# Model-Based Tool Qualification

## The Roadmap of Eclipse towards Tool Qualification

Oscar Slotosch

Validas AG, Munich, Germany

slotosch@validas.de

**Abstract.** In this paper we describe the model-based approach to tool qualification starting from the process model for the determination of the qualification need until the model for test and qualification. The model-based approach can automate many steps from checking the syntactical requirements completeness until the determination whether all requirements have been implemented and successfully tested. Many required documents like the “Tool Requirements Specification” or “Tool Test Specification” can be generated from the model. The model-based approach has been shown to fulfill all requirements from the DO-330 standard which describes tool qualification for avionic, automotive and other industries. Therefore the Eclipse Foundation has chosen this standard and proposed a roadmap to provide support for the development of qualifiable Eclipse-based tools. This paper describes the model-based approach and the roadmap of Eclipse to support this process.

**Keywords:** Tool Qualification, Model-based, Eclipse, Roadmap, DO-330

## 1 Introduction

The amount of software in our world increases very fast. The usage of tools in the development of this software in systems increases also. Therefore the correctness of the software does not only depend on the development process of the software but also of the tools used for the development of the software. For example static analyzers can check the reachability of code or some potential errors in the code that had to be detected using reviews. Test generation tools can automatically generate tests that completely cover formalized requirement specifications and simulators allow to verify software without their surrounding systems. However those tools can introduce or hide safety relevant errors in the software.

This fact is reflected new standards for the development of safety critical systems as the ISO 26262 [ISO26262], the DO-178C / DO-330 [DO330] and the IEC 61508 [61508]. These standards require to analyze all tools that are used within the development process of the software. This includes also the integration and verification of the software. All these standards have a three phase approach for using tools safely:

1) Classification: the tools are classified into classes that describe the confidence (certification credit) they require in the development process of the system. The classification is based on the analysis of potential errors in the tool and their detection or prevention probability within the process. Note that the confidence classes for tools differ among the different standards: tool confidence levels in the ISO 26262, tool criteria in the DO-178C and tool classes in the IEC 61508. Tools that do not require confidence since they have either no impact or a high detection probability for all their potential errors in the process can be used without qualification in the analyzed processes.
2) Qualification: Tools that require confidence in the analyzed processes have to be qualified. Qualification might be restricted to the identified use cases and to show the absence of critical errors. In the ISO 26262 there are many qualification methods suggested (proven in use, process assessment, validation and development according to a safety standard). All qualifications methods require a tracing from the use cases to the known bugs and mitigations. The safety standards are the clearest approach, since they prescribe all actions in detail which does not leave so much room for interpretations and makes the tool qualifications better comparable.
3) Usage: The tools can be used according to the known or found restrictions in the development process. There should be a documentation that contains the constraints from the process that have been considered in the analysis phase and workarounds for all restrictions found during tool qualification.

The tool qualification (required for certification) has been considered to be difficult since there were many unclear specified steps in the standards that left much space for interpretation and since the amount of requirements that have to be considered manually was very high.

In this paper we present a model-based approach for tool qualification. This has the following advantages:

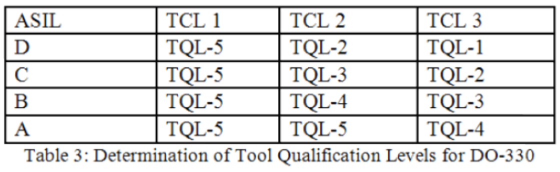
- Clarity: the model has precisely defined elements and leaves (together with it's documentation) not much space for interpretation,
- Reusability and Transparency: since the model clearly states which requirements and functions have been qualified the qualifications can be easily checked for their reuse in different tool chains,
- Completeness: the model covers all phases in the development process and has been successfully traced against all requirements in the DO-330 standard and
- Automatization: the model can be automated in the following ways:
  - Consistency and completeness checks,
  - Inference of confidence requirements from the process model part,
  - Generation of documents from the model and
  - Integration into development environments.

Therefore the model-based approach can reduce the qualification (and certification) efforts dramatically and reduces tool qualification from a research topic to the essence of correct software development which is well known since many years.

This paper presents the model-based approach that has been proposed for the integration into the Eclipse development environment but can also be used in a stand-alone version for tools implemented using other IDEs or programming languages.
The model-based approach consists of the description and the model for the qualification data. The paper is structured as follows: Section 2 describes the purpose and the structure of the DO-330 standard. Section 3 describes the structure of the model and different parts of the model are presented in Section 4, while Section 5 roughly describes the tool qualification process. Section 6 describes the roadmap to enable Eclipse to support the model-based approach in an optimal way by integrating this model into the meta-model of Eclipse plugins. Section 7 describe the used support tools and Section 8 summarizes the approach.

## 2 DO-330

The DO-330 is called "Software Tool Qualification Considerations". It has been created to factor out the tool qualification topic from the standards DO-178C and DO-278A and it is also applicable to automotive and other applications as well. Since these other applications have different risk classes (for example ASILs in ISO 26262) there is an interface to other standards and processes in the DO-330. It is called the tool qualification level (TQL). There are five different TQLs (TQL-1 to TQL-5) that have different rigorous approaches. TQL-1 is the most rigorous level with the most requirements. The decision which TQL shall be used for which standard to create sufficiently tool confidence depends on the risk class and the determined qualification need. This mapping from tool criteria and risk class has to be defined in every standard. DO-178C and DO-278A contain such mappings (see **Fig. 1**), the ISO 26262 currently has no such mapping, since it appeared some weeks after the DO-330. A possible mapping for the ISO 26262 from ASILs and the tool confidence levels (TCL) to the DO-330 TQLs would be the only necessary adaptation of the ISO 26262 to use this standard with all it's positive aspects mentioned in the previous section. Such a mapping could be as defined as proposed in **Fig. 1**.



| ASIL | TCL 1 | TCL 2 | TCL 3 |
|---|---|---|---|
| D | TQL-5 | TQL-2 | TQL-1 |
| C | TQL-5 | TQL-3 | TQL-2 |
| B | TQL-5 | TQL-4 | TQL-3 |
| A | TQL-5 | TQL-5 | TQL-4 |

Table 3: Determination of Tool Qualification Levels for DO-330

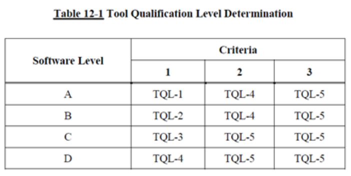
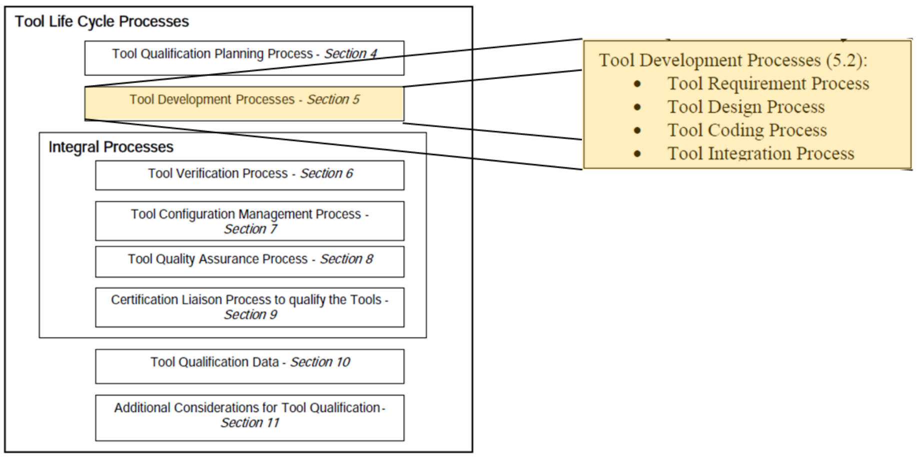


Table 12-1 Tool Qualification Level Determination

| Software Level | Criteria | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| A | TQL-1 | TQL-4 | TQL-5 |
| B | TQL-2 | TQL-4 | TQL-5 |
| C | TQL-3 | TQL-5 | TQL-5 |
| D | TQL-4 | TQL-5 | TQL-5 |

**Fig. 1.** : TQL-Mappings for ISO 26262 (proposed) and DO-178C

The structure of the DO-330 is according to the processes that shall be applied to develop and qualify tools. As every safety standard it does not prescribe a concrete process but poses requirements to processes that have to be satisfied. The structure of the DO-330 is depicted in **Fig. 2**. It contains processes in sections and sub-processes in subsections. The requirements within the DO-330 can be identified quite well using the numbers in it's sections and enumerations.



Tool Life Cycle Processes
Tool Qualification Planning Process - Section 4
Tool Development Processes - Section 5
Integral Processes
Tool Verification Process - Section 6
Tool Configuration Management Process - Section 7
Tool Quality Assurance Process - Section 8
Certification Liaison Process to qualify the Tools - Section 9
Tool Qualification Data - Section 10
Additional Considerations for Tool Qualification - Section 11
Tool Development Processes (5.2):
Tool Requirement Process
Tool Design Process
Tool Coding Process
Tool Integration Process

**Fig. 2.** Structure of DO-330

While the ISO 26262 determines the tool confidence level based on a detailed analysis of use cases, potential errors and applied checks and mitigations the DO-178C and IEC 61508 have a rather fixed classification from the tools into tool criteria and tool classes. However the detailed analysis of the ISO can be applied also for the other standards, since it is a typical safety technique. The DO-330 states in FAQ D.3 that the tool criteria classification can be reduced by analyzing all uses cases of the tool and their potential errors. Therefore we integrated this approach into the model and we can reduce the tool criticality with this analysis for example by using redundancy.

## 3 Model-Based Qualification Approach

Model-based development is increasingly used in more and more processes, for example model-based code generation, model-based testing, model-based risk analyses, model-based design, documentation models, etc. The success of the models is caused by the appropriateness of the models, the automatized model analyses and the outputs of the models.

The Eclipse development tool (and other IDEs) are based on a meta model. It consists of classes, packages, etc. The Eclipse meta model covers also many design aspects using the plugin architecture, their export and import interfaces and contributions of other plugins to tool's functionality. The Eclipse modeling framework allows to generate code from design models. Also the integration process is specified using a meta model. Therefore the current model of Eclipse covers some aspects to satisfy the DO-330 (parts of architecture, design and integration) already. It can be depicted in **Fig. 3**. Note that the development process does not need to be a strict V, but can be differently organized. This meta models have been proved useful in the development with Eclipse, however they cover only small parts of the DO-330. Therefore we propose to extend the current meta model by new elements to cover also the missing phases of the DO-330.

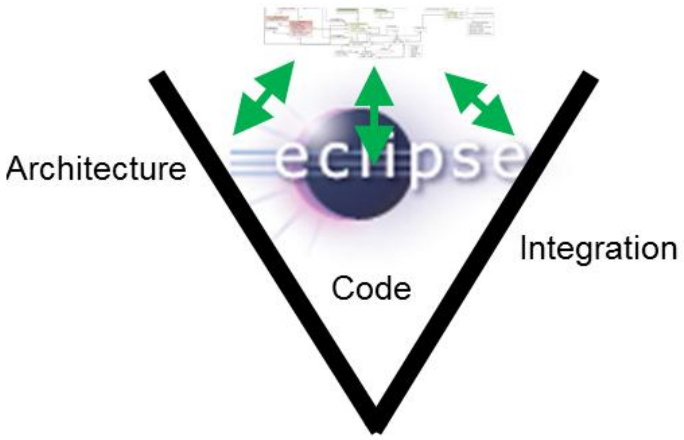


**Fig. 3.** Meta Model and Process Phases of Eclipse

In addition to the extended meta model we also need some instructions how to use the model correctly. These are documented in the following three documents:

- Howto Qualify Eclipse-based Tools [HowTo]: Contains the liaison process between the authority and the developer and some other steps like the application of the qualification kits. Furthermore it contains a complete tracing from all requirements in the DO-330 to the documents describing the model.
- Development Plan for every Qualifiable Eclipse Plugin [TDP]: Describes how to develop the plugins and how to model development artifacts to satisfy the corresponding DO-330 requirements
- Verification Plan for every Qualifiable Eclipse Plugin [TVP]: Describes the generic verification activities and artifacts.

Note that these generic documents can be applied for every plugin of Eclipse. The plugin specific elements (plugin requirements, plugin design, code, tests,..) are modeled as described in the generic documents. All plugin specific documents (from the tool analysis in the plan of software aspects in certification (PSAC), until the verification reports can be generated from the meta model and the integrated verification tools.

The extended meta model covers the tool operational requirements (use cases), the tool requirements (functions and architecture) and the low-level requirements (code documentation in Eclipse) and the verification and qualification data. Also the other processes are covered from the model (see Section 4).

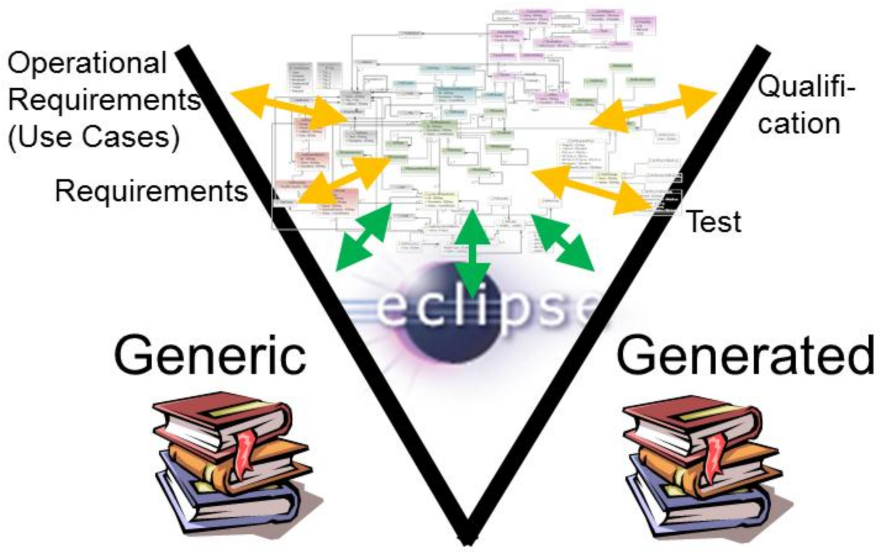


**Fig. 4.** Extended Meta Model, Documents and Process Phases of Eclipse

## 4 The Tool Qualification Model

The tool qualification model contains all data that are required for tool qualification. It is modeled with the Eclipse Modeling Framework [EMF] in a class diagram. Every artifact required for qualification is contained as a class in the qualification model. The tracing between the artifacts is modeled using associations. This allows a tracing from the process model for the usage of the tools via a stepwise refinement into requirements, code and test cases. Verification data classes for reviews and tests are added to the model and have to be populated from the verification activities. Even if the model is integrated in one big class diagram, it needs to be presented in several parts corresponding to different process areas of the DO-330. Therefore the generic tool development and the verification plans (see Section 3) have the following parts that contain the description of details (attributes and relations) of the model:

- Tool Analysis Model: It describes the artifacts and potential errors and error classes for the automated determination of the required tool confidence, see Section 4.1,
- Tool Operational Requirements (TORs): It describes the different TORs required from the DO-330 and the assumptions for the user, see Section 4.2,
- Tool Requirements (TRs): It describes the different kind of requirements of the tool,
- Tool Design: It contains the elements to describe the architecture of the tool and integrates many design elements of plugins, EMF, xText, packages, ecore diagrams, etc.

- Low Level Requirements (LLRs): It contains the LLRs that are an extension of the Javadoc formatted comments within Eclipse code,
- Implementation: It contains stubs to refer to the existing code in Eclipse like classes, methods etc. The stubs will be replaced with the existing models, once the model has been integrated into Eclipse by QPP (see section 6),
- Qualitiy Assurance: It contains the models of existing problem reports, their severity and the relations to tests and potential errors. This model has to be filled from problem reporting system and
- Test and Verification: They describe the model to represent tests, reviews, their results and relations to the requirements, etc.

In this paper we present only some parts of the model in the following subsections. The model has been designed as simple as possible to satisfy the DO-330 using simple strings for the description of the elements like requirements, test cases, problem reports, etc. This can be the basis for the integration of more sophisticated concepts like requirement templates, test specification techniques etc.

## 4.1 Tool Analysis Model

The tool analysis model is used to support the determination of the tool qualification level for the plugin based on an analysis of the functions of the plugin their potential errors and mitigations. The model supports a systematic way to derive the potential errors (black-box and white-box) and allows to compute the TQL automatically (see the tool chain analyzer tool for that purpose in [TCA]) as described in the ISO 26262 and allowed in the DO-330. The model is depicted in **Fig. 5**.

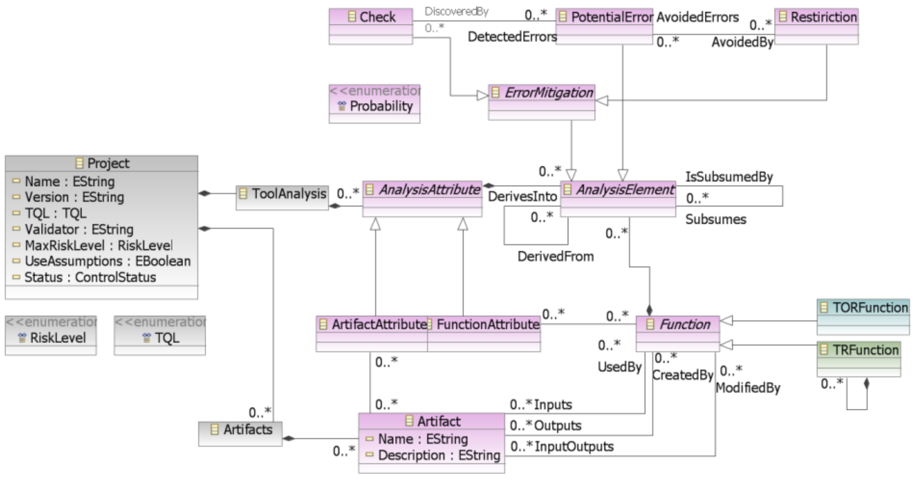


**Fig. 5.** Tool Analysis Model

The tool analysis model is depicted using pink color. It is contained in the plugin project model (gray color) in two containers: ***Artifacts***: contains all artifacts used by this plugin and ***ToolAnalysis*** containing the potential errors, their mitigations etc. Both containers are contained in the ***Project*** model that represents the information of the plugin project required for tool qualification, for example the ***TQL*** and the maximal risk level (***MaxRiscLevel***). The ***TQL*** is computed as described in [ISO26262], while the maximal risk level is a uniform interface to the risk levels of the different standards (Risk class in DO, SIL in IEC 61508 and ASIL in ISO 26262) and is used to compute the TQL from the maximal required confidence according to the table in **Fig. 6**. Note that some standards might have deviations from this table, for example in ISO 26262 a low confidence need (TCL 1) requires no tool qualification.

| Confidence Need | No Impact | LOW | MEDIUM | HIGH |
|---|---|---|---|---|
| Risk Level 1 | No TQL | TQL 5 | TQL 2 | TQL 1 |
| Risk Level 2 | No TQL | TQL 5 | TQL 3 | TQL 2 |
| Risk Level 3 | No TQL | TQL 5 | TQL 4 | TQL 3 |
| Risk Level 4 | No TQL | TQL 5 | TQL 5 | TQL 4 |
| No Level | No TQL | No TQL | No TQL | No TQL |

**Fig. 6.** Determination of the TQL

In contrast to [WPJSZ12] where the functions are guessed manually from an assumed structure or the user manual of the tool, the analysis model in **Fig. 5** has a well-defined interface to the functions developed. The used functions in a plugin are modeled as tool operational requirements (***TORFunction***) and refined to tool functions (***TRFunction***) during the requirements engineering. Those two parts are modeled in different colors (blue and green). Both elements have a common analysis interface ***Function*** that is used to assign the input/output artifacts to the functions and to assign the attributes that characterize the functions (***FunctionAttribute***) to them. Beside the ***FunctionAttribute*** elements that characterize a function (white box strategy), a function can also be analyzed as black box by just considering it's output ***Artifact***s. The ***Artifact***s are also characterized by attributes (***ArtifactAttribute***). Both ***FunctionAttribute***s and ***ArtifactAttribute***s are ***AnalysisAttribute***s and contain the following ***AnalysisElement***s: ***PotentialError***, ***Check*** and ***Restriction***. ***AnalysisAttribute***s can be derived from other ***AnalysisAttribute***s and can be subsumed to simplify the mitigation. Furthermore assumptions can be used in plugins (***UseAssumption*** in ***Project***) and the analysis (***Assumption*** in ***ErrorMitigation***) to model constraints the user of the tool has to respect.

The required confidence of a plugin is determined automatically from the given model as follows: all functions are characterized by function attributes and artifact attributes of their outputs and each analysis attribute has a set of typical errors. Those errors have to be assigned to mitigations that have ***Probability*** elements to detect or prevent the error if it would occur. The best mitigation for an error determines the probability to mitigate the error, while the worst error mitigation probability determines the confidence need as follows:

- If all errors in a plugin have a **high** mitigation probability the plugin has LOW qualification need / or confidence level,

- If at least one error has a **medium** mitigation probability the plugin has MEDIUM qualification need / or confidence level and
- If at least one error has no or **low** mitigation probability the plugin has HIGH qualification need / or confidence level.

The tool transition criteria (see Section 5) ensure that the functions are analyzed systematically within the tool qualification alpha state.

## 4.2 Tool Operational Requirements

The TORs describe how the tool is operating in it's environment. They can be seen as a description of the use cases of the tool. The provider of a plugin describes the developer use cases (from his point of view), while the user of the tool can use those unchanged or adapt it to his needs. The TORs are the top level requirements. The model of the TORs and some related elements is depicted in **Fig. 7**. Note that there are further requirements, not depicted in this overview, for example to describe interfaces, the architecture etc.

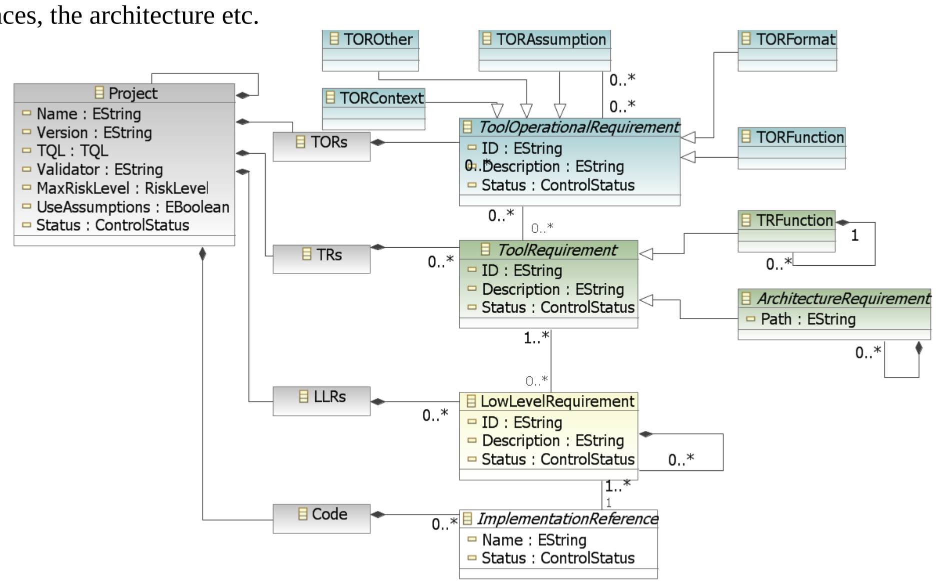


**Fig. 7.** Tool Operational Requirements Model

The TOR model is structured similar to the tool analysis model. All elements are in containers that are contained in the ***Project*** element that contains the plugins qualification data (grey color). The TORs (blue) consists of an abstract class ***ToolOperationalRequirement***. It has different sub classes for different kinds of requirements required from the DO-330. For example the functional operational requirements (***TORFunction***) are required to be considered in the tool operational requirements definition process (DO-330-5.1) in the consistency verification step (DO-330-5.1.2.b) and shall be contained in the tool qualification plan (DO-330-10.1.2.c). The format requirements (***TORFormat***) are required from the DO-330 in section 10.3.1 where the formats of input and output files in the operational environment shall be contained. The ***TORAssumption*** class is used to model assumptions to the usage of the tool, for example to avoid some constructs or to perform checks to verify the output of the tools. The assumptions are the interface from the tool to the user of the tool to ensure a save development process. Furthermore there are operational requirements on the context in which the system should run (***TORContext*** for example required by DO-330-10.3.1.b) and there are other requirements (***TOROther***) that can be used to model other operational requirements of the tool.

All TORs (use cases) have to be mapped to tool requirements (functions) are that implemented in the tool using LLRs (***LowLevelRequirement***) and have references to the code (***ImplementationReference***). The architecture requirements (***ArchitectureRequirement***) are modeled as special cases of tool requirements (***ToolRequirement***). These elements are modeled like the TORs. However **Fig. 7** shows only the abstract elements to illustrate the structure. More details can be found in the tool development plan.

## 5 Tool Development and Qualification Process

The tool development process is not restricted except that it is required to use the model to document the development steps. This can be done before, during or after the development of the code, thus supporting also the qualification of existing tools and plugins. However it is recommended to stick to established software-engineering processes.

The qualification process makes use of the tool qualification model. Especially the tool life cycle and the transition criteria can be determined automatically based on the model. The transition criteria define different development stages of the tool/plugin from the qualification point of view. This has the advantage that the well working development processes of the Eclipse community do not need to be changed, but they are enriched by an addition attribute the "qualification stage" that is independent from the other processes. Qualified tools/plugins must have the stage "Qualification Release"

The following qualification stages are defined for each plugin and have a formal definition that is based on the qualification mode:

- **Unqualified Pre-Alpha Release**: The plugin is an undefined, unknown qualification state, for example if the model is missing,
- **Qualification Alpha-Release:** The TORs are defined and TQL is determined for the plugin. It is "analyzed",
- **Qualification Beta-Release:** All requirements (TORs and TRs) are described and have links to LLRs and Code. The plugin is "feature complete",
- **Qualification Release Candidate:** All required verification steps are defined. No open bugs of the category Blocker are available. The plugin is "Verification Defined" and

- **Qualification Release:** Verification has been successfully executed and is documented within the qualification kit. The plugin is "Successfully Verified".

The qualification stages are based on each other, i.e. the qualification beta-release requires the alpha release and additional constraints. If some elements, e.g. requirements are changed this invalidates the verification data that bases on this element such that the qualification stage is reduced.

The transition criteria are defined on the qualification data using their attributes and relations from the qualification model. For example the transition into the alpha release requires (among other similar checks) that the qualification model of the plugin has at least one ***TORFunction*** element defined in the plugin and that this has a nonempty name and description and the CM status is reviewed. The TORFunction element should have an Artifact element assigned as input or output and at least one potential error. All details of the transition criteria are described in the tool development plan.

The qualification process for a tool is as follows:

1) Determine the plugins required for the tool,
2) Classify the plugins, i.e. reach the qualification alpha state in the corresponding qualification models and determine the TQL of the plugins and
3) Qualify the plugins that require confidence according to the determined TQL.

The assumptions of the required plugins under which the TQL is computed have to be satisfied. If they are not satisfied by construction of the tool they will correspond to assumptions of the tool that have to be satisfied from the user of the tool.

Similar to the development of Eclipse-based tools, where plugins can be integrated and reused the qualification information (classification and qualification kit) can also be reused in several tools. This enables a modular qualification approach and makes the qualification of plugins that are used in many tools to a common interest of all users of those plugins that want to qualify their tools.

## 6 Roadmap of Eclipse

Since currently Eclipse does not support tool qualification but many tools that require tool qualification base on Eclipse there should be tool qualification support for Eclipse to ease the qualification of those tools. Validas AG has developed a roadmap towards the qualification of Eclipse-based tools within the automotive industrial working group of Eclipse [AutoIWG]. It consists of the following steps:

- Goals Agreement: The goals of the roadmap are to enable the tool qualification of Eclipse-based tools according to the DO-330 which is applicable to all domains. This shall not change the normal process of Eclipse, but gives the possibility to analyze, develop and qualify critical parts of Eclipse-based tools.
- Concept Elaboration: The concept is to use the model-based tool qualification on the level of Eclipse plugins as presented in this paper.

- Concept validation: The concept will be validated in two steps, that both will improve the concept:
    - Demonstrator: A small tool will be developed and qualified according to the concept. We selected the implementation of the transition criteria checker of the DO-330 model as demonstrator application. This will show the feasibility of the concept and provide us with estimations on the required effort,
    - DO-330 Review: The DO-330 review shall review the presented concept and the work products produced during the demonstrator and shall verify if the DO-330 is covered correctly by the concept and the demonstrator,
- Qualifiable Plugin Projects (QPP): This will be a typical Eclipse project that will extend the current plugin mechanism such that it supports the creation of the qualification model (with the tools mentioned in section 7) during the development and the generation of the plugin specific documents from the model and
- Qualification: Selected plugins of Eclipse will be qualified. For the selection and the qualification there will a business model and an ecco system of people (“Validators”) that qualify tools and verify the necessary conditions. Note that there are some mechanism of Eclipse that ensure the modularity and the correctness of the approach that are of highest priority to qualify.

Eclipse follows this roadmap towards tool qualification within the work package 5 of the automotive IWG [AutoIWG] for an actual status of the roadmap and some presentations of the topic.

## 7 Tools

There are many tools that can be used with a model-based tool qualification. The tools may also differ in different IDEs. However it is required to integrate them into the generic development plan such that the plan satisfies the DO-330 requirements. For Eclipse we propose a configuration and description of supporting tools that has been successfully checked to fulfill the DO-330 requirements in the chosen configuration. The proposed tools are: Git, [Gerrit], Bugzilla and [CodeCover].

## 8 Conclusion

New Standards like the ISO 26262 allow to classify the tools according to the used processes and also the DO-330 states in FAQ D.3 that this is possible. Therefore the presented model-based tool qualification approach seems to be very promising, especially since the integration into Eclipse can reduce the overhead for qualification significantly. Currently the approach is evaluated by developing a Eclipse plugin according to it. Of course further relevant plugins of Eclipse and some Eclipse parts need qualification as well [HowTo] but this work enables the model-based tool qualification and will be the basis for all further qualification activities of Eclipse.

This classification approach has been applied from many users [HRMSP11], [WPJSZ12], but the combination of the classification model with the development model is new and also for the Eclipse tool there is neither a classification, nor a qualification model available.
The effort for development of qualifiable tools can be split into the effort for developing a reliable tool (starting from the requirements until the test that completely cover the code) and the effort for providing the documentation to make it evident to the user of the tool or a certification authority. While the effort for developing a reliable tool is high, the additional effort for documenting this should be quite low. Therefore the development of qualifiable tools should not be much more work than the development of reliable tools. For non-reliable tools the effort for making it reliable is surely higher.
The development of reliable systems is something quite well studied based on the domain specific standards. Speaking in terms of [Rus11], this work is the construction of a specific safety case [BB98], however focused on tools and not on systems such that the safety claim is that the used tools must not corrupt the safety of the developed systems. The development of reliable tools requires many verification steps like reviews and tests that should be performed by trustworthy persons.
For that purpose we propose to create the role “Validator” in the Eclipse community that is a specialization of the “Committer” role for tool validation. The Validators task is to validate that the required verification steps are executed correctly. This can be supported with a public visible profile like the traders in ebay have. The Validators can gain positive ratings if they successfully verify activities. If their verification turns out to be false they will get negative feedback. Being a positive Validator will be even more desirable than being an active committer.

## Acknowledgments

We would like to thank the Eclipse Foundation and it’s automotive industrial working group for supporting us in the development, especially Mario Driussi. Furthermore we like to thank Stefan Dirix and Natacha Tchebetchou for their contributions to the roadmap and BMW CarIT (Michael Rudorfer und Tilmann Ochs) to support the development of the demonstrator. For review comments we would like Jan Philipps and the unknown reviewers of this paper.
Especially we would like to thank all authors of the DO-330 to provide the first general safety standard for tool qualification that could be adopted for Eclipse so easily.

## References

[61508] International Electrotechnical Commission, IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systmes, Edition 2.0, Apr 2010.
[AutoIWG] Automotive Eclipse Automotive Industrial Working Group for tool qualification, see http://wiki.eclipse.org/Auto_IWG_WP5

[BB98] P. Bishop and R. Bloomfield. A methodology for safety case development. In Safety-Critical Systems Symposium, Birmingham, UK, Feb. 1998. Available at http://www.adelard.com/resources/papers/pdf/sss98web.pdf.
[CodeCover] An open source glass-box testing tool, http://codecover.org/
[DO330] RTCA. *DO-330: Software Tool Qualification Considerations* 1st Edition 2011-12-13.
[EMF] The Eclipse Modeling Framework, see http://www.eclipse.org/modeling/emf/
[Gerrit] Code Review Tool for Git, see http://code.google.com/p/gerrit/
[HowTo] Eclipse (Proposal): How-To Qualify Eclipse-Based Tools, Version 1.0, to be published when reviewed.
[HRMSP11] Joachim Hillebrand, Peter Reichenpfader, Irenka Mandic, Hannes Siegl, Christian Peer: *Establishing confidence in the usage of software tools in context of ISO 26262* In SAFECOMP 2011.
[ISO26262] International Organization for Standardization. *ISO 26262 Road Vehicles –Functional safety–*. 1st Edition, 2011-11-15.
[Rus11] Rushby, J. 2011, 'New Challenges In Certification For Aircraft Software' EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.
[TCA] Tool Chain Analyzer Tool, can be downloaded from www.validas.de/TCA.html
[TDP] Eclipse (Proposal): Tool Development Plan for every Qualifiable Eclipse Plugin, Version 1.0, to be published when reviewed.
[TVP] Eclipse (Proposal): Tool Verification Plan for every Qualifiable Eclipse Plugin, Version 1.0, to be published when reviewed.
[WPJSZ12] Martin Wildmoser, Jan Philipps, Reinhard Jeschull, Oscar Slotosch Rafael Zalman. *ISO 26262 - Tool Chain Analysis Reduces Tool Qualification Costs.* In SAFECOMP 2012.