Adaptation of the Analysability Metric for the Automotive Domain

Yanja Dajsuren*, Eric Bouwers†, Harald Altinger‡
*Eindhoven University of Technology, Eindhoven, The Netherlands
†Squla, Utrecht, The Netherlands
‡Audi Electronics Venture GmbH, Gaimersheim, Germany
y.dajsuren@tue.nl, e.bouwers@squla.nl, harald.altinger@audi.de

***Abstract*—MATLAB/Simulink is a popular model-based development language for the automotive domain. When developing a software system in Simulink, subsystems are used to decompose a (sub-) system to improve its analysability. However, it has not been investigated if decomposing a system into particular number of subsystems has impact on the analysability of the Simulink models. Existing metric known as Component Balance (CB) was designed to define the optimal decomposition of a system into top-level components. The CB metric was used to evaluate software architecture of industrial and open source software systems. Therefore, in this paper we adapt the CB metric for Simulink models. The preliminary results indicate that the CB metric provides useful input to the analysis of the Simulink analysability and the unbalanced (sub-) systems are fault-prone.**

I. Introduction

During software development both project managers and software engineers use software metrics to assess the current situation of their projects and products. There are literally hundreds of software metrics available for general purpose languages, with a varying degree of validation and usage reports. A wide range of software metrics is in use to perform software analysis, e.g. fault prediction. They gained popularity in industry e.g. Curtis et.al. [8], Graves et.al [12] and in research e.g. Zhang et.al [19]. Industry related research such as Zimmermann et.al [20] and Boehm [4] claimed the need for software metrics due to 80% of bugs are contained in 20% of the files which drives a high demand to find them. Model-based development methods such as MATLAB/Simulink drive the need for new metrics preserving knowledge gained during the existing research such as [8], [19], [20].

When validation or usage reports are missing, it becomes harder to determine in which situations a software metric is useful and even whether it is useful at all. Moreover, it becomes impossible to assess whether the software metric can be ported from its original domain, usually the domain of general purpose languages to a more specific domain such as Simulink. Model-based software development has been adapted in the automotive industry to tackle the increasing complexity and to decrease the software development effort and cost [3]. Altinger et.al [2] reported in a recent survey that model-based development method mainly MATLAB/Simulink is widely used across automotive development stages. Simulink is a visual model design language that is widely used to enable the model-based software development of control software in the automotive industry [3].

Although many advantages of the visual model design languages exist, some of the traditional software development features lack in these languages, in particular in Simulink. For example, model analysability (comprehension and effective use/reuse of subsystems) are considered problematic in automotive Simulink models [3]. A subsystem is one of the popular techniques to enable reuse and to tackle increasing complexity and size. Subsystems are broadly used to decompose the system to increase the model comprehension and maintainability. Although subsystems can be of different types, e.g., *enabled*, *triggered*, *virtual*, and *atomic* subsystems, they in general consist of a set of (related) blocks/operations and may contain other subsystems. Thus large models can contain big number of subsystems at one level and significant hierarchies of subsystems as well. The main characteristic of having a subsystem as a virtual element (except e.g., an atomic subsystem) may cause an ad-hoc distribution and decomposition of subsystems in Simulink models. An atomic subsystem can be reused similar to a library method.

Therefore, we explore in this paper the applicability of the CB metric for the Simulink domain. The CB metric is defined to address the decomposition of a system by taking into account both the number of components (subsystems) and their relative sizes [6]. This study would benefit the automotive software engineering community to have an objective view when decomposing the system into subsystems and eventually would help model comprehension and maintainability as well. Curtis et.al [8] performed a test with students how they are able to find bugs easily and showed that the size and structure of the program (as well as the coding style and the amount of documentation) influences the ability to understand the program and locate bugs. Eric et.al [6] made a similar claim for traditional languages. In this paper, the relation between the CB and comprehension/maintenance for Simulink models is investigated. The main contribution of this paper is the application of the CB metric to Simulink as an indicator of the ease of comprehension and maintainability.

In the following sections, we present the relevant Simulink concepts and briefly introduce the CB metric. Then we map CB to Simulink domain. We report on the preliminary results of the CB metric evaluation and discuss the next steps we plan to take on improving the metric and evaluation.

II. Problem Statement

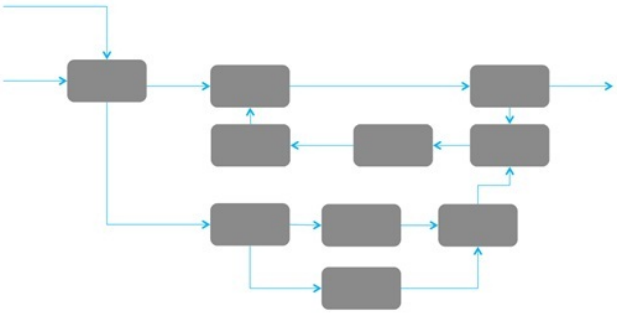
Although Simulink is a popular model-based development language in automotive development stages [2], 'proper' decomposition of a system into subsystems have not been

analysed. To illustrate this problem, Figure 1 shows examples of a system decomposition alternatives. The system consists of 10 blocks. In Figure 1a, the system is not decomposed into any subsystems and simply contains 10 blocks. In Figure 1b, the system consists of one block (in orange color) and a subsystem, which consists of two other subsystems each consisting of five and four blocks. In Figure 1c, the system is decomposed into two subsystems, which contain four and six blocks.

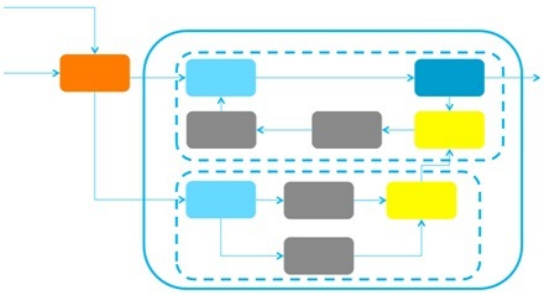
The following research question is defined:
How to adapt the Component Balance metric as an indicator for the success of decomposition of Simulink systems?

III. Simulink Model and Metrics

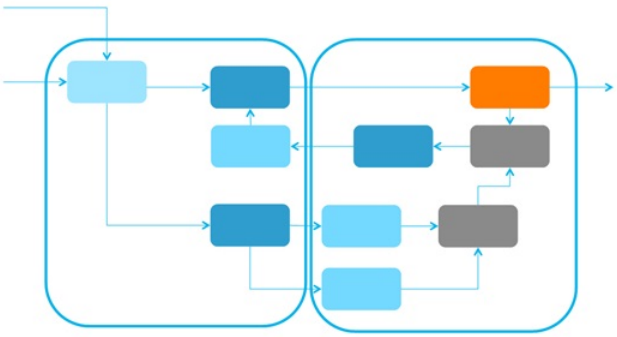
Simulink is a visual model design language and tool for developing, simulating and analysing multi-domain dynamic systems [16]. A *Simulink model* contains a Simulink diagram, which consists of different kinds of blocks. *Subsystems* can be included as blocks, they contain again a Simulink *diagram*, which contains a set of blocks [14]. The subsystem concept enables hierarchical model design, i.e., subsystems can contain other subsystems. The subsystem block can represent a virtual subsystem or a non-virtual subsystem. A subsystem is *virtual* if the subsystem is used for grouping a set of blocks visually and does not execute as a single execution unit. A subsystem is *non-virtual*, if the execution of the contents of the subsystem can be controlled[1] by using a triggered transition (triggered subsystem), a function-call (function-call subsystem), an action (action subsystem), or an enabling input (enabled subsystem). A subsystem can become an atomic subsystem, if its parameter "Treat as atomic unit" is selected. An *atomic subsystem* is a subsystem that can contain a contract [5], which means blocks within an atomic subsystem are grouped together in the execution order (non-virtual).



(a) Alternative A



(b) Alternative B



(c) Alternative C

Fig. 1: Example decomposition alternatives

We defined a special kind of subsystem as a *BasicSubsystem* in [9], if it does not contain other subsystems. *Blocks*, basic elements of a Simulink diagram, communicate via input (*InPort*) and output (*OutPort*) ports. A block can be connected to another block by a *Signal* via its ports. For the sake of diagram readability signals are frequently grouped in buses.Simulink models are directed graphs, following the data flow. *In* and *Out* define the direction.

A. Simulink Metrics

The objective is to define an indicator of the analysability and ease of comprehension of Simulink models. There are several measurement mechanisms and tools included within Simulink, one of them being `sldiagnostics`[2]. 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 [15]. However, all of them are do not indicate if the decomposition of Simulink models are adequate with respect to ease of comprehension.

In the Simulink Verification and Validation toolbox the documentation provides several mechanisms for model coverage, herein cyclomatic complexity. It is defined as a measure of the complexity of a software module based on the number of nodes, edges and components within a diagram [13]. For analysis purposes, each subsystem counts as a single component.

Finally, as one of the mechanism for reducing the complexity of a model Mathworks provided the MAAB guidelines [17]. However, these contribute to creating visually-good design, rather than serve for evaluation purposes.

IV. Adapting CB Metric

To tailor a more general software metric to a specific domain, several questions need to be answered:

- Which attribute and property in the specific domain do we want to measure?

[1] http://www.mathworks.com/help/simulink/slref/subsystem.html
[2] see the Simulink documentation

- Which general software metric can be tailored for this?
- Which characteristics of the specific domain hinder us from directly using the software metric?

After answering these questions, the general software metric can be tailored by taking into account the specific characteristics of the domain. Based on this definition, the tailored software metric needs to be evaluated within the domain using widely known evaluation techniques for software metrics. In the following sections, we present the answer to the questions posed above to come to a tailored software metric for the Simulink domain.

A. Defining the measurement

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. Fenton et.al [10] uses IEEE Standard on measurement to derive a guideline on developing software metrics to measure software. Pfleeger et.al [18] reports on usage of software metrics, but state that practitioner will use whatever they have a standard spreadsheet, even if the method is not correct.

In this work, we want to quantify the *analysability* of a *Simulink (sub)system*. As a quantification of analysability, we look at the way in which the Simulink (sub)system is decomposed into subsystems. Ideally, a Simulink system is composed of a limited number of subsystems which reside on a similar level of abstraction, making it easy to understand the system as a whole. The aforementioned observation considers aspects related to design modularity and analysability of a system, as well as some views on the system related to how it is perceived by humans (visual representation of the models in Simulink). Our goal is not only to include the structure of the system and data flow view, but also interrelations between (sub)systems (meaning subsystems and signals).

In this work we mostly focus on the issues related to decomposition, i.e. system's structure and interrelations, as opposed to the cognitive aspects, which are considered as a future work.

B. Basics on CB Metric in general

We have identified the CB metric as a viable candidate for measuring the analysability of a system. The CB metric varies between 0 and 1 in which higher is better. The metric takes into account the number of components and their relative sizes [6], assigning a higher score to systems decomposed into a reasonable number of components of similar size. In this metric, the size measurement is used as a proxy for the level of abstraction of the components. The usefulness of this metric has been assessed before, with positive results [7].

Although the metric is defined for software architectural/design models, it is language independent and designed to be adapted. In the Simulink model design, there is currently no metric which can provide insight to engineers whether the decomposition of their (sub)system is proper, i.e., there is no indication if it is good to have a subsystem containing hundreds of blocks or hundreds of subsystems containing a few number of blocks.

The CB is defined as the product of *System Breakdown Optimality* and *Component Size Uniformity*, see Equation 1. Both of these metrics are shortly explained below, more details can be found elsewhere [6], [7].

$$CB(S) = SBO(|C|) \times CSU(C) \tag{1}$$

SBO measures how far the number of components deviates from the "ideal" number of components using the Equation 2. In order to instantiate this metric two values need to be established, the "ideal" number of components μ and the maximum number of components ω. Since there is currently no way to determine these numbers objectively, earlier research used a benchmark based approach to approximate these values.

$$SBO(n) = \begin{cases} \frac{n-1}{\mu-1} & \text{if } n \leq \mu \\ 1 - \frac{n-\mu}{\omega-\mu} & \text{if } \mu < n < \omega \\ 0 & \text{if } n \geq \omega \end{cases} \tag{2}$$

CSU quantifies whether the components are roughly equal in size, as defined by the Equation 3. As mentioned before, size is used as an proxy for the level of abstraction of the components, in practice we observe that components on the same level of abstraction are of roughly equal size.

$$CSU(C) = 1 - Gini(\{volume(c) : c \in C\}) \tag{3}$$

C. Tailoring the CB Metric

Two characteristics of the particular domain of Simulink models hinders us from applying the CB metric directly. First of all, the names of the concepts used within the domain clash with the concepts used to define CB, there these concepts first need to be mapped to the particular domain. Secondly, the values for μ and ω need to be established to instantiate the CB metric for this domain.

We first map the concepts between the design model and Simulink model in Table I. The definitions of the design model elements are provided here from [6]. In the design model, a *system* consists of a set of components. A *component* can be divided into modules (e.g., components can be top-level packages or the collection of files). A *module* is decomposed into units (e.g., modules can represent classes in Java or files in a working project). A *unit* is the finest level of the decomposition (e.g., units can be methods in Java or functions in C).

TABLE I: Mapping elements to Simulink domain

Design model elements	Simulink elements
System	Model
Component (e.g., top-level packages or the collection of files)	Subsystem (which are decomposed into subsystems, basic subsystems and/or blocks.)
Module (e.g., classes in Java or files in a working project)	Subsystem (e.g. enabled, triggered, virtual, and function-call subsystems)
Unit (e.g., methods in Java or functions in C)	Simulink blocks or operations

We presented the main Simulink model design elements in Section III. A *system* of the design model is mapped to

the Simulink *model* concept. A Simulink model represents the system into a diagram, which contains a set of blocks. A *component* concept is mapped to a *subsystem* concept, because subsystems are unit of decomposition in Simulink models. A *module* concept is mapped to the *subsystem* concept as well, because subsystems are only model design element, which can be decomposed further in Simulink. The *unit* concept is mapped to the Simulink blocks or operations, because blocks and operations do not contain other model design elements.

As a second step we use a benchmark to establish the values for μ (the ideal number of components) and ω (the maximum number of allowed components). Following the guidelines for the CB metric, we use the central tendency (e.g., the median) of the benchmark values for μ and the 95^{th} percentile for ω. The benchmark data to extract these numbers is derived from a mixed set of public and proprietary Simulink models. This benchmark contains a total of 460 subsystems, with a central tendency $\mu = 7$, and a 95^{th} percentile $\omega = 21$.

V. Preliminary Evaluation

After tailoring the CB metric for the Simulink domain, we carried out a correlation analysis on an industrial application to detect if there is a relation between CB metric and the presence of faults. Its purpose is to detect if unbalanced (sub)systems hinder quality of Simulink models.

We reused the data collected from the industrial application consisting of 41 subsystems [9]. The faults are traced to the top-level subsystems of the models and 20 subsystems contain faults. We use the Kendall's τ correlation test [11], since there are a number of tied values and we are interested in establishing whether any CB metric and the presence of faults are statistically dependent rather than measuring the degree of the relationship between linear related variables.

The significance (Sig.) points to a probability for a coincidence, and a common significance level of 0.05 is accepted for our analysis.According to the Kendall's τ correlation analysis, the CB metric has a negative relation with the presence of faults. Although this analysis is preliminary, it may imply that the (sub)system with a high value of the CB metric (i.e., well-balanced (sub)system) is less fault-prone.

We further evaluated the CB metric by interviewing automotive domain experts. An automotive architect and a control designer measured the CB metrics on a series of real-life automotive Simulink models. They randomly selected the subsystems with different CB values from different versions of the selected models and compared the CB values to their perception of what a balanced (sub)system is.

The results of these interviews showed that measuring the number of blocks contained in the subsystems makes the metric non-transparent to the architect and designer. This is because a Simulink user concentrates on the selected hierarchical level when looking at the models. Thus using the number of blocks as a measurement of size of the underlying subsystem creates a conflicting impression of the balanced and unbalanced subsystems.

VI. Conclusion and Future Work

In this paper we have discussed the steps to adapt the CB metric to Simulink models. The tailored metric has been evaluated using proprietary models, as well as interviews with domain experts.

Analyzing the data from the evaluations we observe that:

1) the tailored CB metric provides useful input to a structured discussion about the balance of (sub)systems
2) there is some evidence of a negative correlation between the presence of faults and the CB values
3) using the number of blocks as a measurement of size of a (sub)system is confusing for metric users in this domain

Given the limited number of (sub)systems used to establish the correlation for observation, we stress that this is preliminary result. It is interesting to see that even though this positive result exists, the domain experts have difficulties to interpret the metric, which hinders the effectiveness of the metric.

To remedy the last observation we plan to quantify the volume of a (sub)system by counting the number of input signals instead of the blocks it contains. This measurement seems to be more aligned by the empirical measure of size used by the domain experts. Validating both the predictive power and usefulness of this revised metrics is planned as future work.

Regarding the tailoring of the metric we observe that it is not sufficient to map the concepts used to define the metric, but that the attribute to be measured should be aligned with the domain experts as well. This experience shows that the involvement of domain experts is an important factor in the tailoring of software metrics to specialized domains.

We plan to present the correlation analysis among the other metrics defined on the automotive dataset [1]. We aim also to analyze further the correlation between the CB metric and the number of faults. This could be of benefit to the fault-prediction analysis.

References

[1] 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.

[2] H. Altinger, F. Wotawa, and M. Schurius. Testing methods used in the automotive industry: Results from a survey. In *Proc.\ JAMAICA*, pages 1–6, San Jose, CA, July 2014. ACM.

[3] M. Bender, K. Laurin, M. Lawford, J. Ong, S. Postma, and V. Pantelic. Signature required-making simulink data flow and interfaces explicit. In *MODELSWARD*, pages 119–131, 2014.

[4] B. Boehm. *Industrial software metrics top 10 list*. IEEE COMPUTER SOC 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264, 1987.

[5] P. Boström, R. Grönblom, T. Huotari, and J. Wiik. An approach to contract-based verification of Simulink models. Technical Report 985, Turku Centre for Computer Science, 2010.

[6] E. Bouwers, J. Correia, A. van Deursen, and J. Visser. Quantifying the analyzability of software architectures. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, pages 83–92. IEEE, 2011.

[7] E. Bouwers, A. van Deursen, and J. Visser. Evaluating usefulness of software metrics: An industrial experience report. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, 2013.

[8] B. Curtis, S. B. Sheppard, and P. Milliman. Third time charm: Stronger prediction of programmer performance by software complexity metrics. In *Proceedings of the 4th international conference on Software engineering*, page 356360, 1979.

[9] Y. Dajsuren, M. van den Brand, A. Serebrenik, and S. Roubtsov. Simulink models are also software: Modularity assessment. In *Proceedings of the 9th international ACM SIGSOFT conference on Quality of Software Architectures*, pages 99–106. ACM, 2013.

[10] N. Fenton. Software measurement: A necessary scientific basis. *Software Engineering, IEEE Transactions on*, 20(3):199–206, 1994.

[11] A. Field. *Discovering Statistics Using SPSS*.

[12] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653661, 2000.

[13] MathWorks. 2014a documentation, types of model coverage - MATLAB and Simulink, 2014.

[14] MathWorks. Componentization guidelines, 2014.

[15] MathWorks. Matlab sldiagnostics – display diagnostic information about Simulink system, 2014.

[16] Mathworks. Simulink. http://www.mathworks.com/products/simulink, 2015.

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

[18] S. L. Pfleeger, R. Jeffery, B. Curtis, and B. Kitchenham. Status report on software measurement. *IEEE software*, (2):33–43, 1997.

[19] M. Zhang and N. Baddoo. Performance comparison of software complexity metrics in an open source project. In *Software Process Improvement*, page 160174. Springer, 2007.

[20] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, page 91100, 2009.