



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 ③
- 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



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

	Mathada	ASIL				
	Methods	A	в	С	D	
1a	Increased confidence from use in accordance with 11.4.7	++	++	+	+	
1b	Evaluation of the tool development process in accordance with 11	++	++	+	+	
1c	Validation of the software tool in accordance with 11.4.9	+	+	++	++	
1d	Development in accordance with a safety standard ^a	t	+	++	++	

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 ++

Here is a hole

were the new

DO-330

standard fits in

. . . .

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
С	TQL-5	TQL-3	TQL-2
В	TQL-5	TQL-4	TQL-3
А	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

Softways Loval	Criteria				
Software Level	1	2	3		
А	TQL-1	TQL-4	TQL-5		
В	TQL-2	TQL-4	TQL-5		
С	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



🔈 Overview General Information **Plug-in Content** This section describes general information about this plug-in. The content of the plug-in is made up of two sections: ID: ToolChainAnalyzer **Dependencies**: lists all the plug-ins required on this plug-in's classpath to compile and run. Kuntime : lists the libraries that make up this plug-in's runtime. 1.5.3Version: %pluginName Name: Extension / Extension Point Content Provider: %providerName This plug-in may define extensions and extension points: Platform Filter: Extensions : declares contributions this plug-in makes to the platform. Activator: Browse... Extension Points : declares new function points this plug-in adds to the platform. Activate this plug-in when one of its classes is loaded Testing This plug-in is a singleton Test this plug-in by launching a separate Eclipse application:

Execution Environments

Specify the minimum execution environments required to run this plug-in.



- Launch an Eclipse application
- 🎋 Launch an Eclipse application in Debug mode

Exporting

To package and export the plug-in:

- 1. Organize the plug-in using the Organize Manifests Wizard
- 2. Externalize the strings within the plug-in using the Externalize Strings Wizard
- 3. Specify what needs to be packaged in the deployable plug-in on the <u>Build Configuration</u> page
- 4. Export the plug-in in a format suitable for deployment using the Export Wizard

Configure JRE associations...

Update the classpath settings

Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF plugin.xml build.properties

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 Validas AG

Validas AG 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"

Validas AG

Contains Requirements (IDs) and tracing to Process/Documents/Model

🔟 🔛 ୨ - ए 🔜 🚰 🖙 🛛 ਮ	lowToQu	alifyEclipseBasedTools.	doc [Kompatibilitätsmo	odus] - Micr	rosoft Word	
Datei Start Einfügen Seitenlayout Verweis	se Se	endungen Überprü	fen Ansicht Ad	id-Ins		^ ?
Image: Arrow of the system Arrow of the	• 🚇 • <u>A</u> •		╪╪╡ <mark>2</mark> ↓│¶ ≣∗│Ѯ҂⊡・	AaBbCcD I Beschrift	AaBbCcI 1 AaE 1 Standard 1 Übersch	Suchen 👻 Ersetzen Markieren 👻
Zwischenablage 🕞 Schriftart	Fa	Absatz	5 E		Formatvorlagen 🕞	Bearbeiten
Navigation Dokument durchsuchen	▼ × ρ -	4.2 Tracing	to Tool Qualifica	tion Plan	ning Process Section	Ca I
		The tool qualific analysis, similar (see FAQ D.2).	cation planning is d to the one propose From this analysis	lone using d by [ISO the TORs	a tool chain analysis model with an error-in 26262] (see part 8, chapter 11) and the [D0 are determined.	mpact D-330]
		DO-330-4.1	Qualification Nee	d be	Satisfied by satisfying sub-items	
1 Document History		DO-330-4.1.a	Identification		The identification (plugin/product name) of	of
2 Definitions			identification.		Eclipse products and plugins is reused	
3 Tool Qualification Process		DO-330-4.1.b	Intended Use		Is done in the TORs model for the main	
▲ 4 Traceability to DO-330					plugin of the tool model (see section 4.3.1	in
4.1 General Considerations					[TDP])	
4.2 Tracing to Tool Qualification Planning Process Sect.	.)	DO-330-4.1.c	Qualification Nee	d	See Tool-Analysis part in the model (see	
4.3 Tracing to Tool Development Life Cycle and Process.	.				section 4.2.2 and 4.2.3 in [TDP])	
4.4 Tracing to Tool Verification Process Section		DO-330-4.1.d	TQLs	:	See Determination in section 4.2.4 in [TDP	2
4.5 Tracing to Tool Configuration Management Proces		DO-330-4.1.e	Stakeholders	:	See "Provider" in MANIFEST.MF and	
4.6 Tracing to Tool Quality Assurance Process Section	1				"Validator" in project model in section 4.1	.1
4.7 Tracing to Tool Qualification Liaison Process Section	5 II				of [TDP] and the Validator in the verification	on
4.8 Tracing to Tool Qualification Data Section	- I				data model (see section 4.2.2 in [TVP])	
4.9 Tracing to Additional Considerations for Tool Qua.	3	DO-330-4.1.f	Tool Environmen	ť	The Environment is defined using the	
4.10 Tracing to Tool Qualification Objectives Section	\leq		Definition		TORContext requirements, see section	
5 Deferences	-				4.3.1.4 in [TDP]	
Silectenes		DO-330-4.2	Life cycle		See sections 5 and 5.1 in [TDP]	
		DO-330-4.2.a	Planning process		See section 5.1.1 in [TDP]	•
		DO-330-4.2.b	Development pro	ocess	See section 5.1.2 in [TDP]	±
		DO-330-4.2.c	Integral process	1	See section 5.1.3 in [TDP]	0
		DO-330-4.2.1	Transition criteria	a s	See section 5.3 in [TDP]	Ŧ

DO-330 Topics



Structure of DO-330



Existing Methods: Requirements



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



R Rec	qlf-Model.rif	R Specification Document 🛛	
R Spe	ecification Docu	iment	
	ID	Description	Link
1	C 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 / ...?
- Validas AG Do we need RIF within Eclipse?

Create DO-330 Conformant Example



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.

Document History

Version 0.2

Validas AG



Create Model for Tool-Requirements



EMF-Metamodel (Draft) for Tool Requirements



Create Example Model



Using the default EMF Editor

Do330 Application		C Descent			
File Edit Do330 Editor V	Vindow Help				
🐼 *TCA.do330 🖂		🗖 🗖 🔲 Pro	perti	es 🖾 🗄 Outline	
Para Resource Set		Prope	rty		Value
▲ 🙀 file:/C:/Users/slotosc	h/Desktop/TCA.do330	Na	ame		Tool Chain Analyzer
Project Tool Chai	n Analuzer			70.01 T	
TR Custo	New Child	•	¥	TR User Instruction	
🔶 TOR Fun 💛	Undo Delete	Ctrl+Z	*	TR Function	
5	Redo Drag and Drop	Ctrl+Y	*	TR Op Mode	
			*	TR Customizing	
et	Cut		*	TR Interface	
	Сору		*	TR Expected Error M	Message
Ê	Paste		*	TR Robustness	
×	Delete		*	TR Performance	
	Validate		*	TR Other	
	Control		*	Definition	
	Controllin		*	TOR Assumption	
	Load Resource		*	TOR Function	
	Refresh		*	TOR Context	
	Show Properties View	v	*	TOR Format	
			*	TOR Other	



All Extensions

Define extensions for this plug-in in the following section.

🗄 👓 org.eclipse.core.runtime.ap	plications		
🗧 🕬 org.eclipse.ui.perspectives			
🗝 🗢 org.eclipse.ui.commands			
🗝 🗢 org.eclipse.ui.bindings			
🗝 🗢 org.eclipse.ui.actionSets			
🗝 🗢 org.eclipse.ui.actionSets			
🗐 💝 org.eclipse.ui.editors			
🗝 🗢 org.eclipse.core.runtime.pr	oducts		
org.eclipse.core.runtime.pr	oducts		
🗧 🗢 org.eclipse.ui.popupMen	: New	,	objectContribution
😑 🕺 ToolChainAnalyzer.e	NOT		
🗄 🔣 Export (menu)	Delete		
Tool (XML) (action	Chan Daariatiaa		-
🖃 💹 ToolChainAnalyzer.e	Show Description		: (objectContribution)
🗄 🔟 Export (menu) 🚽	们 Open Schema		
🔁 🦆 Default Errors (XI 🕇	ኛ Find Declaration		
ToolChainAnalyzer.e	Find References		jectContribution)
🗄 🔟 Import (menu) –			-
Tool (XML) (action o	🗲 Cut	Ctrl+X	
ToolChainAnalyzer.e	📄 Copy	Ctrl+C	: (objectContribution)
Import (menu)	Paste	Ctrl+V	
Z Default Errors (XI -			-
I ToolChainAnalyzer.e	Revert		rt (objectContribution)
	Save		rt (objectContribution)
Export (menu)	Externalize String	5	
		Te alcheiart (- his state to the state (
ToolChainAnaiyzer.edit	or.objectContributio	n i ooiChains (- Taal Chaiao (objectContribution)
I toolChainAnaiyzer.edit	or.objectContributio	n i ooiChainz (- TICh-i-O (objectContribution)
I OOILINAINANAIYZEY.edit	or.opjectContribution	n rooichaiń3 (objectContribution)
Excel 1001-Artifact	Maurix (action)	-TaolChair 4 (object/Contribution)
	or.objectContributio	mooichain4 (objectContribution)

- 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!)

E ToolChain	ſ	👰 Error Log 🧔 Tasks [Problems 📃 Console 🔲 Properties 🛛 🦂
ASIL: ASIL UseAssumptions : EBoolean		EAttribute	
ShowOnlyAssumptions : EBoolean		Mandal	model documentation does here
DetaultAssumptionValueForNewElements IgnoreArtifacts : EBoolean		Model	model documentation goes here
ShowErrorStatistics : EBoolean		Fytended Metadata	
		Advanced	

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 -->
 * 
 * here is the specific code description of the return value '<em>ASIL</em>'
 * 
 * <!-- 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
 *7.
ASIL getASIL();
```

Search

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?)

```
* <copyright>
 * Validas AG
 * </copyright>
 * This class is the Editor Advisor q
 * @author: Oscar Slotosch, Reinhard -
 * TODO: review low level requirement;
 * @link generated from de.validas.to
 *
 * $Id$
 */
package metamodel.de.validas.iso26262
import java.io.File;[]
/**
 * Customized {@link WorkbenchAdvisor
 * <!-- begin-user-doc -->
 * RJ: this class has been generated :
 * <!-- end-user-doc -->
 * @requirements
 * 0
      🗐 @author - author name
     @ @author
publ @ @category
     @deprecated
     @ @see
     @ @serial
     @ @since
     @version
     @ {@code}
     @ {@docRoot}
```

Press 'Ctrl+Space' to show Template Proposals

Page 27

Design Model



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



Test Model





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; 22 230 @Before 24 public void setUp() { 25 fac = new ToolchainanalyzerFactoryImpl(); 26 } 27 280 @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



a 😹 ToolChainAnaly		Kefactor A	Alt+Shift+1 ▶	<pre>new ToolchainanalyzerFactoryImpl();</pre>
▲ 🚑 src dirix 888	2	Import		
Artef	4	Export		
Assur		References	+	st checks if AssumptionModel.getIsAssumption w
		Declarations	•	ed in @LLR 5.1.1.1 Method getIsAssumption
		Find Bugs	+	Lied Test Procedures are @see 4.1 CodeCover,
DOTE	Ŷ	Refresh	F5	2 JUnit, @see 4.4 Factory-Based Model Testing
D Explain		Assign Working Sets		diriy elotoech
D 🛺 TCLC		Use For Coverage Measurement		
⊳ La TCLO		Run As	×	1 CodeCover Measurement For JUnit
		Debug As	۱.	🕂 2 JUnit Plug-in Test Alt+Shift+X, P
metamo		Profile As	•	Ju 3 JUnit Test Alt+Shift+X, T
metamo		Team	۱.	Run Configurations
metamo		Compare With	+	<pre>nptionError2.setIsAssumption(false);</pre>
⊳ 🛺 metamo		Replace With	•	<pre>nptionError.setIsAssumption(true);</pre>
⊿ 🗁 testsrc		Restore from Local History		False (AssumptionModel.getIsAssumption (noAssump
🔺 🥀 metamo		Properties	Alt+Enter	True (AssumptionModel.getIsAssumption (isAssumpt:
JRE System Lil	brary	[JavaSE-1.6] 49	UseCas	<pre>se isAssumptionUseCase = fac.createUseCase();</pre>

Test Coverage



getAllAssumptions	0,0 %	0,0 %	0,0 %	0 ,0 %	-	-
😳 getIsAssumption	100,0 %	= 100,0 %	-	= 100,0 %	-	-
🙆 antIrDeactivated	0.0.9/	- 0 0 °/		00%		

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 %	-	-
G 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->...
- */

3

if

public static boolean getIsAssumption(Object item) {

- if (item instanceof Error) {
 - Error uce = (Error) item;

(item instanceof UseCase) {

UseCase uc = (UseCase) 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());
 - (item instanceof Qualification) {
- if 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());

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



How-To Qualify Eclipse-Based Tools

VALIDAS

Version 0.2 1 Document History 2 Definitions **3 Tool Qualification Process** 4 Traceability to DO-330 4.1 General Considerations 4.2 Tracing to Tool Qualification Planning Process Section 4.3 Tracing to Tool Development Life Cycle and Process Section 4.4 Tracing to Tool Verification Process Section 4.5 Tracing to Tool Configuration Management Process Section 4.6 Tracing to Tool Quality Assurance Process Section 4.7 Tracing to Tool Qualification Liaison Process Section 4.8 Tracing to Tool Qualification Data Section 4.9 Tracing to Additional Considerations for Tool Qualification Section 4.10 Tracing to Tool Qualification Objectives Section 5 References



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



Tool Development Plan for every Qualifiable Eclipse Plugin Version 0.6

Version 0.6
1 Document History
2 Definitions
A 3 Introduction
3.1 Document Extension
3.2 Multi-Function Tools
3.3 Usage of Qualified Plugins
3.4 COTS Tools
4 4 Standards
4.1 Project and General Information
4.2 Tool Qualification Planning
4.2.1 Overview
4.2.2 The Tool Analysis Model
4.2.3 Determination of the Confidence Level
4.2.4 Determination of the Tool Qualification Level
4.3 Tool Requirements
4.3.1 Tool Operational Requirements
4.3.2 Tool Requirements
4.4 Tool Design
▷ 4.4.1 Architecture
4.4.2 Low Level Requirements
▷ 4.4.3 Implementation
4.5 Tool Code Standards
4 5 Tool Life Cycle
4 5.1 Life Cycle Processes
5.1.1 Tool Planning Process
5.1.2 Tool Development Process
5.1.3 Tool Integration Process
5.1.4 Tool Configuration Management Process
5.1.5 Tool Quality Assurance Process
5.1.6 Tool Operational Verification Process
5.2 Life Cycle Stages
5.3 Transition Criteria
5.4 Life Cycle Verification
6 Tool Development Environment
7 References

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
 - ...

🗟 Build Configuration	0 🎄 ≉ 🕐
Custom Build	
Runtime Information	
Define the libraries, specify the order in which they sho compiled into each selected library.	ould be built, and list the source folders that should be
Add Library Up Down	Add Folder
Binary Build	Source Build
Select the folders and files to include in the binary build.	Select the folders and files to include in the source build.
 .classpath .project .settings .settings .et as spath .settings 	 classpath project settings settings META-INF bin bin build.properties model plugin.properties plugin.xml src

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):
 - o At least one TORFunction defined. All TORFunction elements have
 - nonempty ID
 - nonempty Description
 - ControlStatus=Reviewed
 - At least one TORContext defined. All TORContext (
 - nonempty ID
 - nonempty Description
 - ControlStatus=Reviewed
 - o 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
 - o 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*.

Validas AG

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







Generated PSAC Example from TCA



Generate Tool Classification Report

Generate Word (docx)

Tool Classification Report

Error

Discovered Via

required

4 1.4 Tool Chain Analyzer

- 1.4.1.1 Use Case Determinate Tool Confidence Level
- 1.4.1.2 Use Case Generate Tool Classification Report
- 4 1.4.2 Features of Tool Chain Analyzer
 - 1.4.2.1 Feature Build Model
 - 1.4.2.2 Feature Compute Tool Confidence Level
 - 1.4.2.3 Feature Excel Interface
 - 1.4.2.4 Feature Generate DOT
 - 1.4.2.5 Feature Generate Word (docx)
 - 1.4.2.6 Feature Model Validation
 - 1.4.2.7 Feature Xml Interface
- 1.4.3 Potential Errors in Tool Chain Analyzer
- 1.4.4 Restrictions in Tool Chain Analyzer
- 1.4.5 Checks in Tool Chain Analyzer
- 1.4.6 Assumptions
- 4 1.4.7 TCL Determination
 - 1.4.7.1 TCL Determination for Use C
 - 1.4.7.2 TCL Determination for Use C

1.4 Tool Chain Analyzer

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

'ool: 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 assumption

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



Error: Document Generated Wrongly

Document does not fit to the model.

Discovered by the following checks:

"No File Created" from "File Generator"

"Semantic Error" from "File Generator"

"Syntax Error" from "File Generator"

Confirmation Review Of TCLs Detect Wrong TCL

in Generate Word (docx) in Generate Tool Classification Report

Description:

From feature:

Subsumes:

Occurrences:

Error View:

Generate Word (docx)

Table 49 Error: Document Generated Wrongly

- 464 - 14

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

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
 - <u>http://www.eclipse.org/egit/</u>
 - <u>http://progit.org/book/</u>
 - <u>http://www.slideshare.net/stefanlay/eclipse-git-und-gerr</u>

type filter text		Configuration	⇔ • ⇒ • •
General	_		
CDO) Add	a configuration entry	<u>O</u> pen
Ecore Tools Dia EMF Compare Help Install/Update	Add a Please	configuration entry enter a key, e.g. "user.name" and a value	
Java	<u>K</u> ey	ıser.email	
Model Validatio Plug-in Develop	<u>V</u> alue	harutuni@in.tum.de	
Run/Debug			
Tasks Team			
CVS			
Git Configu		OK Cancel	
History Label Dec	orations	New Entry	
Window (Ignored Reso	Cache urces	Value: No configuration entry selected	lete <u>A</u> dd
Usage Data Colle	ctor	Restore <u>D</u> efaults	Apply

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

		ool Development Plan
_	21	Definitions
⊳	31	ntroduction
4	4 9	Standards
	6	4.1 Project and General Information
		4.2 Tool Qualification Planning
		4.3 Tool Requirements
		4.4 Tool Design
		4.5 Tool Code Standards
	4	4.6 Configuration Management Plan
		4.6.1 Configuration Identification
·i ·	F	4.6.2 Baselines
	L	4.6.3 Traceability
		4.6.4 Problem Reporting
		4.6.5 Change Control: Integrity and Identification
		4.6.6 Change Control: Tracking
		4.6.7 Change Review
		4.6.8 Configuration Status Accounting
		4.6.9 Retrieval
		4.6.10 Data Retention
4	51	Tool Life Cycle
	⊳	5.1 Life Cycle Processes
		5.2 Life Cycle Stages
	Þ	5.3 Transition Criteria
		5.4 Life Cycle Verification
⊿	61	ool Development Environment
		6.1 Eclipse Development
		6.2 Eclipse Integration
	4	6.3 Configuration Management (Draft)
		6.3.1 Branches
		6.3.2 Annotated Tags
		6.3.3 Change History
		6.3.4 GitWeb
		6.3.5 Change Control
		6.3.6 Gerrit Review Support
		6.4 Quality Assurance
	7 F	References

Configuration Management



Control

Category by TQL

- 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

I	I			-		-									
			Tool	Оре	erati	ona	I Re	qui	rements Process		1	2	3	4	5
2	Tool Operational Requirements are defined.	<u>5.1.1.a</u>	5.1.2.a 5.1.2.b 5.1.2.c	0	0	0	0	0	Tool Operational Requirements	<u>10.3.1</u>	1	1	1	1	2

Definition of Control Categories (DO-330):

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

Reference	CC1	CC2
7.2.1	•	•
<u>7.2.2.a</u>	•	
<u>7.2.2.b</u>		
<u>7.2.2.c</u>		
<u>7.2.2.d</u>		
<u>7.2.2.e</u>		
<u>7.2.2.f</u>	•	•
<u>7.2.2.g</u>		
7.2.5	•	
	Reference 7.2.1 7.2.2.a 7.2.2.b 7.2.2.c 7.2.2.d 7.2.2.e 7.2.2.f 7.2.2.g 7.2.5	Reference CC1 7.2.1 • 7.2.2.a • 7.2.2.b • 7.2.2.c • 7.2.2.d • 7.2.2.e • 7.2.2.f • 7.2.2.g •

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)



CM: Control Status of Tests (Proposed)





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



- 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

Tool Life (Cycle Processes
	Tool Qualification Planning Process - Section 4
	Tool Development Processes - Section 5
Integ	ral 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

Thank You!







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