On the Applicability of Predictive Maintainability Models onto dynamic Languages

Miguel Lopez[1], Naji Habra[2], Grégory Seront[1]

[1] *CETIC asbl*
Rue Clément Ader, 8
B-6041 Gosselies, Belgium
malm@cetic.be, , gs@cetic.be

[2] Faculty of Computer Science
Namur University – FUNDP
Namur, Belgium
nha@info.fundp.ac.be

Abstract

Growing maintenance costs have become a major concern for developers and users of software systems. Predictive models of maintainability have been developed to reduce such costs. Moreover, predictive models based on internal measurements represent a low-cost means to forecast the maintenance costs and effort.
However, these measurements are not applicable to dynamic languages. Indeed, such languages do not declare the types of the variables. In that context, usual internal measurement and the related predictive models are not relevant.
Current paper analyzes the applicability of predictive maintainability models on dynamic languages, and proposes briefly some ideas of solutions.

Keywords: predictive models, maintainability, dynamic languages, software measurement, metrology, semantic analysis.

1. Introduction

Today, the most important fraction of the IT budget is allocated to the maintenance process. Indeed, 60% of the IT costs are in the maintenance [14],[18]. Regarding this data, a goal to achieve would be to decrease and better control the maintenance costs.

Predictive maintainability models represent a strongly investigated mean to achieve such a goal [1],[9],[11],[13],[15],[20]. The main idea of such methods is to forecast the maintenance effort as early as possible in the product lifecycle.
With such predictions, a suitable planning can be done, and therefore, a better control and a decrease of the costs can be possible. Moreover, predictive models can be used as well qs to explore different possibilities and to choose the less costly in terms of predicted maintenance.

In the literature, many predictive maintainability models can be found [1],[11],[15]. However, and in order to be as clear as possible, two kinds of models can be distinguished in the state of the art.

On the one hand, some models are based on the measurements of attributes like bugs density, effort to modify a program, or to understand it. Such measurements are often called external measurement [6]. In fact, "external" means two different things:

- measuring necessitates to execute the code, *i.e.* bugs density
- the measure is related to the maintenance process, *i.e.* effort to modify.

On the other hand, predictive models can also be based on internal measurements. Internal means that it is not necessary to execute the code to compute the measurement result. For instance, the McCabe cyclomatic number [15] is an internal measurement.

The current paper will only consider the second type of model, that is, predictive models which take internal measures as input.

Obviously, since most of the internal measurements are automatic (done by a software tool), they represent a low-cost means to predict maintainability. This main characteristic represents the rationale behind using such models in order to reduce IT costs.

Nevertheless, internal measurements present important limitations with respect to their applicability to some languages.
Indeed, most of the internal measures have been developed for programming languages like Java or C++. These languages hold an important property that eases the use of these measurements. They are statically typed.
The type of each variable is declared within the code. So, to determine the type of a given object/variable used in a given program some parsing would be needed; furthermore determining the types remains an achievable objective.
Moreover, some internal measurements need to rigorously know the types within a program, *e.g.* coupling. And, the predictive maintainability models often use such measurements.

So, it means that if a given language (e.g., Python, Smalltalk, VB…) does not declare their types, such measurements and their related models are not applicable. These languages called dynamic - because the types are defined at the run-time – do not benefit from the internal measurement, and therefore from the predictive maintainability models

The current paper aims at understanding the limitations within dynamic languages of internal measurement-based predictive models. We highlight the issues due to the dynamic typing of such languages, and give some ideas of solution.

2. Dynamic Languages Issues

Predictive maintenance models based on internal measurement are applicable for static languages like C++, Ada or Java. Indeed, most of the measurements used in such predictive models, e.g. coupling cohesion, need to formally identify the types within the program . In this case, types identification is possible by only parsing the source code.
The coupling measurement is widely used in predictive models. Indeed, measurements like fan out, fan in, or coupling between objects (CBO) represent useful measurement methods in order to predict maintainability.
There are different definitions of the coupling attribute. Roughly speaking coupling is the amount of connections between modules or classes for the object-oriented paradigm. According to Stevens *et al.*, *the measure of the strength of association established by a connection from one module to another*. [21]
Another example is the fan-out attribute which can be defined as follows: *the fan-out of a module M is the number of local flows that emanate from M, plus the number of data structures that are updated by M.*[11]. In other words, the fan-out measurement counts the number of classes or modules used by a given class or module.
However, to compute such a measurement, it is necessary to previously identify the types of each object. Moreover, this type identification must be done by parsing the source code, since the measurement is internal.
The same can be said for the other coupling measurements. So, type identification is crucial for predictive maintenance models.

Another example of internal measurement that needs types identification is cohesion. Module cohesion was introduced by Yourdon and Constantine as *how tightly bound or related the internal elements of a module are to one anothe.r* [22].
An important measurement for cohesion is the set of lack of cohesion, that is, LCOM. Three authors have suggested a definition of LCOM [5], [9],[15].
According to [1], these measurements seem to be closely related to reusability of the source code. In that sense, the reusability quality is relevant within a predictive maintenance model.
To be correctly computed, each of the measurements proposed for cohesion need to identify the types of the variables. Indeed, the type of each variable within the body of a method or function must be determined in order to clarify whether the given class or module uses all the variables. In other words, it is necessary to know the type of all variables within a method in order to compute the cohesion measure.

In that context, dynamic languages represent an interesting and important issue for such predictive models.
Indeed, dynamic languages do not declare the types of their variables. So, internal measurement is strongly impoverished.
Moreover, and since internal measurement is mostly based on static analysis of the source code, measurement like cohesion and coupling are difficult to apply onto such languages.
Therefore, predictive models of maintenance based on internal measurement are hardly relevant for dynamic languages.
A static identification of the types seems to be impossible in those languages. Only the name of the

variables is given by the code, but not their type, not even their name. So, determining the types is made up of uncertainty. In other words, only analyzing the syntax of the source code written in a dynamic language is not enough to collect information about the types.

It is clear that there is a strong need for defining internal measurement applicable to dynamically typed languages. Since internal measures represent a key point of predictive models of maintenance, today, dynamic languages do not benefit of such useful models.

Nevertheless, some applicable solutions exist, even if they are still unused in the industry or the academic world.

3. Some Ideas of Solutions

3.1. Calibration

A first idea of solution can be found in the metrology. In fact the application of suitable metrology concepts such as calibration and error would represent a low-cost means to correct such measurement.

Indeed, since the errors related to dynamic typing arise thoroughly during measurement, the adjustment could be rigorously the same for each execution of the measure.

In that sense, such kinds of errors are well known within the metrology field. They correspond to the well-known concept *systematic errors*, which occur each time the measurement is executed, regardless the conditions.

According to [7], systematic errors are defined as *a component of error which, in the course of a number of analyses of the same measurand, remains constant or varies in a predictable way. It is independent of the number of measurements made and cannot therefore be reduced by increasing the number of analyses under constant measurement conditions*. The measurand is the quantity to be measured.

Moreover, *the systematic error can be controlled in some cases thanks to its predictability. For example, the cause of such an error can be a misconception of the measuring instrument. In this case, a calibration of the instrument can be enough in order to reduce the effects of a systematic error* [16].

This corresponds exactly with the problem we face. Due to a misconception of the measuring instrument, which is not able to infer the types, an error affects systematically the measurement. And, most of the types within a program are not counted.

Metrology suggests an easy means to handle this predictable error, that is, calibration.

In [8], calibration is defined as *a set of operations that establish, under specified conditions, the relationship between values of quantities indicated by a measuring instrument (or values represented by a material measure) and the corresponding values realized by standards*. The result of the calibration can be considered as a correction of the values indicated by the measuring instrument.

An important term is used within the definition, that is, standard. *A standard or etalon can be a material measure, measuring instrument, reference material or measuring system intended to define, realize, conserve or reproduce a unit or one or more values of a quantity to serve as a reference* [8].

Therefore, in our case, the etalon would be a set of pieces of code that show some relevant examples in regard with the type inference for a given dynamic language.

However, things are not as simple as previously explained. Indeed, the need for an etalon for correctly calibrating remains an important issue, which begin to be discussed [1], [16]. And, even thought this topic is a very interesting problem, it is out of the scope to more detail such concept.

When etalons are clearly defined, the measurement of the standards, *i.e.* examples of code, is manually made. Each standard or piece of code is then linked with the exact value, assuming that the manual measurement is correct.

At this point of the calibration process, the correct values of the standards are known. So, now it is necessary to proceed to the measurement with the measuring tool in order to obtain the values given by the tool.

Now, a pair of values is available for each etalon. So, a linear regression can be computed with these values to get the relationship between the correct values (manually obtained) and the values returned by the measuring tool.

An important assumption has been made to compute the linear regression. Indeed, the relationship between the correct and the erroneous values is linear. This hypothesis is acceptable, if and only if the definition domain of the relationship is so small that the curve is locally linear.

Finally, each time the measurement is executed, the correct value is computed within the linear relationship by using the value given by the measuring tool.

3.2. Improved Semantic Analysis

A second idea of solution can be found in semantic analysis. Enhancing the semantic analysis within measurement tools is another way to correctly compute measurements that need formal type identification.

Now, the question is how can we determine the types in a dynamic language, which does not declare its types?

The main idea of such means is to build a set of rules or heuristics that helps identifying the types of the variables. These rules would be build thanks empirical studies of source codes samples, for which different variables typing cases are available.

An example will highlight this second point. Let the following source code in Python.

```
Class A:
        def methodA(self, attr):
                for k, v in attrs :
                        print v
```

Source Code 1 Python Sample

The question is which is the type of the argument *attrs*? This is an important question to compute coupling.
Since there is a loop *for* exploded into two variables *k* and *v*, it seems that *attrs* is a dictionary, that is, a Python array. Now, the question is which is the type of the element within this array?
It is important to know that in such construction *for k, v in attrs*, the variable *k* represent the keys of the array, and the variable *v* is the values of the elements. So, since the value of the variable *v* is displayed at the last line, the type of the variable *v* can be:

- Number: float, integer
- String

In that case, it is not decidable, and a coefficient of uncertainty must be related to the type. This coefficient could be empirically set up.
Moreover, the print statement can be an error of the programmer, and in that case the content of *v* is a memory address. But, based on a static analysis, it is impossible to detect such method, unless a stronger semantic analysis is done.

3.3. Models Using Other Definition Measurement

A third solution to investigate would be to extent the definition of the internal measurements involved in the predictive maintainability models.
For instance, a « static » definition of the coupling attribute is not relevant for a dynamic language, since coupling depends on the history of execution of the program.In that sense, it could be more useful to define and measure another other attribute related to a more « dynamic » coupling concept.
In other words, the model or meta-model of the new coupling attribute would capture all properties of dynamic languages which affect the software maintainability.
Investigating this solution means that the usual predictive maintainability models should be avoided with dynamic languages, and therefore new internal measurements have to be to redefined.

4. Conclusion

In this paper, we study the applicability of the predictive maintainability models for the dynamic languages.
The paper highlights the problematic use of the internal measurements within dynamic languages. The dynamic typing is showed to be a blocking factor that strongly affects the usability of such models.
However, three solution proposals are briefly described in the paper. Firstly, the application of well-known metrology concepts (e.g., uncertainty, systematic, calibration) is explained in terms of improving the internal measurements. Secondly, it is proposed to investigate and improve semantic analysis in order to infer the types. Thirdly, the paper suggests redefining the internal attributes (coupling, cohesion….) involved in the predictive maintainability models in such a manner that these measurements would be relevant for the dynamic languages.

5. Acknowledgement

This research project is supported by the European Union (ERFD) and the Walloon Region (DGTRE) under the terms defined in the Convention n° EP1A12030000072-130008.

6. References

1. A. Abran and A. Sellami. Measurement and metrology requirements for empirical studies in software engineering. Proceedings of the 10th International Workshop on Software Technology and Engineering Practice, 2002.

2. Victor R. Basili, Lionel Briand and Walcélio L. Melo, "A Validation Of Object-Oriented Design Metrics As Quality Indicators", Technical Report, Univ. of Maryland, Dep. of Computer Science, College Park, MD, 20742 USA. April 1995.

3. Boehm, B. *et al*, "Software Cost Estimation with COCOMO II", Prentice Hall PTR, 2000

4. Briand, L., Bunse, C., Daly, J., "An Experimental Evaluation of Quality Guidelines on the Maintainability of Object-Oriented Design Documents", IESE-Report No. 038.97/E, 1997

5. Chidamber, S., R., Kemerer, C., K., A Metrics Suite for Object Oriented Design, IEEE Trans. on Software Eng., Vol.20, No.6, June 1994.

6. ISO/IEC 9126-1 (2001) “Software Engineering—Product Quality—Part 1: Quality model”. June 2001.

7. ISO. Guide to the expression of uncertainty in measurement., 1993

8. ISO. International vocabulary of basic and general terms in metrology, international organization for standardization, 1993.

9. Henderson-Sellers, B., Object-oriented metrics : measures of complexity, Prentice-Hall, pp.142-147, 1996

10. Hind Kabaili, Rudolf K. Keller and Frant;ois Lustman, “Cohesion as Changeability Indicator in Object-Oriented Systems”, Proceedings on Fifth European Conference on Software Maintenance and Reegineering (CSMR’01), 2001

11. Fenton, N. , Pfleeger, S.L., “Software Metrics A rigorous & practical approach”, 2nd Edition, PWS Publishing Company, 1997

12. Marc Frappier Stan Matwin Ali Mili, “Software Metrics for Predicting Maintainability”, Technical Memorandum 2, Canadian Space Agency 1994

13. Marcela Genero, Mario Piattini and Coral Calero, “Empirical Validation of Class Diagram Metrics”, Proceedings of the 2002 International Symposium on Empirrcal Software Engineering (IESE 2002), 2002

14. Huff, S. “Information systems maintenance”. The Business Quarterly 55, 30-32, 1990

15. Li, W., and S. Henry. Maintenance metrics for the object-oriented paradigm. Proceedings of the First International Software Metrics Symposium, Baltimore, MD, May 1993, pp. 52-60.

16. Miguel Lopez, Simon Alexandre, Valerie Paulus, Gregory Seront, “On the Application of some Metrology Concepts to Internal Software Measurement”, 8th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2004), Oslo, 2004

17. McCabe, T, “A Complexity Measure”, IEEE Transactions On Software Engineering, Vol. Se-2, No.4, December 1976

18. Port, O., “The software trap – automate or else”. Business Week 3051 (9), 142-154, 1988

19. Harry M. Sneed Anecon GmbH, Vienna Austria, “A Cost Model for Software Maintenance & Evolution”, Proceedings of 20th International Conference on Software Maintenance (ICSM’04), 2004

20. Dimitris Stavrinoudis, Michalis Xenos, Greece Dimitris Christodoulakis, “Relation Between Software Metrics and Maintainability”, Proceedings of the FESMA99 International Conference, Federation of European Software Measurement Associations, Amsterdam, The Netherlands, pp. 465-476, 1999.

21. W. Stevens, G. Myers, L. Constantine, "Structured Design", *IBM Systems Journal*, 13 (2), 115-139,1974.

22. Edward Yourdon and Larry L. Constantine. Structured Design. Prentice Hall, Englewood Cliffs, N.J., 1979.