Quality Assurance in Model-Driven Software Engineering for Spacecraft

Kilian Hoeflinger, Jan Sommer, Ayush Nepal, Olaf Maibaum, Daniel Lüdtke

DLR - Simulation and Software Technology

Contact: kilian.hoeflinger@dlr.de
PaTaS - Product Assurance with TASTE Study

Motivation
• Improve S/W PA for model-driven development by measuring model quality with model metrics
• Early evaluation/detection of:
  • Flaws in specification
  • Functional requirements
  • Non-functional requirements (Maintainability, Reusability etc.)

Outline of the PATAS study
• One year study
• Development of product quality model with software and model metrics
• Implementation of an end-to-end model-driven software engineering lifecycle demonstrator, based on TASTE
• Evaluation of the demonstrator with mission-critical parts of the onboard S/W of a satellite mission, being modelled and subsequently coded
• Improvement of model-driven S/W PA at ESA
Content

Quality Model

Model Metrics

Demonstrator design and implementation

Conclusions

Next Stop: Model Metricator Tool
Developed Quality Model

- Quality Model is based on existing one of ECSS-Q-HB-80C
- Splitting the product sub-characteristic in a model and software metric
- Graphical and table format representations

Quality model format for recommendation for ECSS-Q-HB-80C
Mapping Formula within the Quality Model

- **Mapping formulae for model to S/W metrics**
  - **Complementary** – Combination of model and S/W metric to derive a quality verdict
  - **Independent** – Model and S/W metric are alone standing
  - Further formulae possible
  - **Nested** - A software metric is nested in a model metric, determining and subsequent handling of special points of interest
# Model Metrics Overview

<table>
<thead>
<tr>
<th>ID</th>
<th>Model Metric Name</th>
<th>Applicable Sub-characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM-01</td>
<td>Adherence to Modelling Conventions</td>
<td>Modularity, Completeness, Self-descriptiveness, Conciseness, Balance, Correctness</td>
</tr>
<tr>
<td>MM-02</td>
<td>Interaction Diagram Coverage</td>
<td>Completeness, Balance</td>
</tr>
<tr>
<td>MM-03</td>
<td>Model Type Instance Weight</td>
<td>Complexity, Balance</td>
</tr>
<tr>
<td>MM-04</td>
<td>Model Coupling</td>
<td>Modularity, Complexity, Balance</td>
</tr>
<tr>
<td>MM-05</td>
<td>Model Type Instances per Use Case</td>
<td>Modularity, Complexity, Balance, Conciseness</td>
</tr>
<tr>
<td>MM-06</td>
<td>Use Cases per Model Type Instance</td>
<td>Modularity, Complexity, Balance, Conciseness</td>
</tr>
<tr>
<td>MM-07</td>
<td>Lines of model code</td>
<td>Complexity, Balance, Self-descriptiveness</td>
</tr>
<tr>
<td>MM-08</td>
<td>Model comment frequency</td>
<td>Complexity, Balance, Self-descriptiveness</td>
</tr>
<tr>
<td>MM-09</td>
<td>Module Fan-in / Fan-out</td>
<td>Modularity, Balance</td>
</tr>
<tr>
<td>MM-10</td>
<td>Requirements Specification Coverage</td>
<td>Completeness, Correctness</td>
</tr>
</tbody>
</table>

*PaTaS model metrics overview*
Model metrics assessment results (1/3)

Model Type Instance Weight
Accumulation of all model type instances, “owned” by a model type instance, considering a model type specific weight factor, determined by any indicator of complexity.

Results
- Large data interfaces are visible, represents good a-priori evaluation possibility for complexity.
- Interface changes are rare and on the highest level not visible.
- Shows creation of service 152 of ONS to ralex service 8 of ONS.

Small TASTE IV example function with correlating ASN.1 interface parameters

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>MTIW value of Function_1</th>
<th>Specific model element</th>
<th>Weight-factor $\omega_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface1</td>
<td>2+1 = 3</td>
<td>Sequence/Choice (ASN.1)</td>
<td>2</td>
</tr>
<tr>
<td>Interface2</td>
<td>2+(2+1+1)+(2+1) = 9</td>
<td>Simple Datatype (ASN.1)</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MTIW result

Applied weight–factor and formula
Model metrics assessment results (2/3)

Model Type Instances per Use Case (MTIpUC)

Amount of model type instances per use case has to be counted. Here, a use case is the implementation of a test for a software requirement.

Results

- Removal of range between min and max shows homogenisation of models
- High values indicate low functional cohesion in system
- Range caused by requirements, when they are to coarse grained defined

Small TASTE IV example system

TASTE MSC use_case1

TASTE MSC use_case2

<table>
<thead>
<tr>
<th>Use Case</th>
<th>MTIpUC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_case1</td>
<td>2</td>
</tr>
<tr>
<td>use_case2</td>
<td>2</td>
</tr>
</tbody>
</table>

MTIpUC referring to TASTE IV functions
Model metrics assessment results (3/3)

Model Comment Frequency
Ratio between number of model comment lines and lines of model code plus number of model comment lines

Results
• Difficult to comment models, when they are very self-explaining, like ASN.1.
• The jitter between the maximum and the minimum is rather big and not closing throughout the lifecycle, which is due to different model views and their technology.
• But all files are above 20%, and the average is almost at 30%.

Lines of model code
Counting the number of model lines per model file (excluding comments and blank lines)

Results
• Result depends on modelling language, ASN.1 requires more lines of code than most custom domain specific languages.
• Transfer of this metric to a graphical model requires re-definition of ‘lines’, e.g. to specific model components.
• Forces the developer to think about a good and logical distribution of a model over multiple files.
• Shows that min to max gap closes over time, increasing balance.
Model-Based Software Development Lifecycle following V-Model

Models

Auto Code

Manual Code
MBSD Lifecycle Demonstrator Design

**Workflow**
1. Define computation independent PUS, communication data and communication test model
2. Refine platform independent model in TASTE Interface View
3. Generate code skeletons from TASTE Deployment View
4. Test-driven implementation of OBSW

**Applied standards and methodologies**
- ECSS PUS, OMG Model-driven Architecture standard, Model-based testing taxonomy, TASTE inherent standards

**Use case**
- Parts of ACS, ONS and CDH of an actual small satellite mission of DLR
- Targeting lab quality (x86), no flight H/W
Traceability of artefacts: Document to Model to Code

- Bidirectional traceability allows reversal of working direction
- Automatic traceability update prevents a loss of the trace

<table>
<thead>
<tr>
<th>Artefact</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case</td>
<td>90 TM/TC messages</td>
</tr>
<tr>
<td>Model size</td>
<td>19,340 lines with PAL: 126 lines</td>
</tr>
<tr>
<td></td>
<td>DTVL: 401 lines</td>
</tr>
<tr>
<td></td>
<td>TASTE IV: 5593 lines (only AADL)</td>
</tr>
<tr>
<td></td>
<td>TASTE DV: 188 lines (only AADL)</td>
</tr>
<tr>
<td></td>
<td>ASN1: 13,032 lines</td>
</tr>
<tr>
<td>Unit-test size</td>
<td>5,928 lines</td>
</tr>
<tr>
<td>Integration Tests</td>
<td>19,723 lines</td>
</tr>
<tr>
<td>OBSW (user mode)</td>
<td>3,334 lines</td>
</tr>
<tr>
<td>OBSW (TASTE mode)</td>
<td>370,887 lines (with PrintTypes.c: 105,925; and PrintTypesAsASN1.c 215,161)</td>
</tr>
</tbody>
</table>
Applications contain services

Services contain telemetry and tele-command subservices

Subservices are linked to ASN.1 messages
ASN.1 editor

All frontend editors
- offer auto completion
- Syntax highlighting
- Syntax validation

ASN1. editor
- Type definition
- Value assignment
- Transforms ASN.1 to Ecore model
- Easy integratable with custom code generator
- Or existing tools to translate Ecore model to X
Data Testing and Verification Language (DTVL) editor

- Allows the description of use cases as black boxes tests
- Exploits the TM/TC interface of satellites
- Enables referencing TM or TC message instances
- Based on Linear Temporal Logic
- Enriched to describe periodic message events
- Could be used to describe the up and downlink of entire mission phases
TASTE Interface and Deployment View

PaTaS use case in TASTE Interface View

PaTaS use case in TASTE Deployment View
Automatic model metric collection

Model Coupling

Metric Description

The goal is to improve the maintainability and the balance of the product by determining the coupling of model type instances among each other. Coupling of a model type instance is determined by counting all coupled model type instances.

Excessive coupling is detrimental to modular design and prevents reuse. Strong dependencies among model type instances can lead to side effects when software gets changed, preventing an effective maintenance.

Timestamp: Thu Oct 26 14:36:10 CEST 2017

<table>
<thead>
<tr>
<th>Module Type Instance Weight metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAN_IN.html</td>
</tr>
<tr>
<td>FAN_OUT.html</td>
</tr>
<tr>
<td>FUNCTIONAL_COHESION.html</td>
</tr>
<tr>
<td>INTERACTION_DIAGRAM_COVERAGE.html</td>
</tr>
<tr>
<td>MODEL_COUPLING.html</td>
</tr>
<tr>
<td>MODEL_TYPE_INSTANCES_PER_USE_CASE.html</td>
</tr>
<tr>
<td>MODEL_WEIGHT.html</td>
</tr>
<tr>
<td>USE_CASES_PER_MODEL_TYPE_INSTANCE.html</td>
</tr>
<tr>
<td>src-gen</td>
</tr>
</tbody>
</table>

Model Coupling metric as example
Recommendations for ECSS

**ECSS-Q-80 (ST+HB)**
- Minor adaptions in various clauses
- Reference model-based software quality model
- 10 Model metrics
- Tailoring recommendations for the model metrciation programme
- Model metrics applicability and thresholds based on criticality
- 3 new sub-characteristics

**ECSS-E-40 (ST+HB)**
- Minor adaptions in various clauses
- Model-based development life cycle considering various development methodologies
- Model Driven Architecture elaboration as standard background
- Differentiation of Modelling standard and Modelling guideline
Model Metric Thresholds

• Finding optimal thresholds for model metrics takes further evaluation/usage

• Thresholds are difficult to determine, as they depend on the used underlying software standard (here: PUS) and the used modelling languages/tools. Model metrics have to be tailored under consideration of the used standards and modelling methods/tools.

• Recommendation: Keep the range in the model metric results as small as possible so that it is well balanced.

• Recommendation: Average values might be a good starting point.

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Proposed target value/ criticality category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Adherence to Modelling Conventions</td>
<td>$1 \leq x \leq 15$</td>
</tr>
<tr>
<td>Interaction Diagram Coverage</td>
<td>$1 \leq x \leq 20$</td>
</tr>
<tr>
<td>Model Type Instance Weight</td>
<td>$x \leq 50$</td>
</tr>
<tr>
<td>Model Coupling</td>
<td>$x \leq 5$</td>
</tr>
<tr>
<td>Model Type Instances per Use Case</td>
<td>$x \leq 3$</td>
</tr>
<tr>
<td>Use Cases per Model Type Instance</td>
<td>$1 \leq x \leq 10$</td>
</tr>
<tr>
<td>Fan-IN/OUT</td>
<td>$x \leq 4$</td>
</tr>
<tr>
<td>Model Comment Frequency</td>
<td>30 %</td>
</tr>
<tr>
<td>Lines of Model Code</td>
<td>$&lt; 300$</td>
</tr>
</tbody>
</table>

Current metric threshold values
Qualitative conclusion: Evaluation Order Matters

- Next to the classification based on their evaluable characteristics, model metrics can be grouped regarding their analytical capability.

- **Analytic capabilities of model metrics:**
  - **Conformance scanning**
    - forces developers to create overview and standard conformance within their models.
  - **Model Comment Frequency, Adherence to Modelling Conventions, Lines of Model Code**
  - **Structural scanning**
    - give detailed insight on the structural design and data flow within the product.
    - **Model Coupling, Model Type Instance Weight, Module Fan-in/out**
  - **Behavioural scanning**
    - related to structural scanning, but targets mainly on the functional requirement and the specification.
    - **Interaction Diagram Coverage, Model Type Instances per Use Case, Use Cases per Model Type Instance**
Further Qualitative Conclusiones

• **Balance** is major driver in the *modelling phases*

• **Complexity** is major driver in the *coding phases*

• **Single-view model metrics are not meaningful** when conducting model-driven development, as the source code can also be evaluated with existing tools

• **Quality** is added mainly in the *modelling phases*, but has to be *maintained in the coding phases*

• **Model metrics also allow an assessment of the software requirements**, as they determine their extent over the system and their granularity

• **It is visible how good the testing regarding fault tolerance is.** There could be even a factor between fault tolerance and expected behaviour test cases
Next stop: Model Metricator Tool

- Work in progress
- Small adaptable tool to evaluate the quality of models
- Adaptable to all technologies
- We search partners, being model owners, who want to have a tool to evaluate their model quality (for free)
- And we search collaborators
- Contact: kilian.hoeflinger@dlr.de