Towards Event-B Specification Metrics

Marta (Pląska) Olszewska, Kaisa Sere

Åbo Akademi University
Joukahaisenkatu 3-5A
FIN-20520 Turku
mplaska@abo.fi

Abstract. In this paper we describe our ongoing research on Event-B specification metrics. We focus on the physical features of specification, such as its vocabulary or length for the Event-B machines. We base our metrics on the syntactic properties of the Event-B language, namely operators and operands that we consider meaningful for our measurement model. Presented metrics are applied for a number of Event-B machines. Obtained results can be analysed in a perspective of an abstract machine and its refinements.

Keywords: Event-B, specification metrics, size, direct measurements, quantitative measurements, Halstead model, complexity.

1 Introduction

Measurements for software systems and their development process are nowadays a good practice in the computer world [1] [2]. They are a part of software projects in order to assure certain quality [3] or even for standardisation purposes. They have been evolved for many years, extended for particular development methods like Object-Oriented programming [4] [5] or narrowed down to meet the needs of specific programming languages, like Java Programming Language [6]. Quality measurements are done not only at the end-product stage, but much earlier, for the requirements [7] or (formal) model of the system [8]. Early quality assessment has major influence on the final product, as there is a thorough control over the whole development process.

Research on metrics for formal specifications has already been done for the Z language. In [9] authors perform static analysis of Z specification notation, whereas in [10] the focus is on the linguistic properties of the notation and predicting erroneous parts of specifications created in Z. There has also been done an assessment for B language [11], where existing metrics concerned mostly traceability and safety analyses, proof related metrics and direct statistics, like number of LOC (lines of code), variables in a component or imported components. However, it has been observed that there is still a need for metrics in the early phase of the development [12]. The primary users of our metrics are anticipated to be managers, who would be able to analyse the project in its initial stage. Obtained information would then serve as a benchmark of how the current approach influences the specification statistics and,

ideally, allow gathering experience on process of developing such specification. The development team could also benefit from metrics, which could be present already when creating a specification and assist developers with specification's analysis later on. To the best of our knowledge only direct statistics, like numbers of LOC, automatic and interactive proofs, invariants, theorems, refinement steps and the like, are gathered. It would be valuable to provide more elaborated metrics that would help specification analysis and assessment and in the future possibly deal with more involved characteristics, e.g. complexity of the specification or effort prediction.

Event-B is a formal method and a specification language, which is used for system-level modelling and analysis [13]. It is supported by a tool called Rodin Platform, which is an Integrated Development Environment based on Eclipse framework. An Event-B specification consists of a *machine* and its *context* that depict the dynamic and the static part of the specification, respectively. The dynamic model defines the state variables, as well as the operations on these. The context, on the other hand, contains the sets and constants of the model with their properties and is accessed by the machine through the SEES relationship [14]. At this point of our research on metrics we concentrate on the machine only, because we consider machine measurements as a first step towards establishing global Event-B specification metrics.

Since Rodin Platform enables users to obtain LaTeX version of the Event-B specifications, herein machines, we can process each of such files in an automated way. We have implemented a script [15], which parses each of the machine LaTeX file and gathers the statistics about its contents. We perform data collection from the *variables* and *events* sections of the machine, as these are crucial for obtaining the information for our metrics. When the metrics are fully developed, they will be incorporated in a Rodin metrics plugin.

In our work we focus on the syntactical properties of the specification at the source code level. We benefit from the existing code metrics and incorporate them to our early stage development measurements. We derive from Halstead's metrics [16], which describe a program as a collection of tokens that can be classified as either operators or operands. These metrics are used to determine a quantitative measure, e.g. program length, vocabulary size, program volume or complexity directly from the source code [17]. Empirical studies show that standard Halstead metric is a good estimate for program length, provided that the data needed for the equations are available. This means that the program should be (almost) completed, which seems to be a downside of this metric that we are aware of. However, it can be useful in gathering the information about a size of a program after the coding process as well. It should also be mentioned that there have been strong debates and criticism on the methodology and the derivations of equations [18]. We think that with certain degree of modification this metric is worth conducting experiments and could demonstrate to be useful in the domain of (formal) specifications. To our knowledge, no experimentation with Halstead metrics for formal specifications has been done yet.

We use the objectives of the Halstead model and carefully adjust them to Event-B language specifics. Our motivation is that the Event-B syntax is appropriate to be experimented with in terms of Halstead metrics, as it contains all primitives needed for further computations. We believe that experimenting with syntactical metrics

originating from a programming language will occur to be successful in Event-B case and the results of this investigation will be meaningful.

Our paper is structured as follows. In Section 2, we describe our concept of metrics for Event-B machines. Then we illustrate it with an example in Section 3. We conclude and present plans for the continuation our work in Section 4.

2 Metrics for Event-B specification

We derive our metric for Event-B machines from the Halstead model [16], which we now shortly depict. The model is constructed based on a collection of tokens: operators and operands, where four primitive measures can be distinguished:

n_1 – number of distinct operators in a program,
n_2 - number of distinct operands in a program,
N_1 – number of operator occurrences,
N_2 - number of operand occurrences.

These primitive measures are a foundation of Halstead model, which consists of equations expressing the vocabulary, overall program length, potential minimum volume for an algorithm and the difficulty level, which indicates software complexity. Moreover, program difficulty, as well as other features like development effort can be specified with given primitive measures. One needs to take into account that accuracy and behaviour of this model varied between its application domains.

At this point of our research we consider only the machine part of the Event-B specification, focusing on variables and events blocks. Although we presume that refinement has an impact on the qualitative and quantitative aspects of the specification, currently, for the simplicity reasons, we do not take it into consideration. It is planned as a part of our future work.

In order to be able to adjust Halstead model to Event-B environment we have to make several assumptions. Firstly, we decide upon meaningfulness of operators [19] [20]. As an operator we consider unary operators, binary operators except functions, range operator (symbol **..**), forward composition (symbol **;**), parallel product (symbol ||) and direct product (symbol ⊗). We also consider quantifiers, except separators for set comprehension and bounded qualification (symbols **|** and **.** respectively), to be operators in our model. The full list of language operators can be found in the language description [21]. We gather the data about the number of distinct operators (n_1), as well as the number of their occurrences (N_1).

Secondly, we determine the operands, which we chose to be witnesses and variables [21]. We count the number of unique operands (n_2) and the number of their appearances (N_2) respecting their visibility rules. If a witness name appears in several events in a single machine, each of such occurrences is distinct due to the scope. Witnesses are visible only inside a single event, whereas variables are global for the machine.

Having defined primitive measures, we identify several metrics for specification's machine measurements and list them after their description. It is worth mentioning that these metrics do not depend on text formatting, like in a case of using LOC as a machine size metric. They are more credible, as the primitive measures are clearly

defined. For comparison in case of LOC it might not be obvious whether to include comments or data definitions to the size computations [22] [23], unless well defined.

We find the metrics of Halstead [16] suitable for our case. The size of a machine's *vocabulary* (*n)* is defined as a sum of distinct operators and operands (1). A machine's *size* (*N)* is a sum of operator and operand occurrences (2). Next metric, a machine's *volume* (*V),* represents the information contents of the program, i.e. the size of the code of a machine. The calculation of *V* is based on the number of operations and operands present in the machine (3). Another metric, *difficulty level D,* representing the difficulty experienced during writing a specification, is proportional to the number of distinct operators n_1 and occurrences of operands N_2, and inversely proportional to the number of distinct operands n_2 (4). Other Halstead metrics are anticipated in our future work.

(1) $n = n_1 + n_2$

(2) $N = N_1 + N_2$

(3) $V = N * log_2(n)$

(4) $D = (n_1/2) * (N_2/n_2)$

3 Experimental setup and preliminary results

We apply presented metrics to an abstract machine and its subsequent refinements in order to perform comparative study and evaluation of the obtained results. This enables us to quantitatively assess the machine development from its physical characteristics point of view.

Our methodology was applied to a number of specifications created by Space Systems Finland within the European Project DEPLOY. We performed a single domain experiment, with the same type of the development and the same staff. This made the investigation more credible, as we reduced the number of experiment variables that could skew the overall results.

Table 1**.** SSF BepiColombo Specification, Version 5.0 – measurements

Metric	M00	M01	M02	M03	M04	M05	M06	M07
n_1	2	5	0	3	4	0	5	9
N_1	7	69	0	7	11	0	44	22
n_2	19	48	13	21	21	18	85	31
N_2	89	360	30	104	61	28	419	116
n	21	53	13	24	25	18	90	40
N	96	429	30	111	72	28	463	138
V	422	2457	111	509	334	117	3006	734
D	5	19	-	7	6	-	12	17
LOC	231	758	455	599	538	477	1132	783

Table 1 presents the data obtained from BepiColombo specification Version 5.0,

uploaded by SSF to the BSCW repository on the 2nd May, 2009. The analysis of gathered statistics considers also the number of lines of code (LOC), which we defined as non empty and non-comment LOC. The hyphen in the *D* row indicates that there were only skip events in the corresponding machine, which arithmetically means that the result of the computation was not a number.

There is a strong correlation between the vocabulary size *n* and length *N* of each machine, where *n* is always less or equal than *N*. This is a direct consequence of the *n* and *N* definitions. Moreover, a positive relationship of *n* and *N* concerning other statistics, like number of events (also actions) can be observed. There is a strong correlation between *volume* and formerly mentioned machine size metrics (*N*, *n*, LOC), which is particularly visible when logarithmic scale is used in the diagrams. Therefore *V* could be used as a meaningful size metric.

The *difficulty level* characteristic is proportional to the number of unique operators in the program, so also to *vocabulary size*. There seems to be a correlation between *D* and Interactive Proof Obligations, however it is not strong enough to be a complete basis for a quality prediction model. Therefore, it needs to be further investigated.

The Halstead metrics for Event-B specifications have to be verified on more examples. We will pursue the search for possible relations with other indicators, e.g. number of requirements covered.

4 Conclusions and future work directions

Our metrics for Event-B should provide better understanding of specifications' physical features. They could facilitate early-stage development analysis of an abstract specification and its consecutive refinements. Such metrics could be used as a basis for further, indirect measurements, like predictions of time and human resources necessary for a development.

In our paper we proposed a quantitative approach to assess Event-B specifications. We described several metrics for the machine part of a specification adjusted to the specifics of the Event-B language. Moreover, we have implemented a script for automatic data collection and ran it against existing specification.

As a continuation of our research we plan to check the validity of presented metrics on a bigger number of machines (specifications) and discuss the obtained results with the designers. We want to modify the models of the machine metrics if needed, i.e. when not fully expressing the results or not entirely reflecting the developer's intuition. We also need to create metrics for the context part of the specification. This would enable us to create a set of global metrics, which take under consideration complete Event-B specification, i.e. both context and machine, so that it can be analysed as a whole. We also need to incorporate refinement into our model in order to analyse the accumulated Event-B developments, which can be done e.g. using the Breiman's random forest theory [24].

Since software is becoming larger and more complex nowadays, the metrics we proposed could be used for building a complexity model for Event-B specification. Handling complexity already at the beginning of the project would benefit not only

the design process as such, but also later on, e.g. in programming or maintenance phases. It would also serve as early development data to managers.

Acknowledgements

This work has been done within the EC FP7 Integrated Project *Deploy* (grant no. 214158).

References

1. Goodman P., *Software Metrics: Best Practices for Successful IT Management*. Rothstein Associates Inc. (2004).

2. Gopal A., Krishnan M.S., Mukhopadhyay T., Goldenson D., *Measurement Programs in Software Development: Determinants of Success*, IEEE Transactions on Software Engineering, Vol. 28, No. 9, (2002).

3. Kandt R., *Software Engineering Quality Practices*. Auerbach Publications (2005).

4. Lanza M., R. Marinescu, Ducasse S., *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer (2006).

5. Martin R.C., *OO Design Quality Metrics. An Analysis of Dependencies.*, , (1994).

6. Metrics 1.3.6, http://metrics.sourceforge.net/

7. Robertson J., Robertson S., *Requirements: Made to Measure*, American Programmer, X (1997).

8. Tang A., Tran M.H., Han J., van Vliet H., *Design Reasoning Improves Software Design Quality*, Quality of Software Architectures. Models and Architectures.. Springer, Heidelberg (2008)

9. Hayes I.J., Mahony B.E., *Using Units of Measurement in Formal Specifications*, Formal Aspects of Computing, 7 (1995), pp.329-347.

10. Vinter R., Loomes M., Kornbrot D., *Applying Software Metrics to Formal Specifications: A Cognitive Approach*, IEEE International Symposium on Software Metrics, (1998), pp.216.

11. El Koursi E.M., Mariano G., *Assessment and certification of safety critical software*, Proceedings of the 5th Biannual World Automation Congress, . IEEE, Orlando (2002)

12. Whitty R.W., *Research in Specification Methods*, IEE Colloquium on Software Metrics, (2002).

13. Event-B.org, http://www.event-b.org/index.html

14. Abrial J.R., *The B-Book: Assigning Programs to Meanings*. Cambridge University Press (1996).

15. http://valhalla.cs.abo.fi/~mplaska/btexcount.rb,

16. Halstead M.H., *Elements of Software Science*. Elsevier North Holland (1977), pp.128.

17. Kan S., *Metrics and Models in Software Quality Engineering*. Addison-Wesley (2003).

18. Hamer P.G., Frewin G., *M. H. Halstead's Software Science - A Critical Examination*, Proceedings, 6th International Conference on Software Engineering, ICSE. IEEE, Tokyo (1982)

19. Jorgensen M., *Software Quality Measurement*, Advances in Engineering Software, 30 (1999), pp.907-912.

20. *User Manual of the RODIN Platform, Version 2.3*. (2007).

21. Metayer C., Abrial J.R., Voisin L., *Event-B Language, RODIN Deliverable 3.2 (D7)*. (2005)

22. Fenton N., Pflegger S., *Software Metrics. A Rigorous and Practical Approach.*. PWS Publishing Company (1997).

23. Fenton N., Neil M., *Software metrics: roadmap*, Conference on the Future of Software Engineering., Limerick, Ireland (2000)

24. Breiman L., *Random Forests*, Machine Learning, Volume 45, Number 1 / October 2001 (2001), pp.5-32.