Tailoring complexity metrics for simulink models

Tailoring Complexity Metrics for Simulink Models

Marta Olszewska[1] Yanja Dajsuren[2,4] Harald Altinger[3]
Alexander Serebrenik[4] Marina Waldén[1] Mark G.J. van den Brand[4]

[1]Åbo Akademi University, Turku, Finland
[2]Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
[3]Graz University of Technology, Austria
[4]Eindhoven University of Technology, Eindhoven, The Netherlands
mplaska@abo.fi, y.dajsuren@tue.nl, harald.altinger@ist.tugraz.at
a.serebrenik@tue.nl, mwalden@abo.fi, m.g.j.v.d.brand@tue.nl

ABSTRACT

The size and complexity of Simulink models is constantly increasing, just as the systems which they represent. Therefore, it is beneficial to control them already at the design phase. In this paper we establish a set of complexity metrics for Simulink models to capture diverse aspects of complexity by proposing new and redefining existing metrics. To evaluate the applicability of our metrics, we compare them with the closed-source metric proposed by Mathworks. Moreover, through a case study from the automotive domain, we relate such metrics to quality attributes as determined by domain experts, and correlate them to known faults. Preliminary assessment suggests that complexity is closely related to analysability, understandability, and testability.

Keywords

Simulink, complexity, metrics, software quality, automotive domain, expert evaluation

1. INTRODUCTION

Since 90% of the innovation in the automotive industry is driven by electronics and software [5], ensuring software quality has become a necessity. In automotive software engineering, MATLAB/Simulink [17] is one of the most popular graphical languages and an integrated environment for modelling and simulating automotive controller software systems. Since control models are organised using hierarchical structure Simulink models conform to this architecture and are represented as hierarchical data flow diagrams. The architecture can be decomposed into layers (top, trigger, structure and data layer) to facilitate workflow. Scalability of an architecture is supported by componentisation. Simulink models can be used for simulation and code generation, e.g to C language.

ECSAW'16 November 28 – December 02, 2016, Copenhagen, Denmark

ACM ISBN 978-1-4503-4781-5/16/11.
DOI: http://dx.doi.org/10.1145/2993412.3004853

Since 85% of the bugs are introduced in the early development phase, it is crucial to develop techniques to detect bugs early instead of causing costly field recalls [12]. Automated source code analysis methods and tools [19, 28, 7] have been developed to perform quality analysis at the early development phase. Since these tools are commercial, the metrics defined for the source code and Simulink models are not publicly available. Therefore, it is difficult to reproduce the complexity evaluation. In consequence, there is a need for well-defined and publicly available complexity metrics for Simulink models, regardless if they are reengineered from existing commercial tools or originate from metrics meant for other applications. To evaluate software complexity, different metrics have been defined. The cyclomatic complexity metric designed by McCabe [27] to indicate system's testability and understandability is one of the most popular complexity metrics. Mathworks Verification and Validation software tool approximates the resulting McCabe complexity of generated code out of Simulink models [24]. Scheible applies Halstead mapping out of the M-XRAY tool to measure average local complexity and global complexity to manage testability and maintainability respectively [32]. However, the Halstead mapping is not described due to the commercial nature of the tool. Furthermore, Olszewska (Pląska) [30, 31] defined structural and data flow complexity metrics inspired by Card and Glass metrics [6], as well as instability and abstractness metrics [29] based on the object-oriented concepts [20].

As Curtis states [8], the design of methods has an influence on the ability of developers to understand a program. If a complexity metric can show which parts of Simulink models are too complex, the model can be redesigned and refactored, therefore complexity metrics can be fundamental. Since early 2000, a number of automotive Architecture Description Languages (ADLs) has been defined in the automotive industry. However, there is still a lack of quality evaluation of the models represented in these ADLs [10].

In this paper we first present the existing complexity definitions and metrics, as well as the ones used for Simulink (Section 2). Then we define and implement a Complexity Metrics Suite for Simulink models (Section 3). It is empirically evaluated in Section 4, where we analyse the results and cross-reference the complexity measurements with the defect data we have for the case study. Finally, in Section 5 we discuss the threats to validity and give a concise description of our results.

2. COMPLEXITY DEFINITION

There has been a lack of consensus on the definition of complexity [2]. A plethora of definitions has been discussed in [9], where the complexity is described as a characteristic addressing many concepts and embracing computational, cognitive and structural perspectives. There is no explicit definition of complexity in the ISO 25010 international standard also known as SQuARE model. Complexity is, however, still used as a sub-characteristic of other quality characteristics e.g. testability [26]. In this paper we investigate complexity in the context of models (and eventually code).

2.1 Complexity Metrics for Code

Kan [18] related the design and code implementation metrics to software quality. He identified the *Lines of Code (LOC)*, *Halstead's software science metrics* [14], and *McCabe's cyclomatic complexity* [27] as key metrics for code implementation. He presents structure metrics for capturing interactions between modules in a software system. In our work we choose the Kan's metrics that are available for source code.

There are also several complexity metrics related to input/output of the system (sometimes referred as fan-in and fan-out), as well as its structure. For instance, *Henry-Kafura* defined *complexity* as a function of fan-in and fan-out to determine the information flow between different modules. There are also some hybrid versions of this metric described in the literature, see e.g. [33] and [16]. Card and Glass, on the other hand, defined *structural, data, and total complexity metrics*, which can be computed on a module and system level. We find the concepts of these metrics particularly valuable for systems of layered architecture and comprised of subsystems.

Indicators of a good design, where "goodness" is inversely proportional to complexity, were defined by Martin [21]. The *dependency metrics of instability, abstractness, and distance* indicate how easily the (sub)system can be changed in terms of its dependency and abstractness. These metrics could aid in assessment of Simulink models in relation to their maintainability.

2.2 Complexity Metrics for Simulink

There are several measurement tools built into Simulink, one of them being *sldiagnostics* [23]. It displays diagnostic information associated with the model or subsystem, providing measurements on number of each type of block, number of each type of Stateflow object, number of states, outputs, inputs, and sample times of the root model, names of libraries referenced and instances of the referenced blocks, as well as time and additional memory used for each compilation phase of the root model [23]. Furthermore, a metric of cyclomatic complexity, defined as a measure of the complexity of a software module based on the number of nodes, edges and components within a diagram [24], is implemented in the *Simulink Verification and Validation toolbox*. Finally, as one of the mechanisms for reducing the complexity of models, Mathworks provided the *MAAB guidelines* [25], which can be interpreted as modelling patterns and contribute to creating an aesthetically pleasing design, rather than serve evaluation purposes.

Scheible's instability metric [32] calculates the average stability of the blocks of a Simulink model and is based on the concepts of blocks and their fan-in and fan-out. It is indicated that a block with more fan-in blocks as fan-out blocks has a higher probability of change [32].

3. SIMULINK COMPLEXITY METRICS

It is essential to clearly define the attribute or property that is to be measured, since the design of a measurement method and metric heavily depends on it. In our work we define the *complexity of a Simulink model* as the property of a system showing the degree to which the (sub)system or its part(s) has a design that is difficult to create, understand, learn, analyse and test the system on a model level, as well as it is numerically challenging to simulate. Therefore, our definition encapsulates the human and machine aspects of complexity of Simulink models, respectively. Our goal is not only to include the structure of the system and data flow view, but also the interrelations between (sub)systems, i.e. subsystems and signals, as well as human perception.

In the following sub-sections we present Simulink-specific metrics we established. Although McCabe and Halstead-based complexity metrics are mentioned by other researchers, the mapping to Simulink model is not publicly provided in the literature (in English). The Halstead mapping is not described in detail due to the commercial nature of the tool [7]. We include the structural and data flow complexity metrics by Olszewska (Pląska) [30, 31], which were inspired by Card and Glass metrics [6], and the instability and abstractness metrics based on object-oriented concepts [20]. The large-scale evaluation of these metrics has not been carried out before, therefore the thresholds for the acceptable values cannot be provided. However, we apply *absolute comparators* (e.g. ranges specified for abstractness and distance metrics) and *relative comparators* (lowest and highest values of other metrics e.g. complexity metrics for the projects under the evaluation) to interpret the metrics.

3.1 Cyclomatic Complexity

Cyclomatic complexity for Simulink is defined based on McCabe's design complexity calculation. We extend the mapping between the C statements and Simulink control logic blocks [22] as shown in the Table 1. Note that the choice of C language is caused by the possibility code generation from Simulink models to C code and it is one of the widely used programming languages in the automotive software development [4].

Table 1: Mapping between C and Simulink concepts.

C statement	Simulink blocks
if-else	If block, If Action Subsystem
for	For Iterator block, For Iterator Subsystem
while, do-while	While Iterator block, While Iterator Subsystem
switch	Switch Case block, Switch Case Action Subsystem

In addition, *For Each Subsystem*, *Atomic Subsystem*, and the number of case statements of the *MultiPortSwitch* block are also counted as decision statements. Hence the following equation is defined as:

$$\text{mcCMX} = \text{P} + 1$$

where $mcCMX$ is cyclomatic complexity and P is the total number of decision nodes (including atomic subsystems).

3.2 Static Syntactical Complexity

Halstead metrics for Simulink are represented in the same manner as in the original version:

- System vocabulary: $n = n_1 + n_2$
- System size: $N = N_1 + N_2$
- Volume: $hCMX\_V = N \cdot log_2(n)$
- Difficulty: $hCMX\_D = (\frac{n_1}{2}) \cdot (\frac{N_2}{n_2})$

where Halstead concepts are mapped to Simulink setting as follows: n_1 is the number of distinct Simulink block types, n_2 is the number of distinct input signals, N_1 is the total number of Simulink blocks, N_2 is the total number of input and output signals. In our work $hCMX\_D$ is an indicator of complexity. We are aware that the original Halstead metrics measure code size. Although the validity of these metrics was criticised [15], we wanted to investigate them in the Simulink setting.

3.3 Information Flow Complexity

For the **Simulink information flow complexity**, we defined the following metrics based on the Henry-Kafura's metrics.

$$hkCMX = size \cdot (fanin \cdot fanout)^2$$

where $hkCMX$ is the information flow complexity of a subsystem, *size* is the number of contained blocks (including subsystem blocks), *fanin* and *fanout* represent the number of afferent and efferent blocks of a subsystem, respectively.

3.4 Structural and Data Complexity

For structural and data complexity of Simulink models, the Card and Glass metrics are used as originally given. However, modules are changed into subsystem blocks, I/O variables are mapped to the signals entering and exiting the subsystem block for input and output, respectively. The metrics are reformulated as:

- *Structural complexity* $CMX\_S = \frac{\sum_{i=1}^{n} f^2(i)}{n}$ is related to coupling of a system, and measures the mean of squared values of fan-out per number of subsystem blocks, where $f(i)$ is fan-out of subsystem block i and n is a number of subsystem blocks in the system.
- *Data complexity* $CMX\_D = \frac{V(i)}{(f(i)+1) \cdot n}$ is related to cohesion of a system, and is a measure of block's interaction with other blocks; it is a function that is dependent on the sum of I/O variables and inversely dependent on the number of fan-out in the subsystem block.
- *Total complexity* $CMX\_T = CMX\_S + CMX\_D$ is the sum of the structural and data complexity.

The intuition behind CMX_S and CMX_D is the following: both are based on the concepts of fan-outs and I/O variables. The reasoning is twofold: the complexity increases as the square of the connections between subsystems, since more fan-out means that functionality is deferred to subsystems at lower levels; and the more I/O signals in a subsystem the more functionality needs to be accomplished by the subsystem, and therefore, the higher complexity.

3.5 Dependency Metrics

The dependency and instability of the Simulink models, just as in case of software architectures, strongly impacts their sustainability and resilience to change, as well as quality in general.

We define the **instability metric** $dCMX\_I$ for Simulink models as the number of efferent couplings between blocks (CeB) divided by the sum of efferent (CeB) and afferent couplings between blocks (CaB), which is given by the equation:

$$dCMX\_I = \frac{CeB}{CeB+CaB}$$

Afferent coupling between blocks (CaB) is measure of the total number of external blocks linked to a given block due to incoming signal within one layer. In other words, it is the number of destination blocks for the block under analysis. Efferent coupling between blocks (CeB) is defined as the number of blocks that are linked to a given block due to outgoing signal within one layer, i.e. it is the number of source blocks for the given block.

Values range from 0 (no incoming signals), $dCMX\_I = 0$ denotes a completely stable (subsystem) block to 1 (only incoming signals), $dCMX\_I = 1$ signifies a maximally instable (subsystem) block.

Scheible [32], on the other hand, defined an *instability metric* calculated as following:

$$Sch\_Inst = \frac{\sum_{b \in blocks} \frac{fanin(b)}{fanin(b)+fanout(b)}}{size}$$

where, *fanin(b)* is the number of fan-in blocks of a block b, *fanout(b)* is the number of fan-out blocks of a block b, and *size* is the number of contained blocks.

The **abstractness metric** $dCMX\_A$ for Simulink Models is defined as a ratio of the number of subsystem blocks NaB to the total number of blocks NB:

$$dCMX\_A = \frac{NaB}{NB}$$

Values range from 0 (a concrete block) to 1 (a completely abstract block).

We define the **distance metric** $dCMX\_D$ for Simulink models as the relationship between instability $dCMX\_I$ and abstractness $dCMX\_A$. It is computed as a normalised sum of these values decreased by one:

$$dCMX\_D = |dCMX\_A + dCMX\_I - 1|$$

Values range from 0 to 1. 0 is considered desirable, because blocks are either totally stable and abstract (scenario $dCMX\_I = 0$ and $dCMX\_A = 1$). 1 indicates entirely instable and concrete block (scenario $dCMX\_I = 1$ and $dCMX\_A = 0$).

4. EVALUATION

We developed a *complexity analysis tool* to automatically measure the complexity metrics defined in Section 3. It is an extension of the tool that measures modularity metrics of Simulink models [11]. The complexity analysis tool reads Simulink MDL files with the standard structural format and generates the metrics files with the list of subsystems and the respective complexity metrics.

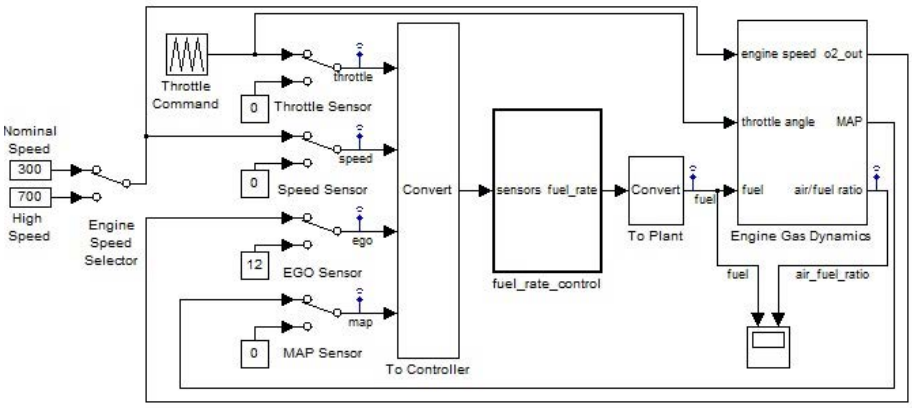


Figure 1: Fault-Tolerant Fuel Control System [1].

4.1 Mathworks Complexity vs. Complexity Metrics Suite

We applied the complexity metric suite to the Fault-Tolerant Fuel Control System (FCS) for a gasoline engine from Mathworks Simulink Example Library [1]. Figure 1 shows the top level of the FCS system. The subsystem *fuel_rate_control* uses signals from the system's sensors to determine the fuel rate. The fuel rate combines with the actual air flow in the engine gas dynamics model to determine the resulting mixture ratio as sensed at the exhaust [1]. The purpose of the evaluation of the FCS system is to compare the complexity metric suite measurement to Mathworks' complexity evaluation of the FCS system. The FCS system consists of 25 subsystems and the maximum hierarchical depth is 5.

As discussed in Section 2.2, *Simulink Verification and Validation toolbox* from Mathworks determines cyclomatic complexity metric based on the number of nodes, edges and components. To elaborate it further, we provide below the equation used by Mathworks [24]:

$$CMX\_M = \sum_{i=1}^{N}(o_i - 1)$$

where CMX_M is the Mathworks complexity, N is the number of decision points that the object (such as a block, chart, or state) represents, and o_i is the number of outcomes for the ith decision point.

The complexity measurement of the FCS is provided in Table 2. The first column lists all the subsystems contained in the FCS system. The second column contains the Mathworks complexity metric values *(CMX_M)*. The third column lists the number of contained subsystems *(NCS)*. The other columns contain the complexity metrics values defined in Section 3.

We carried out the Kendall's τ correlation analysis [13] on the complexity metric suite and Mathworks' cyclomatic complexity metric. We accept a common significance level of 0.05. According to the Mathworks' complexity analysis, *fuel_rate_control*, *control_logic*, and *Engine Gas Dynamics* subsystems are considered the most complex and *To Controller*, *To Plant*, and *validate_sample_time* subsystems are considered the least complex subsystems. We identified that the Mathworks complexity metric (CMX_M) is strongly correlated to the size metric, namely number of contained subsystems (NCS) ($r = 0.734$). Complexity metrics for source code have a strong correlation with the size metrics as well. However, the graphical modelling representation of Simulink requires other complexity attributes besides size.

We identified that the other complexity metrics provide more insight into the complexity analysis. The cyclomatic complexity ($mcCMX$) metric identifies that the subsystem *Throttle* is the most complex, because of higher values of Halstead ($hCMX\_V = 114.7$ and $hCMX\_D = 4.2$), Henry-Kafura ($hkCMX = 14$), and instability metrics ($Sch\_Inst = 0.75$).

4.2 Complexity vs. Quality Attributes

As a second method for validation of metrics we interviewed domain experts on how they perceive complexity. Our expert group consisted of five practitioners with a role of an architect, a developer, and a tester, who evaluated a number of randomly selected subsystems of an automotive application using a Likert-like scale of 1 (least complex) to 10 (most complex). Although there are in general no major inconsistencies between the experts, the developer and tester would provide lower complexity score given their familiarity with the system under review. Prior to the evaluation, definitions of the quality characteristics based on the ISO 25010 international standard were provided to the experts. The following complexity characteristics were recognized from the experts' feedback:

- Complexity vs. Analysability: If difficult blocks (e.g. *flip flop*, *unit delay*, *hit crossing*) and many feedback loops are used, then the model is considered complex and difficult to analyse.
- Complexity vs. Understandability: Too many I/O signals, hierarchical levels, parameters, and unclear naming of subsystems make it complex and difficult to understand. Whenever the number of input signals are high, used algorithms are not complex then the model is considered not complex.
- Complexity vs. Testability: Too many parameters make the model complex and difficult to configure correctly. Many dependencies with other modules make the model complex as well.

Besides the complexity ('ExpCmx'), domain experts evaluated understandability ('ExpUnd'), analysability ('ExpAnz'),

Table 2: Complexity Measurement of Fault Tolerant Fuel Control System.

Subsystem	CMX_M	NCS	mcCMX	hCMX_V	hCMX_D	hkCMX	CMX_S	CMX_D	CMX_T	Sch_Inst	dCMX_I	dCMX_A	dCMX_D
fuel_rate_control	62.0	13.0	3.0	39.0	0.6	6.0	1.3	2.2	3.5	0.5	0.5	1.0	0.5
control_logic	56.0	4.0	1.0	95.1	1.5	0.0	0.0	4.0	4.0	0.33	0.0	0.6	0.4
Engine Gas Dynamics	13.0	9.0	1.0	27.0	0.5	0.0	0.5	3.0	3.5	0.5	1.0	1.0	1.0
Throttle & Manifold	10.0	6.0	1.0	34.9	1.1	0.0	1.0	2.0	3.0	0.5	0.0	0.7	0.3
Throttle	6.0	3.0	6.0	114.7	4.2	14.0	0.0	2.0	2.0	0.75	0.5	0.2	0.3
fuel_calc	4.0	6.0	1.0	20.7	0.6	0.0	0.5	2.8	3.3	0.67	1.0	1.0	1.0
Mixing & Combustion	3.0	2.0	3.0	41.5	2.3	0.0	0.0	2.0	2.0	0.5	1.0	0.3	0.3
Speed.speed_estimate	3.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.67	0.0	0.0	1.0
EGO Sensor	2.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	1.0
f(theta)	2.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	1.0
feedforward_fuel_rate	2.0	1.0	4.0	28.5	1.0	0.0	0.0	0.0	0.0	0.67	0.0	0.0	1.0
g(pratio)	2.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	1.0
Intake Manifold	2.0	2.0	2.0	39.9	2.7	8.0	0.0	3.0	3.0	0.4	0.5	0.3	0.3
MATLAB Function	2.0	1.0	1.0	24.0	1.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	1.0
switchable_compensation	2.0	4.0	4.0	80.0	3.0	0.0	0.0	3.3	3.3	0.75	1.0	0.4	0.4
airflow_calc	1.0	1.0	3.0	225.7	5.8	24.0	0.0	0.0	0.0	0.5	0.5	0.0	0.5
Pressure.map_estimate	1.0	1.0	1.0	11.6	1.0	0.0	0.0	0.0	0.0	0.67	0.0	0.0	1.0
Throttle.throttle_estimate	1.0	1.0	1.0	11.6	1.0	0.0	0.0	0.0	0.0	0.67	0.0	0.0	1.0
disabled_mode	0.0	1.0	1.0	2.0	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	1.0
low_mode	0.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.75	0.0	0.0	1.0
rich_mode	0.0	1.0	1.0	19.7	1.5	0.0	0.0	0.0	0.0	0.75	0.0	0.0	1.0
To Controller	0.0	1.0	1.0	59.2	1.5	0.0	0.0	0.0	0.0	0.8	0.0	0.0	1.0
To Plant	0.0	1.0	1.0	11.6	1.0	4.0	0.0	0.0	0.0	0.33	0.5	0.0	0.5
validate_sample_time	0.0	1.0	1.0	53.3	3.0	0.0	0.0	0.0	0.0	1	0.0	0.0	1.0

and testability ('ExpTst') of the system also using the scale of 1 to 10. According to the expert's evaluation as illustrated in Figure 2, all three quality attributes are related to complexity. This leads to the conclusion that the complexity is a sub-sub-characteristic of several maintainability sub-characteristics, namely analysability, modifiability, and testability.

4.3 Correlation Analysis

We carried out a correlation analysis on the second automotive application to detect a relation between complexity metrics and the number of faults. We used the data collected from the second automotive application consisting of 40 subsystems, from which half of the subsystems contain defects because of proprietary library subsystems. Since there are a number of tied values and we aim to establish if any complexity metric and the number of faults are statistically correlated rather than measuring the degree of the linear relationship between variables, we use the Kendall's τ correlation test as used in our modularity assessment of Simulink models [11].

The correlation coefficient infers the strength and direction of the correlation, i.e., a positive correlation coefficient indicates a positive relation between the metrics and defects and vice versa. The significance points to a probability for a coincidence. We accept a common significance level of 0.05. According to the correlation analysis, the structural complexity CMX_S, data complexity CMX_D, total complexity CMX_T, abstractness $dCMX\_A$ and block instability $dCMX\_I$ metrics are positively correlated with the number of faults. This may imply that the subsystems with higher number of fan-out subsystems and instability can be more prone to faults.

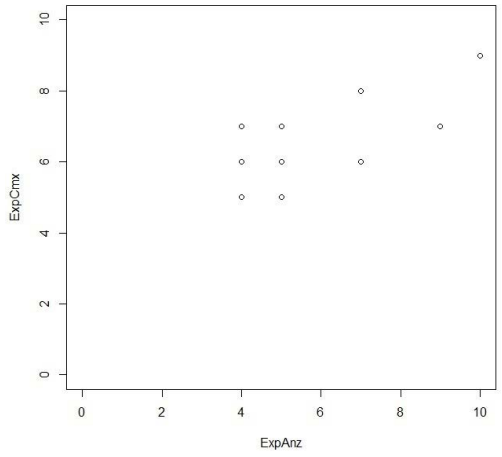
We released a public available dataset on three automotive projects containing model metrics as presented in this paper and traditional source code metrics on the generated code [3]. Early analysis shows a weak correlation between the complexity metrics, which indicates that they measure different aspects of the Simulink models.

5. DISCUSSION AND CONCLUSIONS

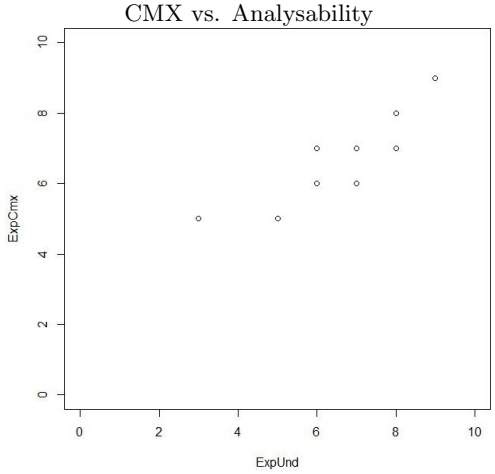
Complexity of produced software systems can become an issue regardless of the application domain. In particular, in automotive domain, where vehicles can be seen essentially as big computer systems consisting of multiple components running various software, quality, and thus complexity, is of utmost importance. There is a need for mechanisms to efficiently evaluate the complexity of such systems, particularly in the early stages of development, at the modelling stage, as it is more economical and beneficial.

In our work we defined and implemented a Complexity Metric Suite for Simulink modelling language based on the well-known complexity metrics from the software engineering field. Although the novelty of the definition of some of our Simulink metrics could be argued, the mapping of the existing complexity metrics from code to Simulink models was achieved. So far, it has not been publicly shared due to (i) the competitive nature of the automotive industry and (ii) closed description of metrics in commercial tools. Since metrics are developed in a proprietary setting, their application to a generic automotive system needs further investigation.

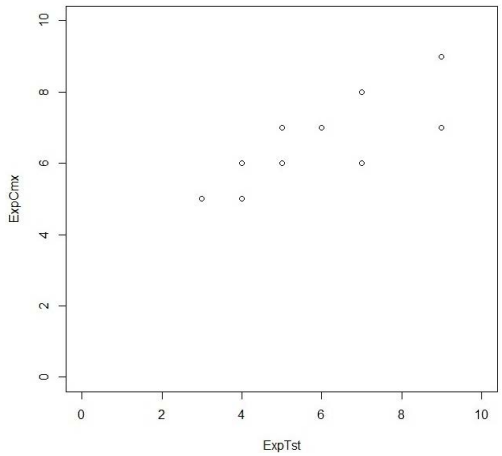
We identified the relation between complexity characteristic and understandability, analysability and testability by involving domain experts in the evaluation. A preliminary analysis provided a strong correlation between some of the complexity metrics and the number of faults. Although we are aware of the limited number of practitioners involved in



CMX vs. Analysability



CMX vs. Understandability



CMX vs. Testability

Figure 2: Expert's evaluation: Complexity (CMX) vs. Quality attributes

the evaluation, they very well represent potential users of the metrics in model engineering tasks.

Our future work continues this research on two levels: (a) refining the established metrics and (b) linking them to standards and architectural complexity models. Complexity metrics can be evaluated further in the specific functional domains of automotive embedded systems. Simulink models developed in other domains can be also used for the evaluation of application of the complexity metrics suite proposed in this paper.

6. ACKNOWLEDGMENTS

The work of the first author is done within ADVICeS project (No. 266373) funded by the Academy of Finland. The roles of female authors are as follows: Marta Olszewska was establishing the metrics suite together with Yanja Dajsuren, who was additionally responsible for the experimental part of the study; Marina Waldén and other co-authors were scientific supervisors in this undertaking.

7. REFERENCES

[1] Modeling a fault-tolerant fuel control system. http://www.mathworks.nl/products/simulink/examples.html?file=/products/demos/shipping/simulink/sldemo_fuelsys.html.

[2] A. Abran. *Software Metrics and Software Metrology*. Wiley, 2010.

[3] H. Altinger, S. Siegl, Y. Dajsuren, and F. Wotawa. A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, Italy, May 2015. IEEE.

[4] H. Altinger, F. Wotawa, and M. Schurius. Testing methods used in the automotive industry: Results from a survey. In *Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-based Testing (JAMAICA)*, pages 1–6. ACM, 2014.

[5] M. Broy. Automotive software engineering. In *ICSE'03*, pages 719–720. IEEE, 2003.

[6] D. Card and R. Glass. *Measuring Software Design Quality*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[7] CQSE. ConQAT. https://www.cqse.eu/en/products/conqat/overview/. (Accessed May 1, 2016).

[8] B. Curtis, S. Sheppard, and P. Milliman. Third time charm: Stronger prediction of programmer performance by software complexity metrics. In *The 4th international conference on Software engineering*, pages 356–360, 1979.

[9] Y. Dajsuren. *On the design of an architecture framework and quality evaluation for automotive software systems*. PhD thesis, Technische Universiteit Eindhoven, 2015.

[10] Y. Dajsuren, M.G.J. van den Brand, A. Serebrenik, and R.G.M. Huisman. Automotive ADLs: a study on enforcing consistency through multiple architectural levels. In *International ACM SIGSOFT conference on Quality of Software Architectures (QoSA)*, pages 71–80. ACM, 2012.

[11] Y. Dajsuren, M.G.J. van den Brand, A. Serebrenik, and S.A. Roubtsov. Simulink models are also software: Modularity assessment. In *International ACM SIGSOFT conference on Quality of Software Architectures (QoSA)*, pages 99–106. ACM, 2013.

[12] Y. Dajsuren, A. Serebrenik, R.G.M. Huisman, and M.G.J. van den Brand. A quality framework for evaluating automotive architecture. In *the FISITA World Automotive Congress*, pages 1–7. FISITA, 2014.

[13] A. Field. *Discovering Statistics Using SPSS*. SAGE Publications, 2005.

[14] M. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[15] P. G. Hamer and G. D. Frewin. M.H. Halstead's Software Science - a Critical Examination. In *Proceedings of the ICSE'1982*, pages 197–206. IEEE CS Press, 1982.

[16] S. Henry and C. Selig. Predicting source-code complexity at the design stage. *IEEE Software*, 7(2):36–44, Mar. 1990.

[17] T. M. Inc. Simulink - Simulation and Model-based Design. http://www.mathworks.com/products/simulink/. (Accessed May 1, 2015).

[18] S. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[19] Klocwork. Source Code Analysis. http://www.klocwork.com/products-services/klocwork/static-code-analysis. (Accessed May 1, 2015).

[20] R. Martin. OO Design Quality Metrics – An Analysis of Dependencies. In *Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*, 1994.

[21] R. C. Martin. OO design quality metrics: An analysis of dependencies. http://www.objectmentor.com/resources/articles/oodmetrc.pdf, 1994. (Accessed May 1, 2015).

[22] Mathworks. Simulink control flow logic. http://www.mathworks.nl/help/simulink/ug/modeling-control-flow-logic.html. (Accessed May 1, 2015).

[23] MathWorks. Matlab sldiagnostics – display diagnostic information about Simulink system. http://www.mathworks.com/help/simulink/slref/sldiagnostics.html, 2014. (Accessed May 1, 2015).

[24] MathWorks. Types of Model Coverage - MATLAB and Simulink. http://www.mathworks.com/help/slvnv/ug/types-of-model-coverage.html, 2014. (Accessed May 1, 2015).

[25] MathWorks Automotive Advisory Board (MAAB). *Control Algorithm Modelling Guidelines Using MATLAB, Simulink, and Stateflow Version 3.0*. MathWorks, 2012.

[26] T. McCabe and C. Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12):1415–1425, 1989.

[27] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, (4):308–320, 1976.

[28] Model Engineering Solutions. MES M-XRAY tool. http://www.model-engineers.com/en/m-xray.html. (Accessed May 1, 2015).

[29] M. Olszewska (Pląska). *On the Impact of Rigorous Approaches on the Quality of Development*. PhD thesis, Turku Centre for Computer Science, 2011.

[30] M. Olszewska (Pląska). Simulink-specific design quality metrics. Technical Report 1002, Turku Centre for Computer Science, Turku, Finland, 2011.

[31] M. Olszewska (Pląska), M. Huova, M. Waldén, K. Sere, and M. Linjama. Quality analysis of Simulink models. In *CONQUEST Conf.*, pages 223–240. Dpunkt.Verlag GmbH, 2009.

[32] J. Scheible. *Automatisierte QualitÃd'tsbewertung am Beispiel von MATLAB Simulink-Modellen in der Automobil-DomÃd'ne*. PhD thesis, UniversitÃd't Tübingen, 2012.

[33] M. Shepperd and D. Ince. *Derivation and Validation of Software Metrics*. International Series of Monographs on Computer Science. Clarendon Press, 1993.