Oscar Slotosch, Validas AG

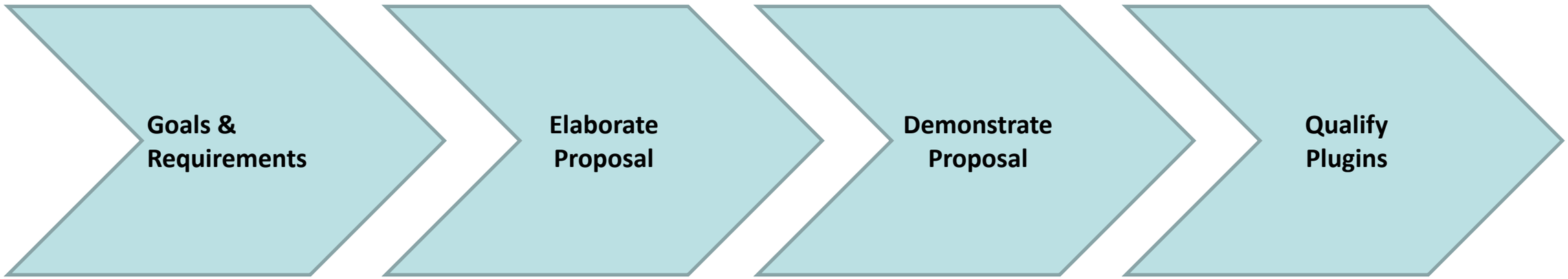
Walk on Roadmap towards Development of Qualifyable Eclipse Tools

Content

- **Roadmap and Proposal Elaboration Process**
- First Steps: Requirements
- Second Steps: Design, Code, Test

Roadmap

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



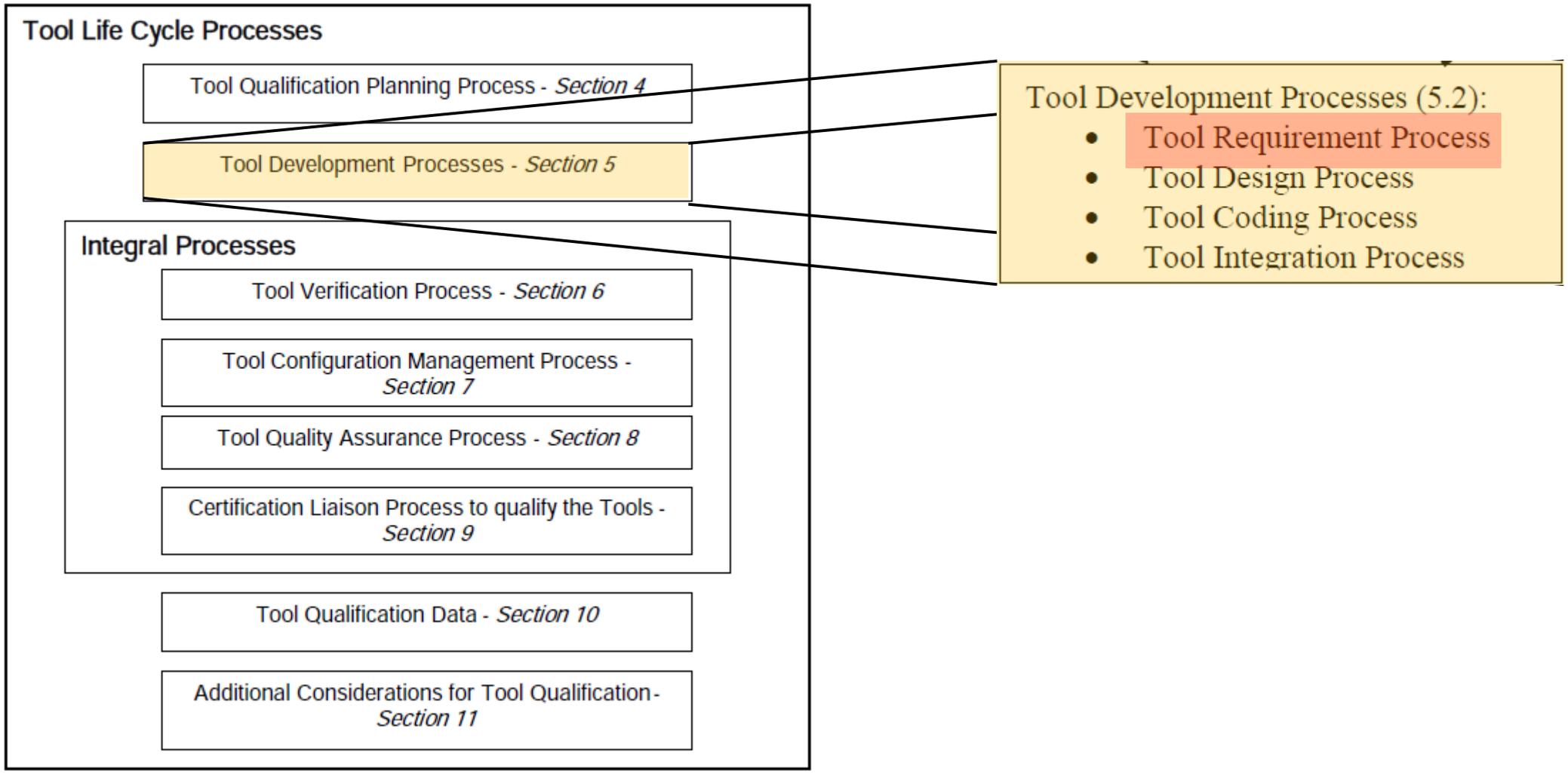
- Is this a Eclipse project? Not a typical ☺
- Is this an Industrial Working Group process?

Proposal Elaboration

1. **Selected DO-330 as general Tool Qualification Standard and Eclipse Plugin Framework (EPF) as modularization for qualification**
2. **Bases on Checklist for DO-330 compliance (documents & requirements)**
3. **Create example documents as required from DO-330**
   A. Generic documents: valid for all plugins
   B. Plugin-specific documents (to be generated from plugin specification/code)
4. **Validate the documents against checklist**
5. **Propose EMF-model to contain all information required to generate the plugin-specific documents**
6. **Create an example for the model**
7. **Topic-based process of modeling and creation of documents**
   1. First Step: Requirements
   2. Second Step: Design, architecture, code and testing
   3. Third Step: Verification
   4. Further Steps: Quality assurance, Configuration Management,…

DO-330 Topics

- **Structure of DO-330**



Checklist for DO-330 compliance

- **Create an Checklist for DO-330 compliance (unvalidated) draft:**

DO330.xlsx - Microsoft Excel

Datei | Start | Einfügen | Seitenlayout | Formeln | Daten | Überprüfen | Ansicht

Normal | Seitenlayout | Umbruchvorschau | Benutzerdef. Ansichten | Ganzer Bildschirm | Arbeitsmappenansichten

Lineal | Bearbeitungsleiste | Gitternetzlinien | Überschriften | Anzeigen

Zoom | 100 % | Fenster einfrieren | Zoom

Neues Fenster | Alle anordnen | Fenster einfrieren | Teilen | Ausblenden | Einblenden | Nebeneinander anzeigen | Synchroner Bildlauf | Fensterposition zurücksetzen | Aufgabenber. speichern | Fenster wechseln | Fenster

Makros | Makros

B46: The Tool Requirements should include all functional and interface-related requirements that will be implemented in the tool.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	ID	Content	Process	Output	TQL1	TQL2	TQL3	TQL4	TQL5	TQL1	TQL2	TQL3	TQL4	TQL5
44	5.2.1.2	Tool Requirements Activities	Tool Development Life Cycle	see subitems										
45	5.2.1.2.a	The Tool Requirements should be developed and docu	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
46	5.2.1.2.b	The Tool Requirements should include all functiona	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
47	5.2.1.2.c	The Tool Requirements should be developed followin	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
48	5.2.1.2.d	The Tool Requirements should be developed using th	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
49	5.2.1.2.e	The Tool Requirements should be verifiable and consi	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
50	5.2.1.2.f	The Tool Requirements should be defined such that al	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
51	5.2.1.2.g	Each Tool Requirement should trace to one or more To	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
52	5.2.1.2.h	Derived tool requirements are those not traceable to	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
53	5.2.1.2.i	The Tool Requirements should provide user instructio	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
54	5.2.1.2.j	The Tool Requirements should identify requirements	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
55	5.2.1.2.k	The Tool Requirements should be defined to a level o	Tool Development Life Cycle	Tool Requirements/ Trace Data	Sat	Sat	Sat	Sat	Private	CC1	CC1	CC1	CC1	Private
56	5.2.2	Tool Design Process	Tool Development Life Cycle	see subitems, 10.2.2										

Requirements & Activities | TCM Process Activities | Tabelle3

Bereit | 100 %

Content

- Roadmap and Proposal Elaboration Process
- **First Steps: Requirements**
- Second Steps: Design, Coding, Test

Create DO-330 Conformant Requirements

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

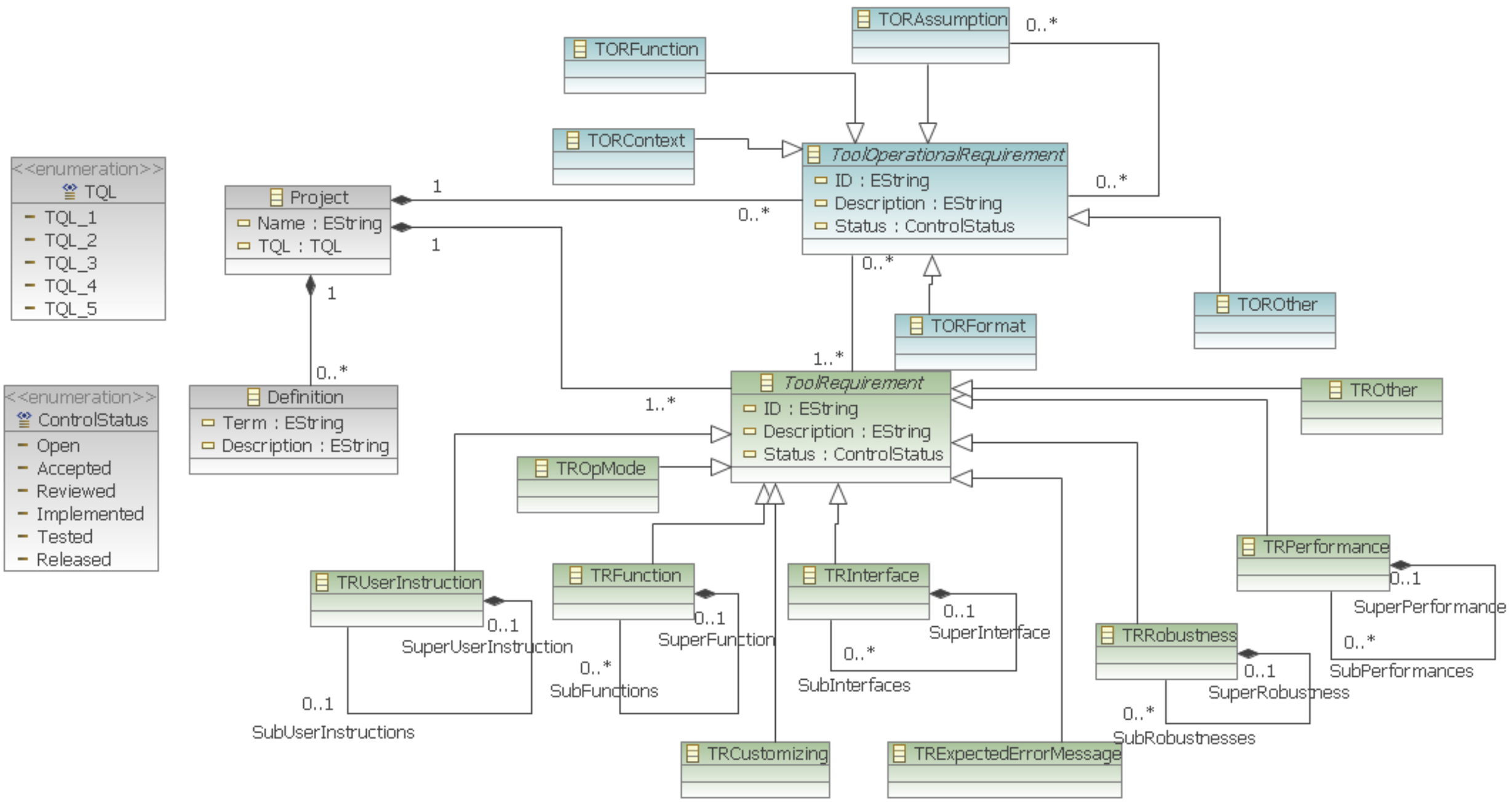
The tool chain models shall be persistent to files. The tool chain analyzer loads models from files and writes the back into files.

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.

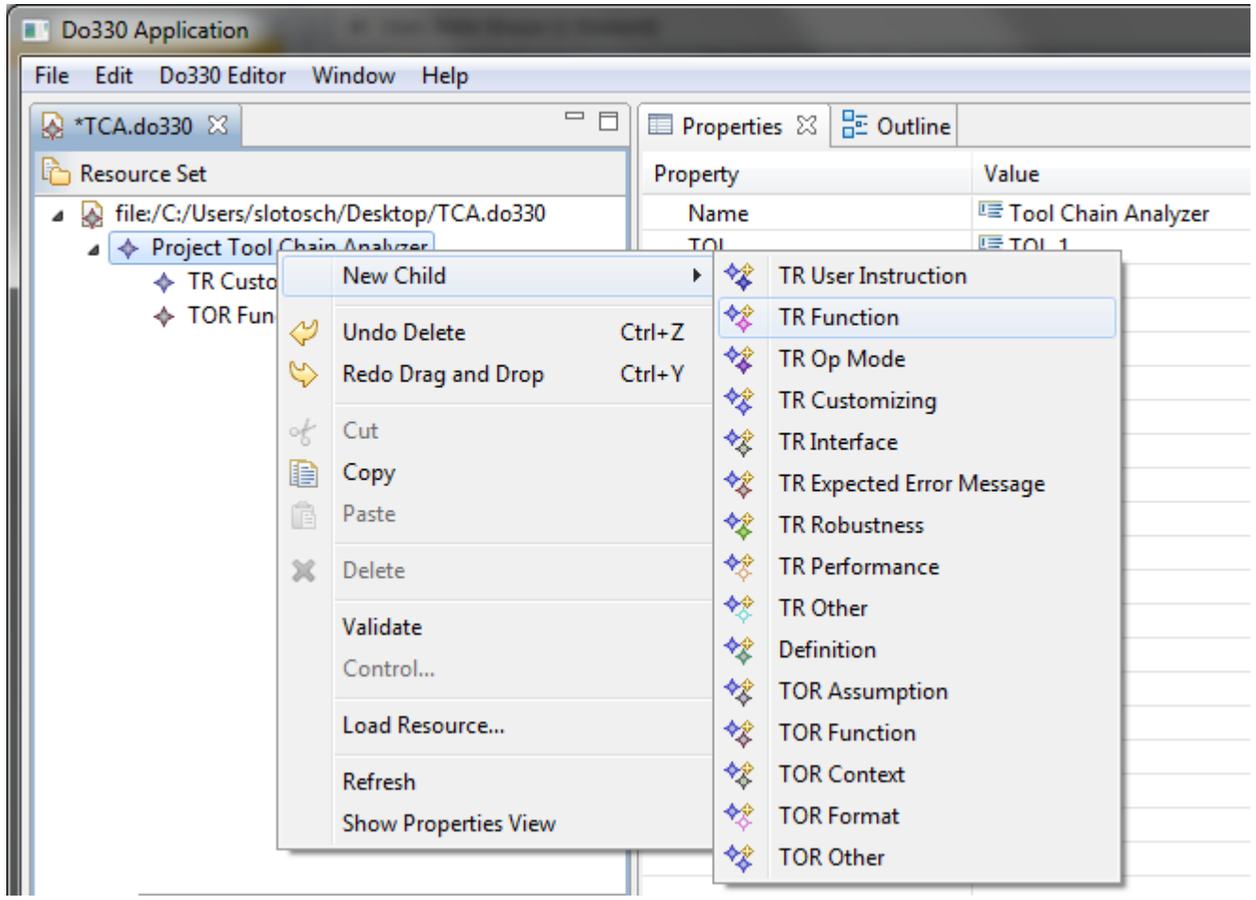
Model for Tool-Requirements

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

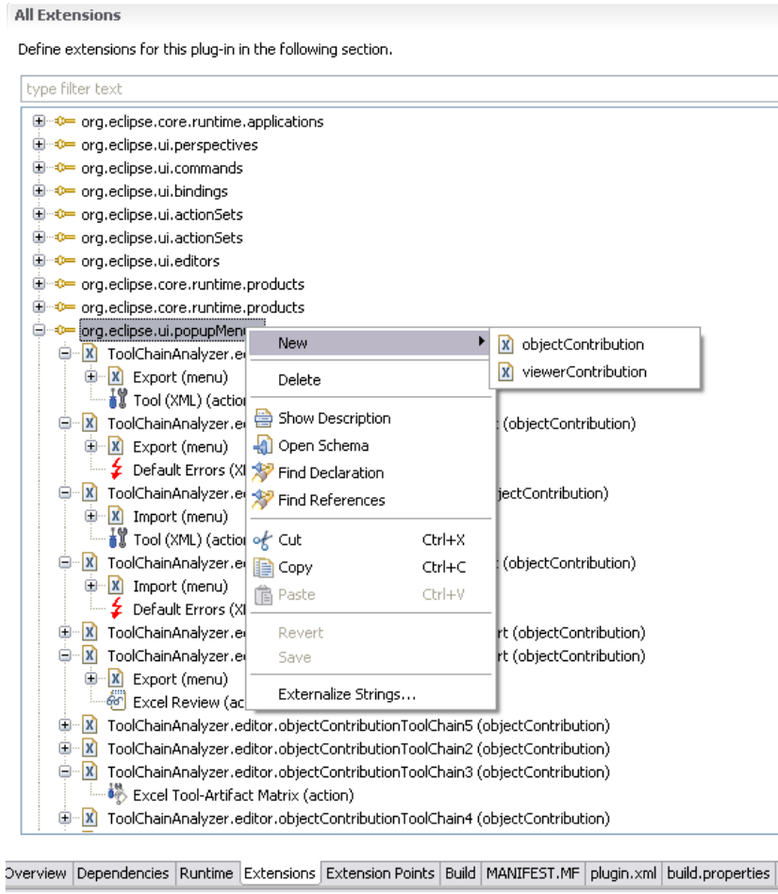


Create Example Model

- **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

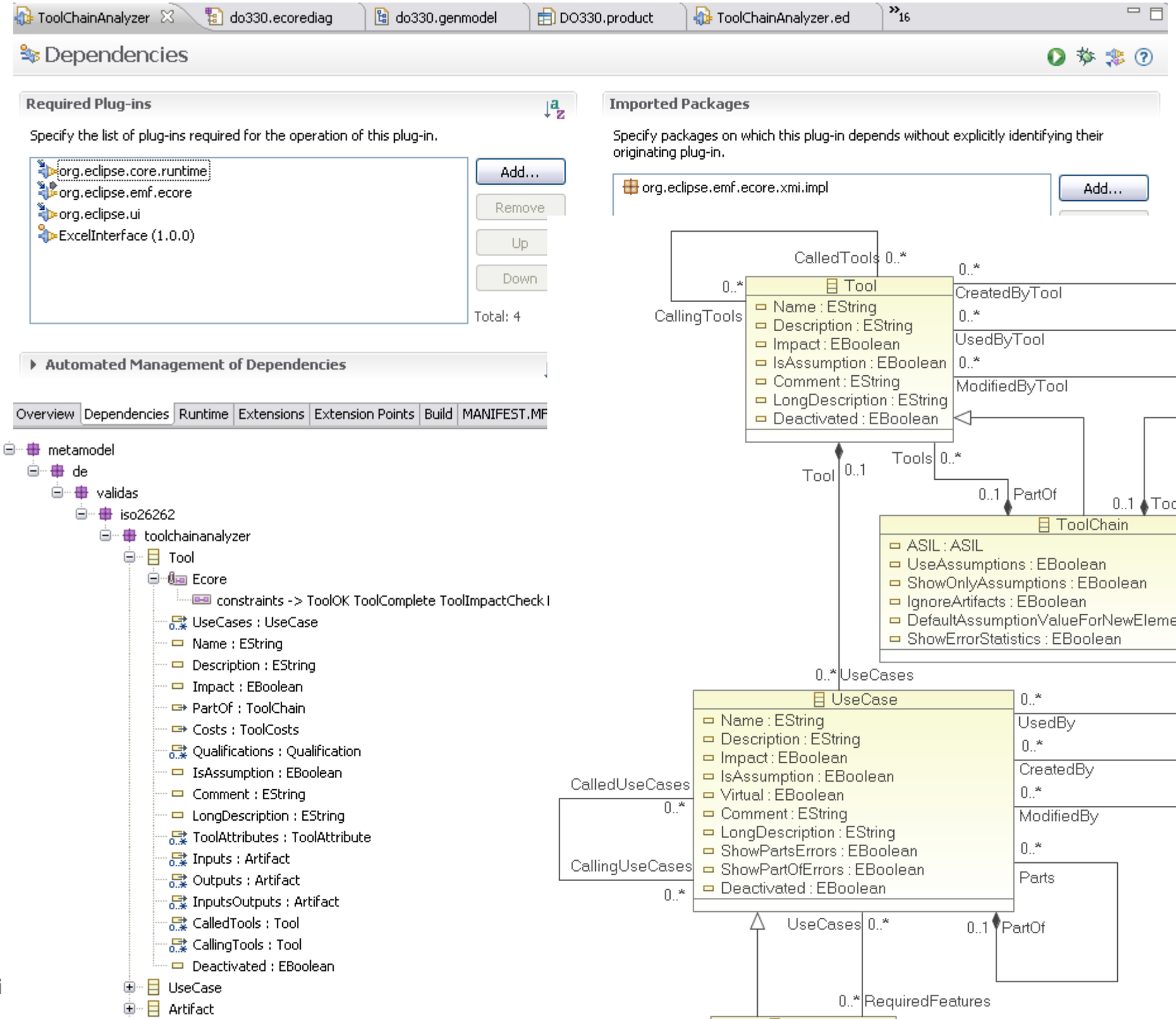
- Roadmap and Proposal Elaboration Process
- First Steps: Requirements
- **Second Steps: Design, Coding, Test**

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**

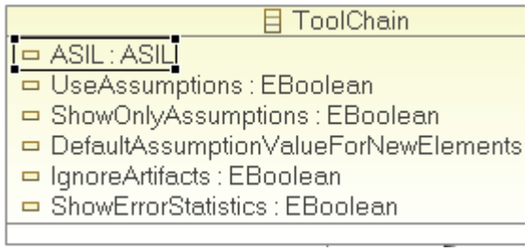
Architecture Examples in Eclipse

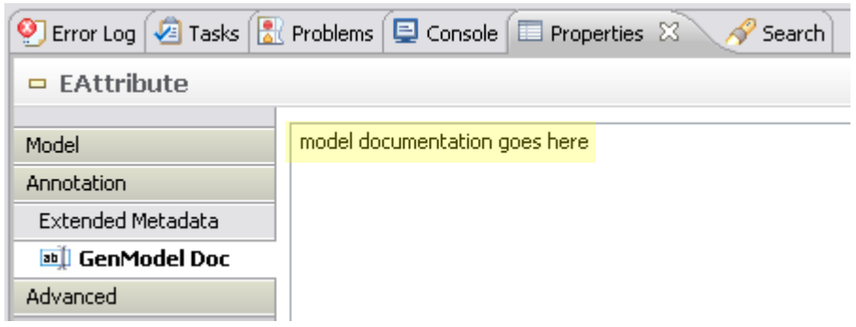
- **Architecture: Plugins & packages, EMF models, xText grammars**



Example EMF Code

- **Model generates code, interface (and description!)**





- **Documentation can be inserted into code and model**

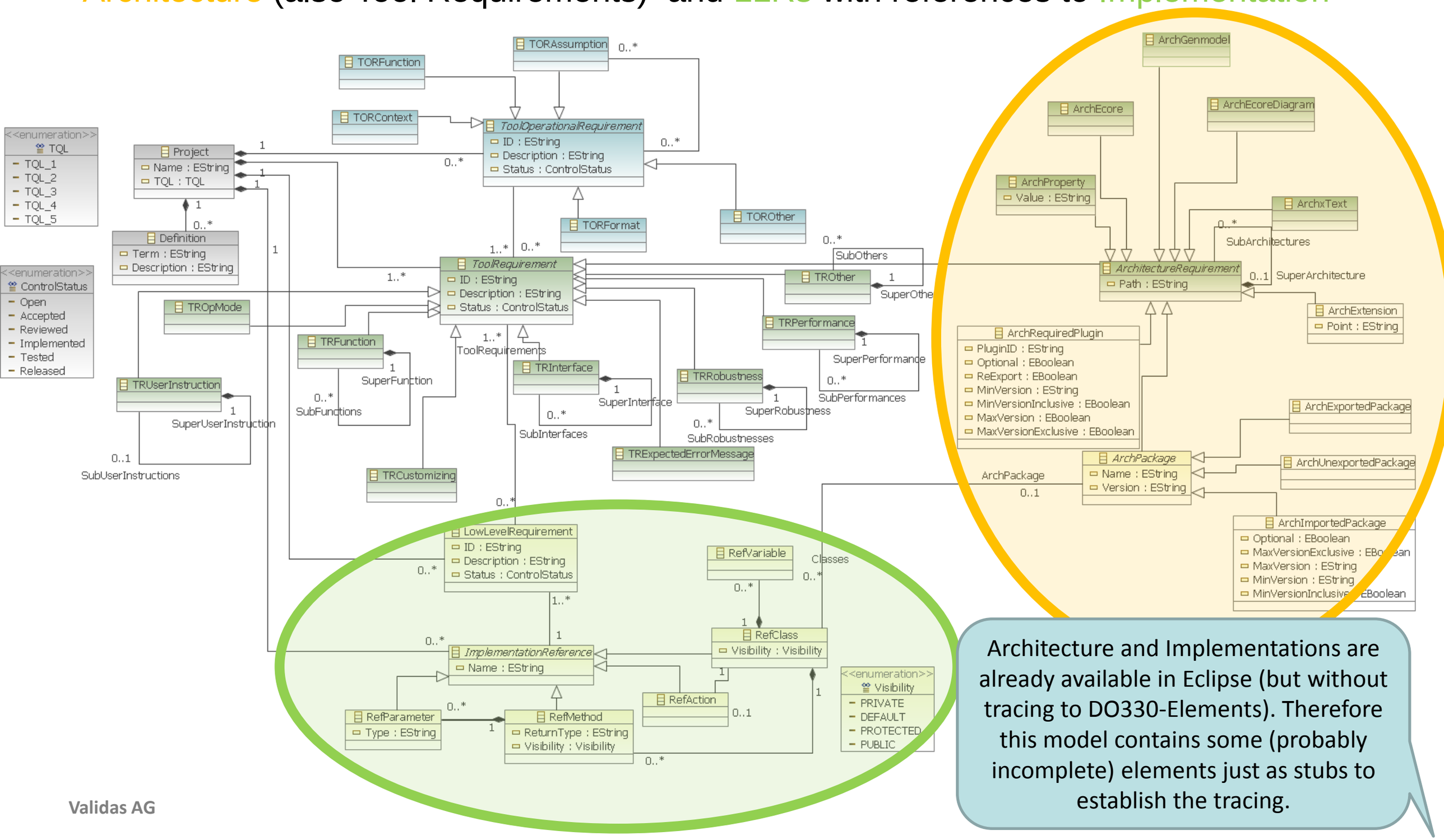
```
/**
 * Returns the value of the '<em><b>ASIL</b></em>' attribute.
 * The literals are from the enumeration {@link metamodel.de.validas.iso26262.toolchainanalyzer.ASIL}.
 * <!-- begin-user-doc -->
 * <p>
 * here is the specific code description of the return value '<em>ASIL</em>'
 * </p>
 * <!-- end-user-doc -->
 * <!-- 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?)

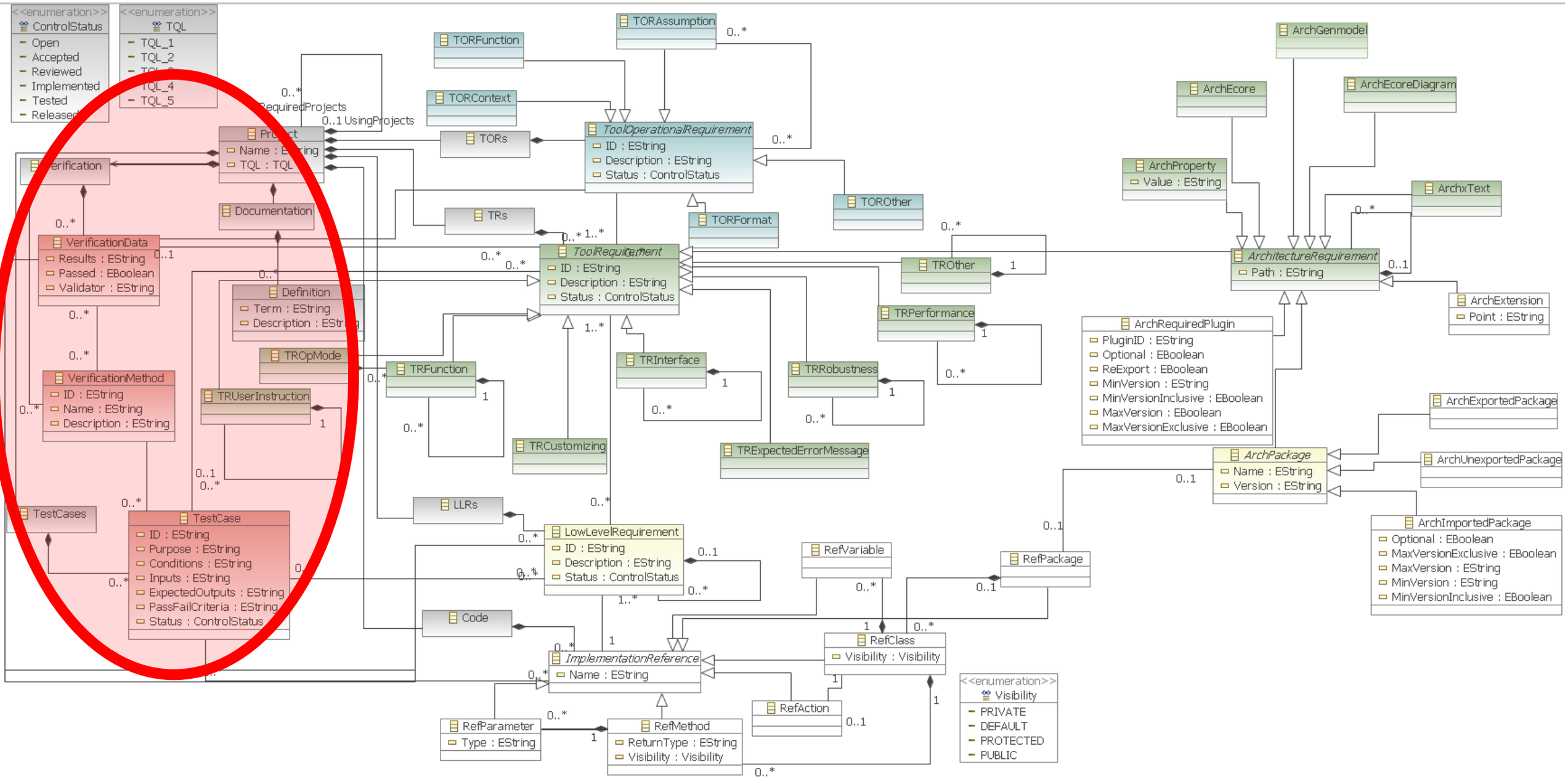
Design Model

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

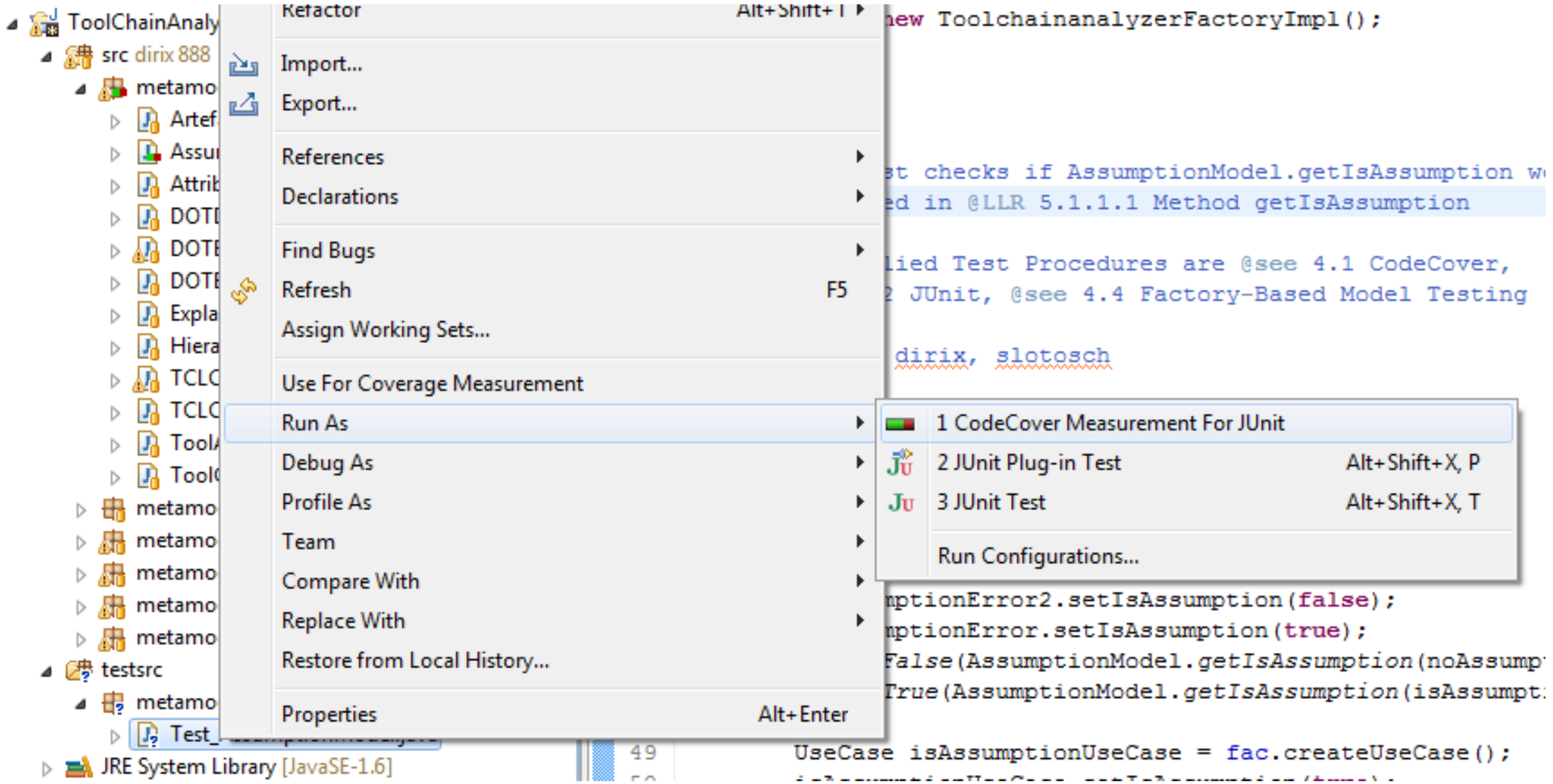

<<enumeration>>
ControlStatus
Open
Accepted
Reviewed
Implemented
Tested
Released
<<enumeration>>
TQL
TQL_1
TQL_2
TQL_4
TQL_5
TORAssumption
TORFunction
TORContext
TORs
ToolOperationalRequirement
ID : EString
Description : EString
Status : ControlStatus
TOROther
TORFormat
TRs
ToolRequirement
ID : EString
Description : EString
Status : ControlStatus
TROther
TRPerformance
TRFunction
TRInterface
TRRobustness
TRCustomizing
TRExpectedErrorMessage
TRUserInstruction
TROpMode
Definition
Term : EString
Description : EString
Documentation
VerificationData
Results : EString
Passed : EBoolean
Validator : EString
VerificationMethod
ID : EString
Name : EString
Description : EString
TestCases
TestCase
ID : EString
Purpose : EString
Conditions : EString
Inputs : EString
ExpectedOutputs : EString
PassFailCriteria : EString
Status : ControlStatus
LLRs
LowLevelRequirement
ID : EString
Description : EString
Status : ControlStatus
Code
ImplementationReference
Name : EString
RefVariable
RefPackage
RefClass
Visibility : Visibility
RefAction
RefParameter
Type : EString
RefMethod
ReturnType : EString
Visibility : Visibility
<<enumeration>>
Visibility
PRIVATE
DEFAULT
PROTECTED
PUBLIC
ArchGenmodel
ArchEcore
ArchEcoreDiagram
ArchProperty
Value : EString
ArchxText
ArchitectureRequirement
Path : EString
ArchExtension
Point : EString
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

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());
        }
```

Thank You!

VALIDAS

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