OETCS/WP7/D7.3

openETCS

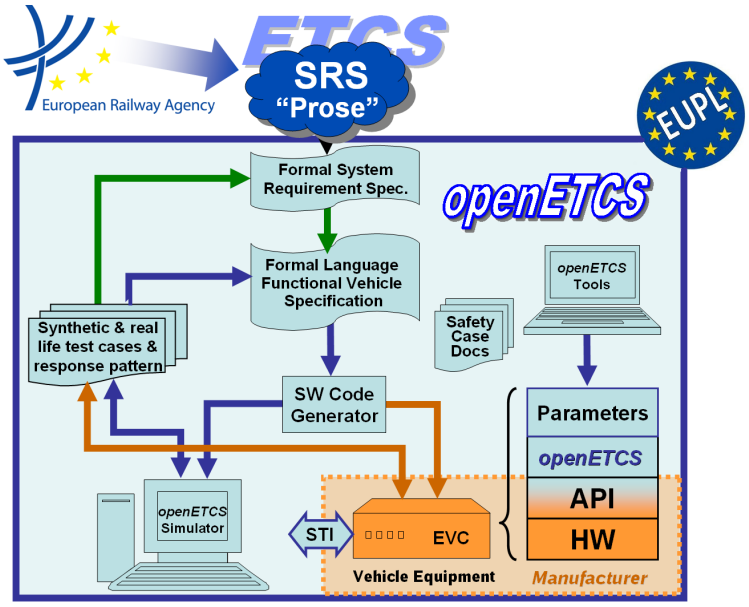
ITEA2 Project
Call 6 11025
2012 – 2015

Work-Package 7: “Toolchain”

Toolchain Qualification Process Description

Cecile Braunstein, Jan Peleska and Stefan Rieger

January 2014



Funded by:

Federal Ministry of Education and Research

Région de Bruxelles-Capitale

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

This page is intentionally left blank

Work-Package 7: “Toolchain”

OETCS/WP7/D7.3
January 2014

Toolchain Qualification Process Description

Cecile Braunstein and Jan Peleska

University Bremen

Stefan Rieger

TWT GmbH Science & Innovation
Ernsthaldenstraße 17
70565 Stuttgart
Germany

Qualification process description

Prepared for openETCS@ITEA2 Project

Abstract: This document presents different ideas of a toolchain qualification. It describes a process for the openETCS toolchain qualification.

Table of Contents

Document Information

Document information	
Work Package	WP7
Deliverable ID or doc. ref.	D7.3
Document title	Toolchain Qualification Process Description
Document version	01.04
Document authors (org.)	Cécile Braunstein and Jan Peleska (Uni.Bremen)

Review information	
Last version reviewed	01.00
Main reviewers	Marielle Petit-Doche, Michael JastramJ, Jaime Paniagua

Approbation			
	Name	Role	Date
Written by	Cécile Braunstein	WP7-T7.3 Sub-Task Leader	12.01.2014
Approved by			

Document evolution			
Version	Date	Author(s)	Justification
00.00	12.01.2014	C. Braunstein	Document creation
01.00	19.01.2014	C. Braunstein	Jan Peleska suggestions
01.01	28.01.2014	M. Jastram	Review
01.02	28.01.2014	C. Braunstein	Change from MPD, MJ JP suggestions
01.03	08.05.2014	S. Rieger	Provided first draft of a tool qualification example
01.04	08.05.2014	C. Braunstein	Provided another tool qualification example
01.05	27.05.2014	S. Rieger	Provided initial list of tool qualification scenarios
01.06	02.07.2014	S. Rieger	Added preliminary tool chain process visualisations
01.07	14.07.2014	S. Rieger	Model-based tool qualification, still incomplete

1 Introduction to Toolchain Qualification

1.1 Tool Qualification

The CENELEC EN 50128 standard [11] defines the tool qualification as follows:
"The objective is to provide evidence that potential failures of tools do not adversely affect the integrated tool-set output in a safety related manner that is undetected by technical and/or organizational measures outside the tool. To this end, software tools are categorized into three classes namely, T1, T2 & T3 respectively."

We recall here the different class definitions:

- Tool class T1: No generated output can be used directly or indirectly to the executable code;
- Tool class T2: Verification tools, the tool may fail to detect errors or defects;
- Tool class T3: Generated output directly or indirectly as part of the executable code.

The deliverable D2.2 [8] summarizes the requirements for the tool needed by the different tool classes. The report highlights that the effort differs depending on the tool class. Furthermore, for the most critical class T3, the evidence should be provided that the output is conform to the specification or that *any failure in the output are detected*.

The standard defined how to classify each tool individually (see [6, 7] as an example). But dealing with a tool chain, integrated within a tool platform, implies extra effort to ensure that the tool integration does not introduce new errors. For example mechanism such as artifacts versioning, time-stamping operations, etc ... should also be considered when qualifying the tool chain. This increased the number of tools to consider during the qualification process.

The effort of qualification depends on the number of tools under consideration, the tool classes and the tool error detection capabilities. To reduce the cost of the toolchain qualification and regarding the fact that our development imply regular releases, a systematic toolchain analysis approach has to be defined.

Table 1 exemplarily lists the tool classes for some of the tools employed in the openETCS project. The tools Papyrus, SCADE and Bitwalker have a direct effect on the generated code whereas ProR and Git do not. The classification for ProR would be different if it would be used for formal requirements that then could be automatically translated to a model (this would yield a T3-classification). The verification tools RTTester and CPN Tools are not in the "direct" chain from requirements to code but they may fail to detect errors in the software or model.

1.2 Toolchain Qualification State of the Art

Some recent works have been done in the field of toolchain qualification from a variety of projects. The next section summarizes the most significant ones.

1.2.1 Slotosh and al. (project RECOMP)

Tool	Purpose	Tool Class
Papyrus	Definition of the model architecture	T3
SCADE	Low-level modelling and code generation	T3
ProR	Requirements management	T1
Bitwalker	Generation of data structures for modelling	T3
Git	Versioning & Traceability	T1
RTTester	Model-based testing	T2
CPN Tools	Model checking and test case generation	T2

Table 1. Classification of some tools in the openETCS tool chain

[9] describes a model-based approach to tool qualification to comply with DO-330 and integration into the Eclipse development environment. The authors claim that the benefits of their method are the following:

Clarity: remove ambiguities;

Re-usability and Transparency: check for reuse in different toolchain;

Completeness: the model covers all parts of the development process and tis traceable;

Automation: Some part of the process may be automated.

Their method is explained in detail in [10]: the toolchain analysis is based on a domain specific toolchain model they have defined. This model is used to represent the toolchain structure as well as the tool confidence. Their goal is to deduce the tool confidence level and to expose specific qualification requirements. Furthermore, their idea is not only to check tool by tool but they follow a more holistic approach that makes use of rearrangement and/or the extension of the toolchain to avoid the certification of all tools. This allows them to reduce the qualification effort by focusing only on the critical tools and making use of already available information. Moreover, checks with respect to inconsistencies, such as missing descriptions, unused artifacts, etc., may be automatically executed on the toolchain model. Finally, automated document generation is addressed.

In [12] the application of tools and methods to an industrial use case to determine the potential errors in the tool-chain is described.

1.2.2 Asplund and al. (projects iFEST, MBAT)

The authors investigate the question if there exists part of the environment related to tool integration that may fall outside the tool qualification defined by the a norm (ISO 26262 here [2]). And if so, how tool integration is affected by ensuring functional safety. One conclusion is that the tool integration may lead to an increase of the qualification effort.

They also state that the standards (EN 50128, DO-178C and ISO26262) are not sufficient to check safety of a toolchain, but some part of a toolchain may be taken into account to mitigate the qualification effort. They highlight 9 safety issues caused by tool integration that also allow to be more exact when identifying software that have to be qualified for certification purpose.

They advocate to use take a “system approach” to deal with the qualification of tool integration within a toolchain. We should not think about individual tools anymore. Their system approach follows these steps:

1. Pre-qualification of development tools (requirements tools, design tools ...): provided by the vendors.
2. Pre-qualification at the tool-chain level: based on step 1 and reference work-flows; decomposition of higher level (project-wide) safety goals on tool level
3. Qualification at the toolchain level: check whether assumptions in step are fulfilled (use cases, environment, process) by the actual toolchain to be deployed.
4. Qualification at the tool level: based on the actual environment when deploying the toolchain.

This approach leads them to separate the parts required for software tool qualification and to identify safety issues related to tool integration.

In [1], they explore the step 2): identifying the required safety goals due to tool integration and obtaining a description of a reference work-flow and tool-chain with annotations regarding the mitigating effort. They propos to use the TIL language, a domain specific language for toolchain models. The model of the tool chain is used to perform a risk analysis and to annotate parts that need mitigating effort for the safety issues due to tool integration.

1.2.3 Biehl and al. (projects CESAR, iFEST, MBAT)

Biehl proposed a Domain Specific Language named TIL for Generating Tool Integration Solutions [5]. A toolchain is described in terms of a number of “Tool Adapters” and the relation between them.

- Tool Adapters: expose data and functionality of a tool
- Channels
  - ControlChannel describes service calls
  - DataChannel describes data exchanges
  - TraceChannel describes creation of a trace links
- Sequencer: describes sequential control flow (sequence of services)
- User: describes and limit the possible interaction
- Repository: provides storage and version management of tool data

This DSL allows early analysis of the toolchain. It may generate part of tool adapter code based on the source and target meta-model.

More recently, Biehl and al. define a standard language for modeling development processes as defined by OMG 2008. The language has been used in [4, 3] together with the TIL language to tailor a toolchain following a process model. The goal is to be able to model both the development process and the set of tools used. A process is defined as follows:

- Process: several Activities
- Activity: set of linked Tasks, WorkProducts, Roles
- A Role can perform a Task
- A WorkProduct can be managed by a Tool
- A Task can use a Tool

Using together the process development language and the toolchain language, in [3], the authors measure the alignment of a toolchain with a product development process. The method proceeds as follows:

1. Inputs:
   - formalized description of the toolchain design
   - description of the process including the set of tools and their capabilities
2. Initial verification graph
3. Automatic mapping links to the verification graph (acc. to mapping rules)
4. Apply alignment rule on the verification graph
5. Apply metrics to determine the degree of alignment between the tool-chain and the process

The metrics and the misalignment list provide feedback to refine the tool-chain design.

2 OpenETCS Toolchain Qualification Process

2.1 Toolchain Analysis

All the methods mentioned above start with a complete definition of the toolchain. In OpenETCS, the development of the toolchain follows an *agile* approach. Hence, for each (major) release we have to deal with an incomplete tool chain. In addition to the methods of the previous section, we need a qualification process that can adapt to the development speed, deal with an incomplete toolchain and can re-use qualification information.

Moreover, as stated by Asplund et al., the toolchain itself may provide some mechanism that may reduce the tool qualification effort [1, 2]. They are descibred as a set of safety goals that the tool chain should ensure. In our context, most of the tool integration effort is made by integrating tools into a tool platform. According to Asplund et al., the tool platform should ensure the following safety-goals that will avoid some extra tool qualification:

- Coherent time stamp information: common time stamps on development artifacts.
- Notification: the user should be notified when artifacts changed.
- Data integrity: avoid use of obsolete artifacts, the data used reflects the current state.
- Data mining: all data used by safety analysis should be available and be verifiable.

2.2 Scenario-based Qualification of Individual Tools

Qualifying a toolchain always requires the qualification of the individual tools it is comprised of. The effort required depends on the type and license of the tool to be qualified. To this end in the following we have identified a number of different scenarios that are of relevance in the context of the openETCS project as depicted in Figure 3.

In the following considerations for the individual scenarios are described.

2.2.1 Self-developed Tool

For a self-developed tool the project needs to provide the means for qualification and quality assurance. As the tool has not been employed in productive use by others the “proven in use” argument [Remark: It needs to be checked whether this argument actually applies to EN 50128] does not apply.

2.2.2 Tool Provided by a Tool Provider

For tools provided by a third party we must distinguish again two cases:

The tool is provided as closed source tool

If the tool is not pre-qualified by the tool provider the options are limited. If the tool provider is not willing or unable to provide the means for qualification, the qualification of the tool will

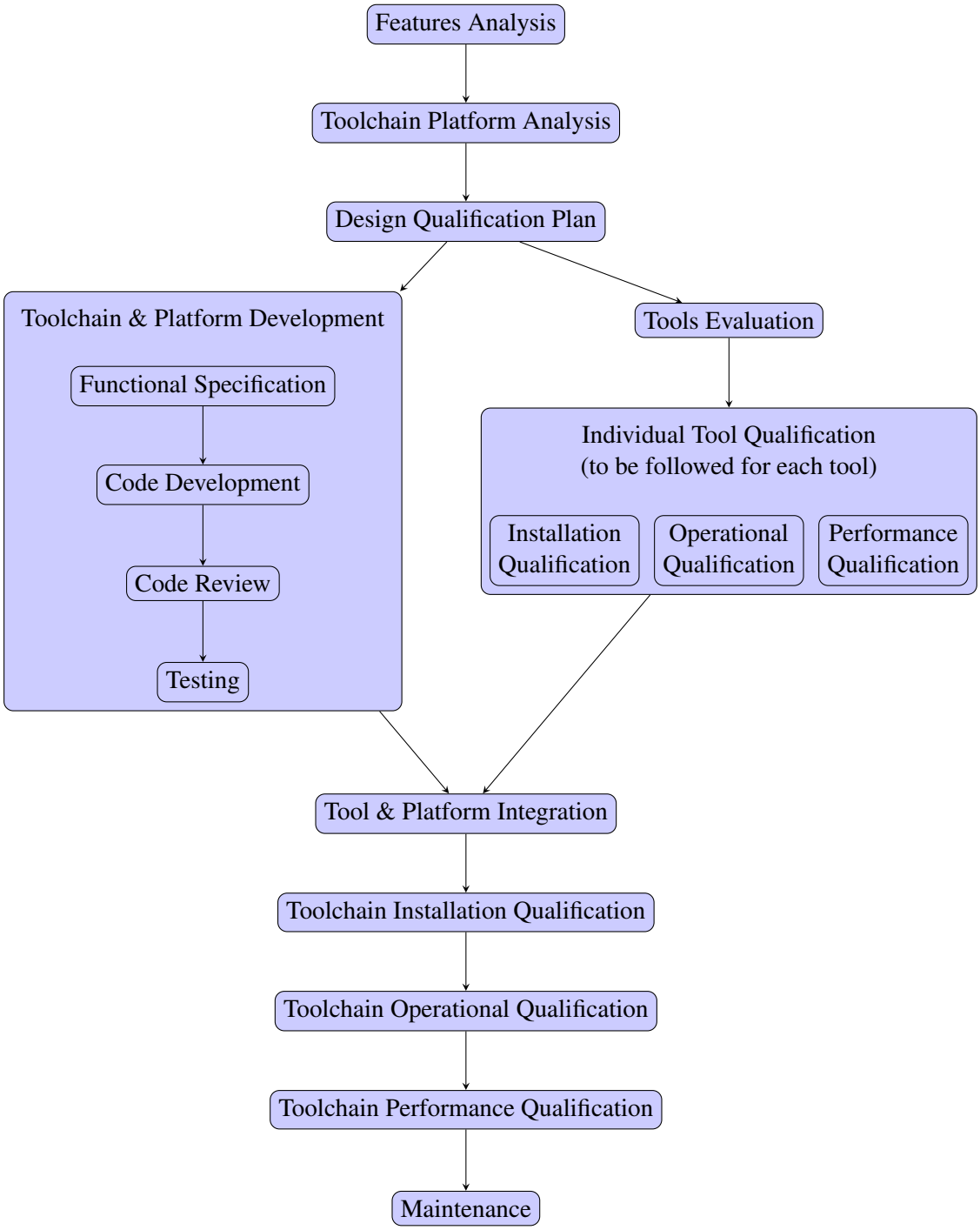


Figure 1. Qualification process as proposed by Izaskun de la Torre

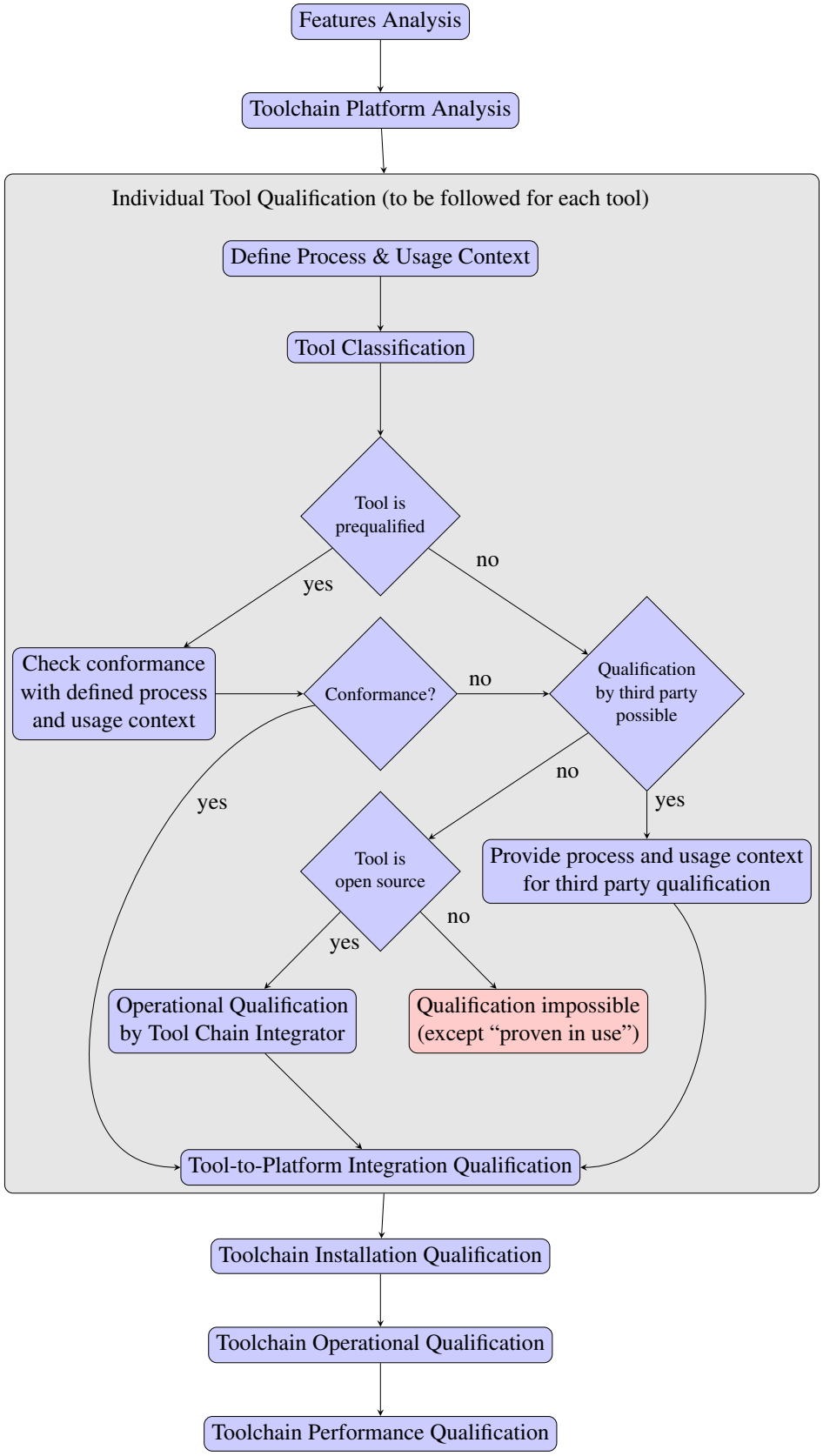


Figure 2. Proposed adaptation by Stefan Rieger (among other changes, the TC development branch was removed as this work is not part of qualification itself)

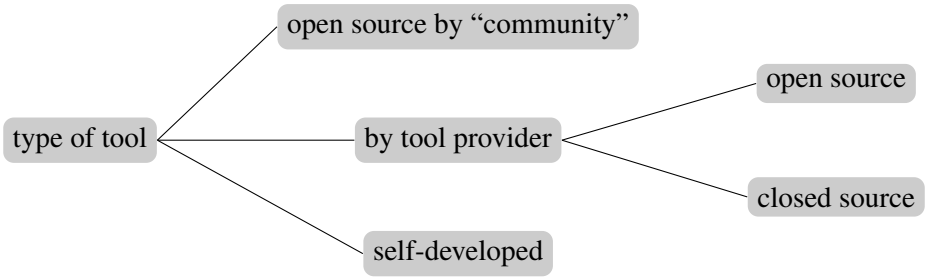


Figure 3. Scenarios to consider when qualifying individual tools

be almost impossible. One exception might be if the tool is already in use on a broad scale in industry and is generally regarded as reliable for the intended purpose. Also the tool class must be considered here.

The tool is provided as open source

In this case it would be possible to adapt the tool and analyse it to enable qualification. However, this involves a huge amount of effort which is best done by the tool provider who as a better insight into the tool's internal workings. Also a third party who has already experience with the tool or is a specialist with regards to qualifying safety-critical tools could be entrusted with the qualification task.

2.2.3 Open Source Tool Provided by the “Community”

If no tool provider can be identified but the tool is developed by the community and distributed under an open source license, the tool qualification has to be conducted by the project. However, many open source tools have a large development and expert community which can certainly be of help. In addition, the “proven in use” argument could possibly be applied (e.g., for tools like GCC).

2.3 Model-based Tool Qualification

This section describes a proposal on how to setup the operational qualification of individual tools. This step of the qualification process (cf. Figure **??**) is required for any tool that is not pre-qualified. With respect to the scenarios described in Section 2.2 it applies to self-developed and third party open source tools. In the case of closed source tools it is not possible to follow this process as the internal workings of the tools are hidden.

We propose setting up the operational tool qualification based on the approach by Slotosch et al. which has been briefly sketched in Section 1.2.1. In particular we will focus on the artifacts and documentation necessary to conduct the qualification steps. Figure **??** shows the different parts of the qualification model according to [9]. Some links are not directly evident in the paper but have been added where reasonable. In the context of openETCS the “Quality Assurance” part mentioned in [9] is covered by using the GitHub issue tracker and is thus not explicitly considered in the following.

2.3.1 Requirements Model

The requirements are represented on three different levels of abstraction interlinked with each other.

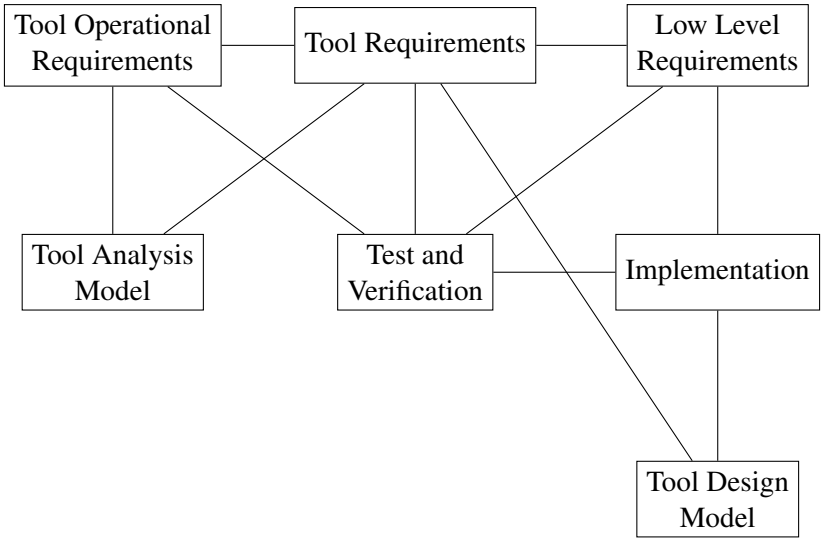


Figure 4. Parts of the qualification model and their linkage according to [9]

Tool Operational Requirements

The tool operational requirements (TORs) are derived from both, the safety standard applicable for openETCS, CENELEC EN 50128, and the operating environment of the tool based on the use cases. They determine the tool confidence level and the tool classification. Each tool operational requirement can be refined by any number of tool requirements.

Tool Requirements

Tool requirements break down the high level TORs down to architectural / design level. Thus, they are interlinked with the design model. Each tool requirement may be linked with any number of low level requirements.

Low Level Requirements

Requirements at code level that are linked directly with the code.

2.3.2 Tool Analysis Model

2.3.3 Tool Design Model

2.3.4 Implementation

2.3.5 Test and Verification

2.4 OpenETCS Toolchain Qualification Process

The OpenETCS tool chain will be defined by the set of its features and a guideline describing how to correctly use it. A SysML block diagram describes the tool chain architecture at a certain point in time as shown in Figure 4.

This block diagram is intended to grow according to new feature requests and the needs of openETCS participants. This diagram will be kept updated as a reference of our tool chain.

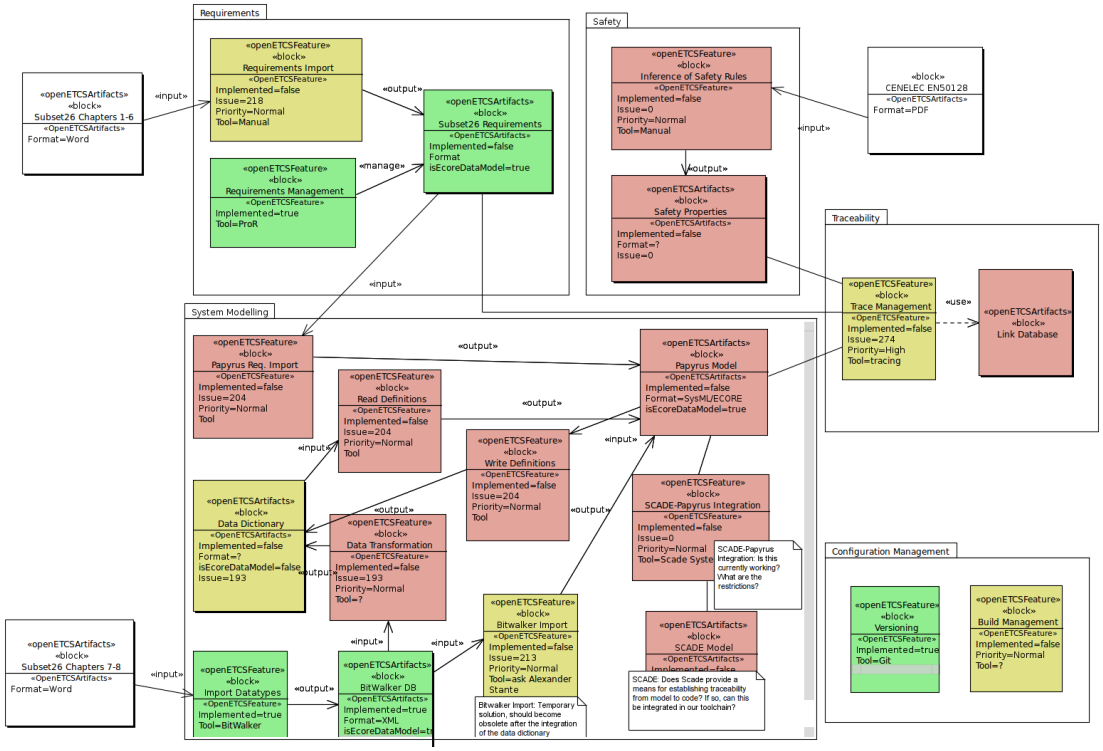


Figure 5. Tool Chain overview (20.02.14) –
Green Block: Implemented
Yellow Block: Work in Progress
Red Block: Not started
White Block: External Artifacts

In any case, the complete information regarding the feature availabilities may be found in the Eclipse product definition.

Each feature of the toolchain is a block with the profile “openETCSFeature” and each artifact is block with the profile “openETCSArtifact”. Each feature realizes at least one use case and may be implemented by one or more tools. Note that in the tool platform features may also be implemented as plug-ins. The diagram also imposes a (partial) order on the tasks. While some may be done in parallel, many tasks are dependent on others. Currently, the diagram neither highlights the use of the tools, nor the order of actions to be performed. This diagram should be completed by guidelines on how to use the tool chain and/or an activity diagram.

To mitigate the qualification process, we will first consider more than one feature at a time considering that errors from a tool A may be detected by a Tool B in the next step of the tool chain process. Furthermore, the toolchain is a collection of features and not only tools. This differs from Asplund and al. in the sense that some of the tool integration mechanisms that automate transformation of data, represent features and are thus not out of the scope of the qualification.

Due to our development process, a “pre-qualification” of tools should be made when integrating a tool.

This high-level list should be detailed.

Tool Integration Process for Qualification

- Define name and version
- Describe use cases
- Provide justification of the tool within the tool chain
- Provide input/output artifacts format (associated with the version)
- Integrate the tool in the SysML model
- Provide tool manual and other available documentation (associated with the version)
- Link with an issue tracker

One possible implementation is to represent all these in formations directly in the SysML model.

This high-level list should be detailed. What I would expect: Which artefacts exist; How they are connected; When artefacts have to be re-validated (e.g. due to changes); What roles exist; who is responsible for what; etc.

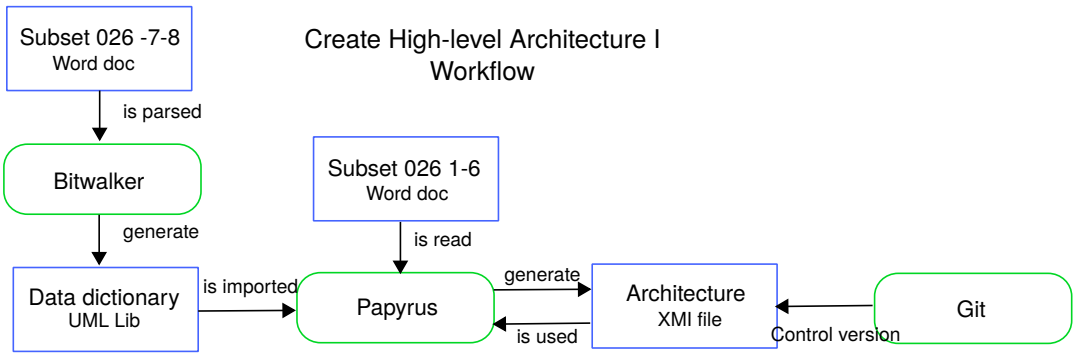
The Qualification Process

1. Feature Analysis
   - This step should assign a class to each feature based on the use cases.
   - Define the potential errors
   - Identify counter measure and/or error detection
   - For T3 tools 2 alternatives: certified compiler/generator or object code checker and/or exhaustive tester
2. Tool platform analysis
   - Provide evidence that the safety-goals mentioned in the previous sub-section are fulfilled[The identification of safety goals is missing in step 1]
3. Toolchain Analysis
   - Defines the work-flow
   - Identify the “hot spots” of the toolchain
   - Rearrange the toolchain if possible
   - Find new measures when needed with combining tools (redundancy with orthogonal codes . . .)
4. Toolchain qualification verification
   - Check consistency of tool version with manuals and other provided feature information
   - Generate table to check if all possible errors has a detection or a correction mechanism
   - Generate the qualification report

3 Tool chain Qualification Example

3.1 Example - Consideration for two openETCS Tool Chain

In this section we will illustrate the qualification process for the openETCS tool chain. As an example we will focus on the sample tool cahin depicted figure 5



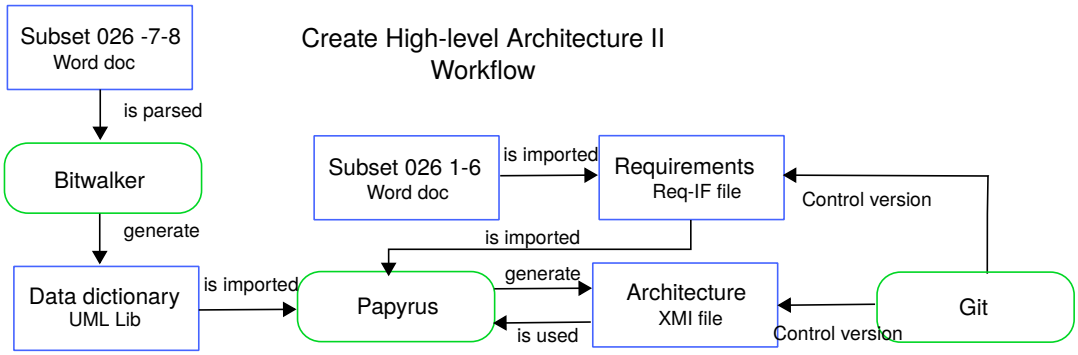


Figure 6. Tool chain samples

The first tool chain : Create High-level Architecture I proposes to create a high level view of the on board unit from the Subset 026 specification. It is composed by two features.

3.1.1 Qualification process for tool chain I

Step 1: Feature Analysis

Feature 1: Data type and Variable Import

Tool 1: Bitwalker

- Tool Identification: Bitwalker v1.0

- Tool Origin: Self-developed
- Tool license: EUPL
- Tool Justification:
- Tool Input/Output:
- Tool dependencies: None
- Use cases:
- Tool Class: T3
- Potential errors: see 3.3
- How to Provide Evidence: see 3.3

Tool 2: Data Dictionary

- Tool Identification: Data Dictionary based on SRS 0.1.201407031006
- Tool Origin: Self-developed
- Tool license: EUPL
- Tool Justification:
- Tool Input/Output:
- Tool dependencies: None
- Use cases:
- Tool Class: T3
- Potential errors:
- How to Provide Evidence:

Feature 2: System Modeling

- Tool Identification: Papyrus 0.10.1.v20130918
- Tool Origin: Eclipse Incubation (CEA)
- Tool license:
- Tool Justification: Papyrus is a modeling tool that allows us to model the high-level architecture of the OBU with SysML.
- Tool Input/Output:
- Tool dependencies:
- Use cases:
- Tool Class: T3
- Potential errors:
- How to Provide Evidence:

Feature 3: Artifacts Versioning

- Tool Identification: Eclipse Egit 3.2.0.2013121812
- Tool Origin: Eclipse foundation
- Tool license: Eclipse Public License Version 1.0
- Tool Justification: Egit provides an implementation of the Git version system.
- Tool Input/Output:
- Tool dependencies: JGit Core
- Use cases:
- Tool Class: T2
- Potential errors:
- How to Provide Evidence:

Tool platform Analysis

This should only be T1 feature ?

- List of services in Use : GUI ...
- Version Compatibility

Tool chain Analysis

- Tool dependency Analysis: All tool needed present with a compatible version (Need a matrix)
- Tool Artifacts interaction matrix: Use for compatibility format and to highlight interactions
- List of new Potential error due to interaction
- How to provide evidence

3.1.2 Qualification process for tool chain II

How to process changes, the different possible change within the tool chain are add, update, replace or remove a tool.

3.2 Example - Considerations for two openETCS Features

3.3 Bitwalker

	Bitwalker - Import of Data Types and Variables
Input	Subset-026-7, Subset-026-8, MS Word Format
Output	Database (XML) representing data types, variables and messages
Tool Class	T3

	The resulting definitions are direct input to the model and thus any fault may be propagated down to the code level
Safety Integrity Level	? [Currently I do not have the EN 50128 standard available]
Potential errors	The following list is not exhaustive and shall just give some ideas. The feature implementor has to provide more detail. • Variable / message / type not detected or missing in output Mitigation: This error can possibly be detected avoided as the modelling process is manual. Required in- or output described in the SRS can be added manually if missing. • Variable or message have wrong type (*) This is a very dangerous error, especially in the case of different precision integers or floats. The error may remain unnoticed and be propagated to the code leading to potentially fatal malfunctions. A mitigation is only possible by manual recheck (which would make an automatic conversion obsolete) of extensive/exhaustive testing or verification of the tool implementing the feature. However the inconsistent nature of the input document (Word) could prevent this. • Missing fields in record / message Mitigation: Can be detected if the functionality of the field is described in the functional description. • Wrong naming of variable / message / type (**) Mitigation: Potentially dangerous if names of variables (of compatible type) are mixed up. The risk is the same as for wrong typing. • ...
Providing evidence	It will be necessary to provide evidence that critical errors such as (*) and (**) can be detected or are not present in the tool. Exhaustive verification will be difficult due to the unreliable structure of the input document. The result must be correct also for ill-formed documents. A solution would be to let ERA acknowledge the converted XML database as a formal document. In this case the tool can be classified T1.

3.4 RT-tester qualification example

3.4.1 Tool identification

Name: RTT-MBT - Model-Based Test Generator

Version : 8.9-4.1.2

3.4.2 Tool Justification

The use of RT-tester allow to tests the C code produced by SCADE, B. It automatically generates tests for the following coverage criteria: basic control state coverage, transition coverage, MC/DC coverage and requirement coverage.

3.4.3 Use cases

Tool classification

Intended prurpose Automated generation of test procedures for embedded HW/SW con- trol systems

Output Test procedures to be executed in a test execution environment (both software testing or hardware-in-the-loop testing environment)

Input SysML test model (XMI format), test cases specification.

Tool Class T2: The tool may fail to detect errors or defects.

Use case description

1. **Generation** Generate test cases, test data and test procedures from model.
2. **Replay.** Replay test execution logs against the model.

Potential Errors

- Hazard 1: undetected SUT failures. The test oracles produced are incorrect and fail to observe a incorect behavior during test execution.
- Hazard 2: undetected coverage failures. The test execution does not cover what it should. The test will pass but the result will not covered what it supposed to.

References

[1] Fredrik Asplund, Matthias Biehl, and Frédéric Loiret. Towards the automated qualification of tool chain design. In *Computer Safety, Reliability, and Security*, volume 7613 of *Lecture Notes in Computer Science*, page 392–399. Springer, 2012.

[2] Fredrik Asplund, Jad El-khoury, and Martin Törngren. Qualifying software tools, a systems approach. *Computer Safety, Reliability, and \ldots*, page 340–351, 2012.

[3] Matthias Biehl. Early automated verification of tool chain design. *Computational Science and Its Applications–ICCSA \ldots*, page 40–50, 2012.

[4] Matthias Biehl and M Törngren. Constructing tool chains based on SPEM process models. In *The Seventh International Conference on Software Engineering Advances (ICSEA2012)*, page 267–273, 2012.

[5] Biehl, Matthias, El-Khoury, Jad, Loiret, Frédéric, and Törngren, Martin. A domain specific language for generating tool integration solutions. In *In 4th Workshop on Model-Driven Tool & Process Integration*, 2011.

[6] Jörg Brauer, Jan Peleska, and Uwe Schulze. Efficient and trustworthy tool qualification for model-based testing tools. In Brian Nielsen and Carsten Weise, editors, *Testing Software and Systems*, volume 7641 of *Lecture Notes in Computer Science*, pages 8–23. Springer Berlin Heidelberg, 2012.

[7] Wen-Ling Huang, Jan Peleska, and Uwe Schulze. Test automation support. Delivrable D34.1, COMPASS, January 2013.

[8] Merlin Pokam and Norbert Schäfer. Report on CENELEC standards. Requirements D2.2, openETCS, April 2013.

[9] Oscar Slotosch. Model-based tool qualification : The roadmap of eclipse towards tool qualification. *Springer*, 2012.

[10] Oscar Slotosch, Martin Wildmoser, Jan Philipps, Reinhard Jeschull, and Rafael Zalman. ISO 26262-tool chain analysis reduces tool qualification costs. *Automotive 2012*, 2012.

[11] European Standard. *Railway applications-Communication,signalling and processing system- Software for railway control and protecton system*. CENELEC EN 50128. DIN, October 2011.

[12] Martin Wildmoser, Jan Philipps, and Oscar Slotosch. Determining potential errors in tool chains: strategies to reach tool confidence according to ISO 26262. In *Proceedings of the 31st international conference on Computer Safety, Reliability, and Security*, SAFECOMP'12, page 317–327, Berlin, Heidelberg, 2012. Springer-Verlag.