Prediction of Software Development Modification Effort Enhanced by a Genetic Algorithm

Gergő Balogh, Ádám Zoltán Végh, and Árpád Beszédes

Department of Software Engineering University of Szeged, Szeged, Hungary
{geryxyz,azvegh,beszedes}@inf.u-szeged.hu

Abstract. During the planning, development, and maintenance of software projects one of the main challenges is to accurately predict the modification cost of a particular piece of code. Several methods are traditionally applied for this purpose and many of them are based on static code investigation. We experimented with a combined use of product and process attributes (metrics) to improve cost prediction, and we applied machine learning to this end. The method depends on several important parameters which can significantly influence the success of the learning model. In the present work, we overview the usage of search based methods (one genetic algorithm in particular) to calibrate these parameters. For the first set of experiments four industrial projects were analysed, and the accuracy of the predictions was compared to previous results. We found that by calibrating the parameters using search based methods we could achieve significant improvement in the overall efficiency of the prediction, from about 50% to 70% (F-measure).

Keywords: software development, effort estimation, modification effort, genetic algorithm

1 Introduction

One of the tasks in software cost estimation, especially in the evolution phase, is to predict the cost (required effort) for modifying a piece of code. A possible approach for such modification effort prediction is to use various software attributes from historical development data and from the current version of the software. The attributes can be expressed in the form of software metrics, both product and process. Product metrics are calculated by preforming the static analysis of the software (a simple example is the logical lines of code), while process metrics can represent time-related quantities collected during project implementation (for example, the Net Development Time of the modifications). The metrics can then be used, with the help of a model, to predict the modification costs.

Numerous articles were written about experiences using process or product metrics [1,2], but researches using both are rare. Our previous works tried to predict the cost of future modifications by applying both product metrics calculated from the source code and process metrics collected during the development [3]. We compared the results when using only one type of metrics with the case when both kinds of metrics had been used. We applied machine learning to create the model for the prediction. In an advanced model we used a more complex metric for expressing the modification effort, which was the aggregation of different kinds of modifications like creation, deletion, and type change. To express the modification effort in a single value, we used different parameters (weights) for the different kinds of modifications. The choice of these parameters was crucial for the accuracy of prediction and their calibration was not simple. In the present work, we

report on our early experiences in applying search based methods to determine these parameters (a basic genetic algorithm (GA) was used for this purpose). A typical improvement of as much as 20 percentage points was achieved in the combined prediction accuracy (F-measure on the precision and recall) when comparing the model with initial parameters to the one obtained after running the search method.

We summarize the experiments with the following research questions: RQ. 1 Is it possible to define a better model for software development effort prediction than the one used in paper [3,4]? In particular, we seek for a better way to express modification effort. RQ. 2 Can the GA be used to improve the precision of the estimation of software development effort compared to the results in paper [3,4]? In particular, what rate of improvement can be obtained after calibrating the crucial parameters of the cost metric and re-applying them to the model?

To achieve the mentioned goals we implemented a framework which is capable to collect and aggregate product and process metrics from various sources including the source code and the integrated development environment. The framework detects the modifications between revisions, and tries to predict the effort of further changes.

2 Overview

The overall process is shown in Fig. 1. The experiment starts with a measurement phase where the used data is collected from various sources: the metrics about the evolution of the software, the source code from the SVN version controlling system, and the metrics estimations which were given by the project manager. This phase has two main tasks; to collect and calculate the process and product metrics, and to detect and group the modifications of the source code between the revisions. The metrics and the modification groups are sent to the GA which prepares a population of individual entities. During the initial set-up of the population and the evolution steps, two metrics are calculated: Typed Modification (TMOD), which was defined as the weighted count of modifications between two revisions; and Modification Effort (MEFF), the ratio of TMOD and the net development time of these modifications. Afterwards the prediction model is tested and its weighted F-measure value is used as fitness to rank the individuals in the population and select the best entities for breeding. As the final step of the evolution cycle, the new weights of the modification are calculated and the model is updated. When the precision reaches an appropriate value, the GA stops and a new, enhanced model is built using the weights of the best entity in the final population. This MEFF prediction model is the output of the execution of the framework.

Data was collected during the experiment from about 800 revision, about 75 days long. Both R&D and industrial projects were analysed and the source code was written in Java language using the Java EE 6 virtual machine and the Seam 2 framework. Altogether 2200 records were collected as learning set.

3 Modification Effort Prediction

Process and product metrics were used together per source file basis as separated entities. Then, the appropriate cost function was calculated from this data. The metrics used as predictors were the followings: (i) Logical Lines Of Code (LLOC) (ii) C&K metric suit defined by Chidamber and Kemerer[2] (iii) Estimated development time of a task, aggregated into 3 groups: short, medium,

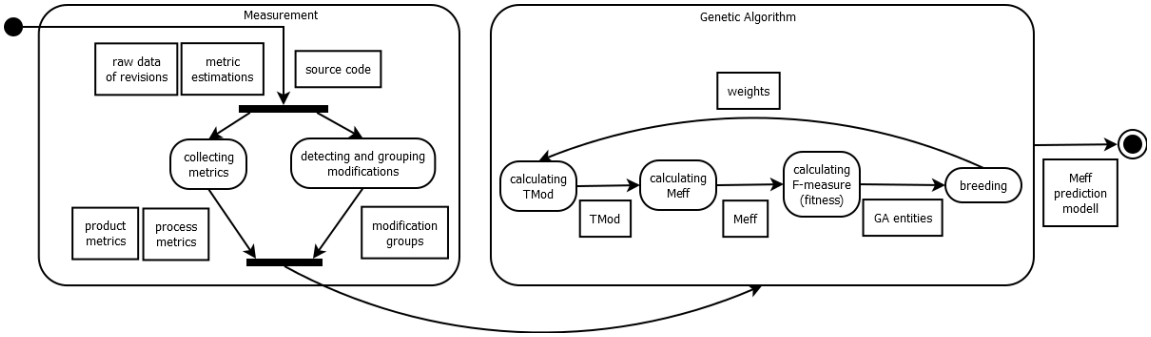

Fig. 1. Overview of the experiment

long. (TT) (iv) Developer's Experience in Programming (DEP) (v) Number of File Access (NFA) (vi) Number of Developers of File (NDF) (vii) Development Time (DT) [1]

To extend our previous framework, a new metric called Modification Effort (MEFF) was defined which was calculated as the following: first, the modifications were grouped, based on the target entity (e.g.: method) and the preformed action (e.g.: creation) [5]. The weighted count of the previously defined modifications was called Typed Modification (TMOD), this expresses the different amount of developer's effort used in the modifications. The ratio of the net development time with the TMOD is the MEFF metric.

The weight was defined on the basis of the per groups of modification. We established a single rule for each disjunct group which determined the contained relation.

The following modification groups were measured.

- class ∘ creation ∘ deletion ∘ accessibility change
- method ∘ creation ∘ deletion ∘ accessibility change ∘ prototype change ∘ return type change ∘ size change
- data member ∘ creation ∘ deletion ∘ accessibility change ∘ type change

Instead of the Level of Modification Complexity (LMC) – defined as the ratio of the Effectively Changed Lines Of Code (ECLOC) for the next change of the file/class and DT, the net development time between two revisions [3,4] – MEFF was used as the target function, which was equally labelled with: low, medium and high values.

The Weka framework [6] machine learning and the 10-fold cross-validation utility were used to evaluate our model. The weight sum of the F-measure was chosen as the fitness value which is the harmonic mean of precision and recall.

4 Genetic Algorithm

SBSE was used to fine tune the weights of the modification. The initial weight-vector was set by our developer experience. The algorithm was iterated on the basis of previously set scenario. Our assumption was that the genetic algorithm should converge to the suitable weights, which should provide a more accurate estimation [7].

[1] detailed in paper [3]

As previously described, a weighted sum of the count of modification group was used as the target function. A genetic algorithm (GA) was used to fine tune the weights of each group.

The individuals identified by its chromosome, which is a vector over the real numbers with the same dimension. In the model each chromosome represents a weight-vector, and every element determines the weight of a single modification group.

The fitness value is calculated for each individual by evaluating the model with the weights defined in that particular individual. The final goal of the GA is to improve the precision of the model. For classification problems, the F-measure value can give a reliable approximation of the accuracy. The base model was not enhanced with the GA, but it was evaluated using the F-measure. Thus the F-measure was chosen to be the fitness value of the GA.

An evolution step starts with the breeding process which consists of two steps, first, the GA selects the two best entities with its fitness value. The crossover operator will apply only this pair. Every call of the crossover operator produces exactly one child. The algorithm repeats the operation to produce more than one child.

We used a uniform crossover logic. During the crossover the algorithm iterates via the elements of the chromosome (vector) and randomly chooses an element from one of the two parents. Every element has the same chance to be copied into the child's chromosome [8].

The chromosome of the children is subject to mutation. A lower limit and an upper limit were determined for the weights of the groups. During the mutation some elements (weights) of the chromosome change. The algorithm gets the half of the distance between the limits and the current selected weight and sets the current value either to the lower or to the upper half point. This way, the two limits are never exceeded. Then, the child is included in the population.

The individuals with the worst fitness value are killed (removed from the population) to maintain the size of the population, this way the current evolution step is completed and the algorithm proceeds to the next generation [9,10].

The above mentioned GA parameters and their values are shown in Tab. 1.

Table 1. GA parameters

initial mutation rate	100%
mutation rate	50%
mutation lower limit	0.5
mutation upper limit	100
birth count	2 child per evolution step
crossover rate	2 crossover per evolution step
population size	200 individuals
generation count	50 generation

5 Preliminary Results

As shown in Fig. 2, the fitness value of the prediction grows in every case. Tab. 2 shows the same fitness values. As it can be seen in Tab. 3, the average grows with about 18 percent. It is also relevant that in the worst case our model proves to be better with about 16 percent.

With these pieces of information the two research questions can be answered.

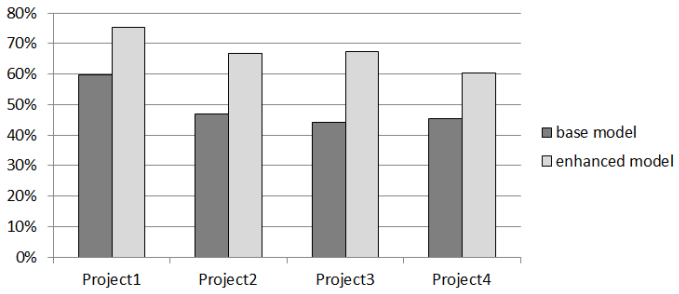

Fig. 2. Fitness values of prediction per project

Table 2. Fitness values of prediction per project

	Project1	Project2	Project3	Project4
base model	59,8000%	47,0000%	44,2000%	45,3000%
enhanced model	75,2630%	66,6425%	67,4835%	60,2791%

Table 3. Comparison of models

	worst	best	average	median
base experiment	44,2000%	59,8000%	49,0750%	46,1500%
enhanced model	60,2791%	75,2630%	67,4170%	67,0630%
difference	16,0791%	15,4630%	18,3420%	20,9130%

RQ. 1 Yes, it is possible. Our model gives a better estimation from the beginning of the evolution and the population average fitness value is higher in every generation.

RQ. 2 Yes, the use of the GA can improve precision. With this simple GA implementation a significant improvement was reached.

The weight of groups was aggregated from all four projects and weighted with the size of the learning set. Two aspects were created to examine the validity of the weights calculated by the GA.

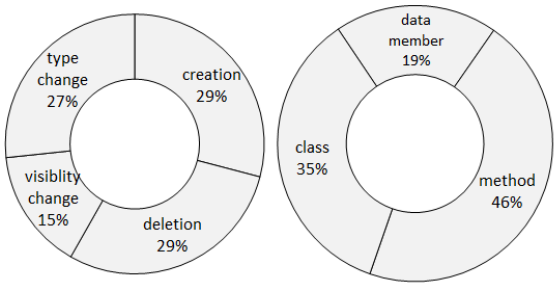

Fig. 3. Aggregated weights of groups

These aspects are shown in Fig. 5. The values can be interpreted as the importance of a modification, i.e. how much gain will be achieved by applying the modification. The diagram on the left shows a by action aggregation. As it can be seen, the creation and deletion are more important than the type and visibility changes. On the right side, a subject based aggregation can be seen. The most important modification was applied on the method elements which included the method body modifications as well.

6 Conclusion and Plans

In this paper, we present two new metrics and a new procedure with which we can increase the effectiveness of our modification cost prediction method based on product and process metrics and machine learning. Our previous results have been improved with the introduction of new metrics (TMod, Meff) used as target function, and by using a genetic algorithm to calibrate certain crucial parameters required by the model. We were able to increase the success of the prediction model significantly. We manually investigated the final parameter values produced by the GA. These parameters seem to be valid based on our own developer experience, but further analysis will be needed to validate the results.

In the future, we plan to prepare a new set of parameters by collecting developer experience within our team. We plan to use a questionnaire and compare the results with the automatically calculated ones, which do not use the bias of developer experiment. We also plan to repeat the experiment on a bigger dataset and apply the method to achieve different goals like comparing projects or developer productivity. We would like to fine tune our GA with the testing of some other crossover operator or fitness value.

References

1. Mockus, A., Weiss, D.M., Zhang, P.: Understanding and predicting effort in software projects. In: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society (2003) 274–284
2. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20**(6) (1994) 476–493
3. Tóth, G., Végh, A.Z., Beszédes, A., Gyimóthy, T.: Adding Process Metrics to Enhance Modification Complexity Prediction. In: Proceedings of the 19th IEEE International Conference on Program Comprehension (ICPC'11), Ieee (June 2011) 201–204
4. Tóth, G., Végh, A.Z., Beszédes, A., Schrettner, L., Gergely, T., Gyimóthy, T.: Adjusting Effort Estimation Using Micro-Productivity Profiles. In: Proceedings of the 12th Symposium on Programming Languages and Software Tools (SPLST'11). (2011) 207–218
5. Abdi, M.K., Lounis, H., Sahraoui, H.: Using Coupling Metrics for Change Impact Analysis in Object-Oriented Systems. In: Proceedings of the 10th ECOOP Workshop on Quantitative Approaches in ObjectOriented Software Engineering QAOOSE 06. (2006) 61–70
6. Holmes, G., Donkin, A., Witten, I.: WEKA: a machine learning workbench. In: Proceedings of ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conference, IEEE 357–361
7. Cavicchio, D.: Adaptive search using simulated evolution. (1970)
8. Syswerda, G.: Uniform Crossover in Genetic Algorithms. (1989) 2 – 9
9. Melanie, M.: An introduction to genetic algorithms. Cambridge, Massachusetts London, England, Fifth (1999)
10. Sivanandam, S.: Introduction to genetic algorithms. (2007)