Adding Process Metrics to Enhance Modification Complexity Prediction

Gabriella Tóth, Ádám Zoltán Végh, Árpád Beszédes and Tibor Gyimóthy
Department of Software Engineering
University of Szeged, Hungary, Árpád tér 2. H-6720
Email: gtoth, azvegh, beszedes, gyimothy@inf.u-szeged.hu

***Abstract*—Software estimation is used in various contexts including cost, maintainability or defect prediction. To make the estimate, different *models* are usually applied based on attributes of the development process and the product itself. However, often only one type of attributes is used, like historical process data or product metrics, and rarely their combination is employed. In this report, we present a project in which we started to develop a framework for such complex measurement of software projects, which can be used to build combined models for different estimations related to software maintenance and comprehension. First, we performed an experiment to predict modification complexity (cost of a unity change) based on a combination of process and product metrics. We observed promising results that confirm the hypothesis that a combined model performs significantly better than any of the individual measurements.**

***Keywords*-Metrics, changeability, effort prediction**

I. Introduction

Reliably estimating expected software project properties like modification cost is desirable during development or maintenance. The modification cost mainly depends on the comprehension time of the program and the difficulty of the modification. The modification cost estimation is often relied upon only on the experience of project leaders, and no methodic approaches are used. However, different current attributes of the project and the product itself can be quite good predictors of the mentioned properties. For example, a large and complex module is usually more difficult to comprehend and modify than a simpler one, hence this property can be an indication of higher maintenance cost. But similarly, past data about the development process, like programmer productivity can also be used successfully to estimate the expected modification cost. In the former case, we use *product metrics* to predict the desired attributes, while in the latter case, *process metrics* are employed. Both approaches are used by software engineering professionals aided by different estimation models [1].

The research on good predictor models is very important though, as often important decisions are made upon these approaches. There is a large body of literature dealing with the mentioned methods, but surprisingly, it can be observed that it is rarely the case that the initial data for estimation includes different kinds of software properties, such as both process and product metrics. We motivate our research on this observation, and work towards verifying the importance of such a combined estimation model, and establishing accurate prediction frameworks and models for specific development and maintenance estimations. In this report, we present our initial results of the research project in which we collected, over a certain period of time, different process and product metrics of a real industrial project, and used them to predict modification complexity, reporting promising results. Namely, the combined prediction using process and product metrics was more efficient than using only one type of metrics. We used machine learning techniques to automatically derive the prediction models. Although the current results require further investigation, and the model is still modest in its prediction capability, the sole fact that the combined model performs much better, strengthens our determination to continue this research.

The paper is organized as follows. After reviewing related work in Section II, we describe our experimental study in Section III. Empirical results are described in Section IV, and we discuss the results and conclude in Section V.

II. Related work

In contrast to most of the previous work, our primary goal is not to analyze the prediction accuracy for a set of predictors, instead, we focus on a comparative analysis between predictions with different kinds of predictors.

An effort estimation model was introduced by Mockus *et al.* [2] which predicts the amount and the distribution over time of the maintenance effort. They used the following factors: the developer herself, the type, the status, the size, and the rate of size and complexity of a change. They found that introducing a new feature is more difficult than repairing previous changes, which essentially means that looking at the progress of time is sufficient.

Product metrics are often used for prediction of maintenance effort. A well-known group of object-oriented product metrics has been proposed by Chidamber and Kemerer [3]. Many empirical studies were performed to validate empirically C&K suite, showing an acceptable correlation between C&K metrics values and software fault-proneness [4], [5] or maintenance effort [6].

Using not only product metrics but also developer's expertise or other process properties has already been investigated in the fault prediction area. Mockus and Weiss [7] applied a model, where the factors of their prediction were the

diffusion and the size of a change, the type of the change and the developers' expertise with the system. They found that change diffusion and developer's expertise were essential to predict failures. Moser *et al.* [8] stated that the most powerful defect indicators are high number of revisions, high number of bug-fixing activities, small sized CVS commits, and small number of refactorings. Yuan *et al.* [9] applied fuzzy subtractive clustering method to predict the number of faults, and they used module-order modeling to classify the modules into faulty/non-faulty classes. They built a model which is based on process and product metrics. Contrarily with other researchers, they stated that process metrics do not improve the classification accuracy and such a model does not provide acceptable results. In the present work, we examine whether the developer properties and other process metrics can improve the estimation model in the area of program comprehension and change effort estimation.

III. The Experimental Study

Our long term goal is to build a framework which can be applied for the prediction of different kinds of maintenance properties from various kinds of metrics. Apart from the theme of our current research, it will be used to predict for example, testability, error proneness and other maintainability and comprehension attributes as well. In this paper we use this framework to predict *modification complexity*, which means how long, on average, the modification of a line in a given class will take at the next change.

We experimented with different prediction models, after identifying the following **research questions:**

- RQ1a: How does the level of modification complexity change during the implementation/maintenance life cycle?
- RQ1b: Is the progress of time sufficient to predict the level of modification complexity? Some researchers argued that the progress of time is a sufficient predictor for some maintenance related attributes [2], so we wanted to verify that hypothesis.
- RQ2: Can the modification complexity prediction by product metrics be enhanced with additional process metrics? In the fault prediction area, it has been shown [7] that it can. Here we examine this hypothesis in connection with effort estimation.

Our **framework** is suitable to answer these questions. Very briefly, it is structured as follows (see Figure 1).

On each workstation, Productivity Plug-in was installed into development environment, which logs the development-related low level information (e.g. active file, perspective, developer, task, elapsed time, etc.). Each log is then uploaded to the Productivity Log Server. The project manager estimated some project metrics (e.g. task time, developers' experience, etc.). Process metrics are calculated from the Productivity Log Server, while the product metrics are obtained using the Columbus tool [10], which analyzes the

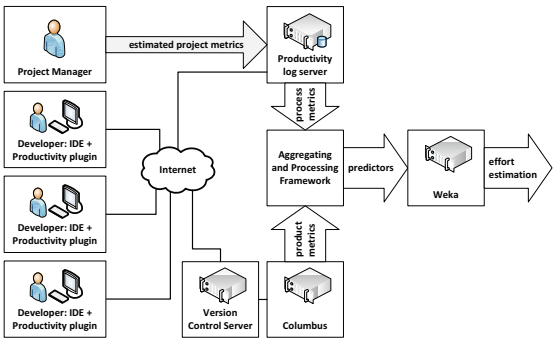

Figure 1. The architecture of the framework.

source code retrieved from the Version Control Server. The framework collects, processes, and aggregates these metrics which serve as input to the Weka [11] machine learning tool, which finally predicts the maintenance effort.

The Project We gathered data from an industrial R&D project in the area of telemedicine, which involves the JBoss SEAM, Symbian and Android technologies. Altogether, 7 developers worked in this project during our experimentation period, which lasted 23 working days. We started the experiment after an initial development of the system was over (revision r954), and finished when the system had been released (revision r1805). During the experiment, both development- and maintenance-type tasks have been performed (the latter being prevalent), the size of the system increased from 16,338 lines to 19,765 lines (Logical-LOC), and from 309 classes to 337 classes.

In the prediction model, three types of metrics have been used. The first category, **Product Metrics** are the following:

- LLOC (Logical Lines Of Code): total non-empty and non-comment lines of the class.
- C&K metrics: **DIT** (Depth of Inheritance Tree), **NOC** (Number Of Children), **CBO** (Coupling Between Object Classes), **RFC** (Response For a Class), **WMC** (Weighted Methods per Class), **LCOM** (Lack of COhesion in Methods).
- ECLOC (Effectively Changed Lines Of Code): the *delta* calculated from the SVN, the number of added, deleted or modified lines by comparing the previous version of the class with the current version.

The **Process Metrics** we used were the following:

- TT (Task Time): Estimated development time of a task, aggregated into 3 groups: short, medium, long.
- DEP (Developer's Experience in Programming): The level of experience of the developer, aggregated into 3 groups: low, medium, high.
- NFA (Number of File Access): Shows how many times a developer accessed (got back to) a file during a modification.
- NDF (Number of Developers of File): Shows how many developers have modified the file before.
- DT (Development Time): The net development time during the modification in minutes.

While product metrics are based on classes, process metrics are based on files, unambiguous association between classes and files was necessary.

We also used the Revision Number (**RN**) as a metric in prediction, which was the number of the revision when the file was modified, essentially the representation of the time. Finally, we define the metric Level of Modification Complexity (**LMC**) as the ratio of DT and ECLOC for the next change of the file/class. This is our target of prediction.

Data collection During the 851 examined commits to the SVN, 1134 file modifications have been made. For each file modification, we collected the C&K and LLOC product metrics calculated by Columbus. These metrics were processed in three ways: we calculated 1) the value of the metric in the given revision and file, 2) the relative value of the metric in the given revision and file compared to the base revision (r954), 3) the relative value of the metric in the given revision and file compared to the previous revision. We determined the ECLOC metric from the SVN repository. The next step was to relate the logs from Productivity Log Server to SVN changes. Unfortunately, we realized that the developers had not used continuously the Plug-in (they simply had not switched it on many times), so a lot of low level development information had been lost, so we were unable to connect all of the SVN commits to developer logs. Eventually, 200 file modifications had connection with both product and process metrics. Finally, we calculated the LMC values for the training set of the machine learner. However, since there were file modifications among the 200 cases where the file was not modified again at all, or the connection of process metrics to the next modification of the file was impossible, several modifications had to be discarded again. At the end, 91 file modifications remained as a training set to predict LMC.

The Prediction The predictor set consisted of one or more of the following groups of metrics: **1)** RN, **2)** Product metrics: processed C&K and LLOC, ECLOC and **3)** Process metrics: TT, DEP, NFA, NDF, DT.

The to-be-predicted LMC metric is a nominal attribute, so we aggregated it into 3 groups: low, medium, and high, to be able to feed it to the learner. To distribute the target classification approximately equally, we defined the limits accordingly, which resulted in having 30, 32 and 29 modification complexity values in the different classes. Finally, we validated the model with 10-fold cross validation.

IV. Preliminary Results

Table I summarizes our main findings from the experiment, in which the number of true positives (from the total of 91) and the weighted average of precision and recall values for each model can be seen. We built 4 models: one using only the revision number as predictor (column 2), one using the revision number and the product metrics (column 3), one using the revision number and the process metrics (column 4), and one using all three kinds of predictors (column 5). The first column shows 3 typical learning methods we used: decision tree, neural network, and regression.

Now, let us evaluate the results following the research questions set up at the beginning of the article.

RQ1: It was interesting to observe, how the values of our target prediction (LMC) actually changed over the range of the dataset. Generally, at the beginning of implementing a file, in the first revisions the developer implemented new features to the class and, in parallel, he/she was also designing future features to it. So, during this time the modification complexity level was usually high. This was followed by a period of smaller corrections, which did not take so long, so the modification complexity level became lower. Finally, when the class became more difficult to change, the modification complexity level was becoming higher and higher. This resulted in an overall 'U' shape, which is similar to what other researchers found [2].

The second part of this research question dealt with the strength of the progress of time as the sole predictor. In our case, it was represented by the revision number. The accuracy of prediction when only this measure was used can be seen in the second column of Table I. We can say that in two cases the accuracy of prediction is higher than the random case (where the precision value would be 33.33%), so we can conclude that progress of time can be an important predictor (sometimes even better than the product or process metrics taken individually). Due to the significance of the revision number, henceforward in the further predictions we always used it together with the other two categories.

RQ2: Our most important research goal was to find out whether the combined use of different types of metrics is any better than using only one type. We can observe in columns 3 and 4 of Table I that product metrics are good predictors of LMC, while the prediction capability based on only the process metrics is low (compared to the random 33.33%). However, the last column of the table shows a clear evidence that **the combination of the product and process metrics results a significantly higher accuracy in prediction**. Compared to the product metrics, in the case of the decision tree the prediction increased by 4.4 percentage points, 3.1 points with RBF network, and by using classification model it increased by 2.7 points.

V. Discussion and Conclusions

We introduced an approach to enhance maintenance effort prediction using low level process information in addition to product metrics-based estimation only. Our primary goal was not to find the best set of predictors for predicting the modification complexity level, but to verify which set of predictor types offers more possibilities for the prediction. We found that using process metrics in addition to product metrics can be obtained a significantly better prediction model. Surprisingly, we observed that the progress of time

J48	Only Revision Number	Product Metrics	Process metrics	Product & Process Metrics
Nr. of true positives	32	39	31	**43**
Precision	31.9%	43.1%	34.2%	47.5%
Recall	35.2%	42.9%	34.1%	47.3%
RBFNetwork				
Nr. of true positives	35	32	29	**36**
Precision	36.6%	36.6%	31.7%	39.7%
Recall	38.5%	35.2%	31.9%	39.6%
ClassificationViaRegression				
Nr. of true positives	36	38	30	**41**
Precision	35.2%	40%	26.7%	42.7%
Recall	39.6%	41.8%	33%	45.1%

Table I
PREDICTION RESULTS.

is more significant than the process metrics alone. However, the best result can be obtained if all three types are combined, which coincides with the result of related works in other areas like fault prediction. We can draw some other conclusions as well, as overviewed below.

Although the learning precision of 40-47% seems not to be very high (although much better than the random case), we think this result is quite remarkable in this early stage of our research. It has a number of potential improvement points, so we are confident that in another, more thoroughly designed experiment even better results will be obtained. The main limitation of this experiment was that the process measurement was not so complete as could have been if the programmers were using the tool more conscientiously. The limitation of the small number of programmers and the small period of time might impose a threat to the generalizability of the results as well.

The usage of the Productivity Plug-in serves us valuable low level process metrics (exact development time, number of file access), while most of related works deal with only version control system, problem tracking system or configuration management database.

Apart from our most important goal to enhance the collection of process metrics with the Productivity Plug-in, we have some other plans for future research as well. We have built a general framework with which we will be able to predict other maintenance or comprehension attributes, like testing effort, development time, error proneness, or the number of faults, involving various other predictors as well. Our other goal is to extend the approach to be able also to compare to each other the effectiveness of different kinds of technologies in terms of productivity or maintainability.

VI. ACKNOWLEDGMENTS

This research was supported by Telenor, Nokia Komárom, the Hungarian national grants TECH 08-A2/2-2008-0089, GOP-1.1.2-07/1-2008-0007, and OTKA K-73688.

REFERENCES

[1] L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach*. Wiley-IEEE Computer Society Pr, 2006.

[2] A. Mockus, D. M. Weiss, and P. Zhang, "Understanding and predicting effort in software projects," in *In 2003 International Conference on Software Engineering*. ACM Press, 2002, pp. 274–284.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, pp. 476–493, June 1994.

[4] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," in *IEEE Transactions on Software Engineering*, vol. 22, no. 10, Oct. 1996, pp. 751–761.

[5] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," in *IEEE Transactions on Software Engineering*, vol. 31, no. 10. IEEE Computer Society, Oct. 2005, pp. 897–910.

[6] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.*, vol. 23, pp. 111–122, November 1993.

[7] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, pp. 169–180, 2000.

[8] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 181–190.

[9] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," *Application-Specific Software Engineering and Technology, IEEE Workshop on*, p. 85, 2000.

[10] R. Ferenc, A. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus – reverse engineering tool and schema for C++," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)*. IEEE Computer Society, Oct. 2002, pp. 172–181.

[11] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. San Francisco: Morgan Kaufmann, 2005.