Specification Metrics for Event-B Developments

Marta (Pląska) Olszewska[1,2], *Kaisa Sere*[1]

[1] Department of Information Technology
Åbo Akademi University
Joukahaisenkatu 3-5
FIN-20520 Turku
{marta.plaska, kaisa.sere}@abo.fi

[2] TUCS - Turku Centre For Computer Science
Joukahaisenkatu 3-5
FIN-20520 Turku

Abstract

In this paper we define several metrics for Event-B specifications in order to assess the maintainability of a rigorous system in its early development stage. We perform measurements of physical features of the specification. Moreover, we derive metrics like difficulty of constructing a specification or effort needed for its creation. The specification is investigated in the perspective of its syntactical characteristics. We base our metrics on the Event-B language primitives, namely operators and operands that we regard meaningful for our measurement model. We also use statistics about proof obligations in our metrics, as we consider the proving activity as a vital element in the process of creating a specification.

Presented metrics are applied to a number of Event-B specifications, both their dynamic and static parts. They are examined in order to empirically confirm appropriateness for management purposes. Obtained results are analysed in a perspective of an abstract specification and its consecutive refinements.

1 Introduction

Measurements for software systems and their development process are nowadays considered as a good practice in the computer world [Goo04] [AGo02]. They are a part of software projects in order to assure certain quality [KRK05] or for the improvement purposes. They have been evolved for many years, extended for particular development methods like Object-Oriented programming [Lan06] [Mar94] or specialised to meet the needs of certain programming languages, like Java Programming Language [Fra09]. Quality measurements are done not only at the end-product stage, but much earlier, for the requirements [Rob97] or (formal) modelling of the system [Tan08]. An early quality assessment has a major influence on the final product, as a thorough control over the whole development process is maintained.

Research on metrics for formal specifications has already been done for the Z language. In [Hay95] authors perform a static analysis of Z specification notation, whereas in [Vin98] the focus is on the linguistic properties of the notation and predicting erroneous parts of specifications created in Z. An assessment for the B language [ElK02], in

which 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, has also been done. However, it has been observed that there is still a need for metrics in the early phase of the development [Whi02].

To the best of our knowledge only direct statistics, like number of LOC, automatic and interactive proofs, invariants, theorems, refinement steps and the like, can be recorded for Event-B in a straightforward way. We believe that providing more elaborated metrics will facilitate a thorough specification analysis and assessment. Furthermore, it will support the improvement of the modelling strategy and possibly enable effort prediction. We anticipate the primary users of our metrics to be managers, who can monitor the project in its initial stage. Obtained information will be a benchmark of how the current development approach influences the specification and, ideally, allow gathering experience on improving the process of creating such specifications. We expect that metrics will be present already when creating a specification and assist developers with modelling and analysis later on.

Event-B is a formal method and a specification language, which is used for system-level modelling and analysis [Eve10]. An Event-B specification consists of a *machine* and its *context* that depict the dynamic and the static part of the specification, respectively [Abr96]. The language is supported by a tool called Rodin Platform [Rod10], which is an Integrated Development Environment based on Eclipse framework. Since the Rodin Platform is a "rich client platform", it enabled us to extend the core with measurement plug-in for automated data collection.

In our work we focus on the syntactical properties of the specification. We benefit from the existing metrics and incorporate them to our early stage development measurements. We derive from Halstead's metrics [Hal77], 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 or program volume. Empirical studies show that standard Halstead metric is a good indicator for program length, provided that the data needed for the equations are available [Kan03]. This means that the program, or in our case specification, 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 specification during the modelling process. It should also be mentioned that we are aware of the strong debates and criticism on the methodology and the derivations of equations [Ham82]. In our research we adapt the general ideas of Halsted to a more abstract, higher-level setting, i.e. formal specifications. Moreover we do not use any estimates, instead we focus on adjusting concrete metrics by including factors specific for formal development. To our knowledge, no experimentation with Halstead metrics for 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 investigated in terms of Halstead metrics, as it contains all the primitives needed for further

computations. These measures should be considered for use during the modelling as outliners of the complexity and development trends. Moreover, the generic version of Halstead metrics, although criticised, is simple and well known to the computer society, as it has been used for almost forty years. We believe that experimenting with the syntactical metrics originating from a programming language will 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 the Event-B language for machine and context. Section 3 depicts the reasoning of the syntax based measurements and presents our concept of metrics for Event-B specifications. In Section 4 we shortly discuss the validity of the given metrics. Section 5 illustrates our research with a large real-life example and analyses the measurement results. We conclude and present plans for the future work directions in Section 6.

2 Event-B characteristics

Event-B [Abr961] is an extension of the B Method formalism [Abr96] and is used for modelling of event-based systems. An Event-B development incorporates the Action Systems formalism [Bac90] [BSe96], which allow the system to be constructed gradually and taking into account refinement rules. Stepwise refinement helps to tackle all the implementation issues by decomposing the problems to be specified and introducing details of the system to the specification in a stepwise manner. In the refinement process an abstract specification *A* is transformed into a more concrete and deterministic system *C* that preserves the functionality of *A*. In order to be able to ensure the correctness of the system, the abstract model should be consistent and feasible.

A specification of a system consists of behavioural part, described by a machine, whereas the data structures are given in a context. The machine 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 [Abr96].

An Event-B machine consists of its *name*, list of distinct variables *var*, the invariants *Inv(var)*, and a collection of events evt_i, including INITIALISATION, and is depicted with Fig.1.

MACHINE *Machine_0*
SEES *Context_0*
VARIABLES *var*
INVARIANTS *Inv(var)*
INITIALISATION *Init*
EVENTS
evt_1
…
evt_N
END

Fig. 1: General form of machine in Event-B specification

All the variables *var* are declared in the VARIABLES clause and initiated in the INITIALISATION clause. Furthermore, they are strongly typed by constraining predicates of invariants given in the INVARIANT, which might also contain properties that should be maintained by the system. The operations on the variables are given as events of the form ***WHEN** guard **THEN** substitution **END***. In case the event is parameterised it is given as ***ANY** witness **WHERE** guard **THEN** substitution **END***, where *witness* is a local variable visible within an event. *Guard* represents a state predicate, whereas a *substitution* is a B statement describing how the event affects the program state and is given in the form of deterministic or nondeterministic assignment over the system variables [Use07].

Event-B classifies events as 'convergent' if they are new events that are expected to eventually relinquish control to an old (refined) event (i.e. it must decrease the variant). Events that are not convergent are classified as 'ordinary' (default). A third classification, 'anticipated', refers to events that will be shown to be convergent in a future refinement. Events can also be categorised as 'extended' or not. In our research we consider all of the classifications, however the 'extended' type is only partially supported with metrics, e.g. when a machine includes it among other types of events.

Context part of Event-B specification is associated with machine by **SEES** relationship. A context is made of its *name*, list of distinct carrier sets *sets*, list of distinct constants *const*, and list of its properties given by axioms *axm* and theorems *th*, which is exemplified in Fig.2. The sets and constants of an abstract context are kept in its refinement, via 'extend' mechanism, therefore they are accumulated.

CONTEXT *Context_1*
EXTENDS *Context_0*
SETS *sets*
CONSTANTS *const*
AXIOMS *axm*
THEOREMS *th*

Fig. 2: General form of context in Event-B specification

3 Metrics for Event-B specification

We derive our metrics for Event-B machines from the Halstead model [Hal77]. The model is based on a collection of tokens, operators and operands, where four primitive measures can be distinguished:

- n_2 - 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 measures are a foundation of Halstead model, which consists of equations expressing the vocabulary, the overall program length, the potential minimum volume for

an algorithm and the difficulty level that indicates software complexity. Moreover, features like the development effort can be specified. One should keep in mind that the accuracy and the behaviour of this model varied between its application domains.

Our tool for automatic data collection deals with the variables, invariants and event blocks of a current machine, and collects the data for each of the machines separately without recognising the semantic relations between the machines. This means that the 'extended' type of event refinement is not considered for the machines, yet. The refinement and extension method is fully supported for the context part of the Event-B specification. Contexts are handled accumulatively, i.e. we acknowledge that considered context is an extension of the previous one. Therefore, the data is collected recursively, with respect to the occurrence of certain set or variable in current context and its presence in previous one. Although we realise that refinement has an impact on the qualitative and quantitative aspects of the specification, currently we consider it in a limited way. Additional issues concerning the metrics acknowledging refinement are given in Section 5 as a part of our future work.

In order to be able to adjust the Halstead model to Event-B environment we have to make several assumptions considering the primitives. Firstly, we decide upon the meaningfulness of operators [Jor99] [Use07] both for the machine and context. 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 [Met05]. 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 [Met05]. In the machine part 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 is declared 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. For the context part of the specification we consider sets and constants as operands. We depict counting of the operators and operands with an excerpt of a simple Event-B code [Abr10] presented in Listing 1 and 2.

Listing 1. Example of Event-B machine

```
MACHINE part_1
REFINES part_0
SEES    part_ctx
VARIABLES j, h, l
INVARIANTS
     inv1   :      j ∈ 0‥n
     inv2   :      h ∈ 1‥n ↣ ℕ
     inv3   :      l ∈ 0‥j
     inv4   :      ran(h) = ran(f)
     inv5   :      ∀m·m∈1‥l ⇒ h(m)≤x
     inv6   :      ∀m·m∈l+1‥j ⇒ x<h(m)
```

```
EVENTS

     INITIALISATION   ≙
BEGIN
        act3   :      j ≔ 0
        act4   :      h ≔ f
        act5   :      l ≔ 0
END

     progress 2   ≙
WHEN
        grd1   :      j≠n
        grd2   :      h(j+1)≤x
        grd3   :      l=j
THEN
        act1   :      l ≔ l+1
        act2   :      j ≔ j+1
END

     progress_3   ≙
WHEN
        grd1   :      j≠n
        grd2   :      h(j+1)≤x
        grd3   :      l≠j
THEN
        act1   :      l ≔ l+1
        act2   :      j ≔ j+1
        act3   :      h ≔ h ⩤ {l+1 ↦ h(j+1)} ⩤ {j+1 ↦ h(l+1)}
END
END
```

Listing 2. Example of Event-B context

```
CONTEXT
     part ctx
CONSTANTS
     n
     f
     x
AXIOMS
     axm1   :      n ∈ ℕ
     axm2   :      f ∈ 1‥n ⇸ ℕ
     axm3   :      x ∈ ℕ
END
```

In Listing 1 we show an extract of the Event-B machine, whereas in Listing 2 we present the code of the context seen by this machine. The machine given in Listing 1 contains 8 unique operators and 25 operator occurrences, as well as 3 distinct operands and 42 operand occurrences. Sample context given in Listing 2 consists of 1 unique operator, which is repeated 3 times, in addition to 3 operands that appear in the code 7 times in total.

Having defined primitive measures, we identify several metrics for machine and context parts of a specification 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 specification 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 [Fen97] [FeN00].

We define the size of a *vocabulary* (*n)* of a specification (machine or context) is defined as a sum of distinct operators and operands (1). A sum of operator and operand occurrences (2) is a *size* of a specification (*N)*. Next metric, *volume* (*V)*, represents the information contents of the program. The calculation of *V* is based on the number of operations and operands present in the machine (3).

1. $n = n_1 + n_2$
2. $N = N_1 + N_2$
3. $V = N * log_2(n)$

We have done numerous experiments by applying this metrics to Event-B specifications. The results confirmed a strong correlation between the size metrics, explicitly the vocabulary size *n* and length *N* of each machine. This is a direct consequence of the *n* and *N* definitions, which implies that *n* is always less or equal than *N*. There is also a correlation between *volume* and the machine size metrics (*N*, *n*, LOC), which is particularly visible when the logarithmic scale is used in the diagrams. Therefore *V* could be used as a meaningful size metric, as long as there exist at least one operator or operand ($n>0$). Size metrics are proportional to the direct Event-B machine statistics, such as the number of events or the number of actions. Moreover, they are proportional to the number of constants and axioms in a context.

Another metric, *difficulty level D* (4), 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. $D = (n_1/2) * (N_2/n_2)$

One should note that in practice, since there is a possibility that no operators are used in a machine (empty events with skip) or in a context (either no axioms or theorems are present or sets and constants are given without their properties), the result of D after computation could be undefined.

Construction of a specification is influenced by the problem that needs to be modelled, the clarity and completeness of requirements, the skill of the modeller, as well as the number of interactive and automatic proof obligations (IPO and PO). The difficulty model, which we presented, is suitable for Event-B specifications, as it correlates with the number of IPOs, or in case there are no IPO, with the number of POs.

We also experimented with the Halstead effort general equation ($E=V*D$, where *V* and *D* stand for volume and difficulty, respectively), straightforwardly applying it to Event-B environment, but no regularity or pattern could be observed. Therefore we approached the effort characteristic from the point of view of practitioners. Constructing a specification consists of several factors, such as modelling and proving activities, and is influenced not only by quality of requirements, but also by the power of proving mechanisms and experience of the modeller (also in the interactive proving tasks). We define effort as dependant on the number of proof obligations (automatic and interactive), the machine's volume *V* and difficulty *D* (5).

5. $E = (1+IPO/PO) * V * D$

In the equation the proving factor (1+*IPO*/*PO*) is placed to deal with a case when all proof obligations are discharged automatically. Two special cases must be considered. When there are no proof obligations, the proving ratio is treated as zero and the formula (5) is reduced to $E=V*D$. If D is equal to zero or is not a number, the formula (5) should be computed without the difficulty factor. It is worth mentioning that although the Rodin tool might not display any POs, there always exist some trivial POs generated and proved by the platform provers. Such cases do not involve any proving effort. However, the modelling effort is still regarded in the metric. One needs to observe that the lower the value of E, the easier and simpler is to change and improve a specification or maintain it.

The effort characteristic has no established threshold, yet. However, it can be used by developers as a source of information for comparison of how much effort is necessary when using different modelling approaches for a given problem. It can also assist when building the experience of the developers on modelling. It will also help to approximate the effort needed to tackle problems similar to the ones that have already been dealt with. As it is computed in parallel with the specification creation process, it can be checked and decided if it is worth to maintain the chosen modelling solution.

4 Discussion on the metrics validity

Here we concisely apply the Evaluation Framework presented in [Kan04] to our metrics. The main purpose of our measurement is to assist in a self-assessment and improvement of the development process in its early stage. One of our goals is to provide a method for evaluation of the Event-B specifications to the developers and managers. The metrics are intended for the design stage of the development and can assist in the analysis and quantitative assessment of specifications with respect to refinement mechanisms. They are intended to be applied throughout the system modelling process.

Our generic objective is to determine the size of an Event-B specification, the difficulty of modelling it, as well as the effort considering its construction and proving. The variability of these metrics is inherent from the direct measurements that we automatically collect, i.e. number of operators and operands, in addition to their occurrences. The tool that we use to count the direct measurements determines the natural variability of readings and reduces the possibility of measurement errors. As for the foreseeable side effect of using our metric instrument, we consider the lack of existing thresholds as the biggest drawback.

We also validate our metrics against several mathematical properties, such as the non-negativity, null value and module additivity. The size measures given in this paper, such as size of the vocabulary n, length N, as well as the direct measurements considering the unique operators and operands and their occurrences have been validated and proved already in [Bri96] as correct size measures. They are measurable on a ratio scale, as they preserve ordering, size of intervals and ratios between entities. They have a zero

element that represents the total lack of attribute and is a starting element in the measurement. Moreover, arithmetic operations can be meaningfully applied. The volume *V* does not fulfil the additivity of disjoint modules with respect to [Bri96]. Assuming that V=0 for an empty specification, we consider the metric as a ratio scale measurement.

The difficulty level *D* is an ordinal scale measurement, since its values can only be compared according to the ranking it represents. The same applies to the effort metric *E*, since it contains the measurements of ordinal scale type. Both metrics fulfil the non-negativity property.

5 Experimental setup and results

Our methodology was applied to a number of specifications created within the European Project DEPLOY. The specifications are modelling subsets of a highly critical, large-scale and complex industrial system within the space sector. We performed a single domain experiment, with the same type of the development and the same workforce. This made the investigation more credible, as we reduced the number of experiment variables that could skew the overall outcome. The results given by the metrics correspond to the perception of the practitioners.

Here we present a comparative study and an evaluation of our metrics applied to an abstract specification and its subsequent refinements. This enabled us to quantitatively assess the progress of the development from the point of view of the physical characteristics of the constructed specification. Fig. 3 and Fig. 4 present the measurements, which are obtained from one of the software specifications provided by DEPLOY's industrial partner, for the machines and contexts, respectively.

In Fig. 3 we can observe that the displayed statistics correlate in most of the machines. The lack of the line for IPO in machines M0, M1, M2 and M5 denotes that the value of IPO is zero, meaning that all of the proofs have been discharged automatically. It cannot be shown on the diagram because of the logarithmic scale used in the figure.

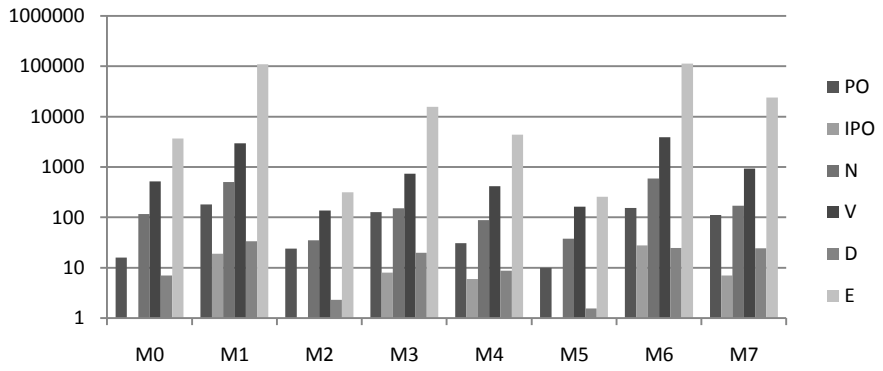

Fig. 3. Statistics on consecutive machines

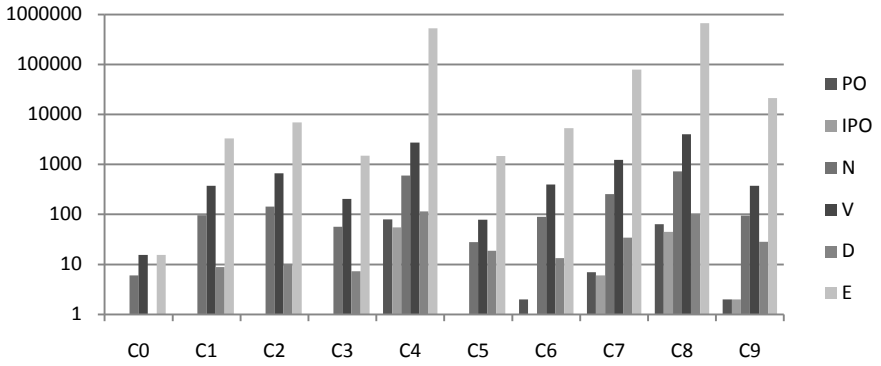

Fig. 4. Measurements on successive contexts

The size measurements *N* and *V* correlate in each refinement step and are proportional to well defined number of LOC (non empty and non comment) or other direct size measures, like previously used size measure – number of specification pages. This analysis was done in order to validate our results against understanding of specification size by developers.

The difficulty level *D* correlates with the number of IPO, and when there are no interactive proves, it correlates with the number of automatically generated proof obligations PO. Consequently, it confirms the perception that the difficulty in creating a machine depends not only on its size (by definition), but also on the proving factor. The effort *E* is a composite of proving, difficulty level and the information contents of the machine, which is visible as correlation of the listed features in the diagram.

Similar analysis can be performed for Fig. 4 presenting the measurements for consecutive contexts of examined specification. As with Fig. 3, the value for missing data is zero, which cannot be represented on a logarithmic scale used in the diagram.

Having the measurements for each of the artefacts at every refinement step we are able to reason about the progress of the development at the early stages. In case the calculated numbers are increasing excessively at certain step, the developers might decide to change their modelling approach and decompose their problem into several smaller ones. This way the product metrics relate to improving the process of system specification.

We are interested in probing the presented metrics for Event-B specifications in different domains, e.g. military or transportation. We will pursue the search for relations between our metrics and other indicators, e.g. effort results and actual man-hours required to create a specification.

5 Conclusions and future work directions

Nowadays (software) systems keep growing in size and complexity. Therefore, moni-

toring complexity already at the beginning of the project benefits not only the design process as such, but also impacts later development phases, e.g. programming or system maintenance. There is a need for metrics, which would assist in controlling complexity at the early stages of the development.

In this paper we described a quantitative approach to assess the dynamic and static parts of Event-B specifications. The proposed measures are defined within the measurement scales and mathematically validated. Moreover, they have been discussed with the practitioners and, as a result, confirmed the perception of the developers. Our metrics for Event-B provide a better understanding of the physical features of specifications. They facilitate an assessment of an early-stage development. By analysing an abstract specification and its consecutive refinements, they support the modelling process. The presented metrics can be used as a basis for further measurements, like predictions of time and human resources necessary for a development. Therefore they can be treated as an early development indicator for managerial purposes.

We consider refinement mechanism as a vital part of the system design. Our metrics for the context part of a specification with 'extend' mechanism already cover the refinement issue. The data for the measurements are collected and computed accumulatively with respect to the previous refinement steps. However, there is a need to expand our metrics considering machines. For now we focus on the machine refinement, where the events are not being an extension of the previous refinement step events. Therefore each machine is handled separately in our measurement computations. We need to elaborate our approach to be able to fully support the 'extended' type of event refinement in machines. In this situation we will be able to analyse the accumulated Event-B machines with respect to all refinement types throughout the modelling process.

We are aware that presented measures may not suffice by themselves as specification metrics and that there is still a need for metrics that encompass the semantic relations within a specification. However, we believe that results of our research are a good starting point for the further investigation in the field of specification assessment.

As a continuation of our investigation we plan to examine the presented metrics on specifications from other critical domains. Our metrics can naturally be customised to fully reflect the intuition of the developers in other fields. Moreover, we intend to create a set of global metrics, which will consider a complete Event-B refinement process, i.e. the machines and the related contexts for each refinement step.

Acknowledgements

This work has been done within the EC FP7 Integrated Project *Deploy* (grant no. 214158). Authors would like to thank Dr Linas Laibinis and Anton Tarasyuk for the valuable insight on the Event-B language and modelling, and Mikołaj Olszewski for his programming involvement.

References

[Abr10] Example by J-R. Abrial, http://valhalla.cs.abo.fi/~mplaska/QuickSort.zip

[Abr96] Abrial J-R, *The B-Book: Assigning Programs to Meanings*. Cambridge University Press (1996).

[Abr961] Abrial J-R, *Extending B without Changing it (for Developing Distributed Systems)*, Proceesings of 1st Conference on the B Method. Springer-Verlag, Nantes (1996)

[AGo02] Gopal A, Krishnan MS, Mukhopadhyay T, Goldenson DR, *Measurement Programs in Software Development: Determinants of Success*, IEEE Transactions on Software Engineering, Vol. 28, No. 9, (2002).

[Bac90] Back RJR, *Refinement Calculus, Part II: Parallel and reactive programs. Stepwise Refinement of Distributed Systems*. Springer-Verlag (1990), pp.67-93.

[Bri96] Briand Lionel, Morasca Sandro, Basili Victor, *Property-Based Software Engineering Measurement*, IEEE Transactions on Software Engineering, (1996), pp.68-86.

[BSe96] Back RJR, Sere K, *Superposition refinement of reactive systems*, Formal Aspects of Computing, 8(3) (1996), pp.1-23.

[ElK02] El Koursi EM, Mariano G, *Assessment and certification of safety critical software*, Proceedings of the 5th Biannual World Automation Congress. IEEE, Orlando (2002)

[Eve10] Event-B.org, http://www.event-b.org/index.html

[FeN00] Fenton Norman, Neil Martin, *Software metrics: roadmap*, Conference on the Future of Software Engineering. Limerick, Ireland (2000)

[Fen97] Fenton NE, Pflegger SL, *Software Metrics. A Rigorous and Practical Approach.*. PWS Publishing Company (1997).

[Fra09] Metrics 1.3.6, http://metrics.sourceforge.net/

[Goo04] Goodman P, *Software Metrics: Best Practices for Successful IT Management*. Rothstein Associates Inc. (2004).

[Hal77] Halstead MH, *Elements of Software Science*. Elsevier North Holland (1977), pp.128.

[Ham82] Hamer PG, Frewin GD, *M. H. Halstead's Software Science - A Critical Examination*, Proceedings, 6th International Conference on Software Engineering, ICSE. IEEE, Tokyo (1982)

[Hay95] Hayes IJ, Mahony BE, *Using Units of Measurement in Formal Specifications*, Formal Aspects of Computing, 7 (1995), pp.329-347.

[Jor99] Jorgensen M, *Software Quality Measurement*, Advances in Engineering Software, 30 (1999), pp.907-912.

[Kan03] Kan SH, *Metrics and Models in Software Quality Engineering*. Addison-Wesley (2003).

[Kan04] Kaner Cem, Bond Walter, *Software Engineering Metrics: What Do They Measure and How Do We Know?*, 10th International Software Metrics Symposium. IEEE Computer Society Press, Chicago (2004)

[KRK05] Kandt RK, *Software Engineering Quality Practices.* Auerbach Publications (2005).

[Lan06] Lanza M, Marinescu R, Ducasse S, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer (2006).

[Mar94] Martin RC, *OO Design Quality Metrics. An Analysis of Dependencies.*, (1994).

[Met05] Metayer C, Abrial J-R, Voisin L, *Event-B Language, RODIN Deliverable 3.2 (D7)*. (2005)

[Rob97] Robertson J, Robertson S, *Requirements: Made to Measure*, American Programmer, X (1997).

[Rod10] Rodin Platform, http://www.event-b.org/platform.html

[Tan08] Tang A, Tran MH, Han J, van Vliet H, *Design Reasoning Improves Software Design Quality*, Quality of Software Architectures. Models and Architectures. Springer, Heidelberg (2008)

[Use07] *User Manual of the RODIN Platform, Version 2.3*. (2007).

[Vin98] Vinter R, Loomes M, Kornbrot D, *Applying Software Metrics to Formal Specifications: A Cognitive Approach*, IEEE International Symposium on Software Metrics, (1998), pp.216.

[Whi02] Whitty RW, *Research in Specification Methods*, IEE Colloquium on Software Metrics, (2002).