Oscar Slotosch, Validas AG

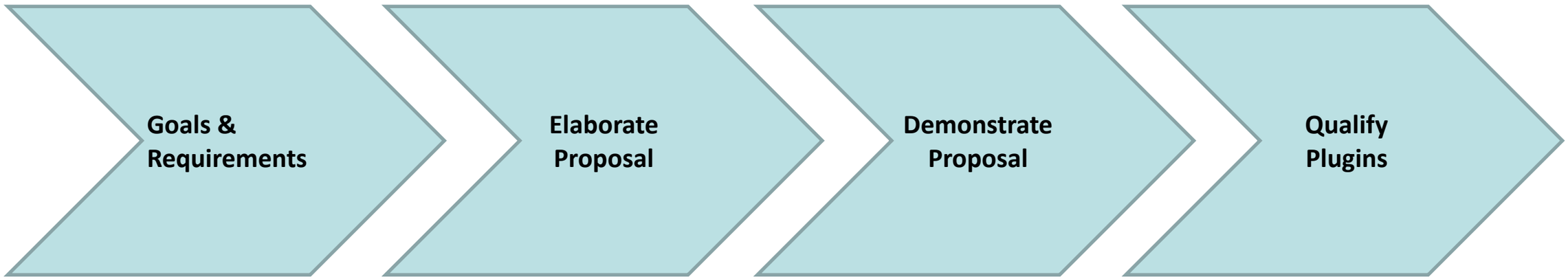
Roadmap towards Development of Qualifyable Eclipse Tools (Eclipse-Project Concept)

Content

- **Roadmap**
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

Roadmap

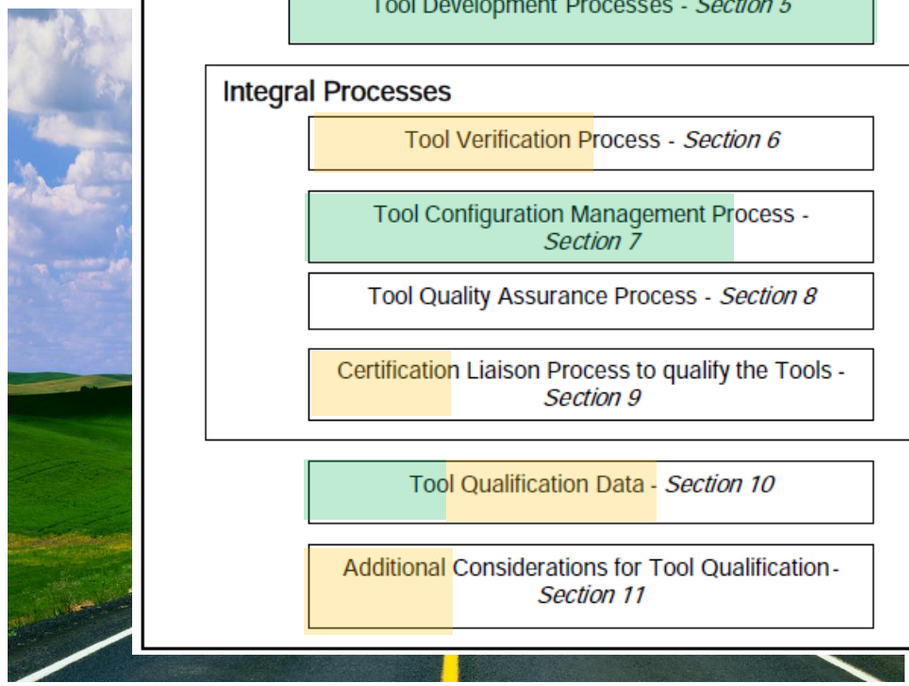
- Identify goals & requirements for tool qualification in Eclipse
- Propose process / project
- Demonstrate tool qualification & improve proposal
- Establish proposal: Qualify (selected) plugins



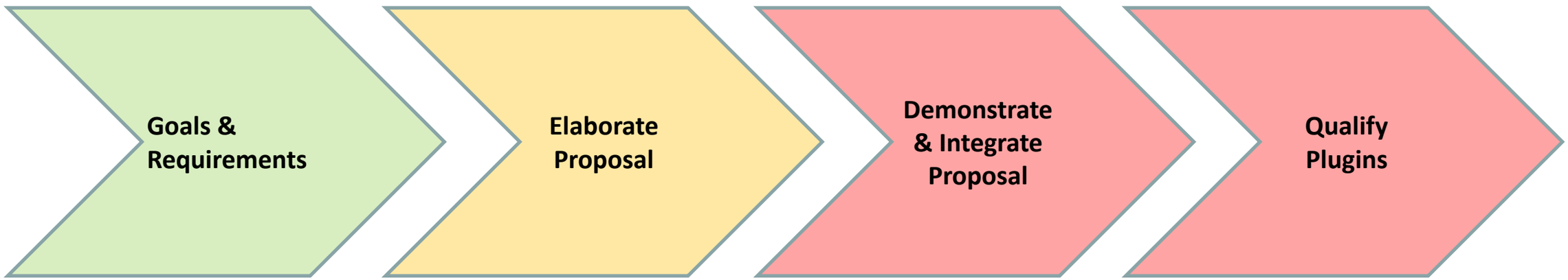
- Is this a Eclipse project? Not a typical ☺
- Is this an Industrial Working Group process?

Roadmap - Status May 2012

1. **Identify goals & requirements for tool qualification in Eclipse**
2. **Propose process / project**
3. **Demonstrate tool qualification & integrate proposal into Eclipse Plugin Framework**
4. **Establish proposal: Qualify (selected) plugins**



- **Status May 2012**



- **Summary: Qualification is feasible and qualification (based on current prototype) could be started now**

Content

- Roadmap
- **Requirements for Tool Qualification (Standards)**
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

Tool Qualification (Summary)

- **Standards require tool qualification: ISO 26262, IEC 61508, DO, EN 50128**
- **Process:**
  - Classify all used tools (Impact, Use-Cases, Artifacts)
  - Qualify critical tools
  - Use tools
- **Qualification Methods ISO 26262**

Table 4 — Qualification of software tools classified TCL3

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Increased confidence from use in accordance with 11.4.7	++	++	+	+
1b	Evaluation of the tool development process in accordance with 11.4.8	++	++	+	+
1c	Validation of the software tool in accordance with 11.4.9	+	+	++	++
1d	Development in accordance with a safety standard[a]	+	+	++	++

Here is a hole were the new DO-330 standard fits in

- **Some tools provide qualification kits for confidence with evidence into**
  - Correctness of functions by testing them “validatio
  - Development process by documentation
  - ….

Since DO-330 is scalable, here could also be a ++

Extension of the ISO 26262?

- **Possible extension / integration of DO-330 into ISO 26262 could look like:**

11.4.10 Development according to a Safety Standard

11.4.10.1 The DO-330 is the first safety standard that is fully applicable to the development of software tools. It is based on Tool Qualification Levels TQL where TQL-1 is the most rigorous level, while TQL-5 is the least one.

11.4.10.2 The mapping from the TCL to the TQL should depend on the SIL level of the system. The mapping is specified in table 4.

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

11.4.10.3 The tool operational requirements, which are the input for tool development according to DO-330, should cover the use cases analysed in clause 11.4.4

- **Similar chapters exist in DO-178C and DO-254**

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

- **Extension is not necessary to apply DO-330 in ISO 26262 but could clarify**

Content

- Roadmap
- Requirements for Tool Qualification (Standards)
- **Proposals for Goals for Eclipse**
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

Goals for Eclipse IWG

- **Exchange & share knowledge**
  - Motivate developers & community to provide qualifyable plugins
- **Provide classification support to users of Eclipse tools**
- **Support the development of qualifyable tools (“Qualification Kits”)**
  - Validation
  - Safety-Standard (DO-330)
- **Apply this to reference tools ARTOP, EMF,… ?**

- **Current status (web-page):**

Auto IWG WP5

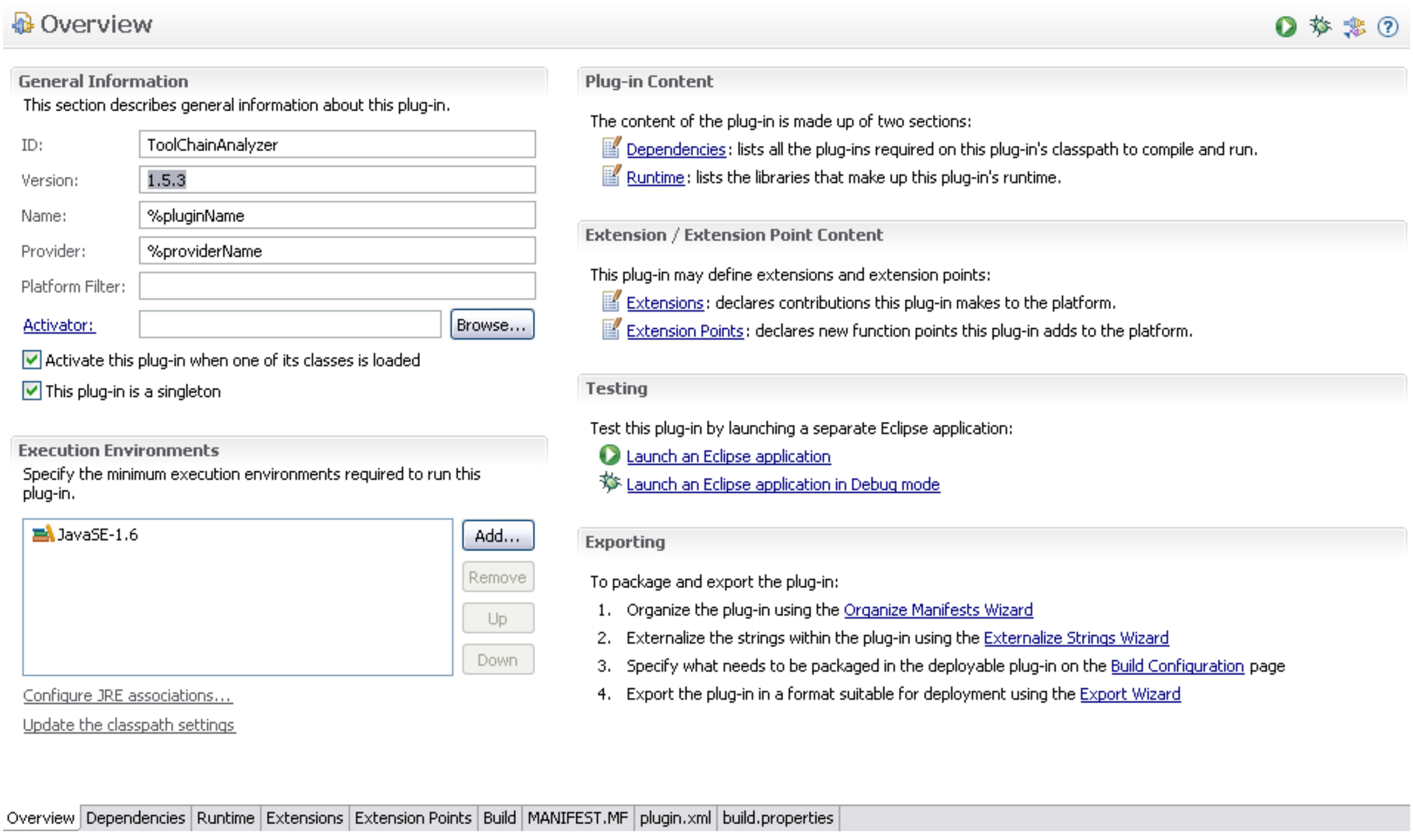
WP5: Eclipse Qualification Kit (ISO26262)

This is work package 5 of the Automotive Industry Working Group.

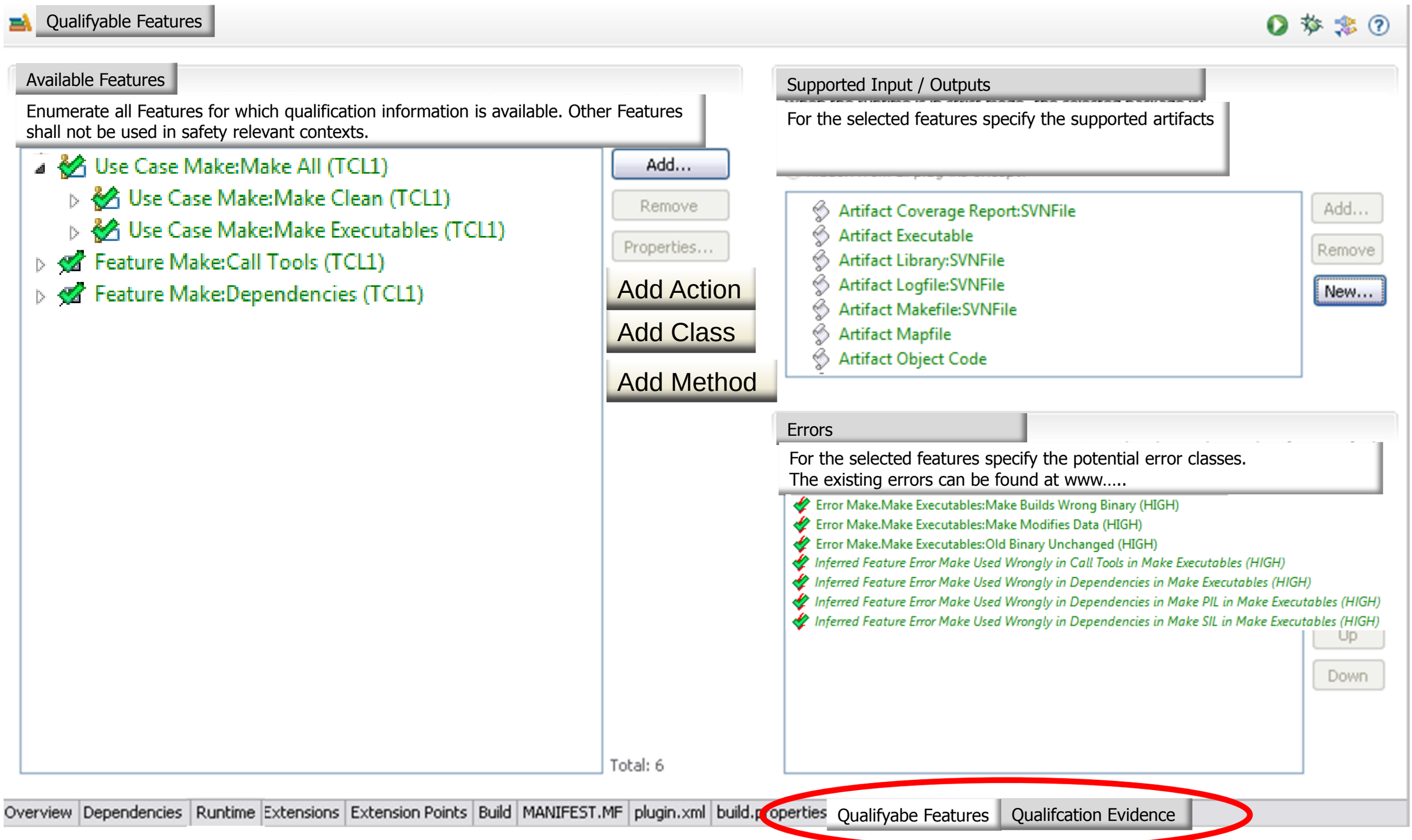
- WP Lead: Bredex (temporary)

Need to share knowledge and resources in the classification/qualification activities of eclipse related products.

Current Eclipse Metadata



Vision: Eclipse Classification Data



Proposed Role: Eclipse Validator

There is much (different) work to do such that we need a new kind of worker: The Validator

- **Should provide confidence**
- **Should be more formalized than a committer**
- **Should have qualifications e.g. by filling out questionnaires on**
  - Eclipse qualification process
  - DO-330
- **Should have responsibilities (answer to questions)**
- **Should earn “credits” for each successful validation action**
  - Executed reviews
  - Formulated requirements
  - Created use/test cases
  - Feedback
  - …
- **Comparable:**
  Confidence in ebay:

slotosch (25 ★)

Positive Bewertungen (der letzten 12 Monate): 100%
[Wie wird der Prozentsatz positiver Bewertungen berechnet?]

Mitglied seit: 01.04.99 in Deutschland

Content

- Roadmap
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- **Proposals for some steps towards Tool Qualification**
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

Proposals

Following activities are necessary to achieve goals:

- **Agree on focus, e.g. “Metadata extension for qualification information”**
- **Provide classification support to users of plugins**
  - Use case
  - Potential errors
  - Possible mitigations for errors
  - TCL inference
- **Provide qualification support**
  - Create checklist for DO-330 requirements (depending on the TQL)
    - Qualification data (general, plugin specific, user adaptable)
    - Requirements (general, development, operational)
  - Check Eclipse against the checklist, create
    - Mapping of Eclipse -> DO-330
    - Identify gaps: missing data/requirements
  - Provide model (EMF?) for the missing data
- **Demonstrate it: Small example e.g. EclipseCon**
- **Validate it: bigger example**

Content

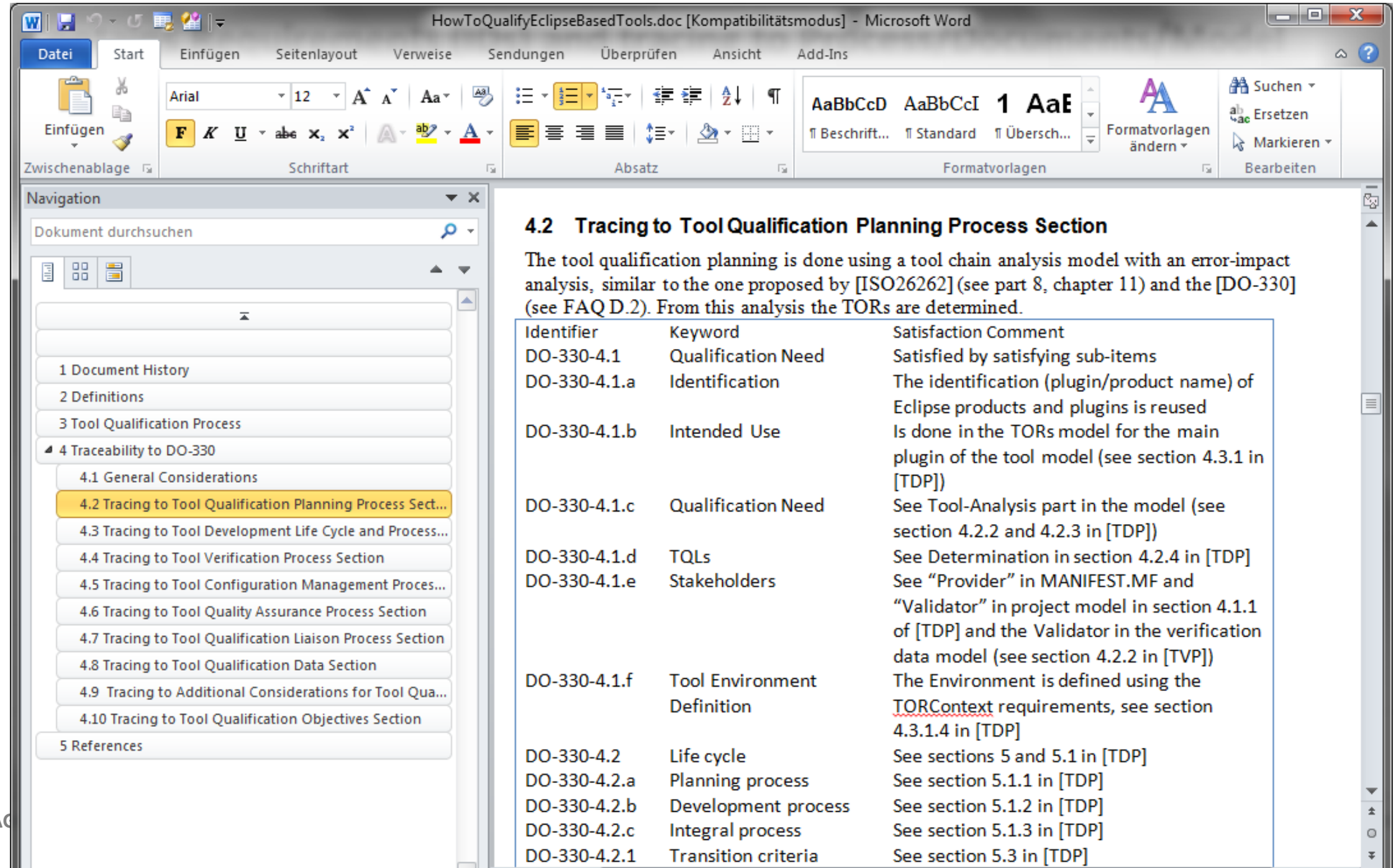
- Roadmap
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - **First steps: Requirements handling**
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

First Steps on the Road

- **Create a checklist to show the DO-330 compliance**
- **Make a/some simple example tool(s) that shall comply with DO-330**
- **Work on selected topics: Requirements, Test, Code, …**
  - Analyze existing Eclipse process
  - Analyze possibilities for the topic e.g. RIF, tracing, tests,…
  - Create example document (eventually based on existing methods)
  - Check DO-330 compliance
  - Create model (for creation of document)
  - Review/Validate for:
    - Expressiveness
    - practicability
    - possible improvements
  - Make proposal for Eclipse integration (part)
- **Until DO-330 is completely satisfyable**
- **Make integrated proposal for Eclipse Extension (EMF,..)**

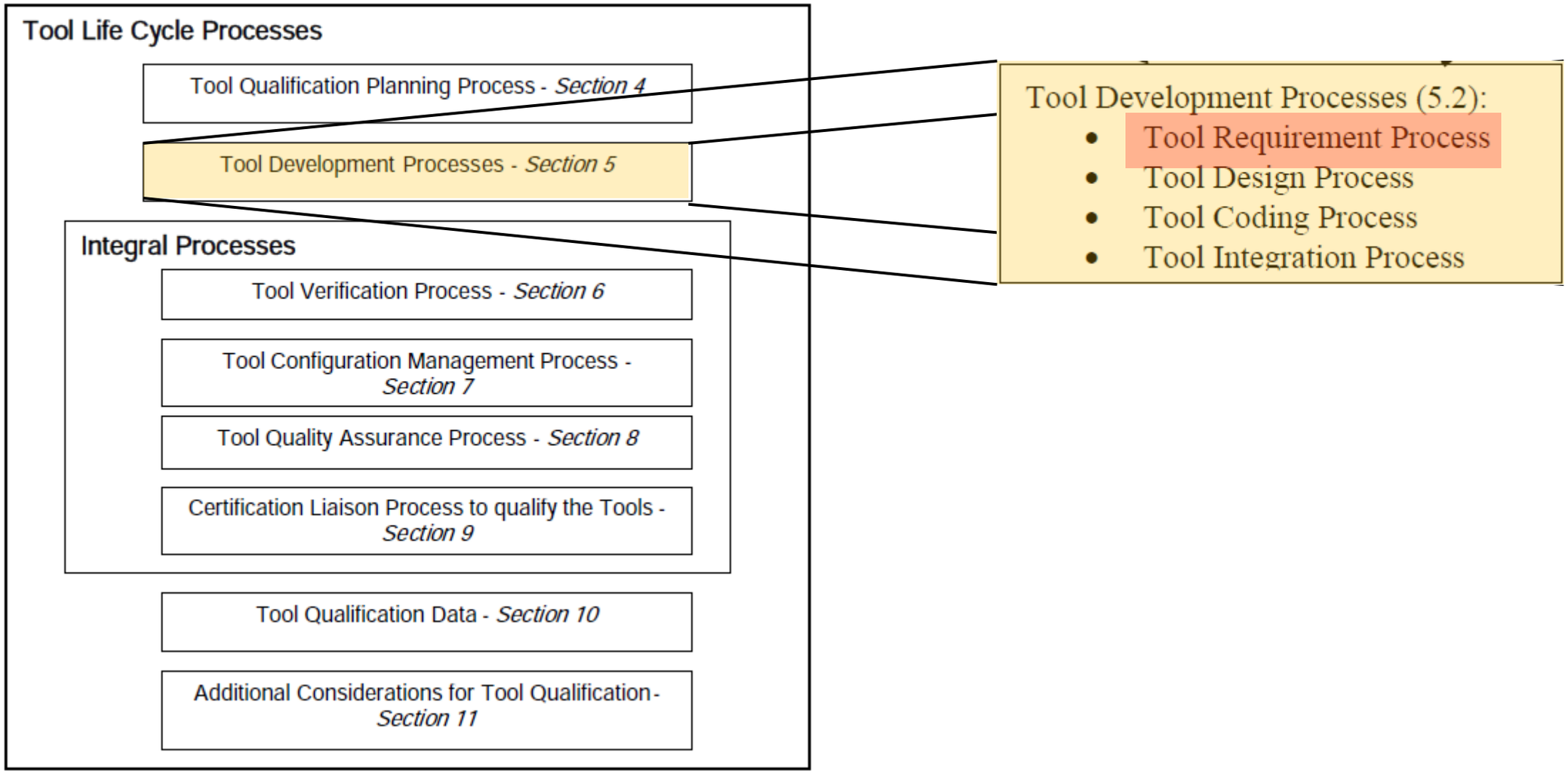
Checklist for DO-330 compliance (refined)

- Document created: “How-To Qualify Eclipse-based Tools”
- Contains Requirements (IDs) and tracing to Process/Documents/Model



DO-330 Topics

- **Structure of DO-330**



Existing Methods: Requirements

- **Currently not practiced in Eclipse**
- **RMF / ProR (Incubation):**

ProR

Reqlf-Model.rif | Specification Document

Specification Document

	ID	Description	Link
1	REQ-1	Dies ist eine Demo von ProR	0 ▷ ▷ 2
	▷		REQ-5
	▷	Links können auch Attribute haben.	REQ-6
1.1	REQ-2	Hierarchien beliebiger Tiefe werden unterstützt.	
1.2	REQ-3	Der Linke Rand hilft bei der Orientierung	
1.2.1	REQ-4	... und die erste Spalte wird eingerückt.	
2	REQ-5	Im Properties-View werden alle Attribute angezeigt.	1 ▷ ▷ 0
3	REQ-6	Im Editor nur die, die man sehen will.	1 ▷ ▷ 0

- **general approach, not tailored for tool requirements**
- **Adoptable to tool requirements by creating corresponding requirements types**
- **First Investigation**
  - Nice usability e.g. for creating new requirements
  - Polymorphic links (any requirement can be linked)
  - Extensible to design / test / …?
  - Do we need RIF within Eclipse?

Create DO-330 Conformant Example

Tool Requirements for Tool Chain Analyzer
Version 0.2
Validas AG

3 General Information

This section contains the general information on the Tool Chain Analyzer generated from the corresponding plugin metadata

- Name: Tool Chain Analyzer
- ID: de.validas.toolchainanalyzer
- Version: 1.5.3
- Provider: Validas AG
- Tool Qualification Level (TQL): TQL-1

from MANIFEST.MF

Overview

General Information

This section describes general information about this plug-in.

ID:	de.validas.toolchainanalyzer
Version:	1.5.3
Name:	Tool Chain Analyzer
Provider:	Validas AG
Qualification Level	TQL-1

?

From Tool Requirements Model

5.4 Customization Requirements

The tool shall be customized to the resources of the computer were it is executed.

5.4.1 Stack Size

The stack size shall be settable. The default stack size should be 400 MB

5.4.2 Heap Size

The heap size shall be settable. The default hap size should be 1000 MB.

5.5 Tool Interface Requirements

The tool chain analyzer shall have the following interfaces.

5.5.1 Graphical User Interface

The graphical user interface consists of different views and property dialogs.

5.5.1.1 Structure View

The structure view represents the tool chain models in a tree view with the structure how the elements are modeled. The structure views also contains the actions to created, move and delete elements. Furthermore it can be used to start actions like the im- and export of tool models.

5.5.1.2 Property View

The property view shows the properties (attributes and relations) of the elements selected in the tree view. They can be edited either directly in the view or in property dialogs the start when the elements are double-clicked.

5.5.1.3 Property Dialogs

The property dialogs are used to edit long text fields or complex relations in the modeled elements. They are started from the property view.

5.5.1.4 Flow View

The flow view shows the information flows within the model, e.g. from one tool to another via the artifact that is written and read. Furthermore the error derivation flow from the general error model to the features and use cases.

5.5.2 File Interface

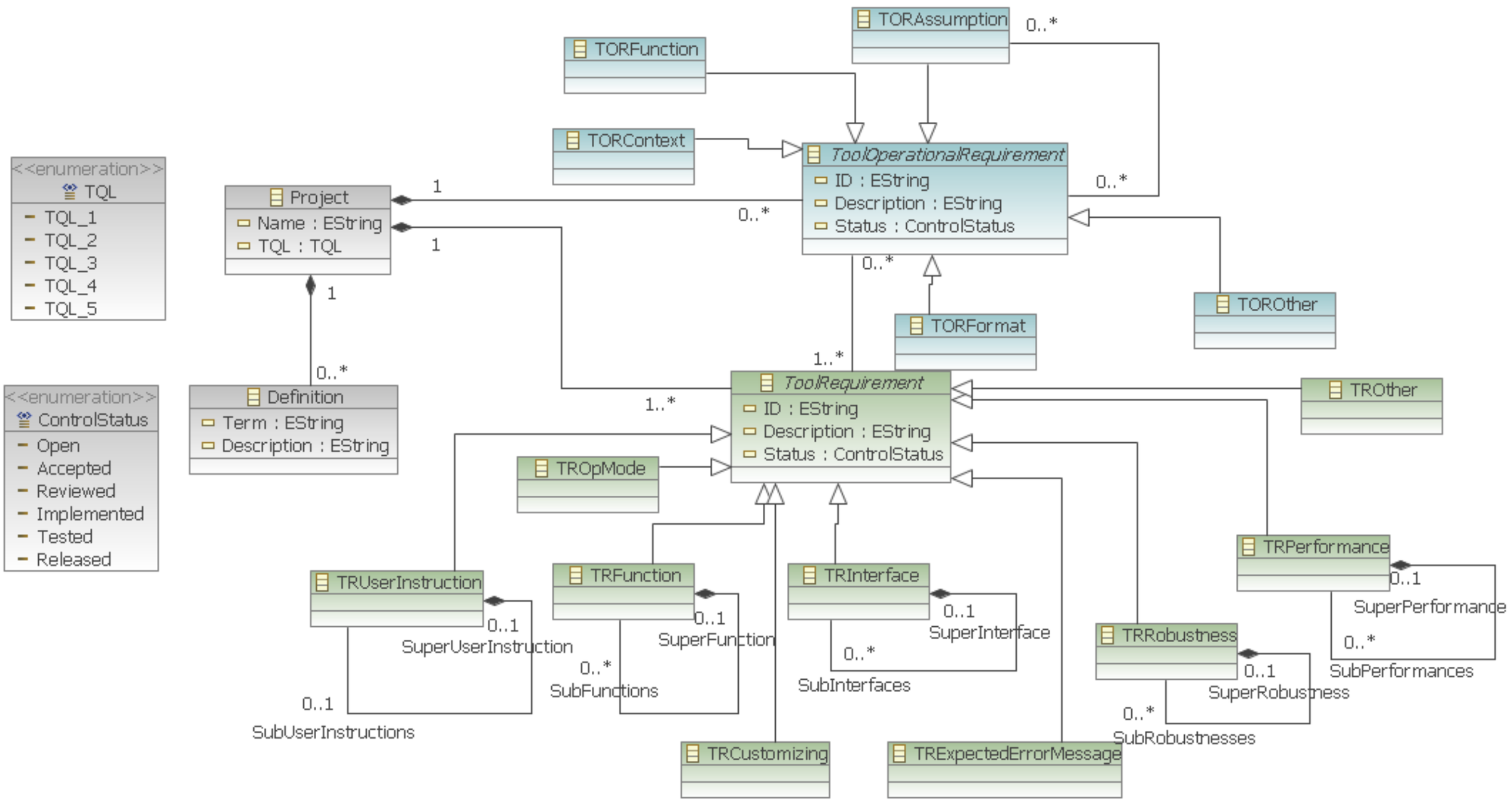
The tool chain models shall be persistent to files. The tool chain analyzer loads models from files and writes the back into files.

5.5.3 DOT Interface

For drawing the images to explain the error flow in the model the graphviz tool with the DOT language. The intermediate files are accessible and can be modified or integrated into other images.

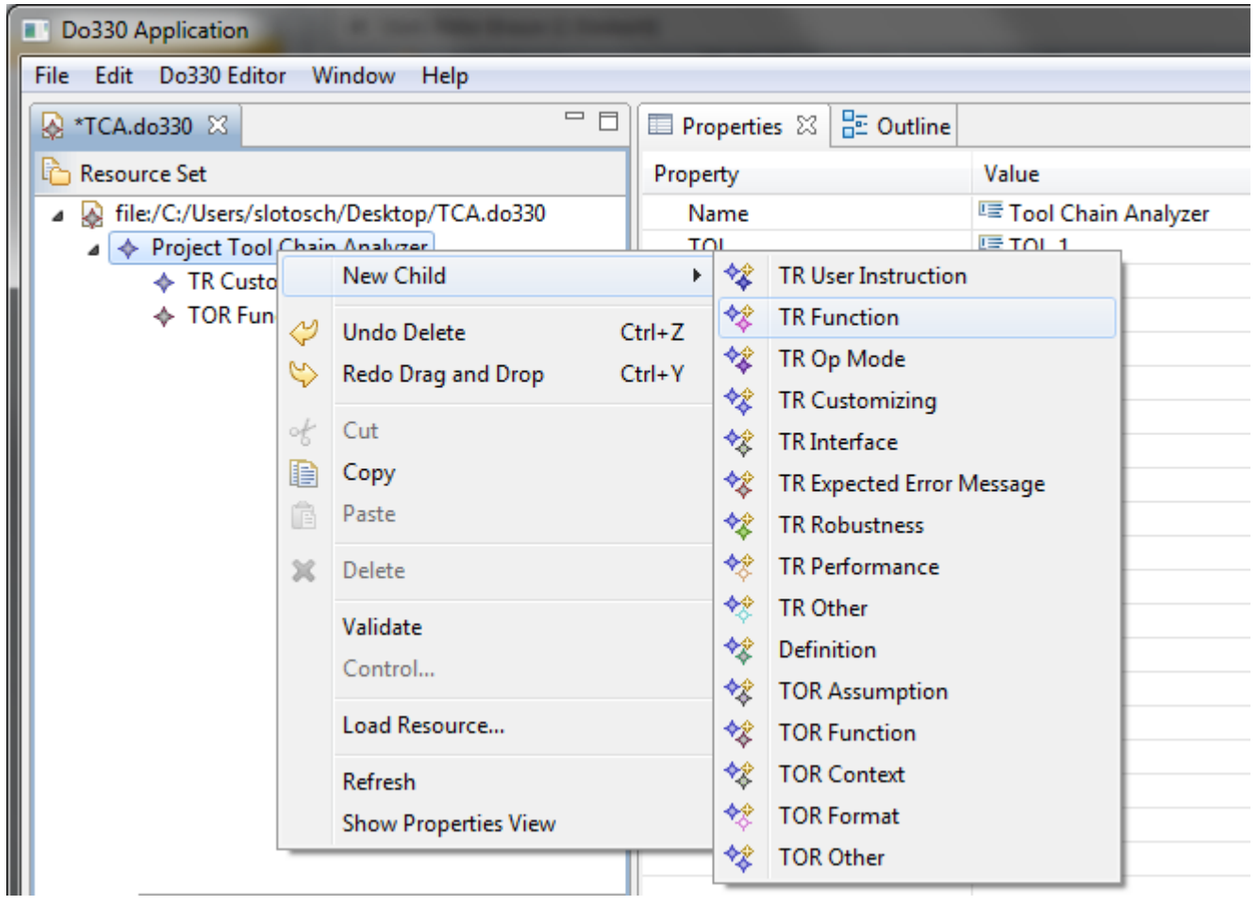
Create Model for Tool-Requirements

- **EMF-Metamodel (Draft) for Tool Requirements**

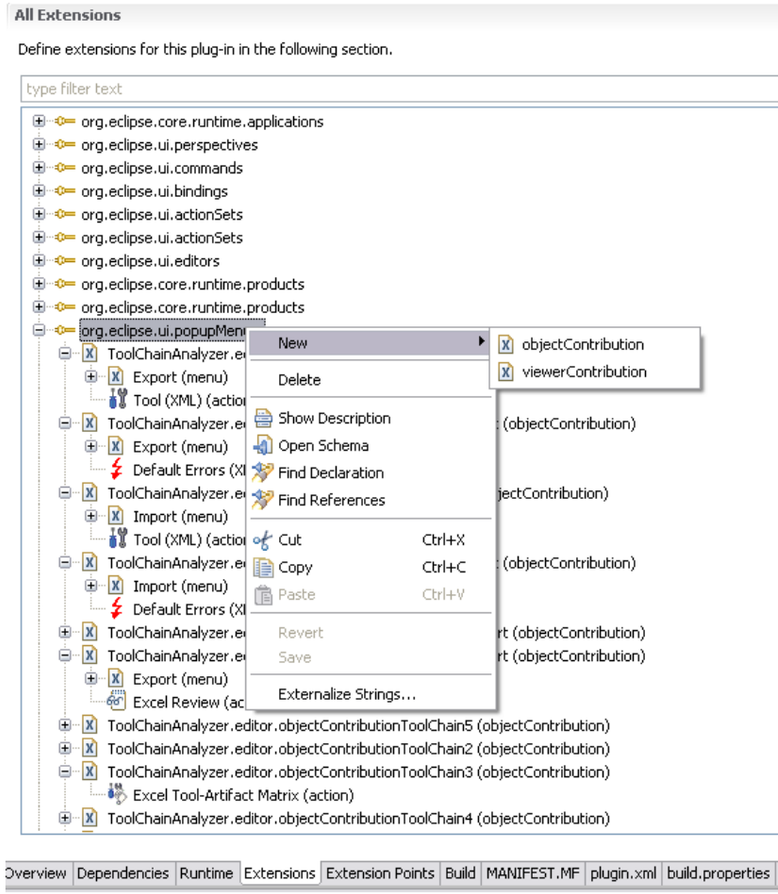


Create Example Model

- **Using the default EMF Editor**



Comparable: Plugin Extension



- **Shows how simple requirements could be created with Eclipse**
- **The example (DO-330 conforming) document can be generated completely from the model**
- **Tracing: TOR <-> TR is done using Eclipse association editors**

Content

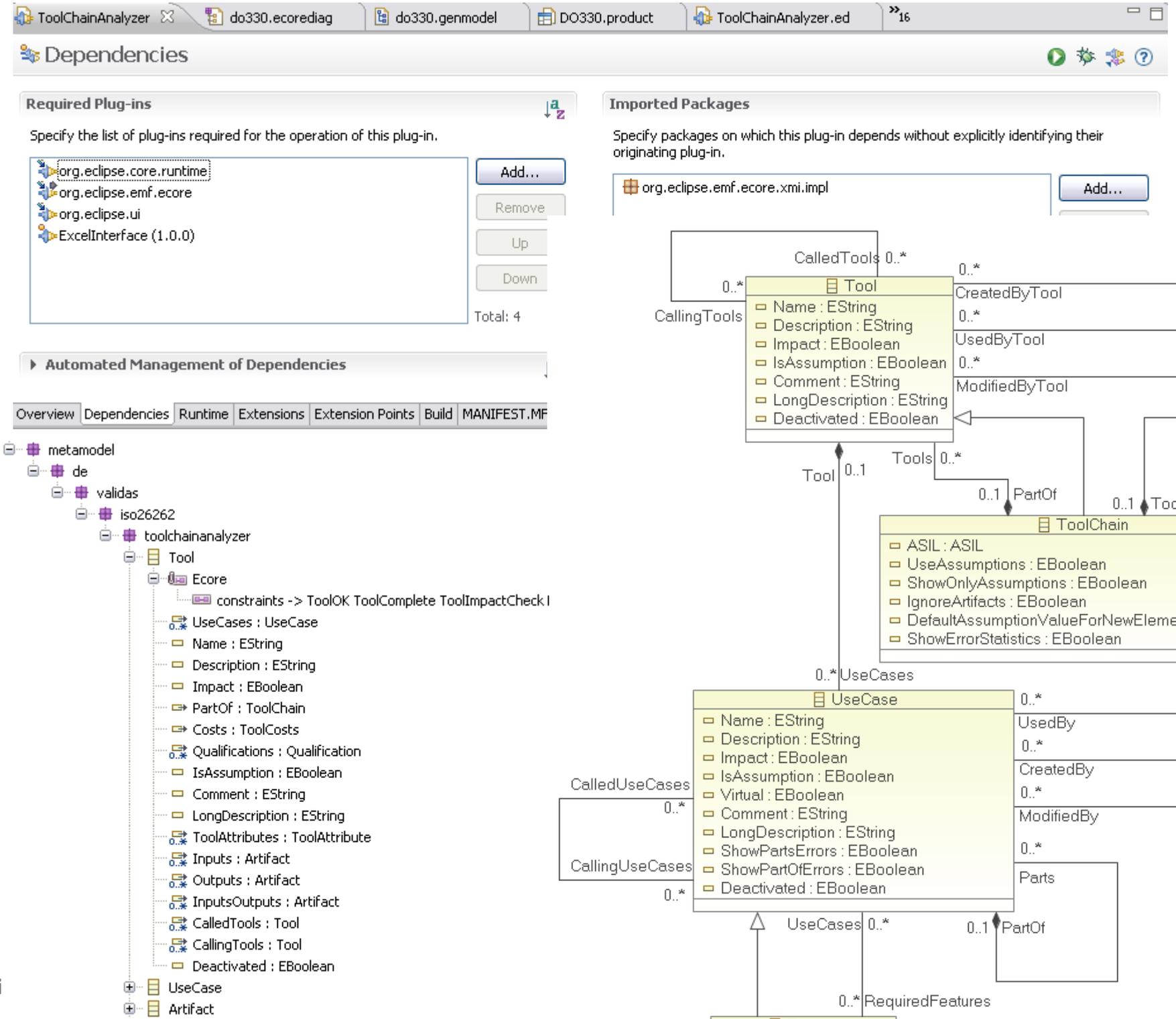
- Roadmap
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - **Second steps: Design, Coding and Test**
  - Third Steps: Planning Tool Analysis and Life Cycle
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- Summary

Design and Coding (5.2.2. and 10.2.2)

- **Design = Architecture + Low Level Requirements (LLRs)**
- **Design Description:**
  - Tool architecture: tool structure to implement TR
  - Detailed Description how TRs are allocated in architecture
  - Input/output description of architecture elements
  - Data & control flow
  - Scheduling procedures
  - Protection (if used)
  - Used components (incl. baselines)
  - LLRs including tracing to TR
  - Derived LLRs (not traceable to TRs)
    - Justification required including
    - No negative impact to TORs and TRs
- **Verifiable and consistent**
- **Compliant to standards**

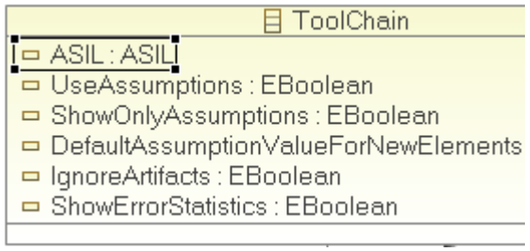
Architecture Examples in Eclipse

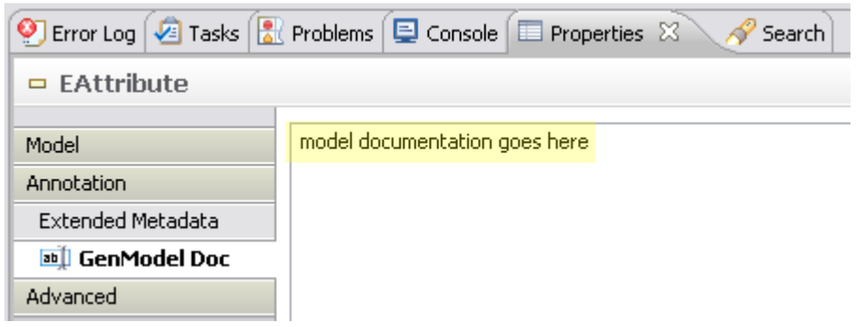
- **Architecture: Plugins & packages, EMF models, xText grammars**



Example EMF Code

- Model generates code, interface (and description!)





- Documentation can be inserted into code and model

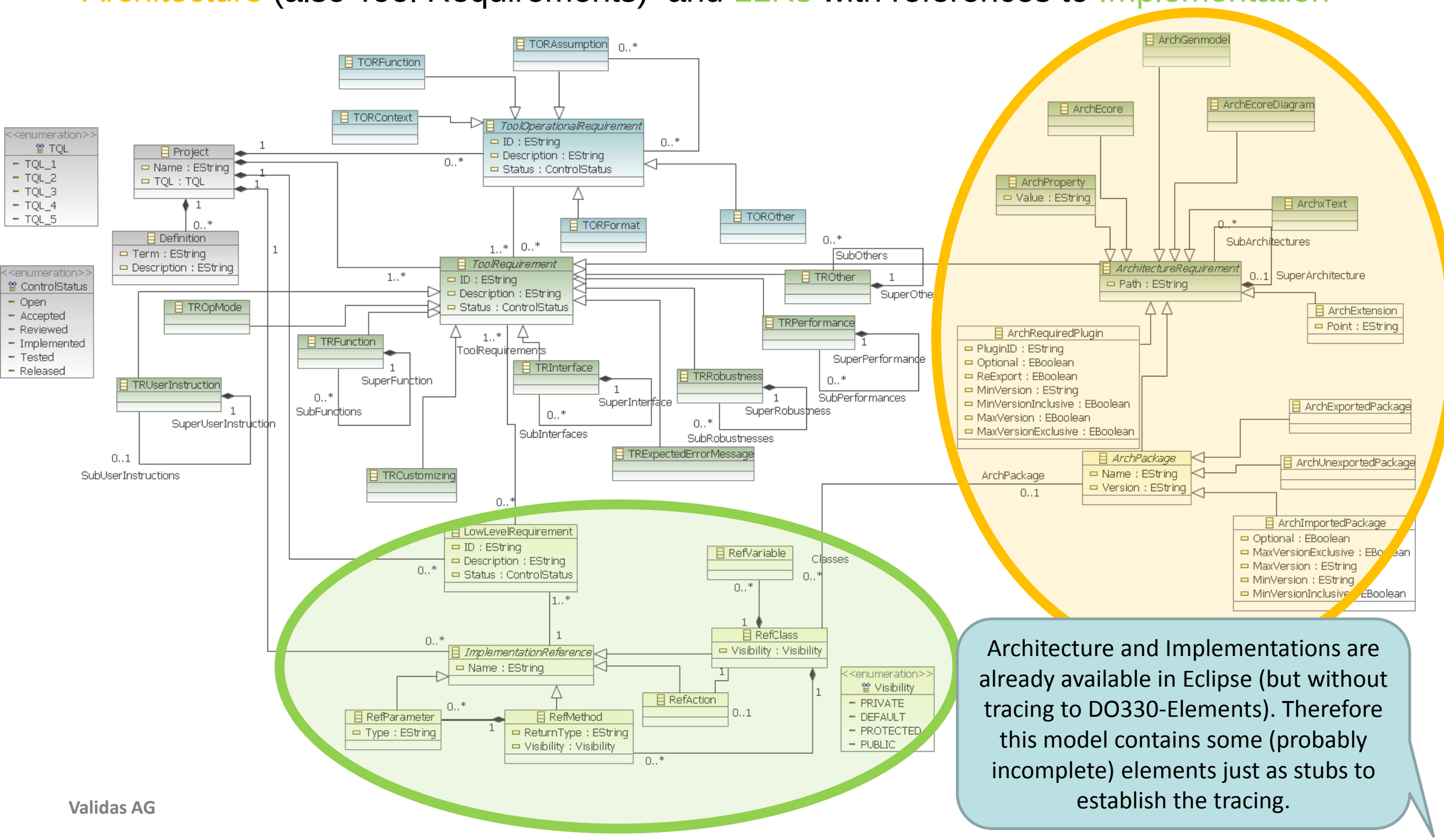
```
/**
 * Returns the value of the '<em><b>ASIL</b></em>' attribute.
 * The literals are from the enumeration {@link metamodel.de.validas.iso26262.toolchainanalyzer.ASIL}.
 * <!-- begin-user-doc -->
 * <p>
 * here is the specific code description of the return value '<em>ASIL</em>'
 * </p>
 * <!-- end-user-doc -->
 * <!-- begin-model-doc -->
 * model documentation goes here
 * <!-- end-model-doc -->
 *
 * @return the value of the '<em>ASIL</em>' attribute.
 * @see metamodel.de.validas.iso26262.toolchainanalyzer.ASIL
 * @see #setASIL(ASIL)
 * @see metamodel.de.validas.iso26262.toolchainanalyzer.ToolchainanalyzerPackage#getToolChain_ASIL()
 * @model
 * @generated
 */
ASIL getASIL();
```

Low Level Requirements

- **Can be directly implemented**
- **Not the code but it's detailed descriptions**
  - Class: Name, super classes, visibility, interfaces, exceptions, purpose
  - Methods: Name, parameters, types, exceptions, visibility, purpose
  - Variables: Name, type, visibility, purpose
  - Contributions:
    - Actions
    - Menus
    - ShortCuts
    - Separators
    - …
- **Currently:**
  - tags & templates in the code
  - No tracing to requirements possible (due to missing requirements?)

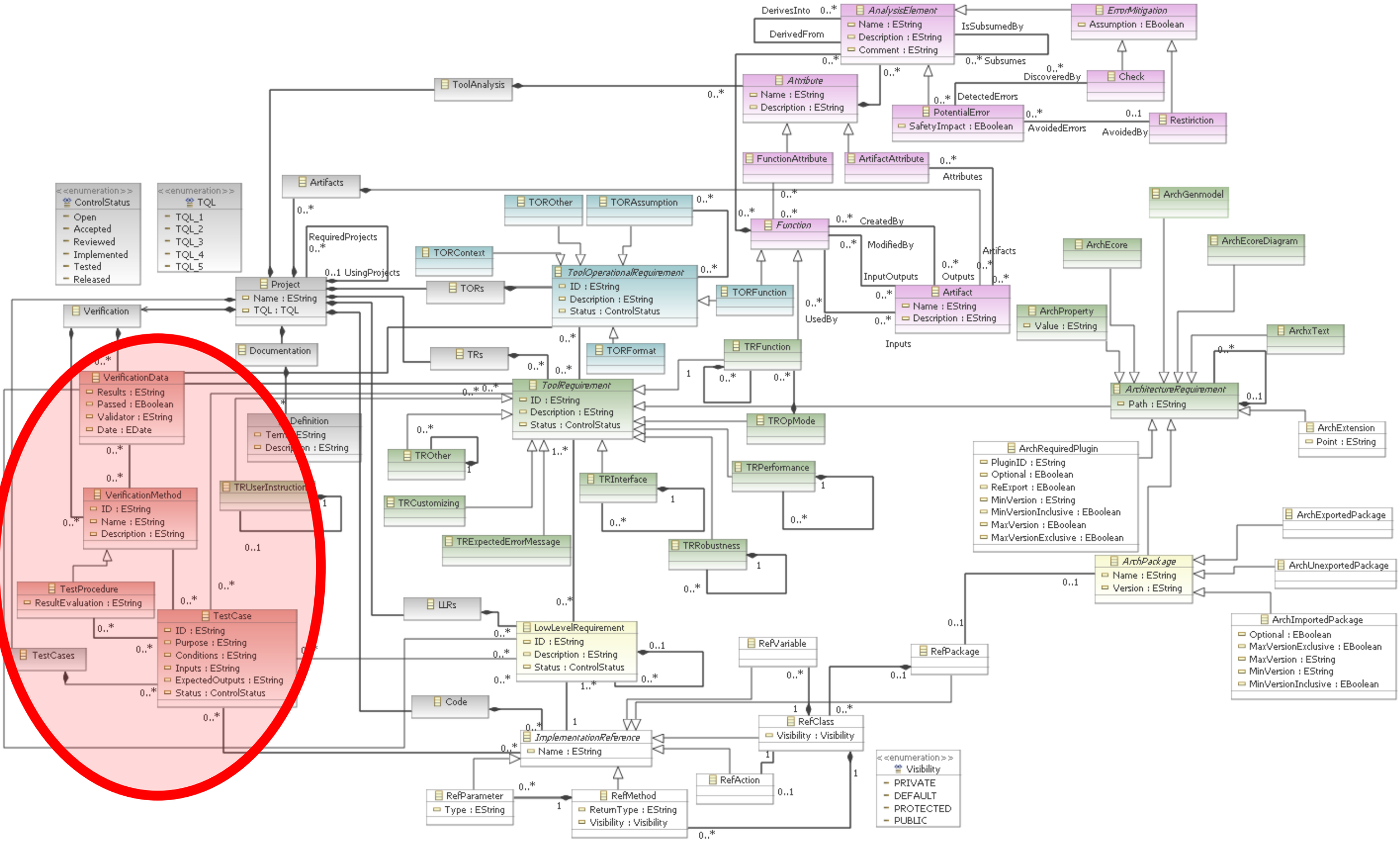
Design Model

The design model extends the requirements model by
Architecture (also Tool Requirements) and LLRs with references to Implementation



Architecture and Implementations are already available in Eclipse (but without tracing to DO330-Elements). Therefore this model contains some (probably incomplete) elements just as stubs to establish the tracing.

Test Model

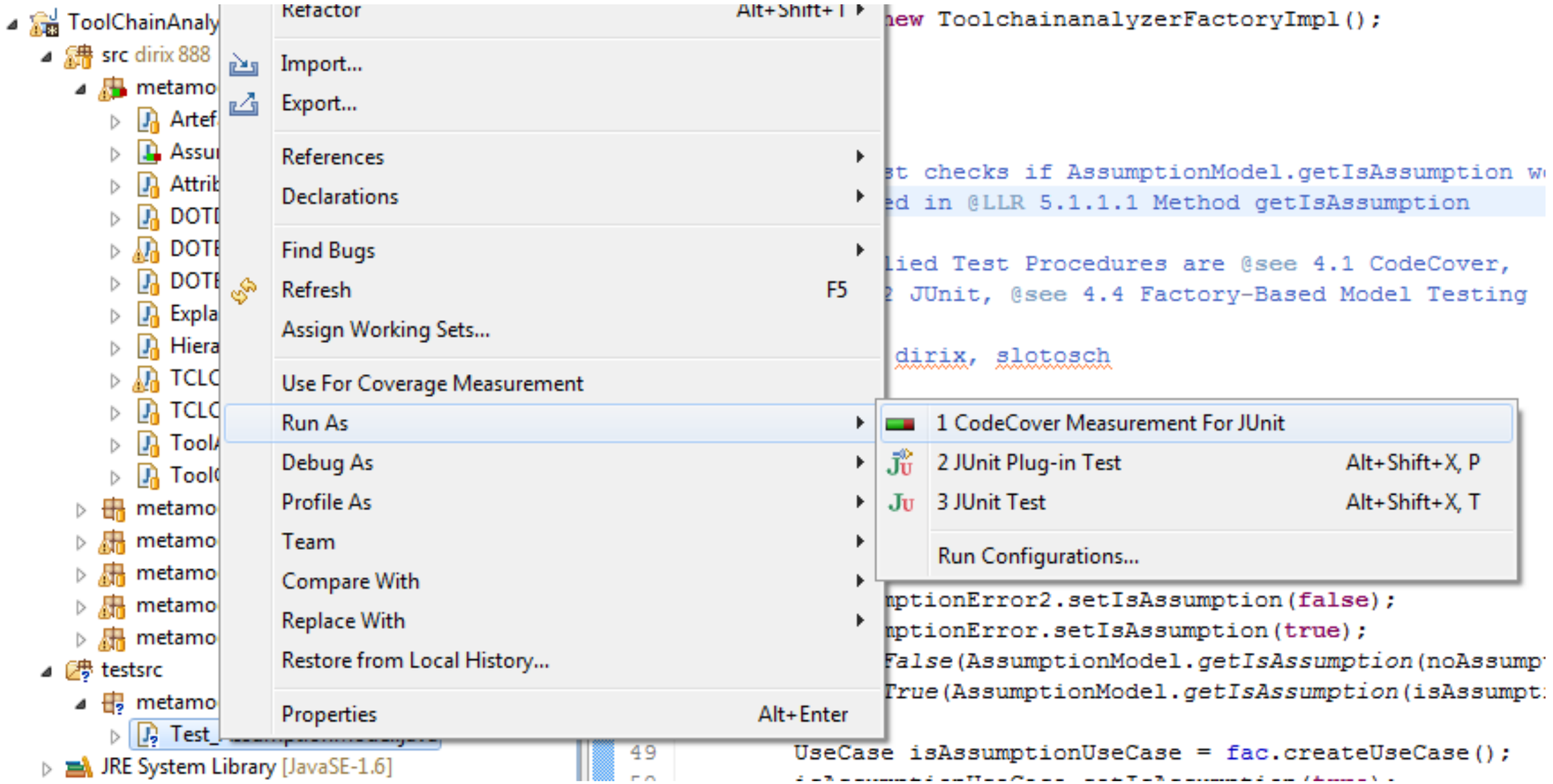

AnalysisElement
Name : EString
Description : EString
Comment : EString
DerivesInto
DerivedFrom
IsSubsumedBy
Subsumes
ErrorMitigation
Assumption : EBoolean
Attribute
Name : EString
Description : EString
ToolAnalysis
Check
DiscoveredBy
DetectedErrors
PotentialError
SafetyImpact : EBoolean
AvoidedErrors
AvoidedBy
Restiriction
FunctionAttribute
ArtifactAttribute
Attributes
Artifacts
TOROther
TORAssumption
TORContext
TORs
Function
CreatedBy
ModifiedBy
InputOutputs
Outputs
Artifact
Name : EString
Description : EString
UsedBy
Inputs
ToolOperationalRequirement
ID : EString
Description : EString
Status : ControlStatus
TORFunction
TORFormat
ArchGenmodel
ArchEcore
ArchEcoreDiagram
ArchProperty
Value : EString
ArchxText
ControlStatus
Open
Accepted
Reviewed
Implemented
Tested
Released
TQL
TQL_1
TQL_2
TQL_3
TQL_4
TQL_5
Project
Name : EString
TQL : TQL
RequiredProjects
UsingProjects
Verification
Documentation
TRs
TRFunction
ToolRequirement
ID : EString
Description : EString
Status : ControlStatus
TROpMode
ArchitectureRequirement
Path : EString
ArchExtension
Point : EString
VerificationData
Results : EString
Passed : EBoolean
Validator : EString
Date : EDate
Definition
TROther
TRInterface
TRPerformance
TRCustomizing
TRExpectedErrorMessage
TRRobustness
ArchRequiredPlugin
PluginID : EString
Optional : EBoolean
ReExport : EBoolean
MinVersion : EString
MinVersionInclusive : EBoolean
MaxVersion : EBoolean
MaxVersionExclusive : EBoolean
ArchExportedPackage
ArchPackage
Name : EString
Version : EString
ArchUnexportedPackage
ArchImportedPackage
Optional : EBoolean
MaxVersionExclusive : EBoolean
MaxVersion : EString
MinVersion : EString
MinVersionInclusive : EBoolean
VerificationMethod
ID : EString
Name : EString
Description : EString
TRUserInstruction
TestProcedure
ResultEvaluation : EString
TestCase
ID : EString
Purpose : EString
Conditions : EString
Inputs : EString
ExpectedOutputs : EString
Status : ControlStatus
TestCases
LLRs
LowLevelRequirement
ID : EString
Description : EString
Status : ControlStatus
RefVariable
RefPackage
Code
ImplementationReference
Name : EString
RefClass
Visibility : Visibility
RefParameter
Type : EString
RefMethod
ReturnType : EString
Visibility : Visibility
RefAction
Visibility
PRIVATE
DEFAULT
PROTECTED
PUBLIC

Test Implementation

*Test_AssumptionMode | ToolInterfaceTest.ja | ToolChainAnalyzer | AssumptionModel.java | ReflectiveCallab

```
13 import metamodel.de.validas.iso26262.toolchainanalyzer.UseCase;
14 import metamodel.de.validas.iso26262.toolchainanalyzer.impl.ToolchainanalyzerFactoryImpl;
15
16 import org.junit.Before;
17 import org.junit.Test;
18
19 public class Test_AssumptionModel {
20
21     private ToolchainanalyzerFactory fac;
22
23     @Before
24     public void setUp() {
25         fac = new ToolchainanalyzerFactoryImpl();
26     }
27
28     @Test
29     /**
30      * This test checks if AssumptionModel.getIsAssumption works as
31      * specified in @LLR 5.1.1.1 Method getIsAssumption
32      *
33      * The applied Test Procedures are @see 4.1 CodeCover,
34      * @see 4.2 JUnit, @see 4.4 Factory-Based Model Testing
35      *
36      * @author dirix, slotosch
37      */
38     public void getIsAssumptionTest() {
39         //testing assumption handling of errors
40         Error noAssumptionError = fac.createError();
41         Error isAssumptionError = fac.createError();
42         Error noAssumptionError2 = fac.createError();
43         noAssumptionError.setIsAssumption(false);
44         noAssumptionError2.setIsAssumption(false);
45         isAssumptionError.setIsAssumption(true);
46         assertFalse(AssumptionModel.getIsAssumption(noAssumptionError));
47         assertTrue(AssumptionModel.getIsAssumption(isAssumptionError));
48
49         UseCase isAssumptionUseCase = fac.createUseCase();
50         isAssumptionUseCase.setIsAssumption(true);
51         UseCase noAssumptionUseCase = fac.createUseCase();
```

Test Execution



Test Coverage

getAllAssumptions	0,0 %	0,0 %	0,0 %	0,0 %	–	–
getIsAssumption	100,0 %	100,0 %	–	100,0 %	–	–

Show methods with Term Coverage > 90,5 %

Name	Statement	Branch	Loop	Term	?-Operator	Synchronized
ToolChainAnalyzer	6,1 %	5,3 %	0,0 %	4,6 %	–	–
metamodel	6,1 %	5,3 %	0,0 %	4,6 %	–	–
de	6,1 %	5,3 %	0,0 %	4,6 %	–	–
validas	6,1 %	5,3 %	0,0 %	4,6 %	–	–
iso26262	6,1 %	5,3 %	0,0 %	4,6 %	–	–
checks	6,1 %	5,3 %	0,0 %	4,6 %	–	–
AssumptionModel	6,1 %	5,3 %	0,0 %	4,6 %	–	–
getIsAssumption	100,0 %	100,0 %	–	100,0 %	–	–

```
* thoses Use Cases that are no assumptions and the Assumed Uses Cases are
* returned only in case it is allowed
*/
public class AssumptionModel {

    /**
     * returns the IsAssumption for an item, ensures that assumption settings
     * are "inherited" from Tool->UseCae->Error->...
     */
    public static boolean getIsAssumption(Object item) {
        if (item instanceof Error) {
            Error uce = (Error) item;
            return uce.isIsAssumption() || (uce.getUseCase() != null && getIsAssumption(uce.getUseCase()))
                    || (uce.getRestriction() != null && getIsAssumption(uce.getRestriction()))
                    || (uce.getCheck() != null && getIsAssumption(uce.getCheck()));
        }
        if (item instanceof Check) {
            Check ck = (Check) item;
            return ck.isIsAssumption() || getIsAssumption(ck.getUseCase());
        }
        if (item instanceof Qualification) {
            Qualification qual = (Qualification) item;
            return qual.isIsAssumption() || (qual.getUseCase() != null && getIsAssumption(qual.getUseCase()))
                    || (qual.getTool() != null && getIsAssumption(qual.getTool()));
        }
        if (item instanceof Restriction) {
            Restriction res = (Restriction) item;
            return res.isIsAssumption() || getIsAssumption(res.getUseCase());
        }
        if (item instanceof UseCase) {
            UseCase uc = (UseCase) item;
            return uc.isIsAssumption() || getIsAssumption(uc.getTool());
        }
```

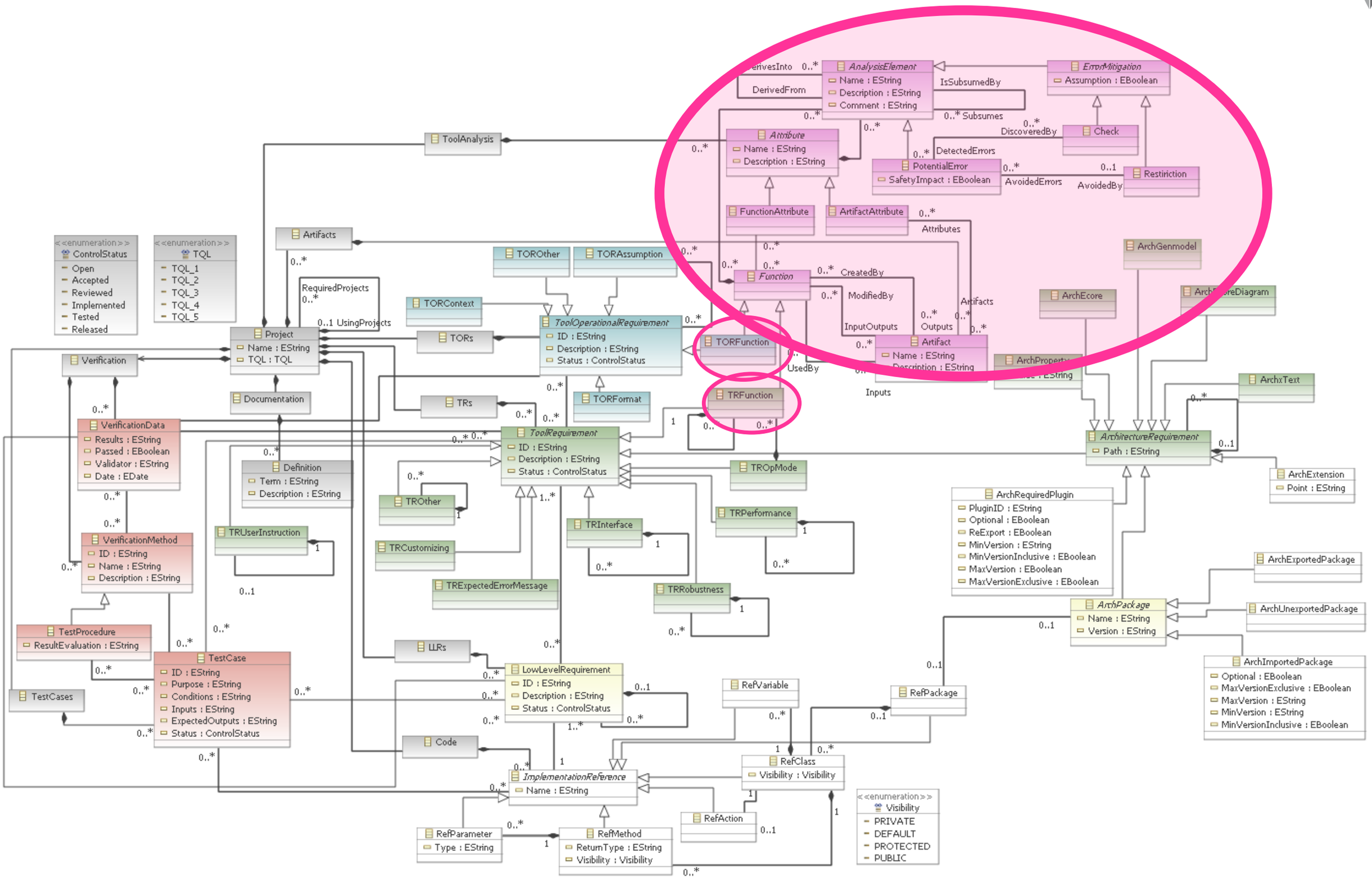
Content

- Roadmap
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - **Third Steps: Planning Tool Analysis and Life Cycle**
  - Fourth Steps: Life Cycle Refined, PSAC Generation, CM
  - Remaining Steps
- **Summary**

Planning: Tool Analysis for PSAC

- **Determines “qualification needs” of used tools**
- **Qualification Need: “Required Confidence => Tool Qualification Level (TQL)”**
- **Required Confidence (and mapping to TQL) depends on domains**
  - DO-178C: Criteria 1 – Criteria 3
  - ISO: Tool Confidence Level based on Error Analysis in Use Cases
  - IEC 61508: Tool Classification: T1 – T3
- **Tool Chain Analysis Method (RECOMP) can be applied in all domains to determine the Required Confidence**
- **Simple Tool Chain Analysis Model has been added to DO-330 meta model**
- **Connections to existing model (“TORFunction”, “TRFunction”)**

Planning: Analysis Model for PSAC



Planning: “How-To Qualify” Document

- **General explanations**
- **Conformance to DO-330 (bidirectional Tracing)**
  - Structures according to DO-330
  - Identification of Requirements
  - Tables for Tracing
  - Tracing against IDs is also contained in other Documents like TDP, TVP,…

eclipse VALIDAS

How-To Qualify
Eclipse-Based Tools
Version 0.2

- **Bidirectional tracing ensures that not too much is models/requested within Eclipse qualification process**

eclipse VALIDAS

Tool Development Plan
for every
Qualifiable Eclipse Plugin
Version 0.6

If the tool plans have been adopted the compliance to DO-330 has
updating the tracing in [HowTo] and their
projects. Satisfies DO-330-T1-6, DO-33

Bidirectional Tracing

Identifier	Keyword	Satisfaction Comment
DO-330-4.1	Qualification Need	Satisfied by satisfying sub-items
DO-330-4.1.a	Identification	The identification (plugin/product name) of Eclipse products and plugins is reused
DO-330-4.1.b	Intended Use	Is done in the TORs model for the main plugin of the tool model (see section 4.3.1 in [TDP])
DO-330-4.1.c	Qualification Need	See Tool-Analysis part in the model (see section 4.2.2 and 4.2.3 in [TDP])
DO-330-4.1.d	TQLs	See Determination in section 4.2.4 in [TDP]

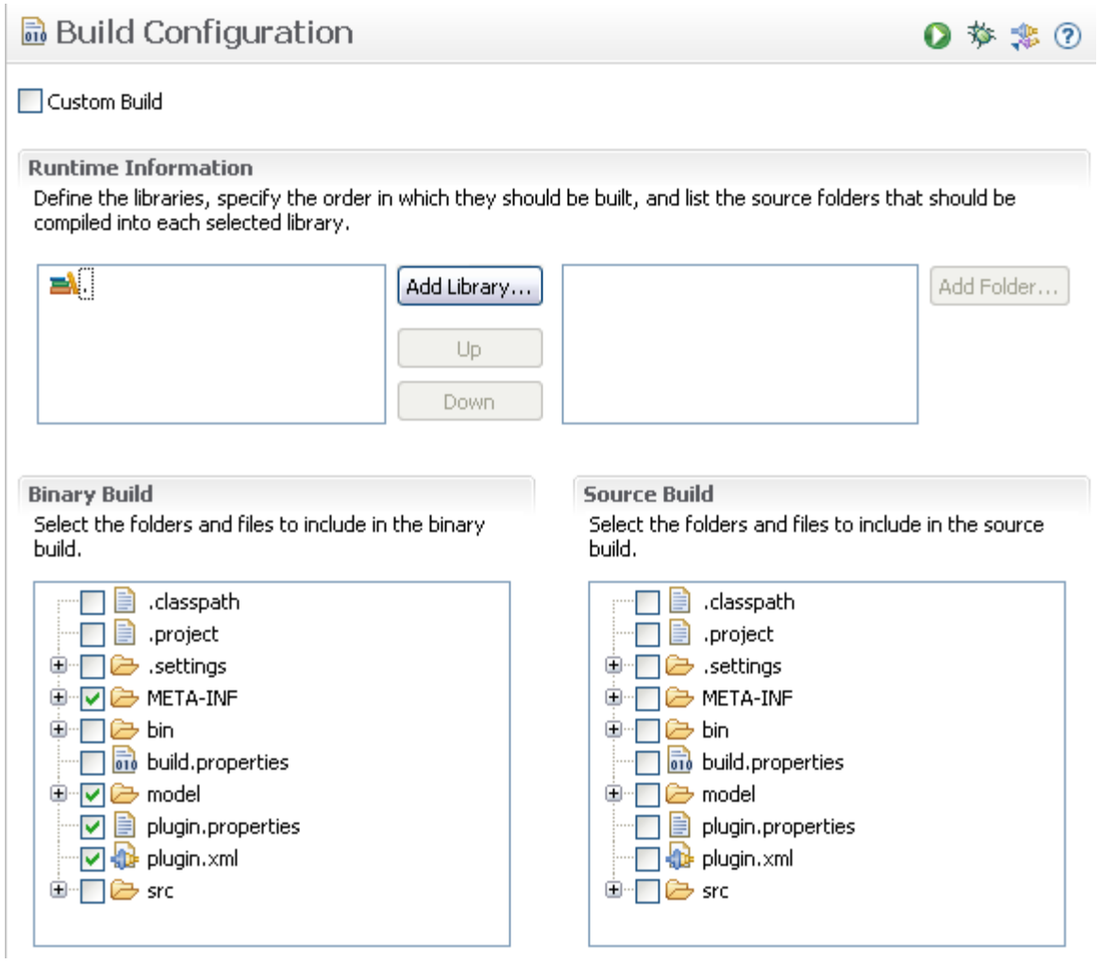
Table 2: Tracing Table to Tool Qualification Planning Process

Tool Development Plan

- **General Process Description for Qualifiable Eclipse Plugins**
- **Compliant to DO-330**
- **Can be adapted by developers (DO-330 compliance!)**
- **Contains description of how to use the model, i.e. standards for**
  - Requirements: TORs, TRs
  - Design, Architecture: TRs, LLRs
  - Implementation
- **Specific documents can be generated from the DO-330 model, the architecture and the (enriched) implementatio**
  - Requirements for <Tool Name>
  - Design for <Tool Name>
  - …
- **Examples for some specific document exists**
- **Similar document for verification: Tool Verification Plan**

Build Qualification Kit

- **Currently: 2 Builds available in Eclipse**
  - Source Build
  - Binary Build
- **Missing: Qualifiable Build Configuration with plugin specific**
  - Qualification information (DO-330 Model)
  - Test Cases / Coverage
  - Verification results
  - Documents
  - …



Content

- **Roadmap**
- Requirements for Tool Qualification (Standards)
- Proposals for Goals for Eclipse
- Proposals for some steps towards Tool Qualification
- Steps on the road
  - First steps: Requirements handling
  - Second steps: Design, Coding and Test
  - Third Steps: Planning Tool Analysis and Life Cycle
  - **Fourth Steps: Life Cycle Refined, PSAC Generation, CM**
  - Remaining Steps
- Summary

Tool Life Cycle for Qualifiable Plugins

- **Combines the following processes:**
  - Planning (TORs)
  - Development (TR, LLRs)
  - Integration (Verification)
  - Configuration Management
  - Quality Assurance
- **Fits to existing processes (Project process, Release Process) by extending them with a "Qualification Stage"**
- **The following stages are defined (and can be determined automatically from the DO-330 model) such that every release has a well-defined qualification stage**

  - **Unqualified-Pre-Alpha Release** ("**Undefined**"): unknown qualification state
  - **Qualification Alpha-Release** ("**Analyzed**"): The TORs are defined and TQL is determined
  - **Qualification Beta-Release** ("**Feature-Complete**"): All requirements (TORs and TRs) are described and have traces to LLRs and Code
  - **Qualification Release Candidate** ("**Verification Defined**"): All required verification steps are defined. No open bugs of the category "Blocker" are available.
  - **Qualification Release**: ("**Successfully Verified**") Verification has been successfully executed and are documented within the qualification kit
- **Transition Criteria are formally defined, based on the DO-330 model**

Life Cycle Transition Criteria

- **Defined in the "Tool Development Plan"**
- **Required by DO-330-4.2.1, DO-330-4.2.2, DO-330-4.3.b**
- **Quite formal definition (can be checked automatically) based on the DO-330 model of the tool**
- **Example (truncated): Transition to Qualification Alpha State ("Analyzed")**

- The *Project* has a nonempty *Name*, *Provider, Validator*,
- The *Project* has a *ControlStatus=Reviewed*
- The *Project* has the following TORs specified (in a *TORs* container):
  - At least one *TORFunction* defined. All *TORFunction* elements have
    - nonempty *ID*
    - nonempty *Description*
    - *ControlStatus=Reviewed*
  - At least one *TORContext* defined. All *TORContext* e
    - nonempty *ID*
    - nonempty *Description*
    - *ControlStatus=Reviewed*
  - At least one *TORFormat* defined. All *TORFormat* e
    - nonempty *ID*
    - nonempty *Description*
    - *ControlStatus=Reviewed*

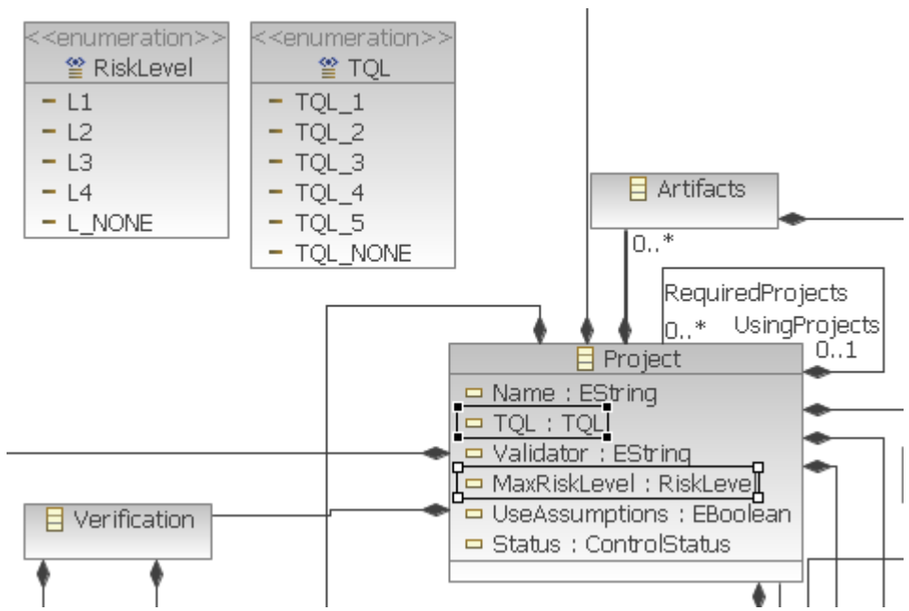
All *TORFunction* elements should have

- at least one *PotentialError* in the *AnalysisElements* composition
- For every potential error in the *TORFunction* which has an assigned mitigation (check/restriction) the shall be an artifact flow (to/from) the mitigation's *TORFunction*, if the mitigation's *TORFunction* is different from the *TORFunction* of the *PotentialError*.
- A set of "derived errors", consisting of
  - all errors (*AnalysisElements* of kind *PotentialError*) of the assigned *FunctionAttributes* and
  - all errors (*AnalysisElements* of kind *PotentialError*) of the *ArtifactAttributes* of the *Artifact* are *CreatedBy* or *ModifiedBy* the *TORFunction*. Note that if a *TORFunction* has several outputs with the same *ArtifactAttribute* element assigned, than the errors of the *ArtifactAttribute* are multiple times in the set with a different *ID* that refers to the *Artifact* in which they can occur.
- For each derived error in the set there is either
  - a copy of the *PotentialError* contained in the *TORFunction* or
  - another *PotentialError* contained in the *TORFunction* that subsumes the derived error, i.e. has the *PotentialError* of the *AnalysisAttribute* in the association *Subsumes*.

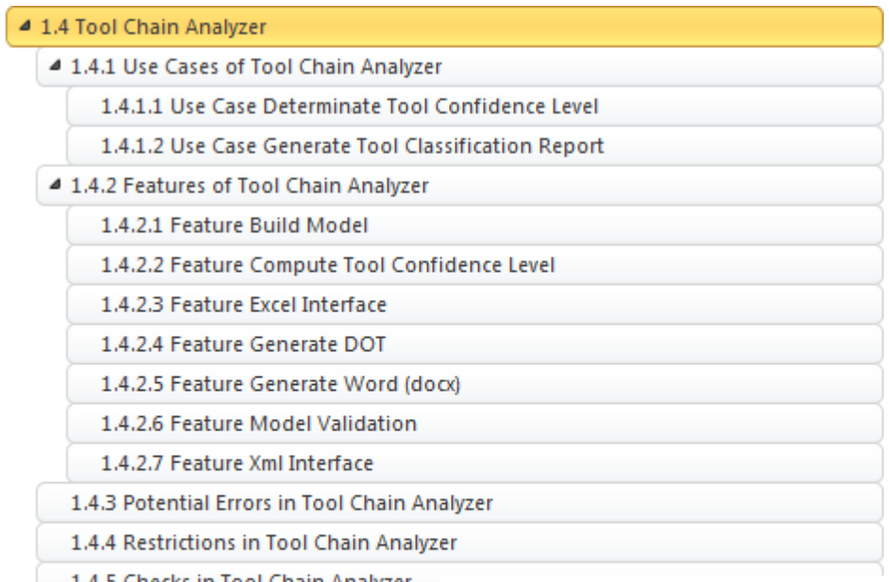
Tool Analysis in PSAC

- **From the Qualification Alpha Release (“Analyzed”) of the plugin/tool**
- **Verify the TQL (from qualification needs and Projects max. “RiskLevel”)**
  - L1: Highest Level (ASIL D, Risk Class A,..)
  - L2, L3
  - L4: Lowest Level (ASIL A, Risk Class D,..)
  - L_NONE for uncritical plugins
- **TQL can also be TQL_NONE for L_NONE or no qualification need**
- **List the assumptions of the analysis**
- **Generate Documentation for the PSAC that justifies the TQL**

Similar to Validas Tool Chain Analyzer: Checks & Computations



Generated PSAC Example from TCA



1.4 Tool Chain Analyzer

This section explains the determination of the Tool Confidence Level (TCL) for the to
Chain Analyzer.

Tool: Tool Chain Analyzer
Description:
This is the Tool Chain Analyzer from Validas AG
Impact:
TI 2 (Impact)
Tool Confidence Level:
TCL 1

Table 25 Tool: Tool Chain Analyzer

The tool Tool Chain Analyzer is modeled with 11 elements which have impact, 0 of th
assumptions. In addition there have been modeled 7 features, 0 of them are assumptior

Elements	Amount (Assumptions)
Use Cases	2 (0)
Checks	1 (0)
Restrictions	0 (0)
Qualifications	0 (0)
Potential Errors	8 (0)

Table 26 Amount of Elements in Tool: Tool Chain Analyzer

1.4.1 Use Cases of Tool Chain Analyzer

This section describes all analyzed use cases of Tool Chain Analyzer in separate subse

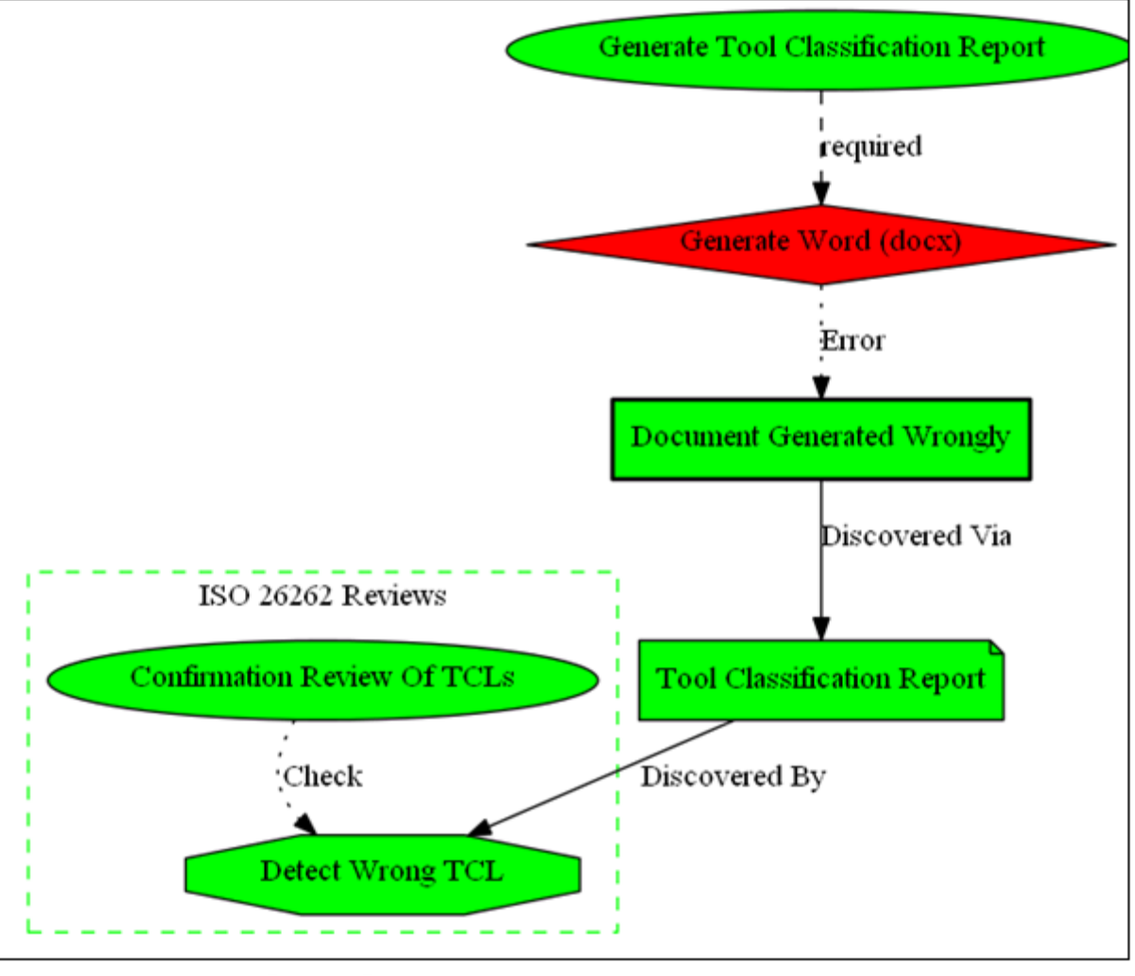
The following use cases of the tool Tool Chain Analyzer are considered:

1. Determinate Tool Confidence Level, see Section 1.4.1.1
2. Generate Tool Classification Report, see Section 1.4.1.2

1.4.1.1 [Use Case Determinate Tool Confidence Level]

This section describes the use case "Determinate Tool Confidence Level".

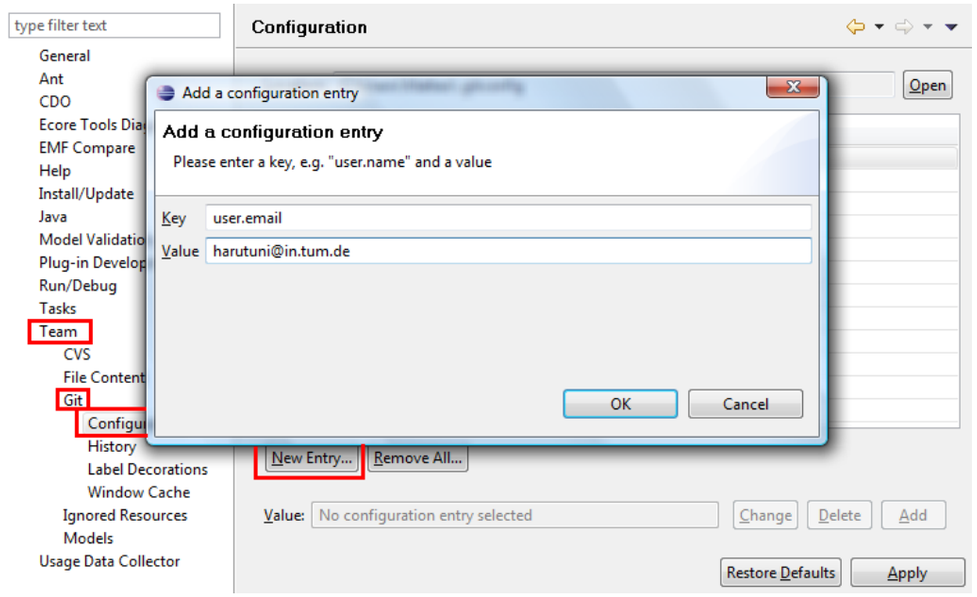
Error: Document Generated Wrongly
Description:
Document does not fit to the model.
From feature:
Generate Word (docx)
Discovered by the following checks:
• Confirmation Review Of TCLs Detect Wrong TCL
Subsumes:
• "No File Created" from "File Generator" • "Semantic Error" from "File Generator" • "Syntax Error" from "File Generator"
Occurrences:
• in Generate Word (docx) in Generate Tool Classification Report
Error View:



[Table 49 Error: Document Generated Wrongly

Configuration Management

- We require eGit and Gerrit to be used for qualifiable Eclipse plugins
- Some details (branch-names, configuration..) to be discussed (currently with BMW-CarIT)
- Informations
  - http://www.eclipse.org/egit/
  - http://progit.org/book/
  - http://www.slideshare.net/stefanlay/eclipse-git-und-gerrit



- Described in Tool Development Plan (Version 0.8)
- Traced against “How-To-Qualify” Document (DO-330)

Tool Development Plan

Configuration Management

- **Configuration Items are all elements within the Qualifiable Eclipse Project**
  - Sources
  - Architecture
  - DO-330-model
    - Requirements (TORs, TRs,
    - Tracing
    - ….
- **Two Control Categories: CC1, CC2. Item's CC depends on TQL**

	Tool Operational Requirements Process										Control Category by TQL				
											1	2	3	4	5
2	Tool Operational Requirements are defined.	5.1.1.a	5.1.2.a 5.1.2.b 5.1.2.c	❍	❍	❍	❍	❍	Tool Operational Requirements	10.3.1	①	①	①	①	②

- **Definition of Control Categories (DO-330):**

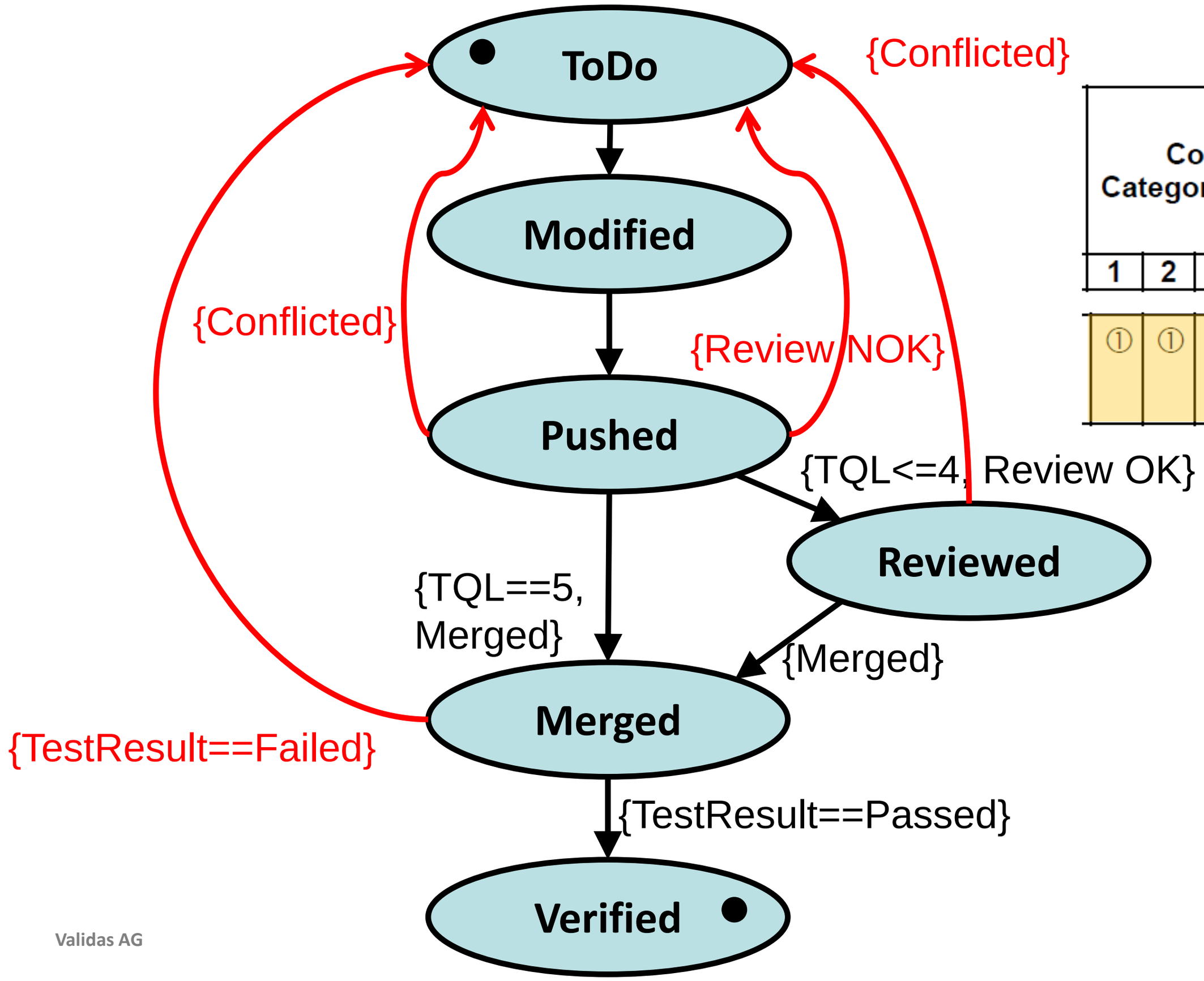
Table 7-1 TCM Process Activites Associated with CC1 and CC2 Data

TCM Process Activity	Reference	CC1	CC2
Configuration Identification	7.2.1	•	•
Baselines	7.2.2.a 7.2.2.b 7.2.2.c 7.2.2.d 7.2.2.e	•	
Traceability	7.2.2.f 7.2.2.g	•	•
Change Review	7.2.5	•	

Example: TORs **changes** have to be **reviewed** for TQL-1 to TQL-4 but not for TQL-5

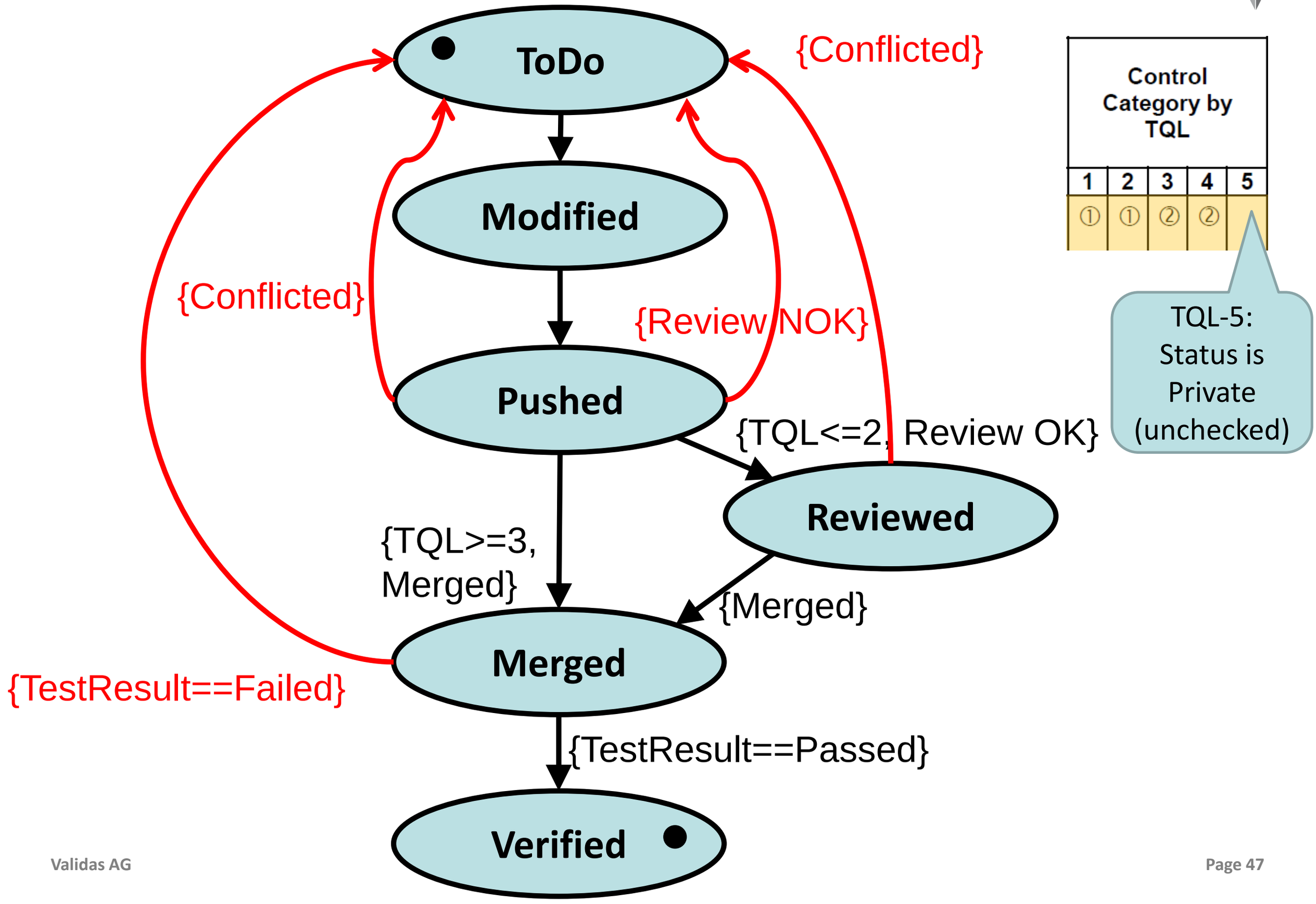
Plugin Extension has to know this (Transition Criteria!)

CM: Control Status of TORs (Proposed)



Control Category by TQL				
1	2	3	4	5
①	①	①	①	②

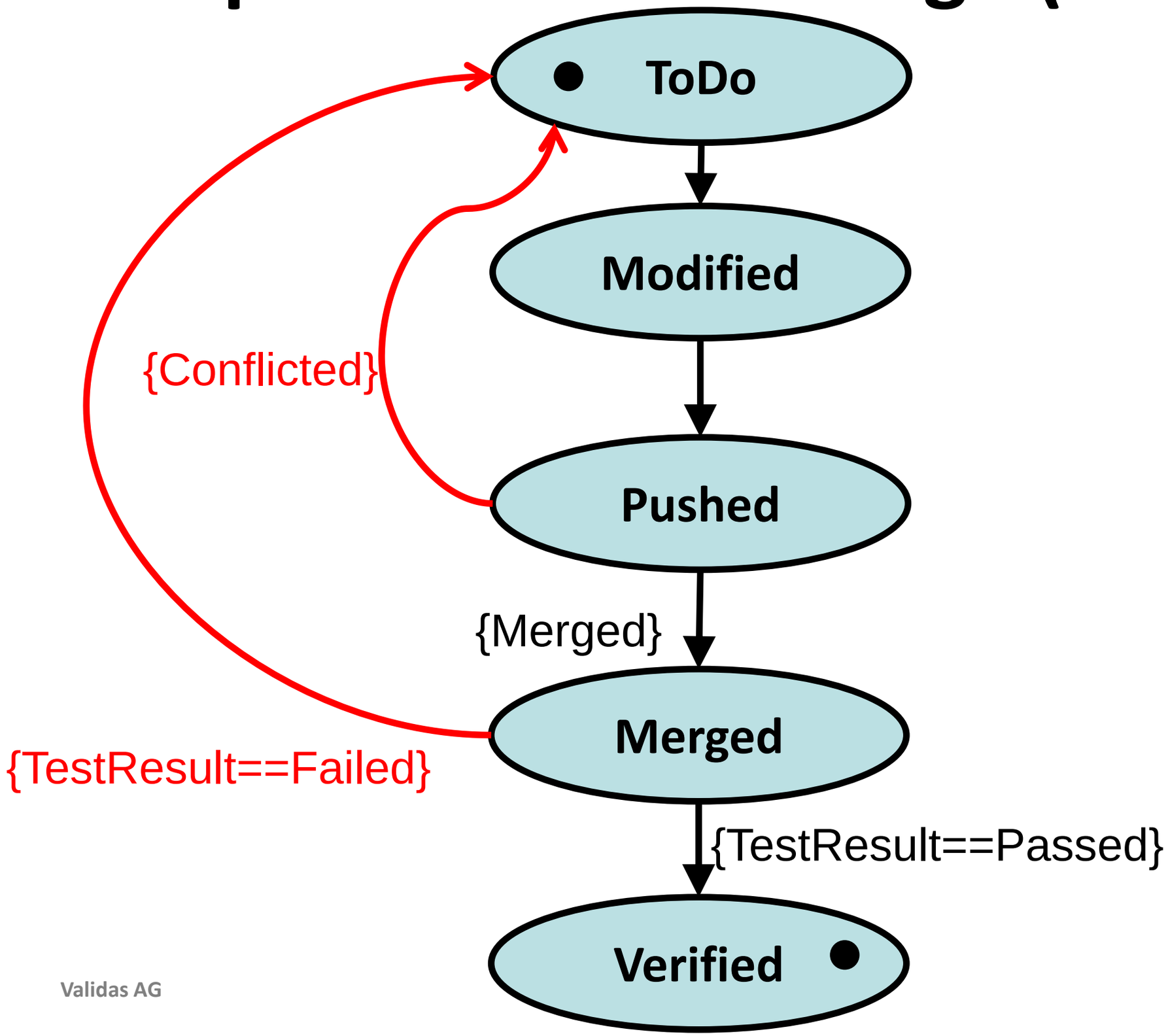
CM: Control Status of Tests (Proposed)



Control Category by TQL				
1	2	3	4	5
①	①	②	②	

TQL-5: Status is Private (unchecked)

CM: Control Status of TRExpectedErrorMessage (Proposed)

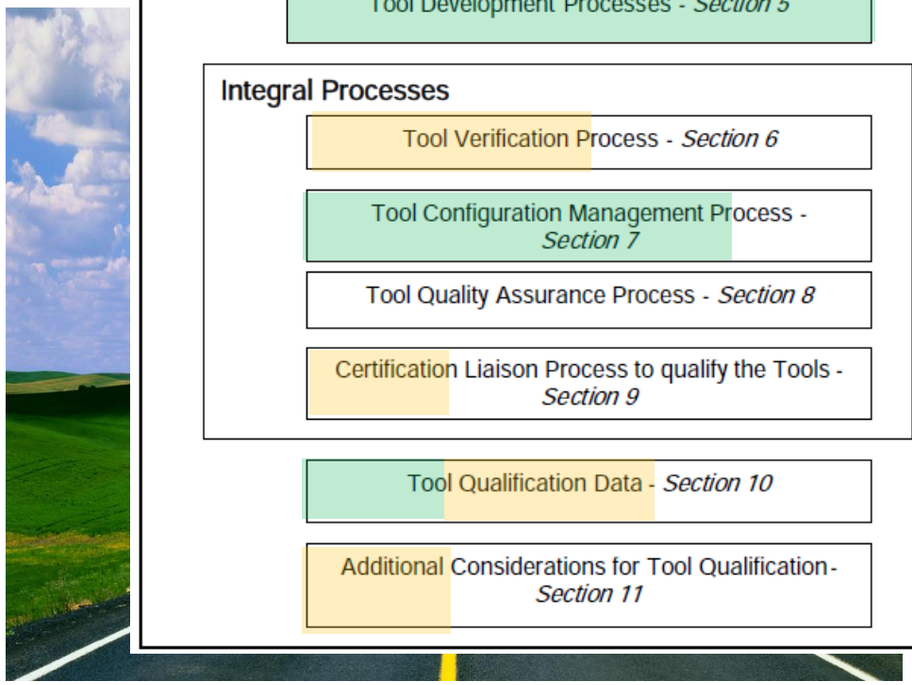


Control Category by TQL				
1	2	3	4	5
②	②	②	②	

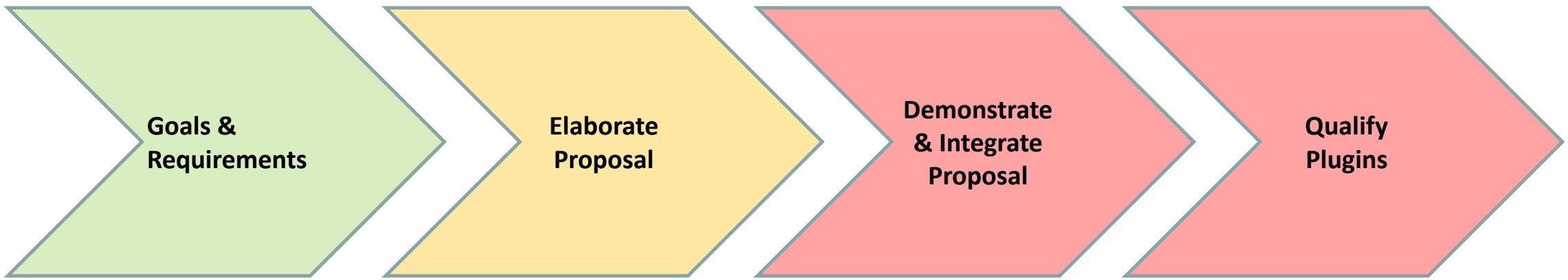
TQL-5: Status is Private (unchecked)

Roadmap - Status May 2012

1. **Identify goals & requirements for tool qualification in Eclipse**
2. **Propose process / project**
3. **Demonstrate tool qualification & integrate proposal into Eclipse Plugin Framework**
4. **Establish proposal: Qualify (selected) plugins**



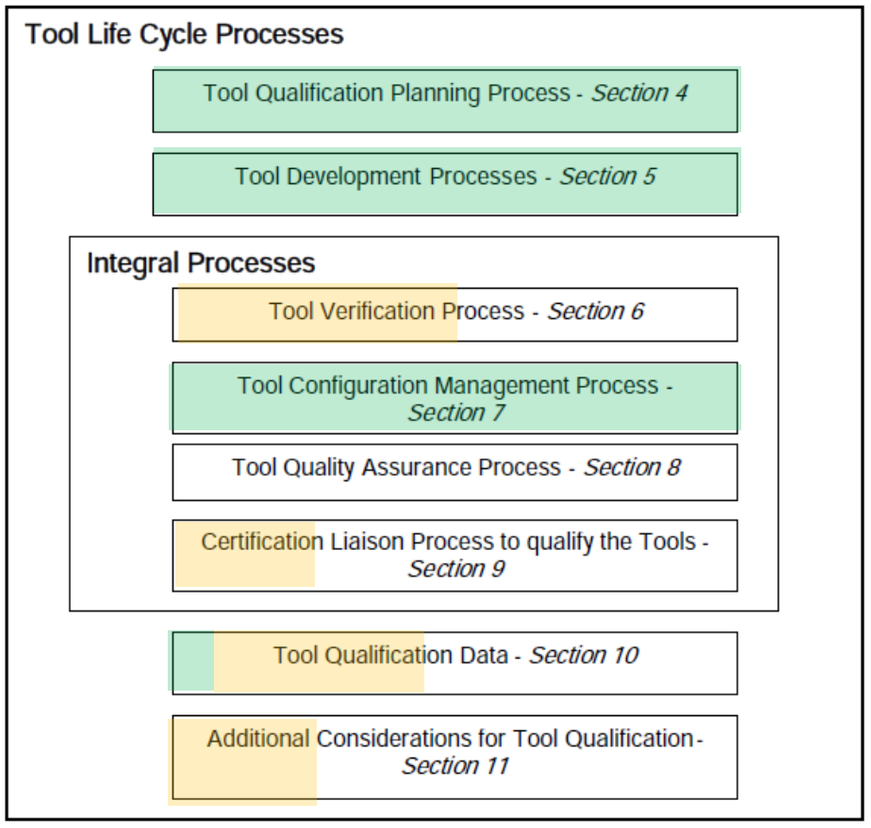
- **Status May 2012**



- **Summary: Qualification is feasible and qualification (based on current prototype) could be started now**

Summary

- **Roadmap towards development of qualifiable Eclipse tools & plugins**
  - Classification: Tool Analysis -> Planning Process
  - Qualification: Process & Model for Qualifiable Plugins
  - Usage: Fulfill Assumptions and apply qualification kits
- **Applicable to all relevant standards (ISO 26262, IEC 61508, DO-178C, EN 50128,..)**
- **Metadata extension for qualification information of plugins: DO-330 model**
- **Much work in progress**
  - Tracing to How-To-Qualify Document
  - Modeling: gaps to current meta-information
  - Create documentations (TDP,TVP,..)
- **First, second, thirds, fourth steps performed**
- **Proposed new role for that work: Eclipse Validator**
- **Validas contributes**



Thank You!

VALIDAS

Arnulfstraße 27
80335 München
www.validas.de
info@validas.de