PaTaS

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

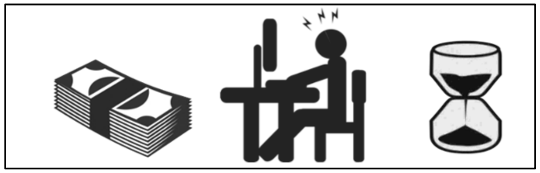
Motivation

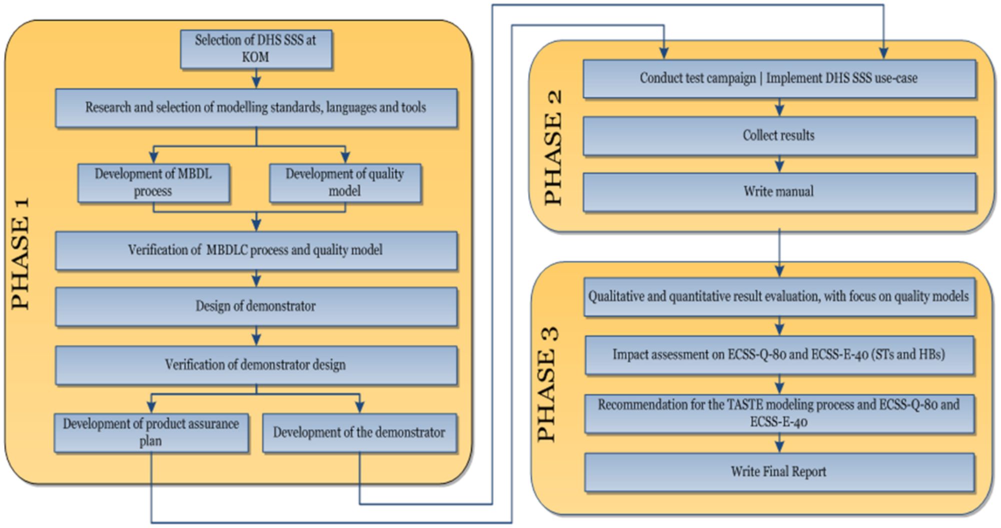
- Improve S/W PA for model-driven development by measuring model quality with model metrics
- Early evaluation/detection of:
  - Flaws in specification
  - Functional requirements
  - Non-functional requirements (Maintainability, Reusability etc.)

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

You save…





PHASE 1

Selection of DHS SSS at KOM

Research and selection of modelling standards, languages and tools

Development of MBDL process

Development of quality model

Verification of MBDLC process and quality model

Design of demonstrator

Verification of demonstrator design

Development of product assurance plan

Development of the demonstrator

PHASE 2

Conduct test campaign | Implement DHS SSS use-case

Collect results

Write manual

PHASE 3

Qualitative and quantitative result evaluation, with focus on quality models

Impact assessment on ECSS-Q-80 and ECSS-E-40 (STs and HBs)

Recommendation for the TASTE modeling process and ECSS-Q-80 and ECSS-E-40

Write Final Report

Workflow of PATAS study

Credit for GIFs: openclipart.org

Content

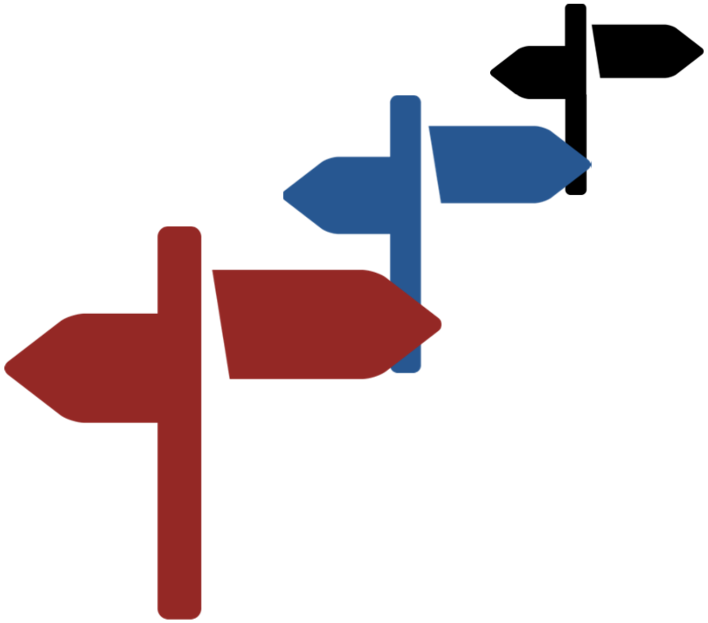
Quality Model

Model Metrics

Demonstrator design and implementation

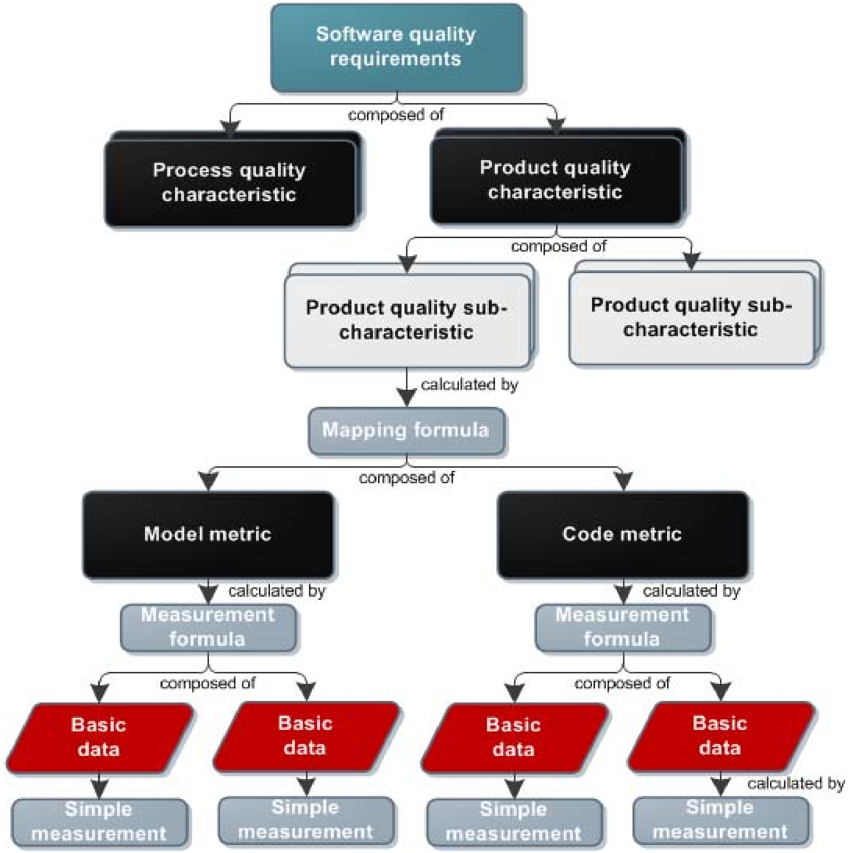
Conclusions

Next Stop: Model Metricator Tool



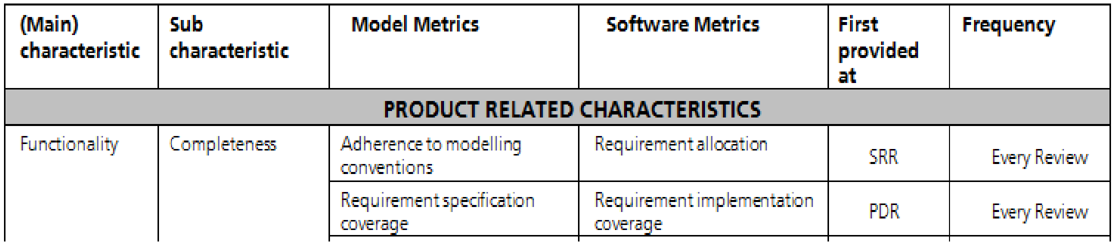
Credit for GIFs: openclipart.org

Developed Quality Model



Quality model for model-based software development

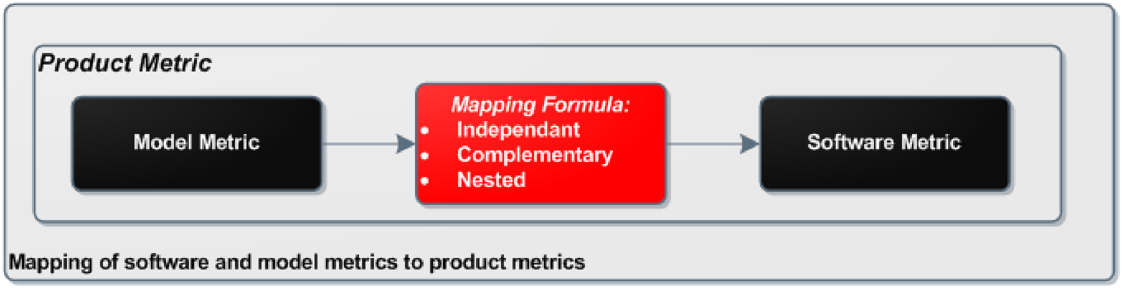
- Quality Model is based on existing one of ECSS-Q-HB-80C
- Splitting the product sub-characteristic in a model and software metric
- Graphical and table format representations



(Main) characteristic	Sub characteristic	Model Metrics	Software Metrics	First provided at	Frequency
PRODUCT RELATED CHARACTERISTICS					
Functionality	Completeness	Adherence to modelling conventions	Requirement allocation	SRR	Every Review
		Requirement specification coverage	Requirement implementation coverage	PDR	Every Review

Quality model format for recommendation for ECSS-Q-HB-80C

Mapping Formula within the Quality Model



Mapping of software and model metrics to product metrics

- **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

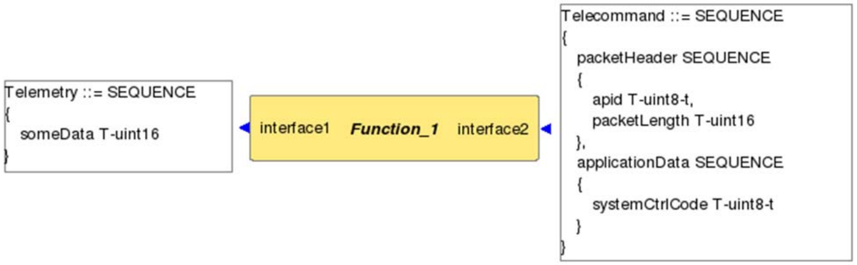
ID	Model Metric Name	Applicable Sub-characteristic
MM-01	Adherence to Modelling Conventions	Modularity, Completeness, Self-descriptiveness, Conciseness, Balance, Correctness
MM-02	Interaction Diagram Coverage	Completeness, Balance
MM-03	Model Type Instance Weight	Complexity, Balance
MM-04	Model Coupling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

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



```
Telemetry ::= SEQUENCE
{
  someData T-uint16
}
```

```
Telecommand ::= SEQUENCE
{
  packetHeader SEQUENCE
  {
    apid T-uint8-t,
    packetLength T-uint16
  },
  applicationData SEQUENCE
  {
    systemCtrlCode T-uint8-t
  }
}
```

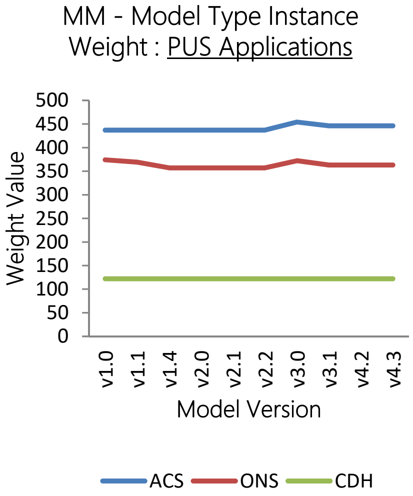
Small TASTE IV example function with correlating ASN.1 interface parameters

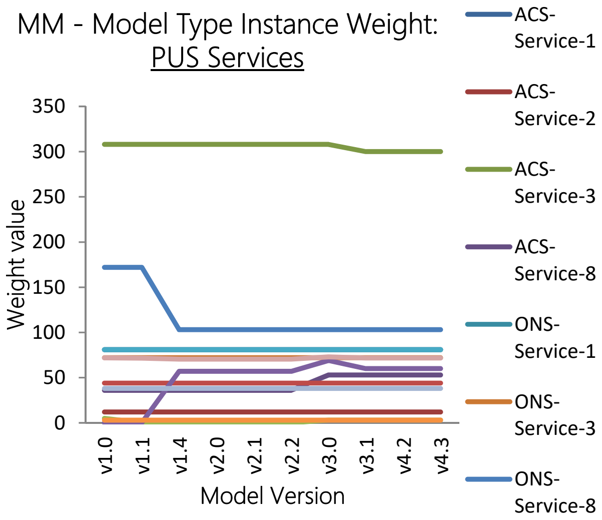
Interfaces	*MTIW* value of Function_1
Interface1	2+1 = 3
Interface2	2+(2+1+1)+(2+1) = 9
Total	12

MTIW result

Specific model element	Weight-factor ω_k
Sequence/Choice (ASN.1)	2
Simple Datatype (ASN.1)	1

Applied weight–factor and formula





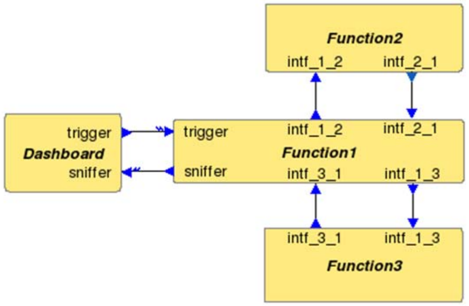
Results

- 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

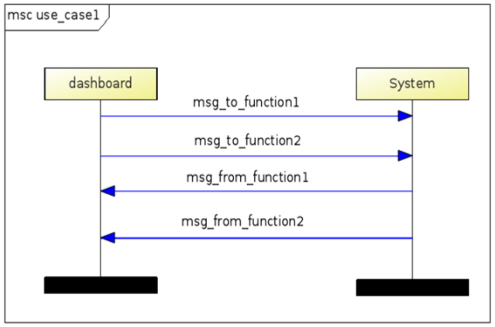
Model metrics assessment results (2/3)

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



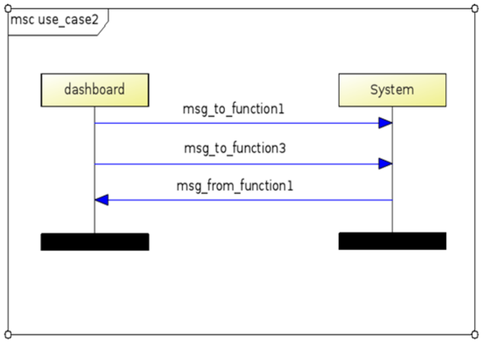
Small TASTE IV example system



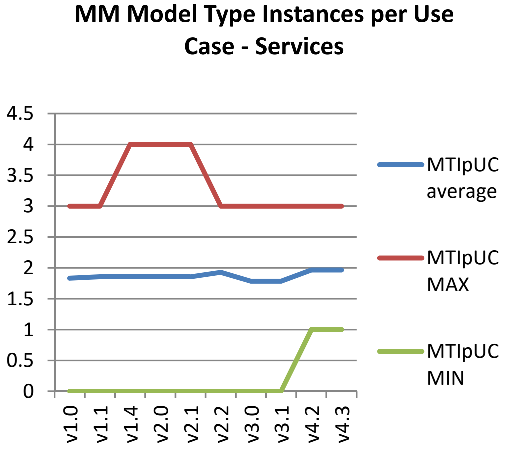
TASTE MSC use_case1

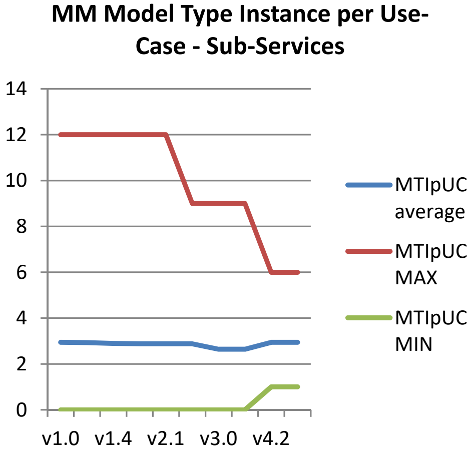
Use Case	MTIpUC Value
use_case1	2
use_case2	2

MTIpUC referring to TASTE IV functions



TASTE MSC use_case2


MM Model Type Instances per Use Case - Services


MM Model Type Instance per Use-Case - Sub-Services

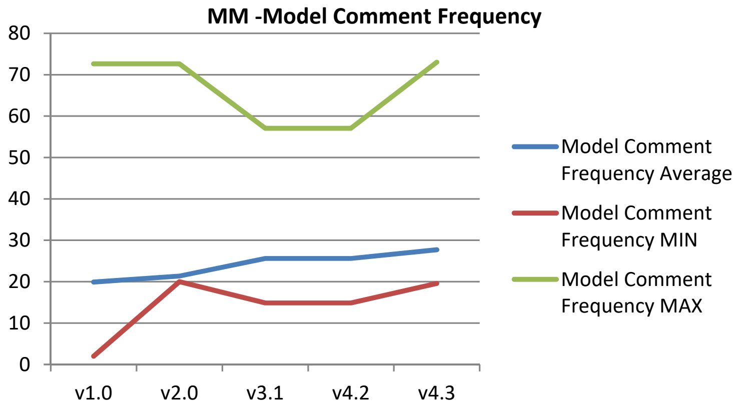
Results

- 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

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

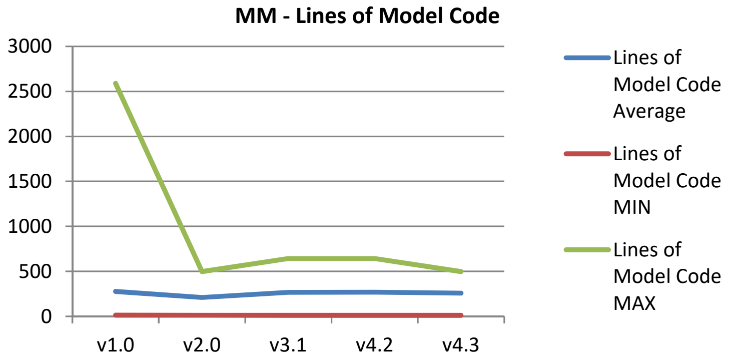


Results

- 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

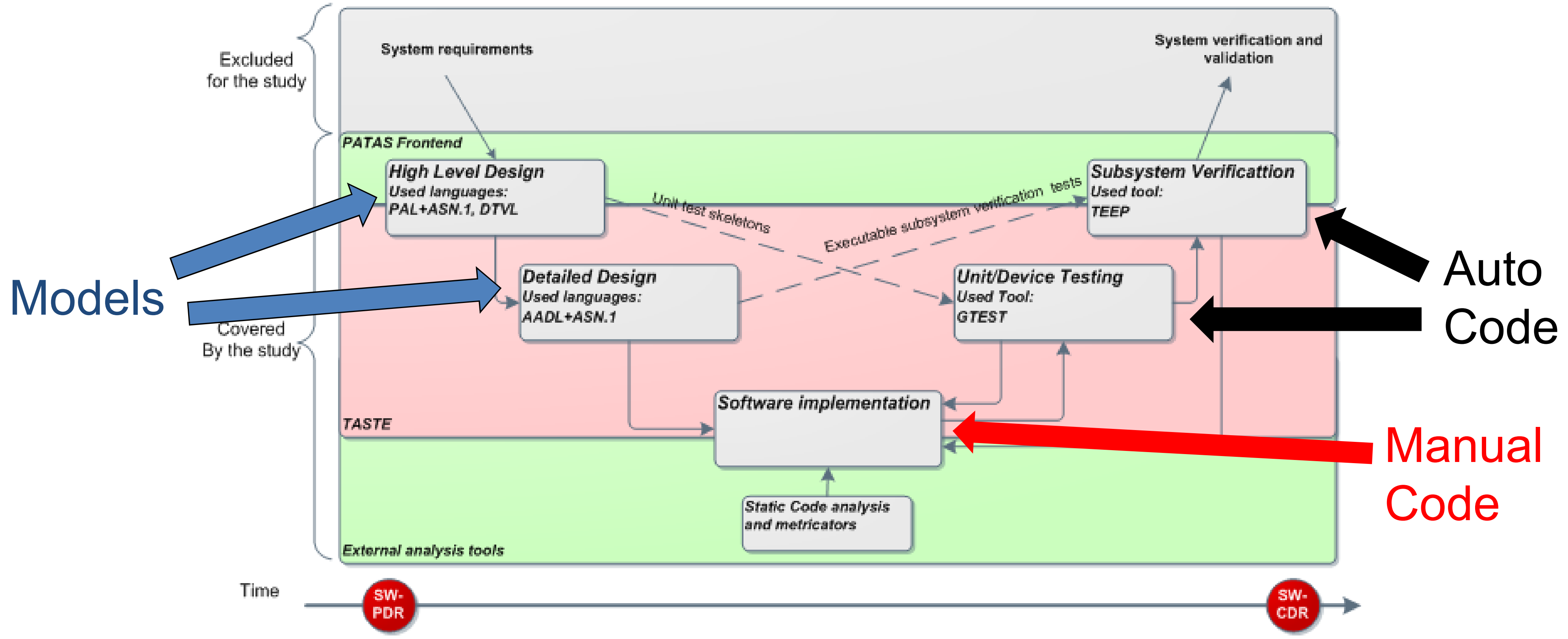
Counting the number of model lines per model file (excluding comments and blank lines)



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



MBSD Lifecycle Demonstrator Design

Workflow

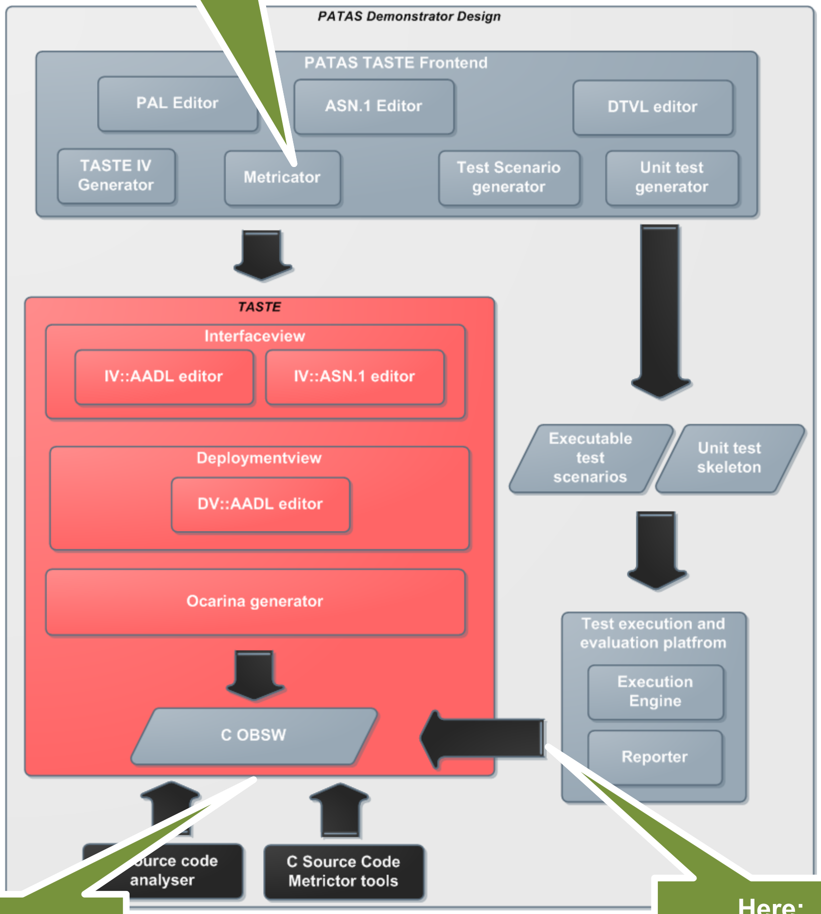
1. Define computation independent PUS, communication data and communication test model
2. Refine platform independent model in TASTE Interface View
3. Generate code skeletons from TASTE Deployment View
4. Test-driven implementation of OBSW

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



Here: Collecting Model Metrics

Here: Collecting S/W metrics

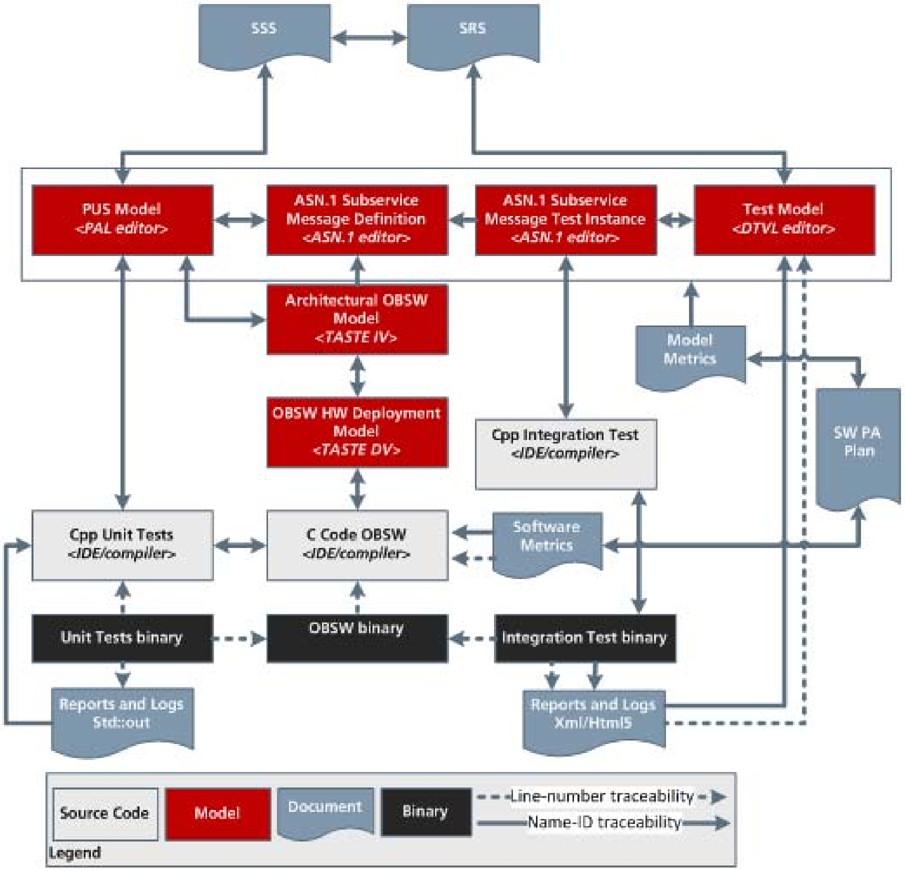
Here: Collecting functionality reports

PaTaS demonstrator design

Traceability of artefacts: Document to Model to Code

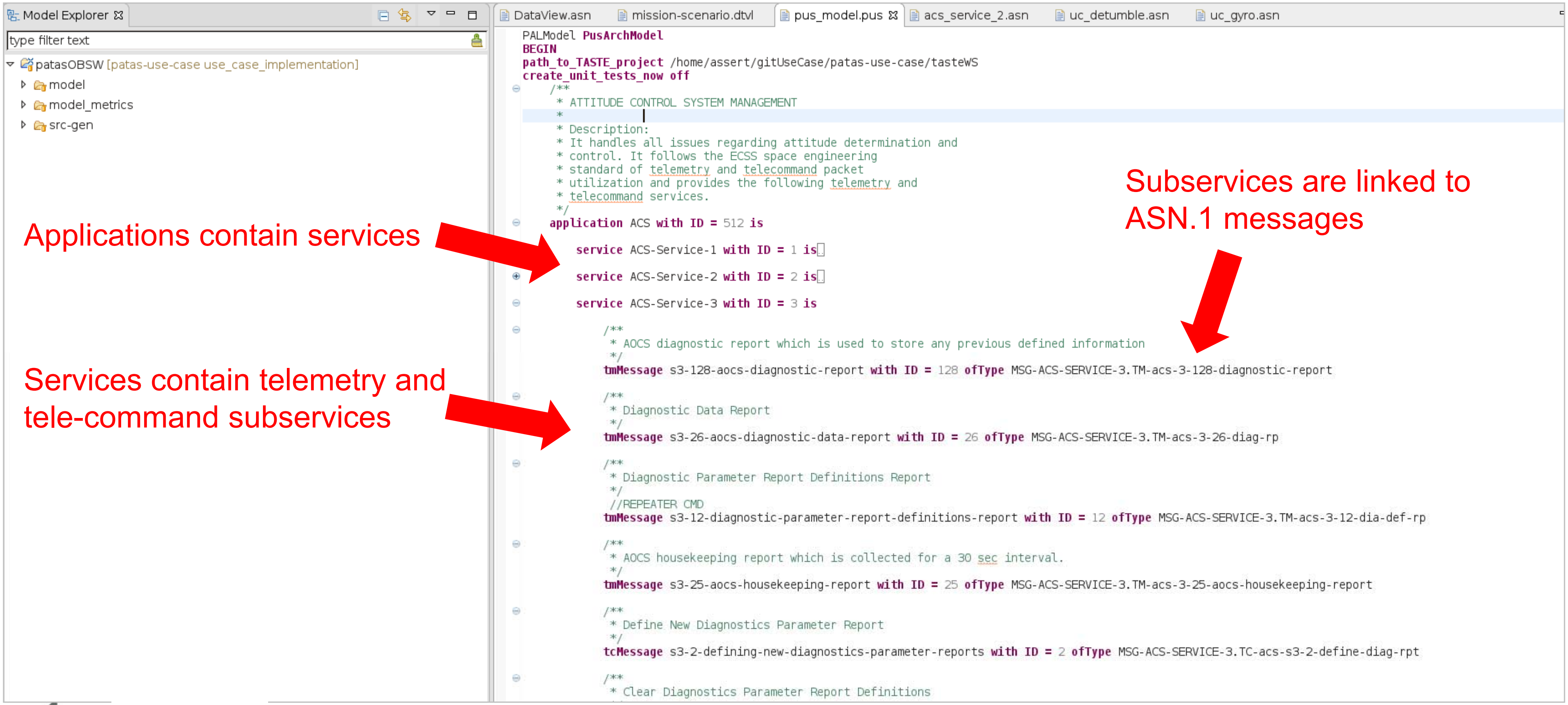
- Bidirectional traceability allows reversal of working direction
- Automatic traceability update prevents a loss of the trace

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



Taraceability of the artefacts of the demonstrator

PUS Architectural Language (PAL) editor



```
PALModel PusArchModel
BEGIN
path_to_TASTE_project /home/assert/gitUseCase/patas-use-case/tasteWS
create_unit_tests_now off
    /**
     * ATTITUDE CONTROL SYSTEM MANAGEMENT
     *
     * Description:
     * It handles all issues regarding attitude determination and
     * control. It follows the ECSS space engineering
     * standard of telemetry and telecommand packet
     * utilization and provides the following telemetry and
     * telecommand services.
     */
    application ACS with ID = 512 is

        service ACS-Service-1 with ID = 1 is

        service ACS-Service-2 with ID = 2 is

        service ACS-Service-3 with ID = 3 is

            /**
             * AOCS diagnostic report which is used to store any previous defined information
             */
            tmMessage s3-128-aocs-diagnostic-report with ID = 128 ofType MSG-ACS-SERVICE-3.TM-acs-3-128-diagnostic-report

            /**
             * Diagnostic Data Report
             */
            tmMessage s3-26-aocs-diagnostic-data-report with ID = 26 ofType MSG-ACS-SERVICE-3.TM-acs-3-26-diag-rp

            /**
             * Diagnostic Parameter Report Definitions Report
             */
             //REPEATER CMD
            tmMessage s3-12-diagnostic-parameter-report-definitions-report with ID = 12 ofType MSG-ACS-SERVICE-3.TM-acs-3-12-dia-def-rp

            /**
             * AOCS housekeeping report which is collected for a 30 sec interval.
             */
            tmMessage s3-25-aocs-housekeeping-report with ID = 25 ofType MSG-ACS-SERVICE-3.TM-acs-3-25-aocs-housekeeping-report

            /**
             * Define New Diagnostics Parameter Report
             */
            tcMessage s3-2-defining-new-diagnostics-parameter-reports with ID = 2 ofType MSG-ACS-SERVICE-3.TC-acs-s3-2-define-diag-rpt

            /**
             * Clear Diagnostics Parameter Report Definitions
```

Applications contain services

Services contain telemetry and tele-command subservices

Subservices are linked to ASN.1 messages

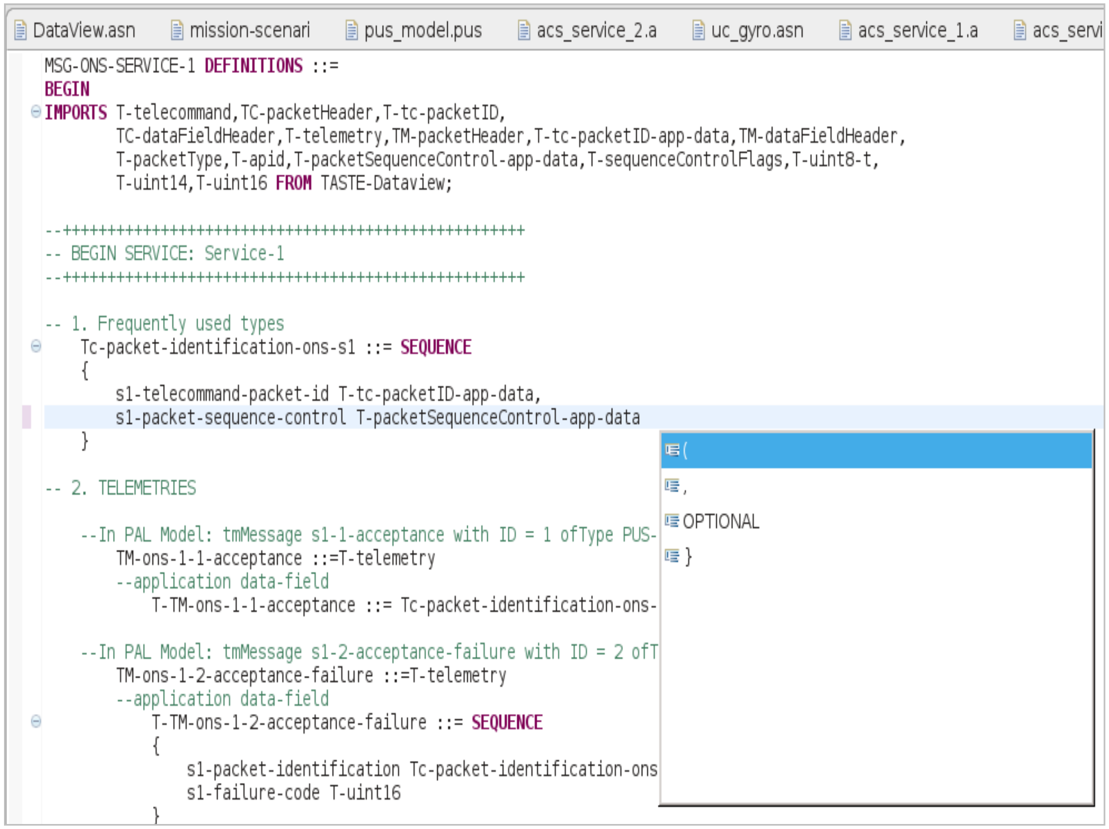
ASN.1 editor

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

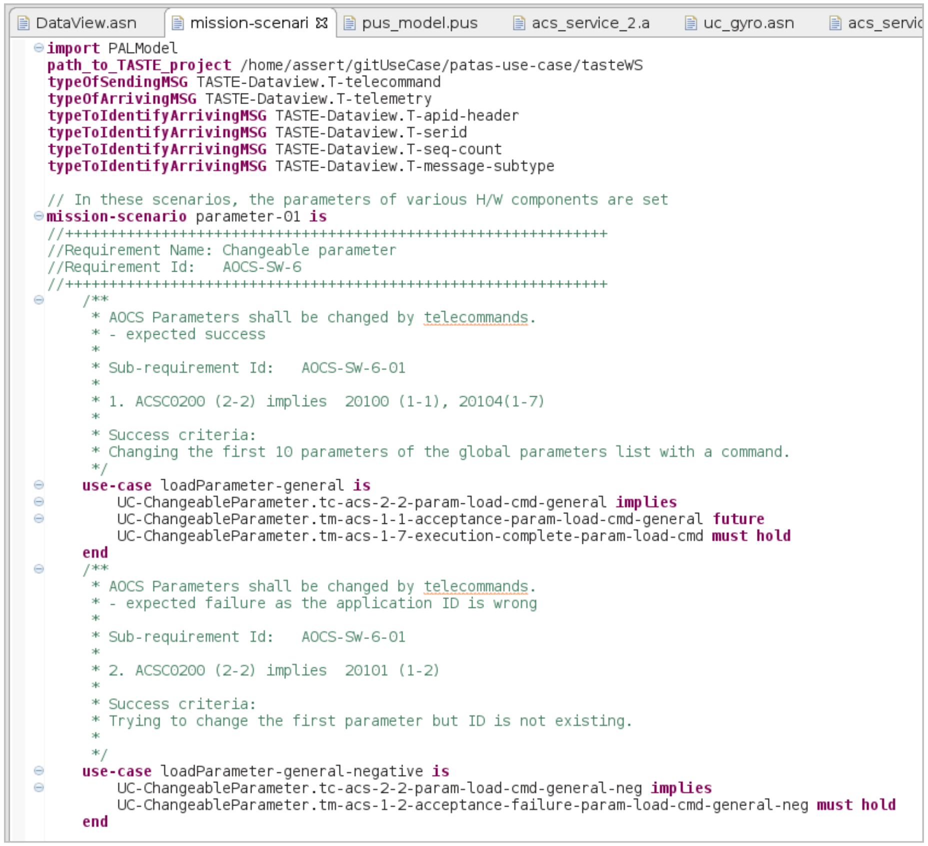


```
MSG-ONS-SERVICE-1 DEFINITIONS ::=
BEGIN
IMPORTS T-telecommand,TC-packetHeader,T-tc-packetID,
        TC-dataFieldHeader,T-telemetry,TM-packetHeader,T-tc-packetID-app-data,TM-dataFieldHeader,
        T-packetType,T-apid,T-packetSequenceControl-app-data,T-sequenceControlFlags,T-uint8-t,
        T-uint14,T-uint16 FROM TASTE-Dataview;

--+++++++++++++++++++++++++++++++++++++++++++++++++++++
-- BEGIN SERVICE: Service-1
--+++++++++++++++++++++++++++++++++++++++++++++++++++++

-- 1. Frequently used types
    Tc-packet-identification-ons-s1 ::= SEQUENCE
    {
        s1-telecommand-packet-id T-tc-packetID-app-data,
        s1-packet-sequence-control T-packetSequenceControl-app-data
    }

-- 2. TELEMETRIES

    --In PAL Model: tmMessage s1-1-acceptance with ID = 1 ofType PUS-
        TM-ons-1-1-acceptance ::=T-telemetry
        --application data-field
            T-TM-ons-1-1-acceptance ::= Tc-packet-identification-ons-

    --In PAL Model: tmMessage s1-2-acceptance-failure with ID = 2 ofT
        TM-ons-1-2-acceptance-failure ::=T-telemetry
        --application data-field
            T-TM-ons-1-2-acceptance-failure ::= SEQUENCE
            {
                s1-packet-identification Tc-packet-identification-ons
                s1-failure-code T-uint16
            }
```

ASN.1 editor

Data Testing and Verification Language (DTVL) editor

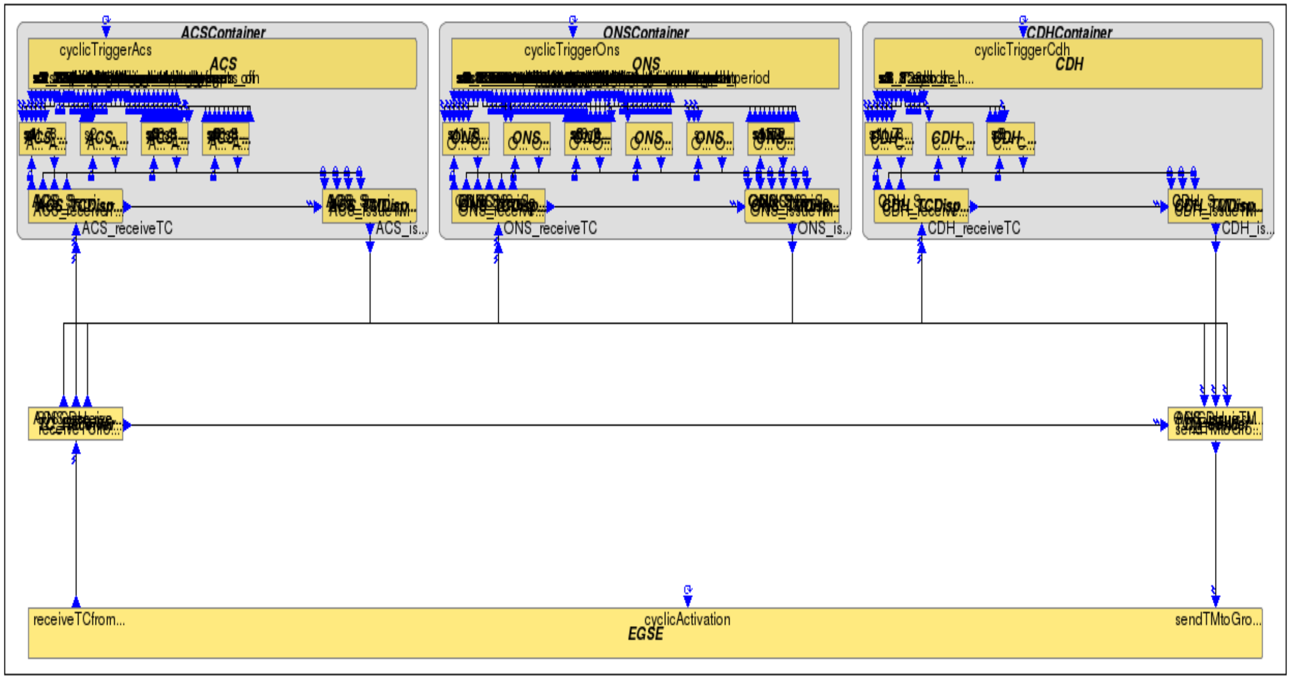
- Allows the description of use cases as black boxes tests
- Exploits the TM/TC interface of satellites
- Enables referencing TM or TC message instances
- Based on Linear Temporal Logic
- Enriched to describe periodic message events
- Could be used to describe the up and downlink of entire mission phases



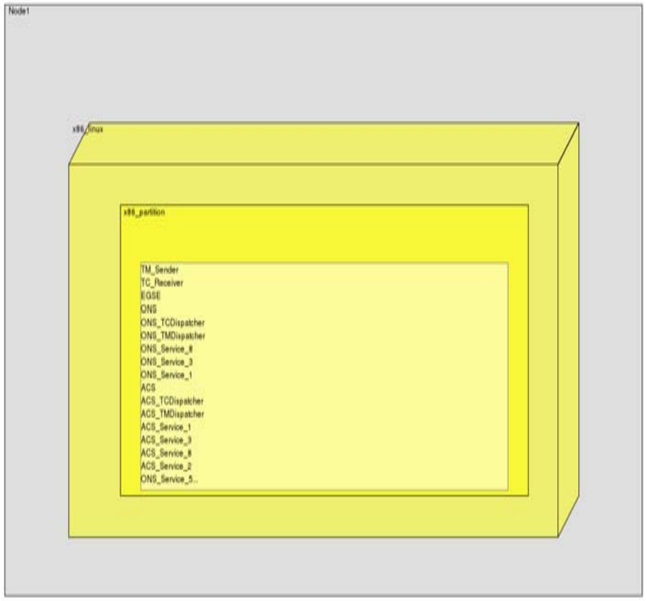
```
import PALModel
path_to_TASTE_project /home/assert/gitUseCase/patas-use-case/tasteWS
typeOfSendingMSG TASTE-Dataview.T-telecommand
typeOfArrivingMSG TASTE-Dataview.T-telemetry
typeToIdentifyArrivingMSG TASTE-Dataview.T-apid-header
typeToIdentifyArrivingMSG TASTE-Dataview.T-serid
typeToIdentifyArrivingMSG TASTE-Dataview.T-seq-count
typeToIdentifyArrivingMSG TASTE-Dataview.T-message-subtype

// In these scenarios, the parameters of various H/W components are set
mission-scenario parameter-01 is
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Requirement Name: Changeable parameter
//Requirement Id:   AOCS-SW-6
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    /**
     * AOCS Parameters shall be changed by telecommands.
     * - expected success
     *
     * Sub-requirement Id:   AOCS-SW-6-01
     *
     * 1. ACSC0200 (2-2) implies  20100 (1-1), 20104(1-7)
     *
     * Success criteria:
     * Changing the first 10 parameters of the global parameters list with a command.
     */
    use-case loadParameter-general is
        UC-ChangeableParameter.tc-acs-2-2-param-load-cmd-general implies
        UC-ChangeableParameter.tm-acs-1-1-acceptance-param-load-cmd-general future
        UC-ChangeableParameter.tm-acs-1-7-execution-complete-param-load-cmd must hold
    end
    /**
     * AOCS Parameters shall be changed by telecommands.
     * - expected failure as the application ID is wrong
     *
     * Sub-requirement Id:   AOCS-SW-6-01
     *
     * 2. ACSC0200 (2-2) implies  20101 (1-2)
     *
     * Success criteria:
     * Trying to change the first parameter but ID is not existing.
     *
     */
    use-case loadParameter-general-negative is
        UC-ChangeableParameter.tc-acs-2-2-param-load-cmd-general-neg implies
        UC-ChangeableParameter.tm-acs-1-2-acceptance-failure-param-load-cmd-general-neg must hold
    end
```

Data Testing and Verification Language editor

TASTE Interface and Deployment View



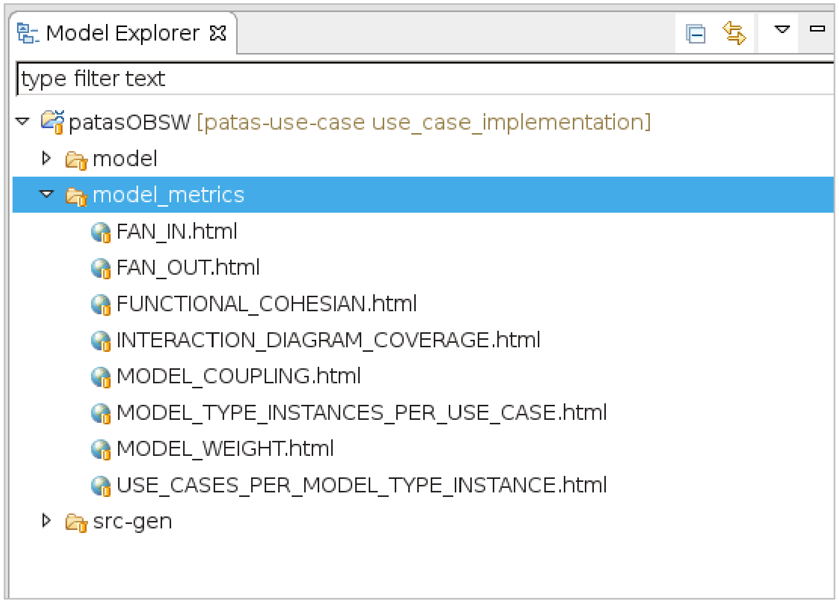
PaTaS use case in TASTE Interface View



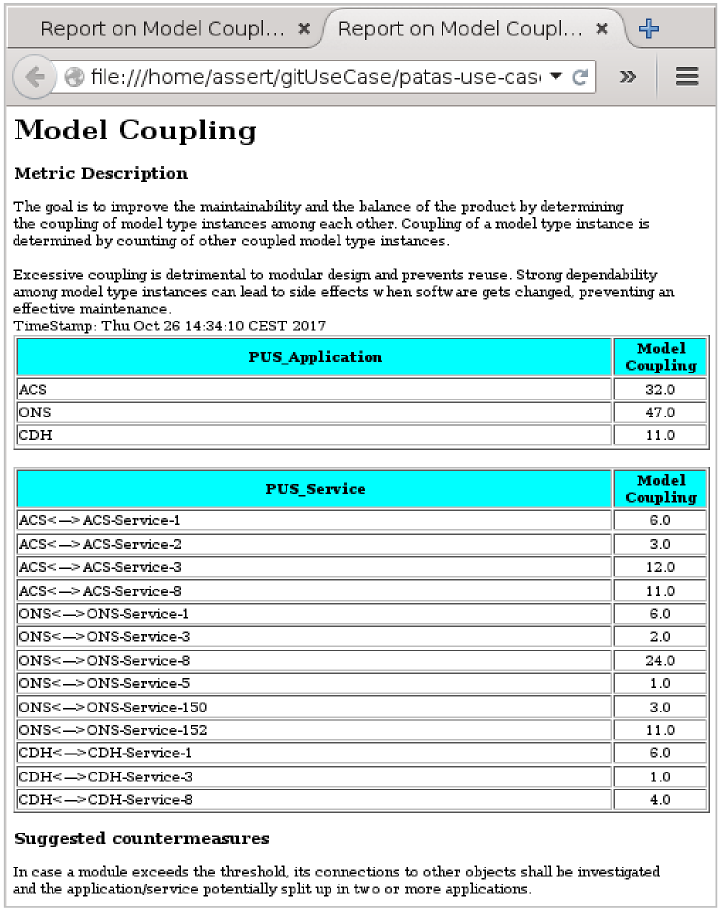
TM_Sender
TC_Receiver
EGSE
ONS
ONS_TCDispatcher
ONS_TMDispatcher
ONS_Service_8
ONS_Service_3
ONS_Service_1
ACS
ACS_TCDispatcher
ACS_TMDispatcher
ACS_Service_1
ACS_Service_3
ACS_Service_8
ACS_Service_2
ONS_Service_5...

PaTaS use case in TASTE Deployment View

Automatic model metric collection


Model Explorer
type filter text
patasOBSW [patas-use-case use_case_implementation]
model
model_metrics
FAN_IN.html
FAN_OUT.html
FUNCTIONAL_COHESIAN.html
INTERACTION_DIAGRAM_COVERAGE.html
MODEL_COUPLING.html
MODEL_TYPE_INSTANCES_PER_USE_CASE.html
MODEL_WEIGHT.html
USE_CASES_PER_MODEL_TYPE_INSTANCE.html
src-gen

Module Type Instance Weight metric



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 of other coupled model type instances.

Excessive coupling is detrimental to modular design and prevents reuse. Strong dependability among model type instances can lead to side effects w hen softw are gets changed, preventing an effective maintenance.
TimeStamp: Thu Oct 26 14:34:10 CEST 2017

PUS_Application	Model Coupling
ACS	32.0
ONS	47.0
CDH	11.0

PUS_Service	Model Coupling
ACS<—>ACS-Service-1	6.0
ACS<—>ACS-Service-2	3.0
ACS<—>ACS-Service-3	12.0
ACS<—>ACS-Service-8	11.0
ONS<—>ONS-Service-1	6.0
ONS<—>ONS-Service-3	2.0
ONS<—>ONS-Service-8	24.0
ONS<—>ONS-Service-5	1.0
ONS<—>ONS-Service-150	3.0
ONS<—>ONS-Service-152	11.0
CDH<—>CDH-Service-1	6.0
CDH<—>CDH-Service-3	1.0
CDH<—>CDH-Service-8	4.0

Suggested countermeasures

In case a module exceeds the threshold, its connections to other objects shall be investigated and the application/service potentially split up in two or more applications.

Model Coupling metric as example

Recommendations for ECSS

ECSS-Q-80 (ST+HB)

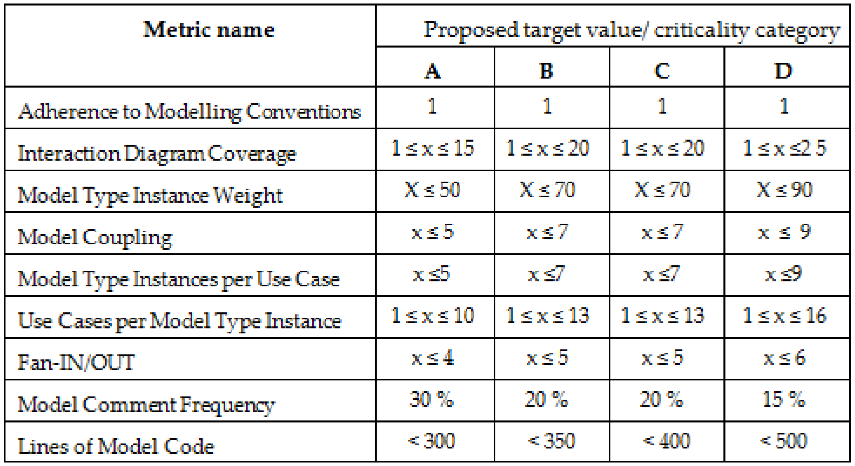
- Minor adaptions in various clauses
- Reference model-based software quality model
- 10 Model metrics
- Tailoring recommendations for the model metrication programme
- Model metrics applicability and thresholds based on criticality
- 3 new sub-characteristics

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

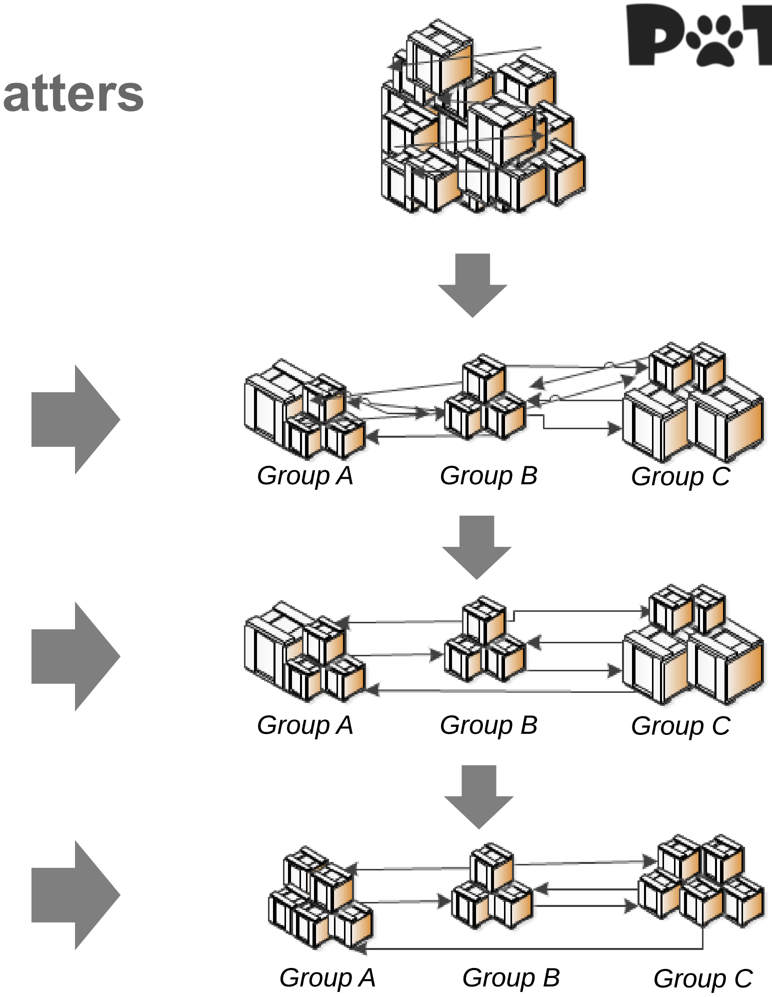


Metric name	Proposed target value/ criticality category			
	A	B	C	D
Adherence to Modelling Conventions	1	1	1	1
Interaction Diagram Coverage	1 ≤ x ≤ 15	1 ≤ x ≤ 20	1 ≤ x ≤ 20	1 ≤ x ≤2 5
Model Type Instance Weight	X ≤ 50	X ≤ 70	X ≤ 70	X ≤ 90
Model Coupling	x ≤ 5	x ≤ 7	x ≤ 7	x ≤ 9
Model Type Instances per Use Case	x ≤5	x ≤7	x ≤7	x ≤9
Use Cases per Model Type Instance	1 ≤ x ≤ 10	1 ≤ x ≤ 13	1 ≤ x ≤ 13	1 ≤ x ≤ 16
Fan-IN/OUT	x ≤ 4	x ≤ 5	x ≤ 5	x ≤ 6
Model Comment Frequency	30 %	20 %	20 %	15 %
Lines of Model Code	< 300	< 350	< 400	< 500

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

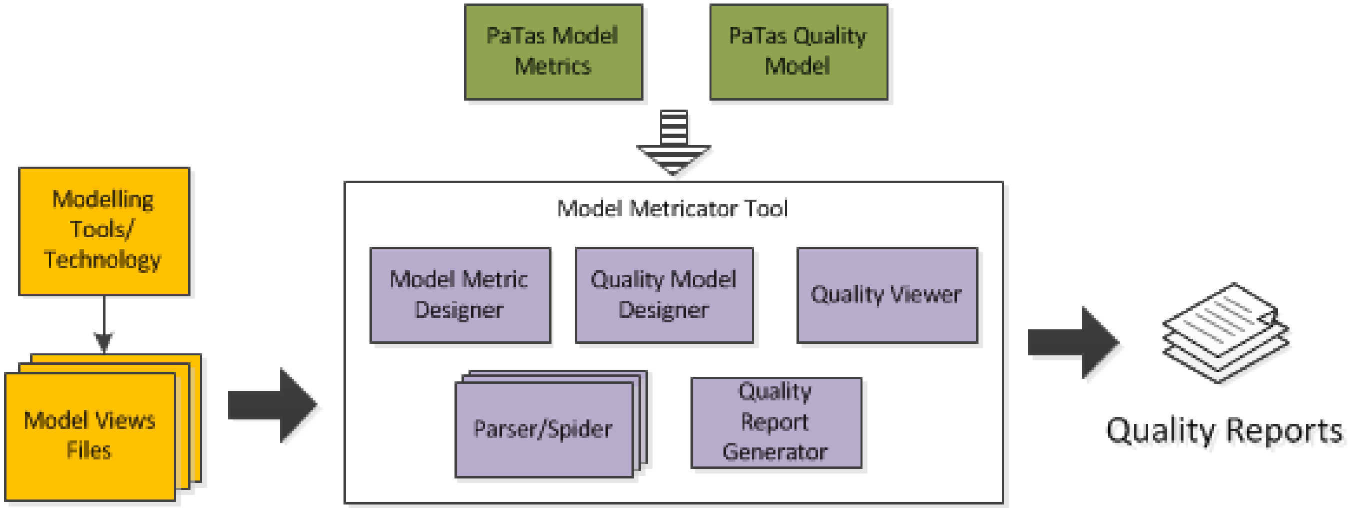


Further Qualitative Conclusiones

- **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

Credit for GIFs: openclipart.org

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**