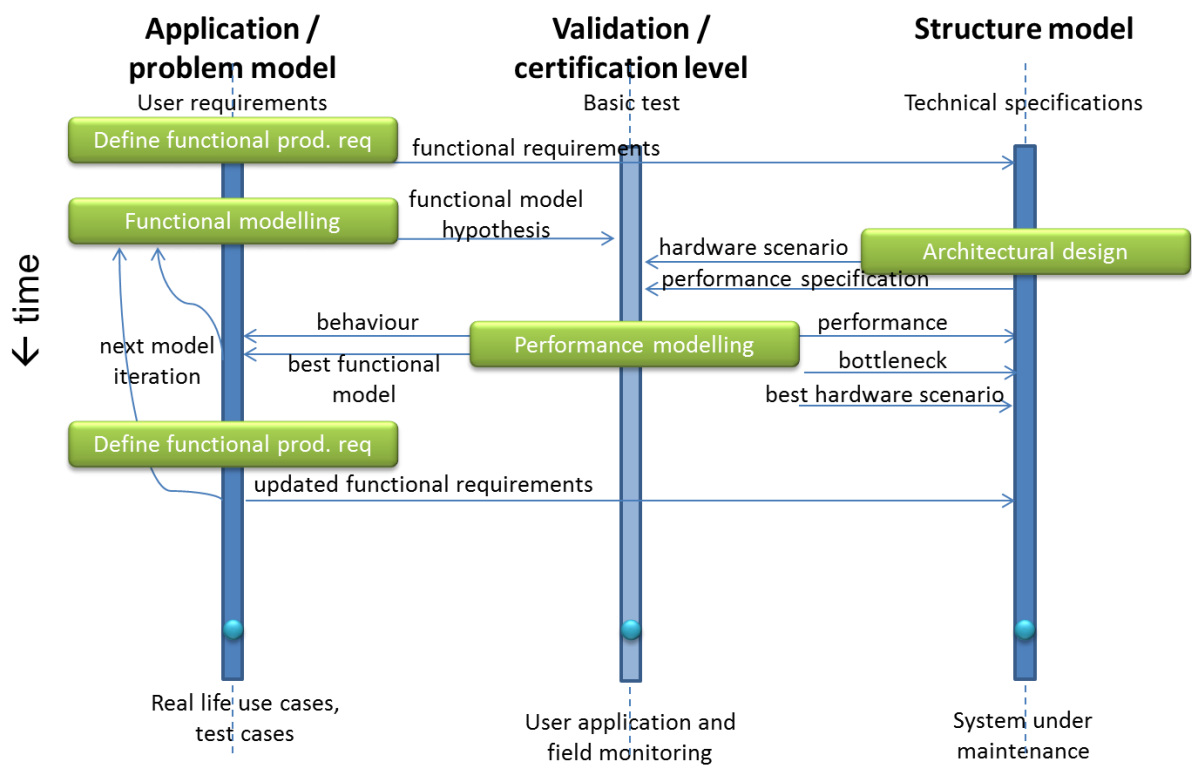


Figure 4-5: Interaction between the tools over time.

4.3.5 Engineering Method: Modular Software Components for QT and gStreamer (Work in Progress)

In order to support modularity in the composition of the video application, Barco has decided to develop a flexible and configurable middleware based on the following public domain tools: the Qt framework (version 5.2) and gStreamer (version 0.10 and 1.0). The key idea behind this framework is that a streaming pipeline of a video application can be decomposed into several imaging components (where a component is typically a complex signal-processing algorithm). These components can then be connected to each other by creating a pipeline. This allows for a reuse of algorithms in different setups with support for a wide range of products. Since the gstreamer framework supports different hardware platforms, different processing pipelines and different levels of parallelism can be supported on a wide range of off-the-shelf platforms, whenever gStreamer and Qt can be deployed on these platforms.

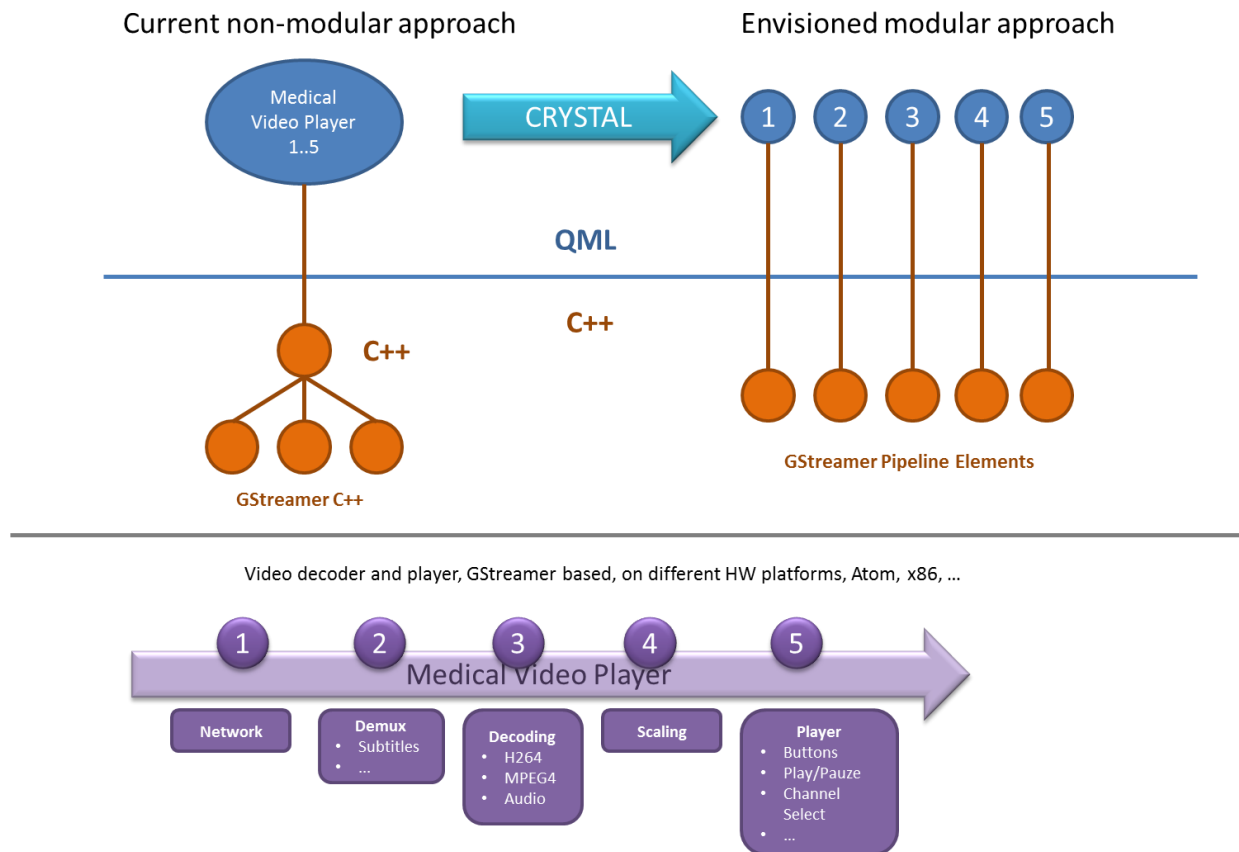


Figure 4-6: Envision modular software approach for the Medical Video Player.

The integration of Qt and gStreamer in the development process brings some new challenges for interoperability. Firstly, we must investigate how the QML scripting language (part of the Qt framework) can be used to flexibly specify the composition of the streaming pipeline. Secondly, we investigate how the specified pipeline can be mapped to the underlying gStreamer elements, for example, using C++ bindings and the corresponding gStreamer API descriptions. This means that signal processing algorithms, being imported from other tools such as Simulink, need to be wrapped in a so-called plugin that implements a proper gStreamer interface. Finally, in order to integrate gStreamer in the development flow (through Qt programming), we need to match the execution model of gStreamer and the selected underlying operating system with the execution model being used at the stage of performance modeling. That is, the execution models of gStreamer and the execution model of the performance model must be aligned in order to predict performance metrics cost-efficiently.

It is expected that interoperability between the models for simulation (e.g., using the Dynaa-tool under maintenance at TNO) and real execution of components brings the following benefits. Firstly, we can profile and test components in isolation. The gStreamer framework comes with various tools and methods to assess the performance of an application or a component running on real hardware. Subsequently, we can fill in these profiling results as properties of the corresponding component in order to tighten the application's analysis. Hence, the simulations can be repeated and various allocation strategies of system resources (such as memory and processor time) can be explored in simulation, so that extra-functional application requirements are met. In this way, the gStreamer framework can be used to compose image-pipeline applications on programmable off-the-shelf hardware platforms in a modular way.

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_0020	Enable early functional evaluation of holistic system behaviour	Must	Enable early prediction of system behaviour and the impact of decisions by simulating the impact of these decisions to the functional behaviour of the system under development. This is to be achieved by holistic simulations that integrate all relevant functional behaviour in one scenario.	TECH_CORE_REQ_0028 TECH_CORE_REQ_0041 TECH_CORE_REQ_0042 TECH_CORE_REQ_0044
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 they need to interface with.	
TECH_CORE_REQ_0022	Provide IOS interface for embedded tool environment	Must	IOS for the embedded tool environment.	
TECH_CORE_REQ_0023	The IOS shall support variability/variant management.	Should	Variability information must be communicated between different tools.	
TECH_CORE_REQ_0030	The model of the system shall be written in a high level modelling description language	Must	Use high level modelling description language to model the system	
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_0042	System architect wishes to rapidly and easily edit models in	Should	The system architect goes through many iterations of	

	case of changes in requirements, architecture or design.		functional and performance modelling. He should not be discouraged by his tools.	
TECH_CORE_REQ_0043	Software architect wishes early validation of executable functional models by formal analysis to proof absence of deadlocks and unwanted parallel states.	Should	By performing a formal analysis of an executable version of the functional model (a state machine), the software architect can proof that the functional model does not lead to unstable situations like deadlocks or unwanted parallel states.	
TECH_CORE_REQ_0044	System architect wishes to analyse system performance using performance simulation.	Must	The system architect needs to decide on high level function allocation (or 'mapping'). To estimate performance, he needs simple to use performance models of the key performance criteria. Performance models may be based on previous projects, which enhances the validity of the simulation, or can be built from scratch.	
TECH_CORE_REQ_0045	Functional analysis of executable models using interactive simulation. System architect wishes to perform functional analysis of a system using interactive functional simulation	Should	Simulate executable models in order to analyse the functionality of the modelled requirements, architecture or design. The functionality of a modelled system and its internal workings can be analysed by interactively simulating some (typical and atypical) interactions.	
TECH_CORE_REQ_0046	Synchronized simulation of executable models running in different tools	Must	Combine executable models of different system parts (such as hardware and software) that are simulated using different tools. Different system aspects are often	TECH_CORE_REQ_0066

			modelled in different specialized tools. To analyse the overall system behaviour using co-simulation, these tools need to synchronize their clocks and events.	
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_0101	Metadata of simulation models.	Should	IOS should provide means to assess metadata of simulation models such as inputs, outputs, model elements and validation level. For reuse of simulation models the characterization and quality of the model should be available.	TECH_CORE_REQ_102
TECH_CORE_REQ_0103	System architect wants to perform joint functional and performance modelling.	Must	The system architect wants to obtain feedback on system performance while exploring various functional models for a system.	
TECH_CORE_REQ_0104	Cross-tool document generation.	Must	Template based generation of certification documentation based on data that is available in different tools is important.	TECH_CORE_REQ_0050
TECH_CORE_REQ_0105	Model reuse.	Should	It shall be possible to build a library to reuse models.	
TECH_CORE_REQ_0106	Model Consistency Checking.	Should	It shall be possible to check the consistency of the models based on modifiable consistency schemas.	

TECH_CORE_REQ_0107	Model versioning and base lining.	Must	It shall be possible to define different versions of a model and create baselines in time.	
TECH_CORE_REQ_0108	Model traceability.	Must	It shall be possible to trace model elements to requirements and test cases.	

The first implementation works have started for these selected Tracks and Engineering Methods, the status of the installed SEE is shown in section 5.1, the implementation results are described in section 6.

5 System Engineering Environment

Section 5.0 will first give a high level overview of the System Engineering Environment and the position of all our CRYSTAL activities on this overview. The following sections (5.1, 5.2 and 5.3) show the SEE and the selected tools for each Track in more detail. In section 5.4 we give a picture of the envisioned SEE based on the activities that have been planned so far.

5.0 Introduction: High level overview of the WP404-405 SEE at M12

Figure 5-1 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.

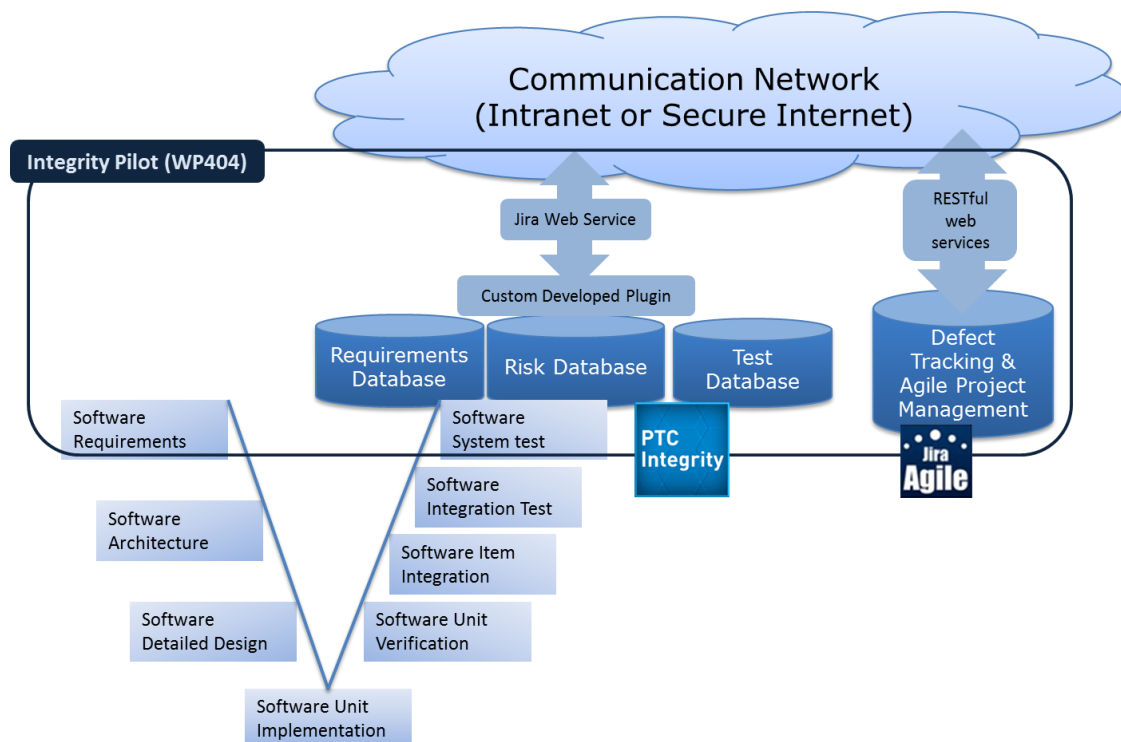


Figure 5-1: Positioning the WP404 Pilot project on the high level SEE overview.

Figure 5-2 is giving an overview of all WP405 activities on a high level overview of the SEE.

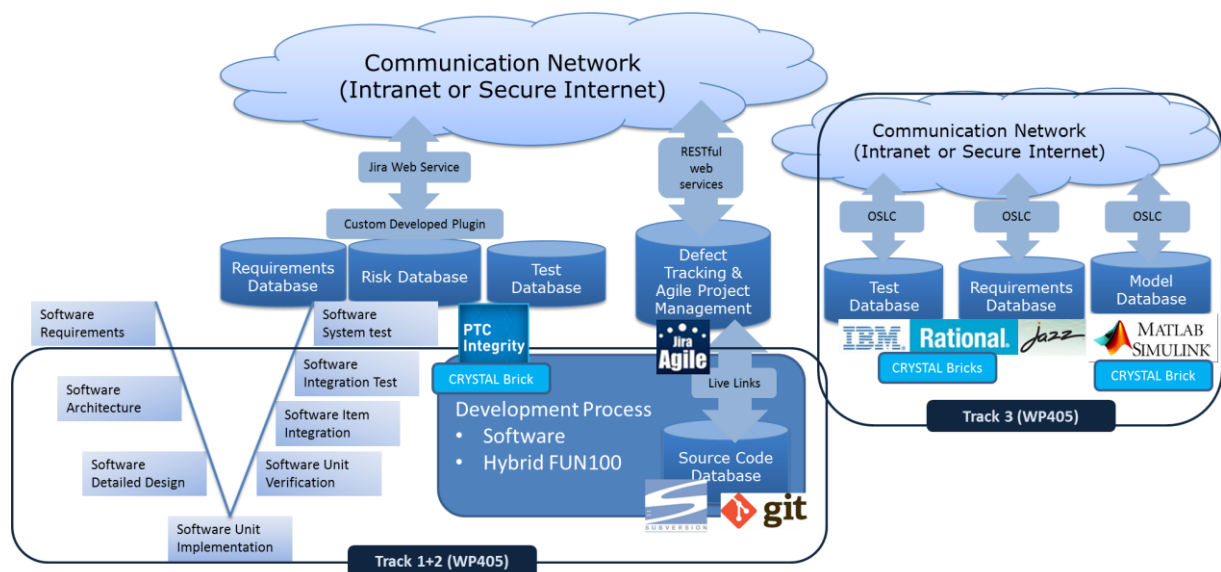


Figure 5-2: Positioning the WP405 activities on the high level SEE overview.

5.1 Track 1: SEE for the New Software Design Process at M12

Figure 5-3 gives a detailed overview of the SEE and the first implementation work on the selected engineering methods for the first implementation track, the New Software Design Process.

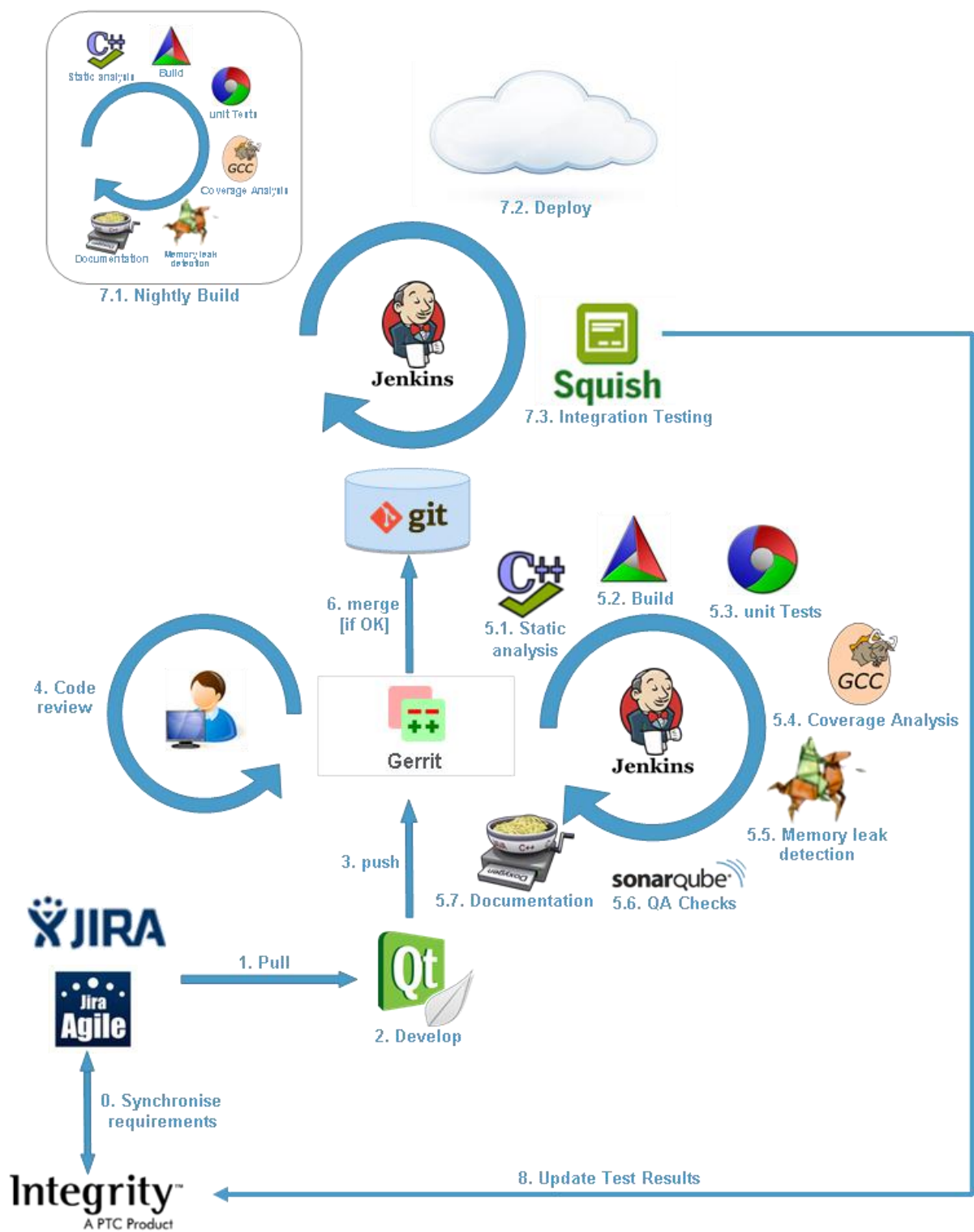


Figure 5-3: SEE setup for New Software Design Process.

Tool Selection

Following tools were selected for these new engineering methods:

- Automation framework: Jenkins.

- Source Control Management: Git.
- Agile Project Management: Jira Agile.
- Development Platform: Qt
- Documentation tool: Doxygen (Work in progress).
- Dynamic memory analyzer: Valgrind.
- Compiler tool chain: GCC.
- Integration testing: Squish.
- Code review: Gerrit.
- Quality management platform: Sonarqube.

More details on the first implementation result are described in section 6.1.

5.2 Track 2: SEE for FUN100 at M12

5.2.1 Test Framework for FUN1000

Figure 5-4 shows the new implemented engineering methods Software Engineering Testing and Component (Integration) Testing, both indicated in green, on the V-model for the FUN100 platform.

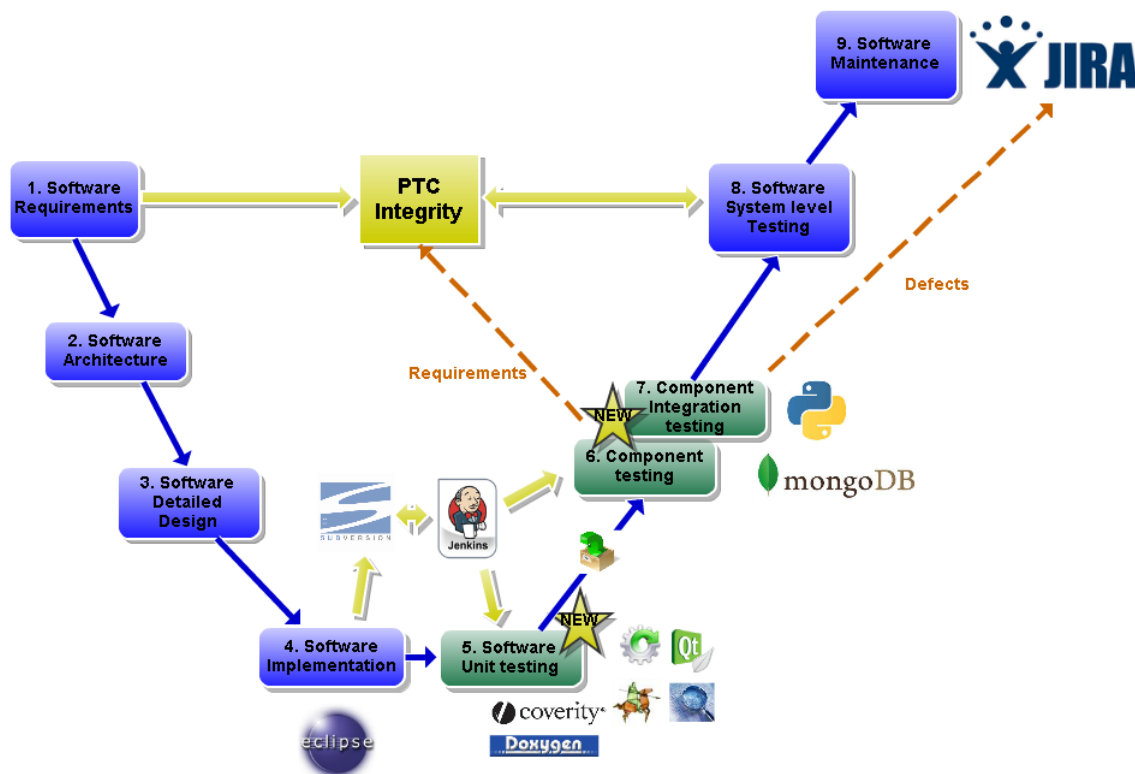


Figure 5-4: New Test Engineering Methods for FUN100.

Tool selection

Following tools were selected for these new engineering methods:

- PTC Integrity: Tracing requirements and system levels tests.

- Jira/itrack: Tracing of defects; lower level project planning.
- Jenkins: Continuous integration: automation of builds, tests.
- Subversion: Source control management tool: configuration management and versioning of source code and test code.
- GCC: C++ compiler.
- Yocto: Overall build system and linux distribution generator.
- QtTest: Unit test framework.
- Valgrind: Memory analysis tool.
- Gcov: Code coverage of unit test.
- Coverity and Cppcheck: Static code analysis tools.
- Doxygen: Source code documentation generation.
- Python and Pytest: Development language for component and system level tests.
- MongoDB: Database to keep component test data and results.

A more detailed view on the FUN100Test SEE, the workflow and artefacts can be found in section 6.2.

5.2.2 Architectural design for FUN100

For the engineering architectural design for the FUN100 we started with a new methodology, next to this new tools were installed to optimise our architectural design process, Figure 5-5 illustrates all tool of this engineerin method, new tools are indicated, mapped onto the FUN100 V-model design process.

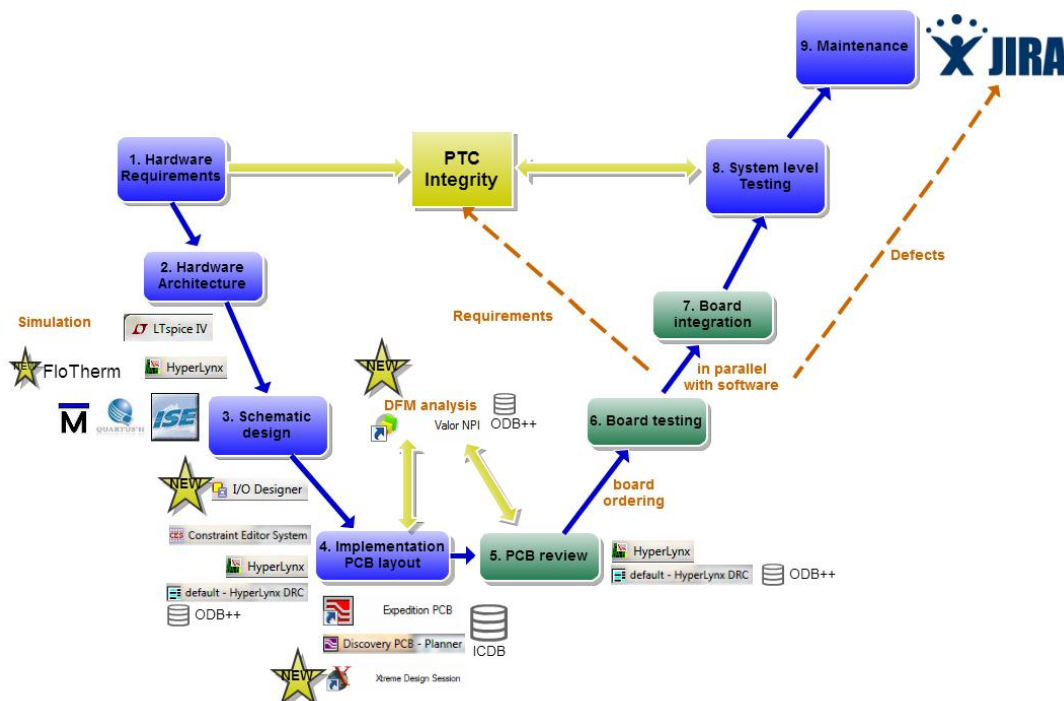


Figure 5-5: New tools for the FUN100 architectural design.

Tool selection

- LTspice: analog simulation tool.
- Hyperlynx: pre and post layout Signal Integrity/Power Integrity simulation tool.

- CES: constraint entry system (layout constraint management).
- HyperlynxDRC: automated PCB reviewing tool.
- Expedition: layout tool.
- Discovery PCB - Planner: manual PCB reviewing tool.
- Valor NPI: Design for manufacturing tool.
- Altera Quartus: FPGA design tool.
- Modelsim: VHDL simulation tool.
- XilinxISE: FPGA design tool.

New Tools

- FloTherm: thermal simulation tool.
- IODesigner: FPGA pin mapping tool to keep schematic and VHDL into sync.
- Xtreme Design Session: Multiple Expedition sessions on the same PCB database ICDB.

For More details about the new methodology for architectural design and the workflow with the new tools we would like to refer to 5.2.2

5.3 Track 3: SEE for Flexible SW Centric Display at M12

Figure 5-6 gives an overview of the System Engineering Environment setup for the selected engineering methods of Track 3.

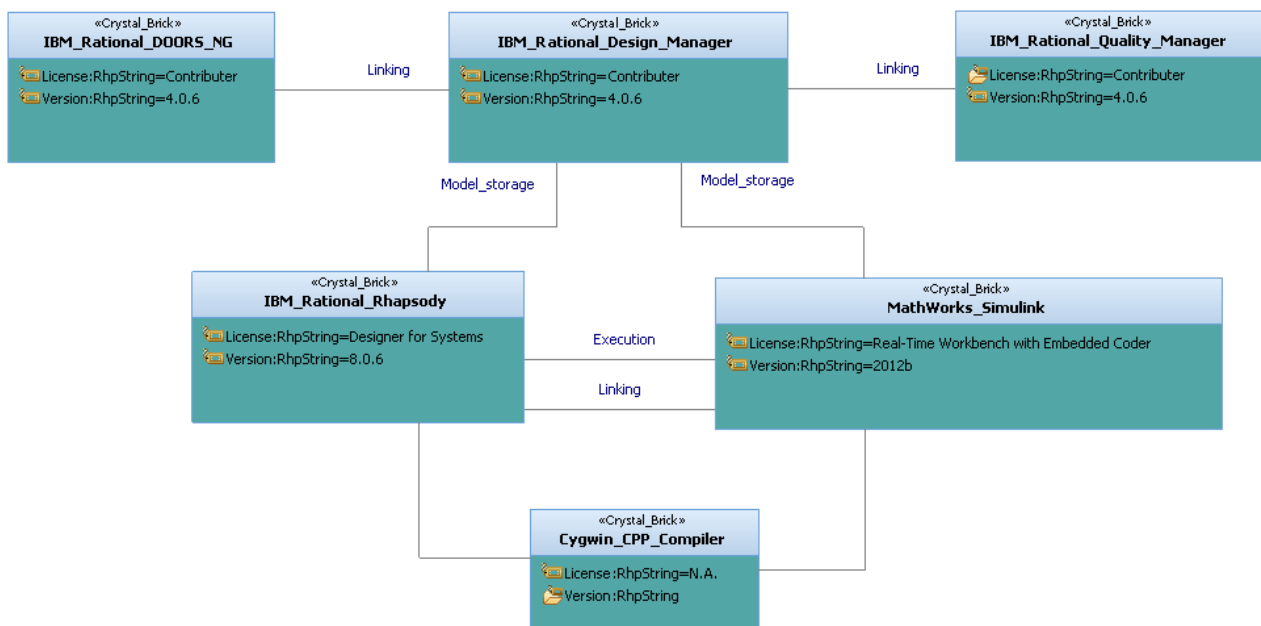


Figure 5-6: SEE setup for the Flexible SW Centric Display.

Tool selection

Following tools were selected for these new engineering methods:

Brick	Description	Interfaces	Process reference
-------	-------------	------------	-------------------

Brick	Description	Interfaces	Process reference
IBM Rational Doors NG	Requirements Management	Linking (OSLC) of: <ul style="list-style-type: none"> Specifications (documents) Requirements 	<ol style="list-style-type: none"> 1. Requirements 2. Release Planning 3. Sprint Planning 4. Systems Engineering 5. Components Engineering 7. System Integration Test 8. System Acceptance Test 10. Impact analysis
IBM Rational Design Manager	Model Management	Model input/output Linking (OSLC) of: <ul style="list-style-type: none"> Model elements 	<ol style="list-style-type: none"> 4. System Engineering 5. Components Engineering 7. System Integration Test 10. Impact Analysis
IBM Rational Quality Manager	Quality Management	Linking (OSLC) of: <ul style="list-style-type: none"> Test plans Test cases Test results 	<ol style="list-style-type: none"> 7. System Integration Test 8. System Acceptance Test 10. Impact Analysis
IBM Rational Rhapsody	Discrete Functional Modelling & Simulation	Model input/output Collaborative Model Execution (FMI) C++ code	<ol style="list-style-type: none"> 4. System Engineering 5. Component Engineering 6. Component Integration 10. Impact Analysis
Mathworks Simulink	Continuous Modelling & Simulation Performance Simulation	Model input/output Collaborative Model Execution (FMI) C++ code	<ol style="list-style-type: none"> 4. System Engineering 5. Component Engineering 6. Component Integration 10. Impact Analysis
Cygwin CPP Compiler	C++ compiler	C++ code Executable	<ol style="list-style-type: none"> 4. System Engineering 5. Component Engineering 6. Component Integration

- **IBM Rational Solution for Systems and Software Engineering**

The IBM Rational Solution for Systems and Software Engineering (SEE) helps you specify, design, implement, and validate complex products and the software that powers them with an integrated set of tools, services, and practices. The integration is based on OSLC and implemented through the Jazz Platform for tool integration. The services that are provided by the SEE are Project Management, Change Management, Configuration Management, Quality Management, Requirement Management, Model Management and Engineering Lifecycle Management. The individual tools that provide the services are described in the next paragraphs.

- **Rational Team Concert (RTC)**
RTC provides Project, Change and Configuration Management. RTC integrates task tracking, source control, and agile planning with continuous builds and a configurable process to adapt to the way you work.
- **Rational Quality Manager (RQM)**
Rational Quality Manager provides Quality Management. RQM is used to plan, develop, execute, and report on your test plan.
- **Rational DOORS Next Generation (RDN)**
RDN provides Requirement Management. RDN is used for complex software and systems engineering environments, helping engineers to manage requirements across disciplines, time zones, and supply chains.
- **Rational Rhapsody with Design Manager (RDM)**
RDM provides Modeling and Design Management capabilities to architect, design, and develop software and systems in an iterative and collaborative way. Models can be shared, reviewed and managed to stay current with others, and understand downstream impacts of design changes. Additionally to that Rhapsody provides model simulation and executable code generation.
- **Rational Engineering Lifecycle Manager (RELM)**

RELM visualizes analyses and organizes engineering lifecycle data and data relationships. It enables product development teams to search and query engineering data that is stored and managed in multiple sources and locations. Rational Engineering Lifecycle Manager helps teams make better use of engineering data for more effective engineering decisions and more efficient compliance with industry standards.

- **Mathworks Simulink:**
Tool for modeling, simulating and analysing multidomain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries
- **Cygwin CPP Compiler**
C++ Compiler environment.

5.4 Conclusion: Envisioned SEE

Based on the identified engineering methods and future planned activities the envisioned SEE is shown in Figure 5-7.

Future integration works include:

- Open Standards interface for Integrity
- Integrating the design process for the Flexible SW Centric Design Process.
- Integration with our PLM system (PTC Windchill).
- Integration with Enterprise Architect.
- Integration with Matlab Simulink.
- ...

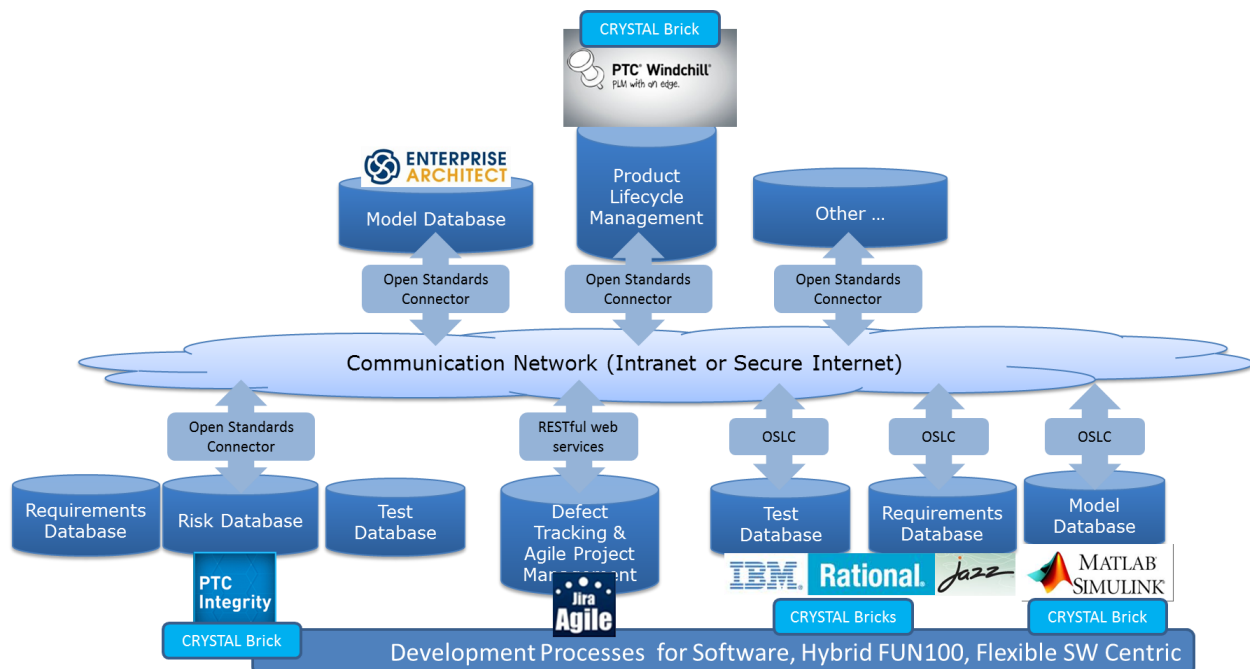


Figure 5-7: Envisioned SEE.

6 Implemented Engineering Methods

In this section we show the results of the first implementation work regarding the selected engineering methods given in section 4. These implemented engineering methods give us a profound understanding of interoperability issues and effectiveness on product quality and development agility.

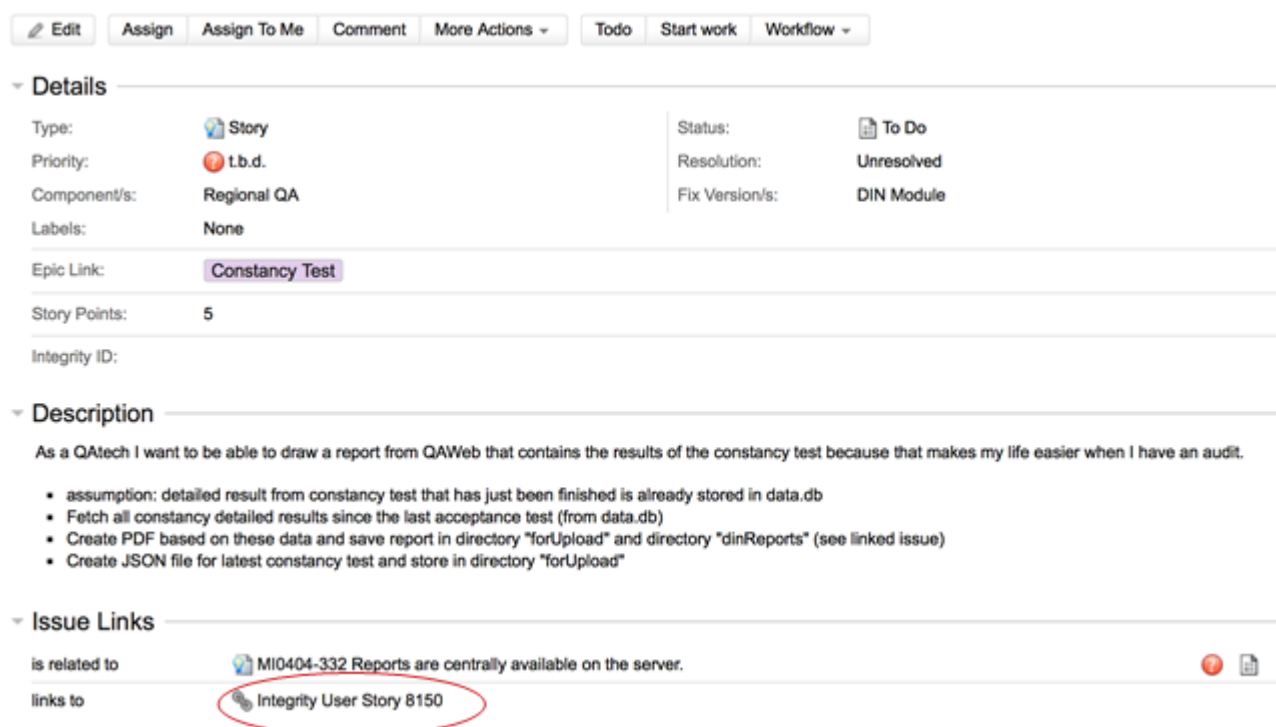
6.1 Track 1: New Software Design Process

6.1.1 Engineering Method: Requirements Traceability

Given the PTC Integrity Pilot project described in detail in CRYSTAL Deliverable D404_010, the next challenge was to create traceability between requirements in Integrity and the Jira Agile backlog items.

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.

Therefore we decided to map the requirements to epics. The Jira Agile tool was used to further break up the requirements into manageable user stories. From user stories we then create sprint tasks. A Java plugin was written to synchronise approved requirements with epics in Jira Agile. The epics that are created in Jira have a link back to the original requirement (Figure 6-1) and the original requirement has a link to the Jira epic.



The screenshot shows a Jira Agile issue page. At the top, there are buttons for 'Edit', 'Assign', 'Assign To Me', 'Comment', 'More Actions', 'Todo', 'Start work', and 'Workflow'. Below these is the 'Details' section, which includes fields for Type (Story), Priority (t.b.d.), Component/s (Regional QA), Labels (None), Epic Link (Constancy Test), Story Points (5), and Integrity ID. The 'Description' section contains a text block and a bulleted list of tasks. The 'Issue Links' section shows a link to 'Integrity User Story 8150', which is circled in red.

Details

Type: Story
Priority: t.b.d.
Component/s: Regional QA
Labels: None
Epic Link: Constancy Test
Story Points: 5
Integrity ID:

Description

As a QAtech I want to be able to draw a report from QAWeb that contains the results of the constancy test because that makes my life easier when I have an audit.

- assumption: detailed result from constancy test that has just been finished is already stored in data.db
- Fetch all constancy detailed results since the last acceptance test (from data.db)
- Create PDF based on these data and save report in directory "forUpload" and directory "dinReports" (see linked issue)
- Create JSON file for latest constancy test and store in directory "forUpload"

Issue Links

is related to: MI0404-332 Reports are centrally available on the server.
links to: Integrity User Story 8150

Figure 6-1: Link between epics in Jira and original requirement.

For tracing requirements to coding artefacts we force all developers to add the Jira user story number into the commit message when they push code to the version control system.

Commits that lack this number are rejected (Figure 6-2):

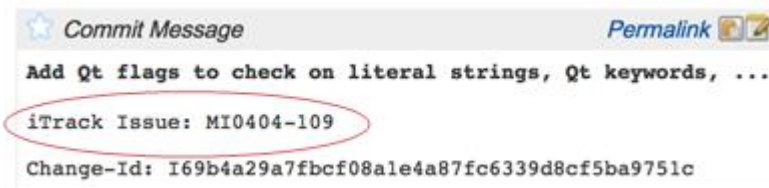


Figure 6-2: Rejected commit.

Browsing through the commits in the version control system shows a link to the original user story (Figure 6-3).

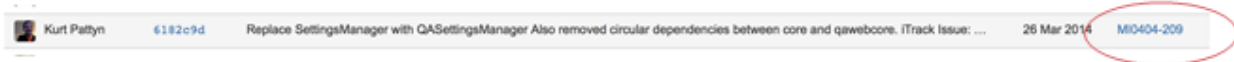


Figure 6-3: Link to the original story.

6.1.2 Engineering Method: Key Quality Metrics

One of the most challenging risks in product development is bugs. Especially in medical devices, bugs can lead to life-threatening situations. Therefore it is of utmost importance that bugs are detected during the development of the product. It is also known that solving bugs becomes more and more expensive as the product approaches its release.

In order to keep the risk under control, we came up with the following base metrics (to be expanded as the experience grows):

Unit Test Code Coverage

One of the best ways to detect bugs in software in a very early stage is to use unit tests. We determined that the minimum required coverage should be 70% (this will be adapted as experience learns us if this number is too low).

Documentation

It is known that code is more read than it is written. When maintaining software it is very important that the code is well understood. We decided that all code must be documented.

Bugs per 1000 lines of code (KLOC) Safety critical software is considered to be reliable when the bug rate is no more than 0.01 bugs/KLOC for deployed code. We decided to use Bugs/KLOC as a quality metric. It will be used to adapt unit test code coverage and enhancing regression testing.

Static Code Analysis

A lot of common mistakes can be detected by so-called static code analysis tools. To supplement our unit tests, we decided to implement static code analysis, so that common mistakes like uninitialized variables, array overflows, memory leaks, and so on can be detected in a very early stage.

Code Review

Not all bugs can be found by unit tests or static code analysis. Misinterpretation of a requirement is a bug and can only be detected by a human. Therefore peer code review will be introduced to detect another field of potential bugs. A side-effect will be that peer reviewers will learn to know the code and can point to copy-paste implementations.

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	53 of 75

6.1.3 Engineering Methods: Iterative Development & Process Automation

We introduced the SCRUM process in the Barco software development team. The purpose was to produce products in small increments (called sprints). This means that the whole lifecycle is executed in every sprint: analysis, design, implementation, testing and acceptance.

The major goals were:

1. Have better control over the release planning.
2. Have early feedback on the correct implementation of the requirements.
3. Deliver a potentially releasable product.

By introducing SCRUM we were able to continuously measure our progress, leading to a better estimation of when a version 1.0 can potentially be released. We also measure our velocity so a better estimation can be made about when a certain amount of requirements can be delivered.

By having regular reviews by product management, we get very early feedback on the correct implementation of requirements. It also reveals shortcomings in requirements, and hence early corrections can be made.

The goal to deliver a potentially releasable product forces us to implement a complete product lifecycle. So during a sprint, all stages of product development are gone through: analysis, design, development, testing. The only thing that is not done is the actual release. But acceptance tests are in place, so that we don't have to wait until the product is declared finished to do acceptance tests.

Because we strive to deliver a potentially releasable product, we also have to take care of QA rules and regulations, like risk management files, regulatory compliance documentation, traceability, and so on.

The SCRUM process and all QA checks have been automated, so that we have a repeatable and demonstrable way of product delivery.

Requirements from PTC Integrity are synchronized with epics in Jira Agile, which in their turn are broken down in user stories and further down into sprint tasks.

Our code is maintained in Git with a code review system guarding the commits. Every commit must contain a link to the Jira Agile sprint task.

When code is committed to Git, the code review system (Gerrit) takes over. It starts with running some basic quality checks, like testing if a link to the Jira Agile task is present in the commit message, line lengths do not exceed 80 characters, and so on. The process that is responsible for this is called the QA Bot. If a problem is found, then the commit is rejected and cannot be merged into Git. If no problem is found, the Build Bot takes over. The Build Bot builds the software, executes unit tests and coverage analysis, generates documentation, runs a static analysis and finally check a number of thresholds:

1. Number of warnings and errors from the compiler must be zero.
2. Documentation of all code must be complete.
3. The number of problems found by the static code analysis must be smaller than 5.
4. This number was chosen empirically and will be changed to zero in the future.
5. The technical debt is analysed. The technical debt is measured from the TODO and FIXME tags in the code, together with the number of outstanding bugs in the Jira issue tracking database.
6. Currently, this number is only calculated and visualised but no thresholds have been set.

If no problem is found, the Review Bot takes over. The Review Bot makes sure that code is reviewed by a number of developers. In our case we require that 2 independent developers review the code. If they both approve the commit, then the commit is merged into Git.

This way of working has a number of advantages:

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	54 of 75

1. It ensures that a number of bugs are discovered early. Studies indicate that code reviews can accomplish an 85% defect removal rate with an average rate of about 65%¹.
2. Developers learn to know the code and can detect copy-paste parts in the code. In implementing new things, developers can leverage knowledge about the already committed code.

Finally the results of the analysis are published. Below is a snapshot of the quality dashboard (Figure 6-4).

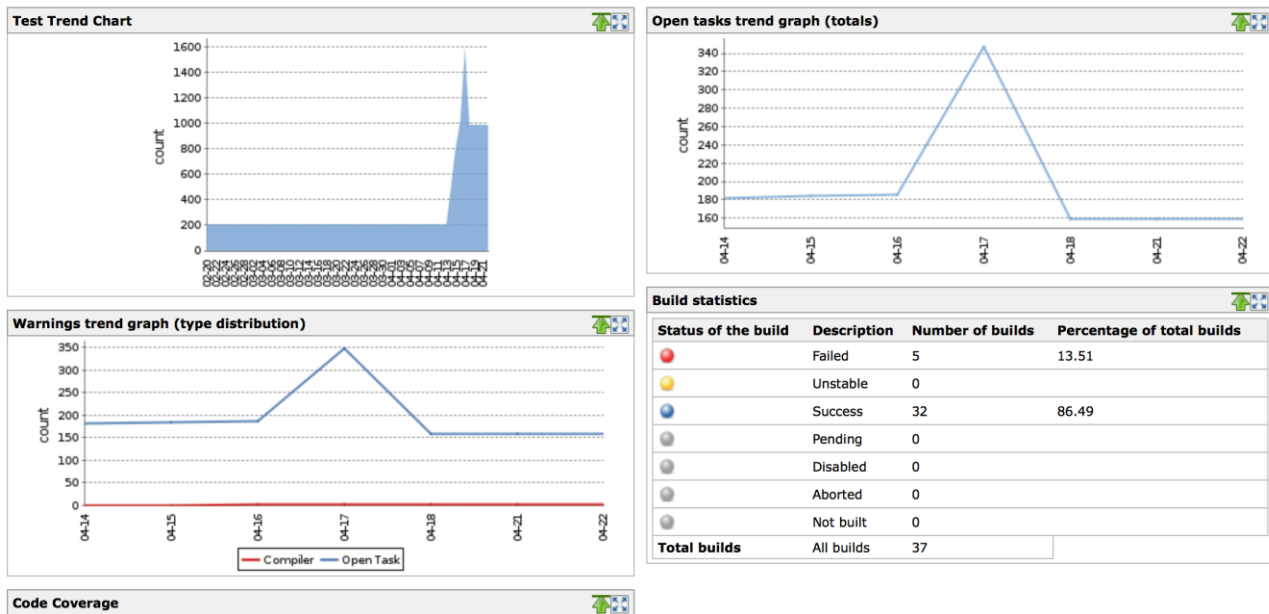


Figure 6-4: Quality Dashboard screenshot.

Below (Figure 6-5) is a snapshot of the code coverage report.

Name	Packages	Files	Classes	Lines	Conditionals
Cobertura Coverage Report	86% 32/37	82% 137/167	82% 137/167	72% 5959/8292	51% 16331/32106

Name	Files	Classes	Lines	Conditionals
base.src.libraries.libBarcoCalibration.src.compliance	100% 1/1	100% 1/1	100% 35/35	55% 75/136
base.src.libraries.libBarcoCalibration.src.displayfunction	100% 12/12	100% 12/12	84% 229/274	72% 110/152
base.src.libraries.libBarcoCalibration.src.lookuptable	50% 1/2	50% 1/2	36% 24/67	59% 38/64
base.src.libraries.libBarcoCalibration.src.main	0% 0/1	0% 0/1	0% 0/2	N/A
base.src.libraries.libBarcoCore.src.codexcs	100% 1/1	100% 1/1	64% 9/14	50% 12/24
base.src.libraries.libBarcoCore.src.event	92% 12/13	92% 12/13	73% 529/720	52% 733/1399
base.src.libraries.libBarcoCore.src.global	100% 1/1	100% 1/1	67% 2/3	N/A

Figure 6-5: Code coverage report result.

6.2 Track 2: FUN100 Design Process

6.2.1 Engineering Method: Software Unit Testing

Automation tool selection

The continuous integration tool Jenkins is used for automation and is the glue that ties everything together. In general Jenkins is responsible for triggering a build or test or check either at a certain time, for example during the night, or trigger by a change of source via a commit to the source control tool. Once the build /

check / test is done Jenkins gathers the status and results and makes them available via its web interface. In case of a failure Jenkins sends out mails to notify the users that made any changes that may have contributed to the failure.

Jenkins ensures tests/checks are continuously being run without need for manual intervention.

Unit testing activities are typically triggered automatically by Jenkins based upon a commit of new source code to SCM. Upon which builds, checks and tests are executed (Figure 6-6).

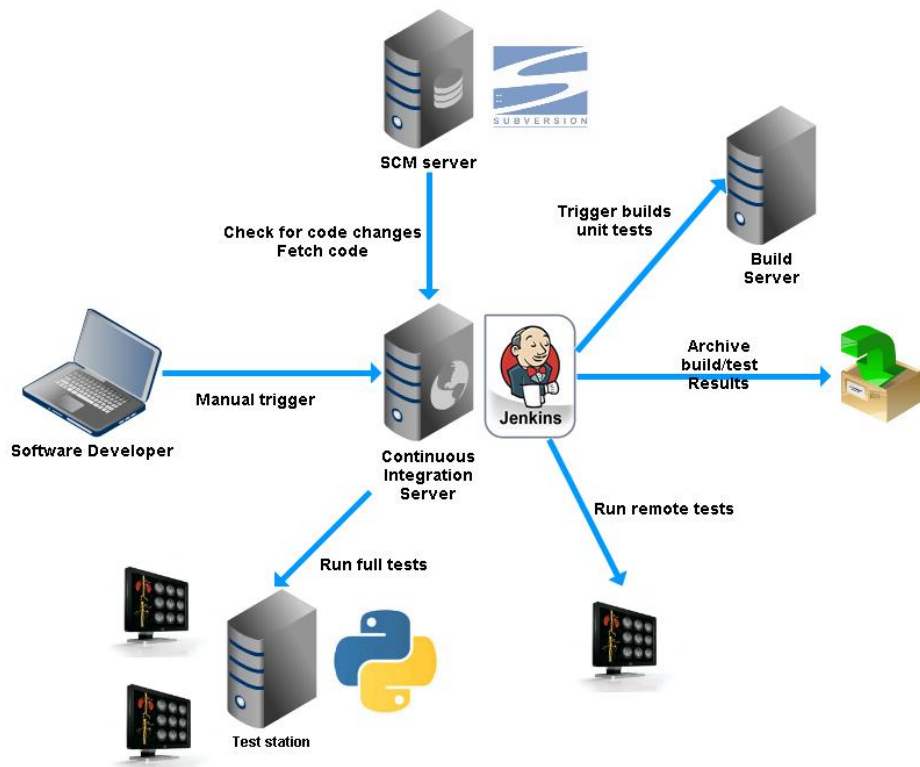


Figure 6-6: Jenkins as automation tool.

Tools

- Automation framework: Jenkins
- Source Control Management: Subversion and Git.

Test tools and checkers:

- GCC cross compiler tool chain: used to do the embedded builds.
- GCC compiler tool chain: used to for native builds, and building and running the unit test.
- Coverity: a static code analyser, doing checks on the embedded code.
- CppCheck: a static code analyser, doing checks on the native code.
- Doxygen: A tool to generate documentation of the source code (Work in progress).
- Gcov/Lcov: Used to measure code coverage of the unit tests.
- Valgrind: a dynamic memory analyser used to detect memory leak and corruption while executing the unit tests.

- QtTest unit test framework: The unit test framework used for unit testing.

Workflow for software development/implementation

1. A developer checks out code from the subversion Source Control Management system.
2. A developer make changes to the code, implements new features, writes unit tests for them.
3. When done a developer commit the code back to the subversion SCM. At this point the code becomes available for other developers. This is also the point the automated unit testing and associated activities kicks in.

Workflow for unit testing activity

Whenever a software developer commits new code or a code change to the Subversion SCM repository, this will be automatically detected by the continuous integration tool Jenkins.

Depending on the software component Jenkins will trigger a build check and the execution of a number of tests and checks.

And when these are successful, Jenkins will furthermore trigger an incremental Yocto build of the full software resulting in a software deliverable that can be installed on a target.

Individual steps in the testing activities:

1. Jenkins, the continuous integration tool/server automatically detects changes to source control and will trigger the tests builds, typically within 5 minutes of the change
2. First the source code is build both for an embedded target and the so-called native target, to check for build integrity and compiler warnings and errors.
3. The unit tests are run, this is done only natively.
4. Two static code analysis tools are released upon the source code: Coverity & Cppcheck
5. The unit tests themselves are run within Valgrind which checks for memory leaks, etc. during the execution of the unit tests.
6. Doxygen is used to automatic generated source documentation.
7. Code coverage is active during the unit tests to check which code is covered by the unit tests and which code is note.

Input and output

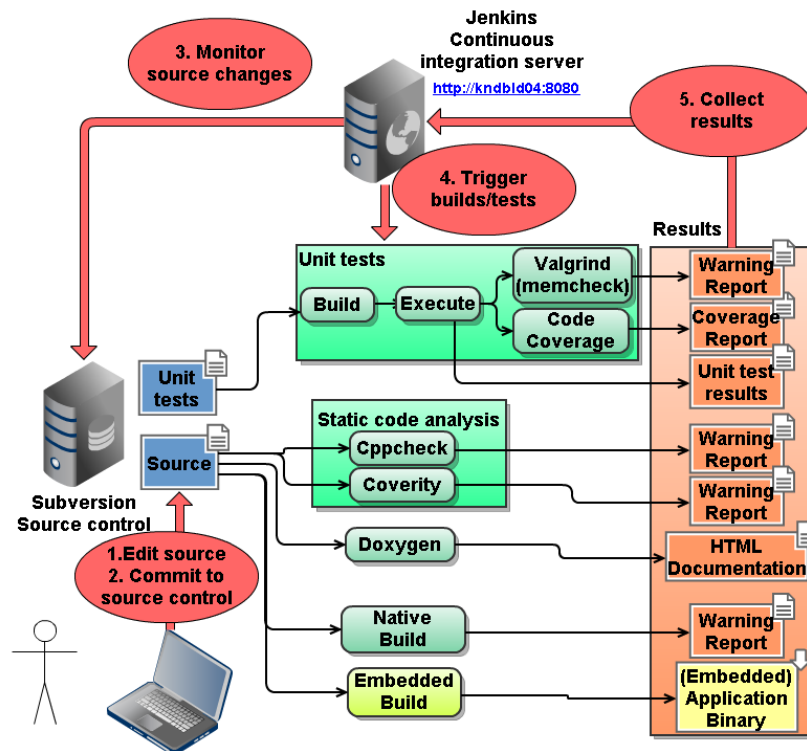


Figure 6-7: Workflow and artefacts for the FUN100 Unit Testing.

6.2.2 Engineering Method: Component Testing

Test items (Input)

1. Software components and the interface between the software components. This is the primary goal of the component tests.
2. User stories. User stories typically have a direct link to the software requirements and cover a certain functionality typically involving multiple software components. The goal however is not to test all software requirements only the requirements fully within the scope of the FUN100 platform.
3. Fixes for defects. Again the component tests won't cover all defects, but only the defects that are fully within the scope of the FUN100 platform.

Requirements based testing: functional and non-functional

Any component test case which is designed to test a specific requirement or has a direct relationship with a requirement should reference the software requirement ID in its test case description.

Figure 6-8 shows the link to requirements and bugs in the Jenkins environment.

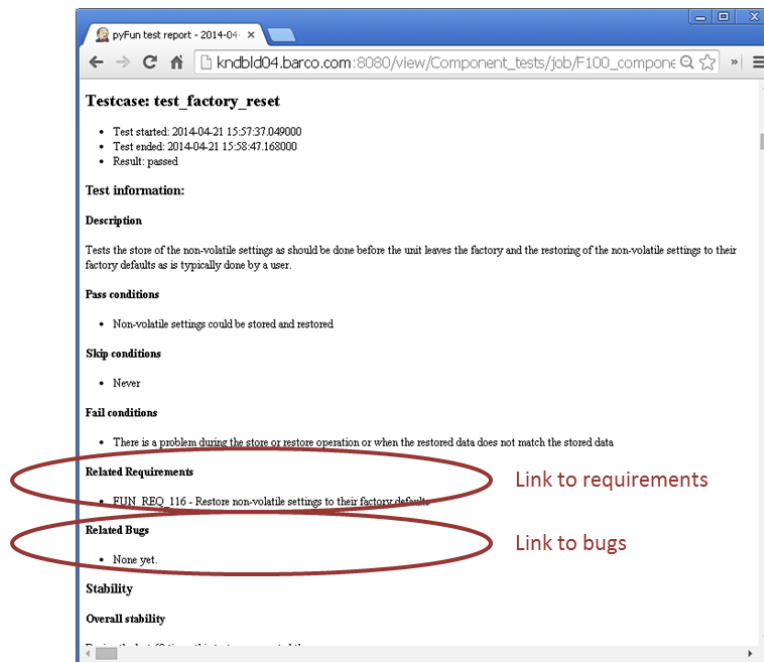


Figure 6-8: Link to requirements and bugs in Jenkins.

Exit criteria (Output)

- Tests run during the software development.
 - For the component tests run during the software development as part of the nightly build and check:
 - For every test which fails for 5 work days in a row or which is unstable (at least 3 failures within 10 consecutive test runs) an iTrack (bug tracking) issue must be created.
- Tests run on a software release or software test release
 - For every failed test as part of the testing of a software test release or software release an iTrack (bug tracking) issue must be created.
- Tracking of system parameters.
 - System parameters are to be evaluated for every (test) release against the historical data of a previous (test) release.
 - Any deviation deemed significant enough has also to be reported via an iTrack issue.
- Reporting

Each test report shall include:

 - Device under test configuration.
 - Product name, display part number and serial number.
 - Version, number (K54....) and timestamp of the FUN100 software used.
 - Test setup configuration:
 - Configuration file name.
 - Test machine hostname and username.
 - Test start and end time.
 - Reference to the revision of the test source.
 - An overall summary:
 - PASS/FAIL. A pass indicating all run test cases passed.
 - Number of tests passed / failed or skipped.
 - A summary per component.

- A detailed overview of all tests and the indication whether they were passed / failed or skipped.
- Details of every test case including:
 - Description.
 - Pass / Fail / Skip conditions.
 - Related requirements if any.
 - Related bugs / defects.
 - Info about the historical stability of the test.
 - Info about the historical evolution of a system parameter if any.

Figure 6-9 shows the test report result in the Jenkins environment.

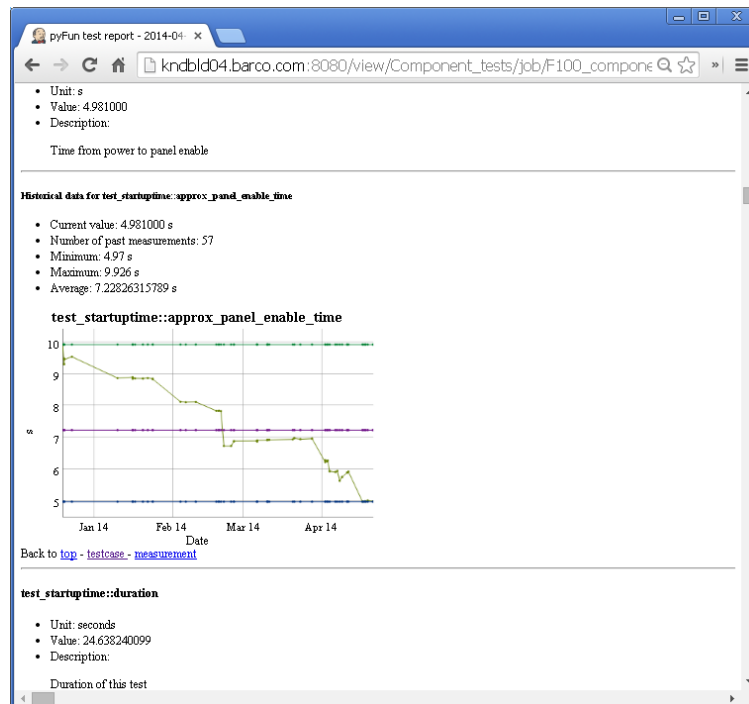


Figure 6-9: Test report result in Jenkins.

6.2.3 Engineering Method: Architectural Design for FUN100

This section describes the first implementation result of the architectural design process for FUN100.

New Design Methodology

In a first phase the requirements of the existing platform were thoroughly reviewed. In cooperation with product management, we were able to rewrite the requirements for better efficiency. All the overhead was removed. Provisions for various possible extensions were removed when they did not add value. And the architecture was completely redrawn in a layered and modular way. It was the intention to make the architecture as independent as possible from the hardware so that a dedicated hardware could be derived as needed to support displays.

Different functional modules were identified, as well as the interfaces between them. Where functionality had to be different over the various implementations, the modules were modified with parameters.

The hardware of the platform 900 was designed to allow re-use over various display types. However, this meant that a lot of overhead was on the board to support the various options. This resulted in fairly large boards, which were expensive. And in the final product, lots of the functionality was not used.

A very important aspect of the new approach (Figure 6-10) is that a lot of effort is spent in the concept phase. Contrary to classic design, the platform concept and architecture was seriously challenged during the concept phase. Any doubt on the implementation was eliminated.

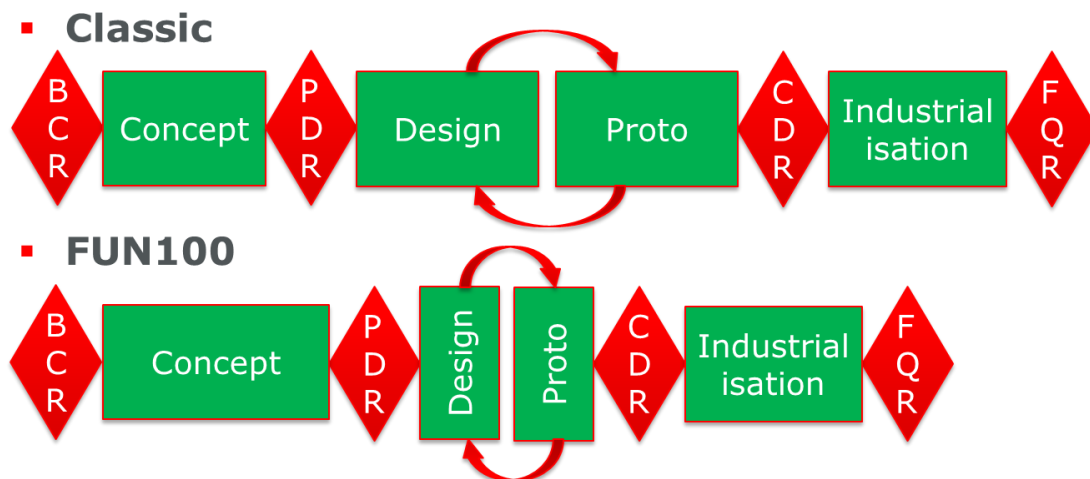


Figure 6-10: FUN100 Architecture Design Methodology.

In order to achieve this, a new design methodology was implemented.

1. For the software, the complete architecture was documented before implementation was started. The parts of the previous platform, which were in scope for re-use, were identified. Also open source software was identified.
2. The automated build and test environment was set-up. Once the development started, the automated environment helped to test and debug the software instantly. Very little iterations were needed. The code itself is heavier and looks more complicated at a first glance. However the modularity and layered structure make it a lot easier to understand. The investment in the overhead was easily gained back during development.
3. The hardware layer was also more efficient as it was clearly separated from the rest of the functionality.
4. For the hardware, the design was de-risked upfront. No implementation was started until all the functional blocks were simulated, or reviewed by peers. A significant amount of effort was spent on signal integrity and power integrity simulations. Also thermal simulations were used. All this was required as the board space is very limited due to cost reasons. It was also the intention to have the PCB layout first time right. Therefore, all of the constraints had to be worked out upfront, so that the PCB layout was only an execution phase. This made it very fast and effective. No iterations were needed, and the boards worked fine the first time.
5. Simulation also allows making critical choices in an early phase, without the need of having the hardware available. The different hardware blocks were conceived as modules on concept level. The actual implementation was filled in only when the exact electrical-digital constraints were known. Simulation was done first on an abstract level, but become more tangible as more details were filled in.

Workflow

1. During the hardware architecture phase, the design is de-risked by intense simulation. All the risk in the architecture is reduced by simulating the design with appropriate tools. Simulation is done for analogue and power design, digital VHDL design, thermal design and digital signal integrity.

2. Only when the design is fully de-risked will the schematic entry begin. During the schematic entry, the hardware components are selected. A big part of the schematic is the FPGA, which has a large number of high speed interconnects. These are connected using I/O designer, assisted by FPGA tools and signal integrity.
3. Next the PCB layout can start. Multiple designers can place and route the PCB. A DFM analysis is performed daily.
4. Hyperlinx DRC is run on a regular basis to perform design rule checks. Critical parts can be indicated by Hyperlinx DRC, and can be simulated in detail by Hyperlinx.
5. ICDB is the common database between schematic and PCB layout.
6. ODB++ is the open format PCB database for external access.
7. After the PCB is finished, a final PCB review is organized. When no remarks are left, the board can be ordered.
8. Once the board is started up, it will become part of the software test environment, so that the hardware is testing is integrated with software testing.

6.2.4 Engineering Method: Software Engineering for FUN100

This section describes the first evaluation results for the new methodologies for the software engineering process for FUN100.

Portability

Coding language choice

Different source code languages are used such as C, C++, Open CL and VHDL. This creates barriers to port implementations from one platform to the other, for instance from FPGA with source code written in VHDL to CPU where C or C++ is often used.

By developing a library of common functions within used languages such as VHDL, Open CL and C++ a common platform can be created to enable a unified description of image processing (IP), independent from its host device technology and even independent from its native software of hardware character. The library can be restricted to a few selected types of applications. It must be created in house for each of the required languages at least once, potentially optimized to support multiple host device types (for instance GPU types developed by NVidia, ATI and AMD and FPGA types developed by Xilinx and Altera).

Although the underlying functions will be language specific, the code structures, the algorithms and the architectures for the image processing application build on such a library would be reusable for the different types of platforms. The goal here is to be capable to port an application from one platform to another without (considerable) modifications.

New developments based on classic languages and their toolset have to be investigated as these evolved higher level languages could help to achieve such a unified description. Note that the toolsets offered by the FPGA vendors have dramatically evolved the last couple of years in order to be capable to synthesize more complex IP blocks and map them more efficiently to the FPGA hardware. Therefore VHDL anno 2014 can be considered as a higher level language than 10 years ago.

This also includes investigation of the new classes developed for C such as SystemC and the tools which extract hardware acceleration code for FPGA implementation such as the AutoESL tool which promises to extract optimized VHDL code for a specific (Xilinx or Altera) FPGA starting from high level C source code. System C will be used to bridge between the FPGA implementation and the “golden” software model. If possible we will use tools to automatically extract hardware circuitry from the C based model.

Ideally a single source code within a single programming language would be used for all imaginable implementations and verifications. That source code would also automatically document and analyse itself. Currently, most image processing applications in Barco use a combination of FPGA based hardware acceleration and CPU based software control, in most cases nearly static but sometimes near real-time (for

instance software algorithms can be part of a dynamic colour calibration module to adjust the PWM values of LEDs in a LCD backlight “as frequently as possible”). While the FPGA functionality is written in VHDL (in most cases), the CPU software as part of the same system is written in C (in most cases).

Within the scope of this platform the research covers a feasibility study of VHDL extraction tools from C and this would solve the problem of integrated system level validation, on top of the benefits from increased portability from FPGA to GPU based hardware.

One of the main drawbacks of C to VHDL conversion is the absence of any generics in the resulting code, which means that any existing VHDL design with many connected generics needs to be precompiled first with placeholders (or black boxes) with the same generics in order to set these values in the C code before useful VHDL for this specific project can be extracted. While this is not an issue for very simple modules such as AES encryption, it could lead to a nearly unmanageable work flow when one needs to integrate a number of more elaborated IP cores within an existing VHDL top level, or even the same module with several sets of generics (for instance a scaling component can be implemented with different quality specs for main display and preview purposes).

Therefore we also explore the possibility to keep the high level code in VHDL, using intensively some of the major improvements that came with the recently improved synthesis tools from both FPGA vendors and related to the VHDL 2008 standard: the capability to globally propagate locally generated statics, a feature that enables to configure modules based on its configuration capabilities in a distributed way.

Multiple design units can communicate with a single module and they can “negotiate” which module can configure which parameter. The subsidiarity principle is used to link modules in a certain design. The main advantage with this methodology is that the same source code can be reused for different top level compilations as long as these FPGA design top levels continue to exist in VHDL. Once these top levels can be converted into the C code, the developed methodology for generic IP configuration should be applied to the C code as well.

The main reason for continuing with research based on VHDL is the assumption that many FPGA designs will continue to have VHDL based top level descriptions for at least a “couple of” years. While with C-based synthesis we cannot yet explore a generic methodology for complete designs, we can do this in VHDL, which is fully compatible with any existing VHDL code.

Coding methodology for an entire IP chain

The new methodology also includes exploring the possibilities of the evolved VHDL syntax, as for example defined by the VHDL-2008 standard. Interesting new capabilities are defined here such as the possibility to define locally static variables distributed over multiple sub-entities (for instance a colour space converter as part of a 3D colour profile) and use these as globally static generic configuration parameters. We expect such a feature to be vitally important to build hierarchical image processing chains allowing for top-down configuration of sub-entities based on bottom-up declaration of building block capabilities. This methodology can be based on new features offered by the VHDL language, or by building our own library with “standardized” functions. In any case these functions should ideally be compatible with some form of C, like C#, C++, SystemC or OpenCL.

The model of the image processing chain must make an abstraction of the host hardware and software stack. In addition, the model must support host devices with heterogeneous processing hardware. It can be for instance a System on Chip, a CPU and GPU combination, an APU, or CPU and FPGA or DSP combination. The image processing chain elements must implement a stable, preferably standardized (as in non-proprietary), binary interface, which is used by the framework to discover, configure and control the elements in the pipeline. This allows using third party components, purely as binaries, as plug-ins in the framework, for instance a DDR memory controller offered by an FPGA vendor.

The image processing chain is modelled as a pipeline of interconnected elements and those elements; on their one must also be ... a processing chain of smaller, more elementary processing steps. This property is called self-similarity and a well-known example is the pictures of fractals. Obviously, this property is not infinitely recursive in this case, but it is however not possible to define upfront the exact number of levels one

could find. Simple use cases can lead to very complex pipelines. A consequence of this is that one can group certain elements together and abstract them away in a larger container, which can be regarded as a single entity and can be placed into another pipeline as one element. One doesn't even need to deal with the complexity of the original processors any more. The element has become a black box. The image processing chain is configured on a very low level in a way that a given (virtual) chain of processing blocks complies to a specific requirement.

An example of such a requirement is the perceptually equal quantizing intervals obtained by a DICOM calibration of the display. Obviously such a perceptual quantizing step tolerance spec can only be fulfilled when all processing blocks in the chain are accurate enough. However, the embedded software which communicates with the FPGA (for instance to load the contents of a look up table) should not be aware of the details of the precision, the parallelization and/or the timing of specific interfaces between underlying blocks.

Capability negotiation between the elements of the image pipeline

The elements of an image processing pipeline typically have a range of supported image or data traits. For example: an electro optical transfer function may support images in various colour spaces, with various bit depth, image width and height. These ranges of supported data traits are called the element capabilities.

At certain stages the framework must be capable of communicating and comparing all supported capabilities. This process is called capability negotiation. Capability negotiation is typically done in a hierarchical fashion: first from bottom-up and followed by a top-down phase. The goal of the negotiation process is to select those capabilities that comply with an optimal balance of several aspects including interoperability, device resources, computational power, image quality (as defined by application specific quality metrics), added latency, memory transfers (and their efficiency)...

Scalability

VHDL coding techniques to obtain scalability

Current implementations of existing image processing functionalities have to be adapted every time a new product has to be developed. A lot of valuable time is wasted with every new project to change "some details" in the previous source code to adapt the code for the current project.

In the context of this project we will explore the possibilities created by improved VHDL synthesis tools as well as the VHDL 2008 standard in order to create more generic code and thus to find a way to shorten future FPGA development times, reduce design risks and improve the video processing quality as the generic IP should offer "the best of" our proprietary IP using a model driven approach.

The rationales for developing improved IP coding techniques within the scope of this IP platform and its eventual goals for future product development are:

- Shorter time to market (reuse of IP and source code, earlier FPGA choice possible, speed up software development based on included driver models).
- Lower risk (no surprises during design process so planning becomes more predictable, power dissipation can be predicted earlier, resources predictions will be very accurate)
- Lower device cost (IP should optimize the usage of FPGA, DRAMS and other devices, design must be portable between FPGA vendors).
- Higher reliability (better validated design when implemented in products, included validation/simulation framework, regression testing).
- Better quality (generic IP should include the best of all Barco IP, higher level description allows for merging of functionalities which can increase quality).
- Simple integration in existing generic VHDL designs while preserving the generic character of the design. The current ModHDL methodology can be used for implementations.
- Validation at compile time which can instantly validate specs according to the currently applied generics in VHDL (top level). The compliancy to the specs of the configuration which is fixed at compile time can be immediately verified by the synthesis tool.

Version	Confidentiality Level	Date	Page
V01.00	R	2014-04-30	64 of 75

Many image processing functionalities are used in Barco in many different products, different FPGA families in different hardware environments, different applications/markets...

Amongst the factors that cause the same image processing to be implemented differently, often with different or even entirely new source code are:

- Quality requirements can be very different between implementations.
- Physical board space constraints can lead to different FPGA or DRAM choice.
- Device cost constraints.
- Device resources reserved for other functionalities to be integrated with current IP functionality (within the same FPGA device or using the same memory devices).
- Some IP functions (sometimes provided by third parties) are only available in 1 device family, so the device choice can be forced by unrelated functions.
- Performance aspects (data throughput or latency specs).
- Image sources required by the system.
- Development time.
- Available devices (and their evolution).
- Designers/co-designers preferences/experiences/skills.
- Available software licenses: Modelsim, Sinplify, Leonardo, Precision, DSP packs in Maple, MatLab...
- User interface or API requirements.
- Power dissipation budget.

Highly configurable

The pipeline topology

The image processing framework must support changes of the pipeline topology at runtime. A simple example is the 3D DICOM colour profile. When dynamically switched on or off, other part of the processing chain must adapt to the presence or absence of this specific processing block. To support such a feature, the framework must be able to reconfigure the pipeline on the fly and insert new elements that process the same video in a new parallel or serial path. The image processing framework must be capable of estimating upfront the consequences regarding quality of experience of specific changes in pipeline parameterisation and pipeline topology.

The following description is applicable for both design time and run time configurations stages. In some markets, consider for instance diagnostic displays, the source signal characteristics and quality requirements are known upfront and thus the implementation decisions can be done during the design of the product and can be synthesised in HW to achieve the best possible performance at the lowest cost.

In other applications, the framework must be able to processes a large variety of data, and thus many of the decisions are done at runtime, for instance inputs can vary in bit depth (8, 10 or 12 bit inputs), colour profiles (sRGB or DICOM)... Every sub-entity would be optimally configured based on the outcome of this "best choice" scenario based on a hierarchical analysis of the image processing chain.

Such a configuration would include the optimization of calculation accuracies and intermediate processing formats including several floating point notations, bit width, excursion scale, head- and foot room definitions. Ideally it would allow eliminating all capabilities unused by the host application per entity. For instance if a colour space converter has the capability to handle out-of-gamut colours but the host application "knows" no colours can go out-of-gamut this processing should not be implemented.

The concept of this platform will accommodate for a variety of display types (such as Fusion, Nio and Unity) and sources in the future. Furthermore the high level description of the host application should be capable of eliminating redundant steps in the image processing chain, for instance by combining successive look up table transfer functions into a single pre-calculated LUT or by combining cascaded FIR filters into a single FIR filter step, such as for sub-pixel display X-talk compensation. This would not only reduce the computational resources but would at the same time increase the image quality and potentially reduce the application's latency. Based on these generics the entity (function or sub-function) would return its constrained capabilities.

Testability

A first evaluation learned that some IP tends to be quite difficult to validate on the actual FPGA hardware implementation. The reason for this is that the hardware processes a massive amount of data in parallel. When an image processing block for instance performs a correction based on data from both the current incoming frame as well as the previous frame, then at first sight you would need to be able to generate all these stimuli (so both frames) as well as to monitor the response (the outgoing frame). The overhead to be able to do so would cost too much FPGA resources.

To overcome this problem some modifications were done within the image processing chain architecture on several strategic locations. These modifications include:

- Insertions of static stimuli at specific nodes within the image processing chain.
- Monitoring of specific nodes within the image processing chain.

The insertion of static stimuli at multiple strategic nodes within the image processing chain allows for emulation of complex stimuli, as if they were changing rapidly (on a frame by frame base). Combined with a monitoring functionality at specific nodes within the image processing chain this allows for quite powerful verifications of the IP on the actual FPGA hardware implementation.

For the FUN 100 platform we have a strong focus towards testability.

The first reason is because we must be able to prove at each release that we tested and validated all of the software requirements to comply with the IEC-62304 standard.

A second reason is that we need to keep our release cycles as short as possible. Especially when supporting two boards with a completely different FPGA architecture.

We can verify a lot of FPGA functionality by simulation. If it doesn't work in simulation, then you can be pretty sure it will not work on the actual platform either.

However after evaluation we concluded that a working simulation doesn't guarantee a working system. Therefore we added strategic test hooks in our FPGA design which our software can use to implement unit and component tests.

This approach has the advantage that the effort needed to implement the test is significantly less.

Furthermore we will be able to run the same test on for instance a F100 and an U100 verifying the actual system.

6.3 Track 3: Flexible SW centric display platform Design Process

6.3.1 Engineering Method: Functional Modeling

Tool selection

- IBM Rational Rhapsody.
- Mathworks MATLAB/Simulink.
- IBM Rational Jazz environment for System and Software Engineering.

Overview of artefacts and Interoperability

Input	Requirements, Use Cases, Actors
Output	Executable behavioural model
Tools	IBM Rational Rhapsody, MathWorks Simulink
Interoperability	Model elements in Rhapsody <-> Model elements in Simulink Requirements in DOORS <-> Model elements in Rhapsody Requirements in DOORS <-> Model elements in Simulink Model execution in Rhapsody <-> Model execution in Simulink

Table 6-1: Artefacts and interoperability for Functional Modelling.

6.3.2 Engineering Method: Performance Simulation

Performance simulation of the image pipeline enables on an empirical manner exploration of architecture design alternatives. In order to be able to implement this performance simulation, one needs performance models of the algorithms consisting of multiple tasks, and the hardware architecture in question. A task is an elementary part of an algorithm specified by parameters such as input data size, number of floating point operations, parallelism and output data size.

On the other hand, the number of operations can be analysed by counting the additions (or subtractions) and multiplications (or divisions) that are needed. With respect to memory, one analyses the amount of needed memory by counting the number of values that need to be present as temporal values. Besides the core algorithms and tasks, one needs to model buffers where needed. The detail level of the models depends on the depth of the performance simulation.

Algorithm

The performance model of an algorithm should specify the input and output parameters, the specific functionality modelled and optionally the settings of the algorithm. The performance is modelled based on these parameters. The use cases describe the performance measures of interest. The performance models should take into account the possible hardware specifications. Important aspects with regard to this performance modelling are the effect of parallelism. Conditions on when to start an algorithm (e.g. is all the input data available) and, is all the output generated.

An easy example of an algorithm consisting of multiple tasks is shown in Figure 6-11 where an image is normalised. Normalisations transforms an image with a minimal (Min) an maximum (Max) intensity into a new image with a new minimum (newMin) and maximum (newMax) intensity according to formula:

$$I_N = (I - \text{Min}) \frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

The first sub-task of this algorithm is to calculate the minimum and maximum intensity of the image. The next task is calculating the new pixel value for each pixel of the new image. Schematically this look like:

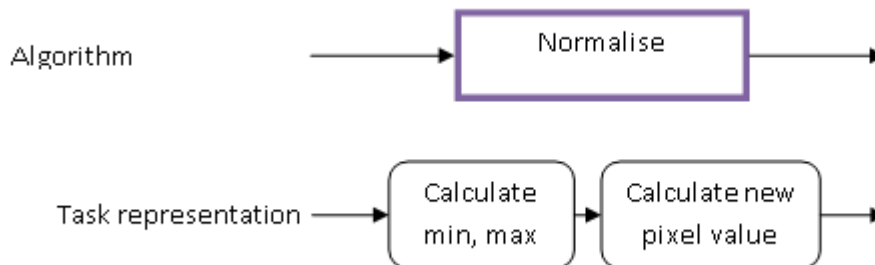


Figure 6-11: Schematic view of a normalisation algorithm.

In the normalisation example one can see that the first sub-task needs to finish before the second can start. One can also see that the first sub-task is finished only when it has read the whole image, and that the second sub-task can be performed in parallel for each pixel.

Another example is Scaling: in the image pipeline performance simulation scaling has to do with when the dimensions of the input image are resized to the dimensions of the display. In our analysis we assume a linear interpolation as a starting point.

Hardware

For the components of interest the hardware specifications are described in order to carry out the performance simulation. Particularly the configuration of the hardware must be known. Aspects of interest in the case of a processor are number of cores, clock frequency per core, memory size, cache size and bus speed. In addition the instruction sets of a particular processor must be taken into account in the performance simulation: Some processors are capable of performing various complex tasks in one cycle than other processors. This may directly affect the performance simulation analysis.

Buffers

In the image pipeline performance modelling, some algorithms may need to process the image after another algorithm has worked on the image and hence buffering is needed. When this buffering at the input or output ports is needed, it can be modelled as an extra delay.

Overview of artefacts and Interoperability

Input	Task map describing the elementary calculation steps of the algorithms including dependencies in in- and output. Hardware design describing the processing units, memory, interconnections, etc., etc., of the system.
Output	Performance metric of interest requested by the performance use cases, e.g. latency, ...
Tools	Rhapsody, MATLAB/Simulink.
Interoperability	Dynamic interface between Rhapsody and MATLAB/Simulink using compiled C-code.

Table 6-2: Artefacts and Interoperability for Performance Simulation.

6.3.3 Engineering Method: Combining Functional Modeling & Performance Simulation

In functional modelling, functional requirements and insights in the desired behaviour of the system are laid down in use cases, activity diagrams, state diagrams or sequence diagrams. Based on this, a decomposition of functions is proposed (as described in section 3.2). This is used as an input for performance modelling. Possibly, a set of function models is provided to performance monitoring for comparison.

Performance modelling combines the functional model with a hardware scenario. Without a hardware scenario, no performance numbers can be attached to performing a certain function. For example, what is the performance on a Pentium vs. a Quad Core processor? Possibly, a performance specification has been defined (as a technical specification) or a performance specification per function is defined (an architectural decision). This can be used as a success/fail criterion in Performance modelling.

Inputs	Functional requirements use cases (optional), hardware scenario(s), performance specification (optional).
Output	Best functional model, best hardware scenario, best function mapping onto hardware (optional), functional behaviour, performance compared to reference, performance bottlenecks.
Model dimensions	<ul style="list-style-type: none"> • 1 – n functional models. • 1 – n hardware scenarios. • 1 – n mappings of functions onto hardware elements. • 0 – n performance specifications.
Tools	Rhapsody, SimuLink
Interoperability	Requirements in Doors <-> Functional model in Rhapsody (available) Functional Model in Rhapsody <-> Performance model in Simulink (provided, not functioning yet). Hardware scenario in tool x <-> Performance model in Simulink (missing). Performance specification in Word <-> Simulink (available as static one way link; to be improved) .

Table 6-3: Combining Functional Modeling & Performance Simulation.

The performance is compared to a reference. This can be:

- Other functional model (which is the best?).
- Other hardware scenario (which is the best?).
- Performance specification (does the performance of the system or individual functions meet the specification?).

The performance model uses these inputs and a model of the functional decomposition is made that includes the necessary data, mass or energy inputs or flows, the transfer characteristics of the function and the resulting data, mass or energy outputs or flows.

Performance modelling provides feedback on functional behaviour (e.g. is a state change criterion met or not) and can provide a best functional model: the functional model with highest performance (in case several functional models are provided). This may lead to new insights into the desired functional behaviour and to requirement changes.

Performance modelling provides feedback on performance in comparison with a reference (e.g. 10ms latency compared to 8ms latency specification). It may also identify the most critical component in the hardware scenario or functional model (bottleneck). This can be used for a next iteration in functional modelling.

When technical specifications are met sufficiently, integrated functional modelling and performance modeling is stopped and architectural design can start.

Integrated functional and performance is subject to the following dynamics:

- Changes in functional requirements.
- Changes in desired behaviour.
- Changes in hardware specifications or availability.
- Changes in desired performance specification.
- Changes in mapping of functions onto hardware.

This leads to new iterations of the modelling.



7 Demonstrator description

As a first demonstrator, we have selected the combined engineering method Functional Modeling & Performance Simulation.

For the demonstration the function model design in Rhapsody will use the performance simulation from Simulink of the algorithms in the image pipeline. Also the interaction of the algorithms and the influence on performance parameters due to extra buffers will be simulated in Simulink. Currently the hardware is not designed in Rhapsody. The performance simulation needs multiple task mappings to make a trade off simulation. Here tasks can be algorithms of parts of algorithms mapped to a CPU or GPU or even to a core of processing unit.

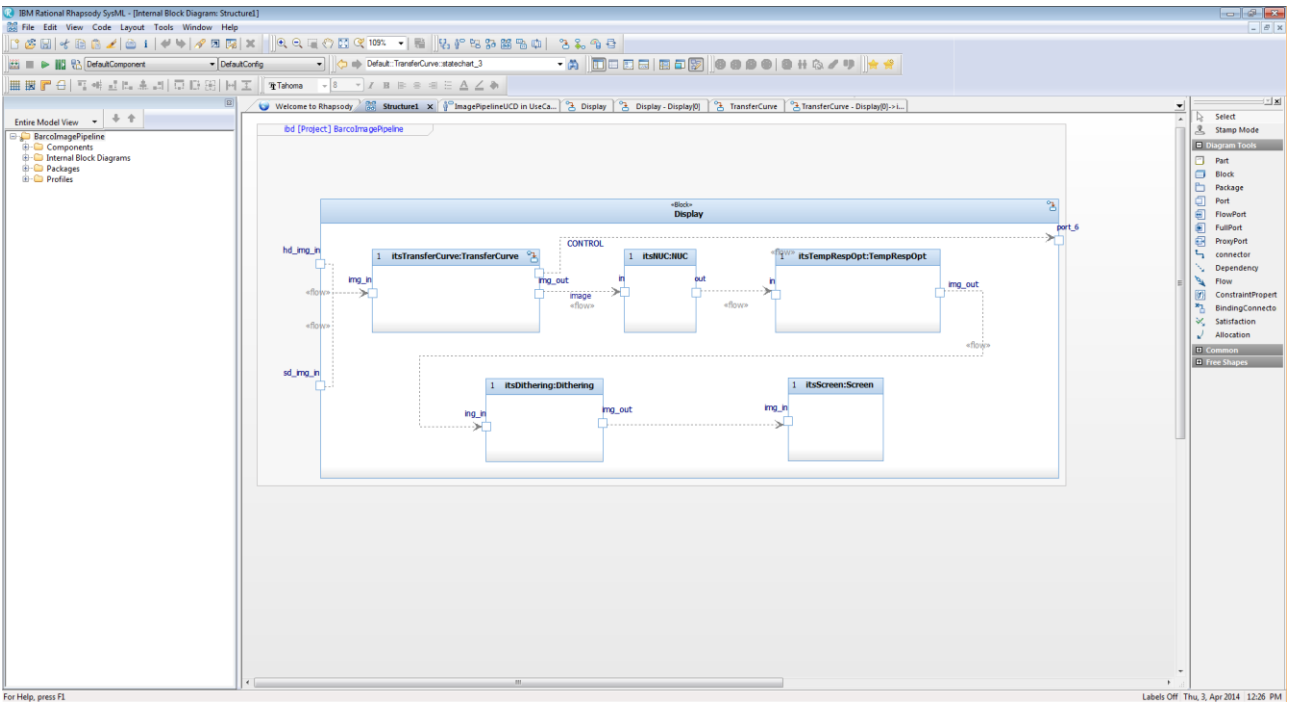


Figure 7-1: Functional Model of Image Pipeline design in Rhapsody.

The performance simulation in Simulink needs information on the current hardware design and the functional model. The performance of each function block will be simulated in Simulink. At the end the total performance will be calculated by Rhapsody.

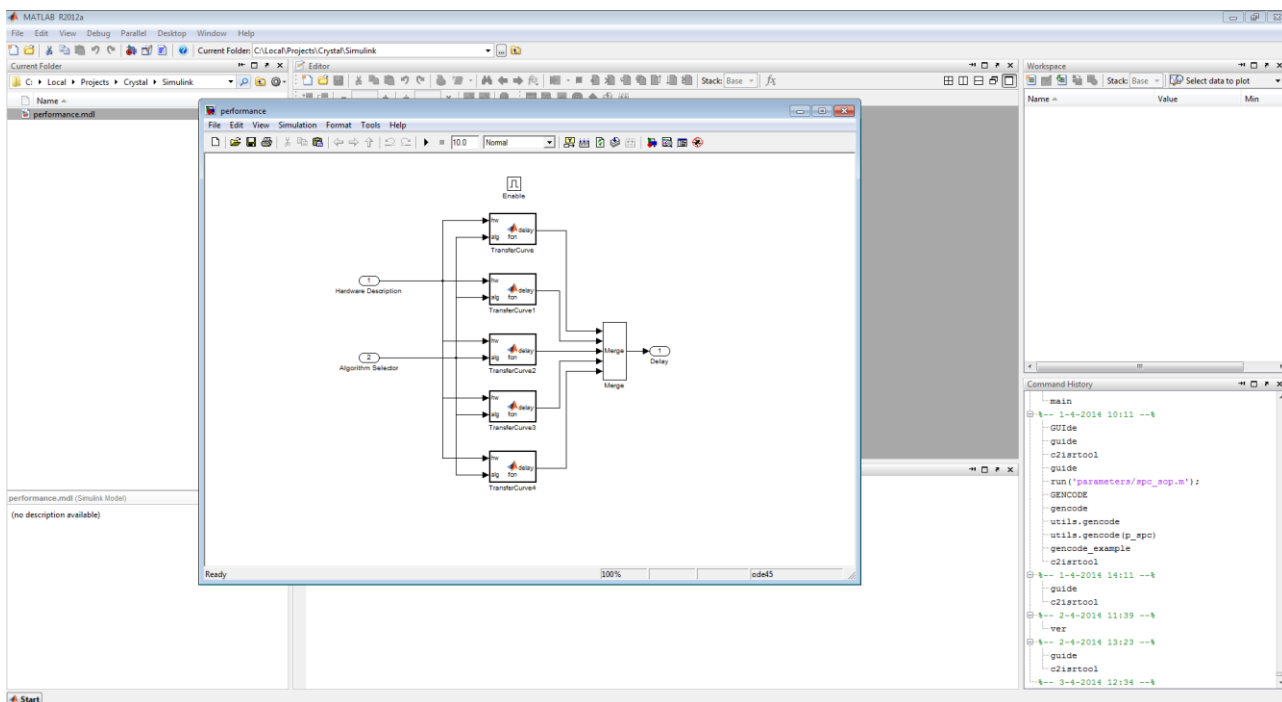


Figure 7-2: Performance simulation in Simulink.

As a result multiple functional designs and hardware designs can be simulated with the tool chain and the use case requirements can be tested. Due to the modular approach of the performance simulation, the performance simulation of the functional block can be replaced by real code for validation and verification. Using this technique also the performance models can be test and tuned to a higher level.

Currently we have not enough knowledge on the coupling between Rhapsody and Simulink to describe the demonstrator in more detail. We can define two parallel processes:

1. Functional modeling
 - a. Make functional model (Rhapsody).
 - b. Specify hardware design (Not performed in any tool).
 - c. Map functionality to hardware components (Rhapsody).
2. Performance simulation
 - a. Design performance models of hardware components (Matlab).
 - b. Design performance models of algorithms (Matlab).
 - c. Do performance simulation of functional model (Simulink).

While these two processes proceed and we gain more experience in the coupling between Rhapsody / Simulink, and Matlab the data exchange will be clear.

8 Conclusions and Way Forward

Section 8 gives an overview of the conclusion of the first implementation activities, the identified interoperability issues and the way ahead for the future planned activities for the next phase of the project.

8.1 Conclusion

The implemented engineering methods in the new software design process are now running for about 4 months, and effects are already noticeable:

1. The bug rate went down from around 1/KLOC to 0.4/KLOC during development.
2. Code coverage went up from 45% to 72% percent.
3. Documentation coverage went up from 42% to 100%.
4. Estimation accuracy went from ± 6 months to ± 2 months, and is still improving as we gather more statistics.
5. The software development process is in a good shape to become fully IEC 62304 compliant.

The first product that is defined is the F100, which aims at displays with a larger volume, but with a lot of pressure on the cost. The hardware was designed to be very cost effective with only minimal requirements.

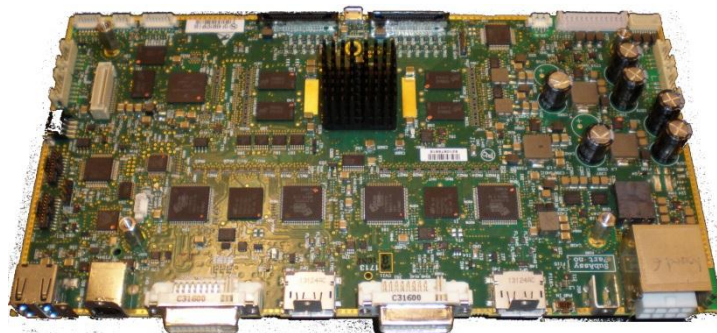


Figure 8-1: FUN100 F100 Board.

The result of this improved way of testing is that we now have implemented regression component tests for the F100 boards. These tests validate the actual image processing FPGA hardware implementation on a nightly base. These tests are also run for each software (test) release deliverable. This methodology will be used in the future to fully validate COTS solutions.

8.2 Interoperability issues

In general, the integration of the different tools went very smooth, especially open source tools. Integration problems are seen with commercial tools like PTC Integrity, Sparx Enterprise Architect, Microsoft Visual Studio, and so on. The next focus should be on finding a solution for integrating those third-party tools into the development chain in an open way.

Tool experiences with Rhapsody – Simulink combination

In the Crystal Barco use-case we are interested in coupling IBM Rhapsody with Mathworks Simulink for performance simulation of the software centric design of the image pipeline. We tested the coupling between Rhapsody and Simulink by using the available Rational Rhapsody and the Simulink integration tutorial.

We went through the example specified by Rhapsody step by step and resolved issues related to automatic source-code generation, compiler and linker version, build scripts, dependencies and licenses. The reason for this have to do with the possession of correct Matlab licenses, in combination with incompatible versions of the tools used in de coupling.

Currently the examples and tutorials work the best with 32 bit version of Rhapsody (8.0.5), 32 bit version of Cygwin.

There are different ways of integrating Rhapsody and Simulink, with different set of Mathworks licenses needed for each approach. It would be desirable to have a good overview of the integration types, the licenses needed, and a list of extended examples to do this.

Currently we are working on the functional model of the image pipeline in Rhapsody, including the coupling with Simulink. In Matlab and Simulink we are working on the analysis of the algorithms, mapping to hardware, and performance simulation. We are also investigating how to solve the remaining issues with regard to Rhapsody and Simulink integration.

Technical Refined Requirements

The input for the performance simulation is the functional model design in Rhapsody. The functional block representing algorithms and tasks are parameterized by their characteristics.

Also the hardware design together with the input signal is specified and modeled in the performance simulation.

The performance simulation returns the output in the form of the performance metric. In this use case one could think of the delay of an algorithm or task on a certain hardware design.

8.3 Way Forward

The following optimization and implementation activities are planned for the next phase of the project.

- 1.
2. Solve interoperability issues as part of the CRYSTAL WP6 activities.
3. Further optimization on Hardware and firmware development integration (simulation & test).
4. Replace Git and Gerrit with Atlassian Stash as this integrates better with Atlassian Jira.
5. Automate Testlink to automatically provide test results to Jira and back to PTC Integrity.
6. In a next phase the work for the work for the U100 will be defined in detail, which aims at our high end display range, with a limited volume.
7. For the FUN100 we want to switch all SCM activities to GIT to optimize the software development cycle even more.
8. Methodology to guarantee real-time execution of critical processing on platforms (WP6.3.11).
9. Consolidated Crystal variability management approach (WP6.10.1).
10. PLM interaction scenarios (WP6.8.9).
11. Document generation for IEC 62304.
12. Automatic regulatory compliance testing.

9 Terms, Abbreviations and Definitions

Table 9-1 provides an overview terms, abbreviations and definitions used in this document.

ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CAPA	Corrective Actions Preventive Actions
CI	Continuous integration
DDR	Double Data Rate
DICOM	Digital Imaging and Communications in Medicine
DVI	Digital Visual Interface
EM	Engineering Method
FDA	Food and Drug Administration
FMEA	Fault Tree and Effect Analysis
FMI	Functional Mock-up Interface
FPGA	Field-programmable gate array
FTA	Fault Tree Analysis
GSDF	Grayscale Display Function
HW	Hardware
IOS	Interoperability Specifications
IP	Image Processing
KLOC	1000 lines of code
LAN	Local Area Network
LUT	LookUp Table
MDD	Model Driven Development
NRE	Non-recurring engineering
OSLC	Open Services for Lifecycle Collaboration
PACS	Picture archiving and communication system
PCIe	Peripheral Component Interconnect Express
PTC	Parametric Technology Corporation (Crystal partner)
QA	Quality assurance
QA Web	Quality assurance service (Barco software)
QoS	Quality of Service
R&D	Research and Development
sRGB	Standard RGB colour space
SCM	Source Control Management (system)
SOUP	Software Of Unknown Provenance
SW	Software

Table 9-1: Terms, Abbreviations and Definitions.