Adjusting Effort Estimation Using Micro-Productivity Profiles

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

Abstract

We investigate a phenomenon we call micro-productivity decrease, which is expected to be found in most development or maintenance projects and has a specific profile that depends on the project, the development model and the team. Micro-productivity decrease refers to the observation that the cumulative effort to implement a series of changes is larger than the effort that would be needed if we made the same modification in only one step. The reason for the difference is that the same sections of code are usually modified more than once in the series of (sometimes imperfect) atomic changes. Hence, we suggest that effort estimation methods based on atomic change estimations should incorporate these profiles when being applied to larger modification tasks. We verify the concept on industrial development projects with our metrics-based machine learning models extended with statistical data. We show the calculated micro-productivity profile for these projects could be used for effort estimation of larger tasks with more accuracy than a naive atomic change oriented estimation.

1 Introduction

Software estimation is often required in development and maintenance tasks in various forms [2, 11]. The prediction of the required effort to implement a specific change is directly related to software costs, hence it is very important. However, estimation based purely on experience is typical, and little or no systematic approaches are often used. This is partly due to the very complex nature of the problem: the required effort depends on a number of parameters which are difficult to quantify. Usually it is easier to get an accurate estimation of a smaller change than of a larger change task (a series of smaller changes). Hence, the estimators often predict a larger task by aggregating a series of smaller change estimations [3]. Different estimation models (such as those based on statistical models or employing machine learning) also tend to scale poorly, *i.e.*, they may be fairly accurate in the short term, but become less and less so when they are applied for longer modification periods. Moreover, the volume of example data available to automatic prediction models using historical data is much bigger related to atomic changes than to larger change tasks. Throughout the paper we will distinguish the following effort estimation types:

- '**Change task estimation**' This refers to the common situation when a project manager or a developer needs to estimate the cost of implementing a specific change request, and generally, the manager is not interested in the individual costs of performing atomic changes. For this type of estimation, global information about the organization and the project is required also.

- '**Atomic change estimation**' In this case, we are interested in the cost of implementing a unit, general atomic change. This kind of estimation is important for short term project control, but insufficient for change task estimation (typically simpler methods and less historic data are enough).

- '**Atomic change sequence estimation**' When the same software entities are changed in a series of subsequent steps of atomic changes, we can aggregate the individual cost estimates up to the estimated size of the change in order to approximate the cost of the whole change sequence. Note that the expected extent of the change needs to be estimated separately.

However, we argue that for the latter the notion of what we call *micro-productivity decrease*,[1] should be taken into account, otherwise the aggregation will be imprecise and provide false estimations of the whole change task. Micro-productivity decrease is an observation that the cumulative effort to implement a series of changes is larger than the effort that would be needed if we made a single modification to achieve the same result in one step. The reason for the difference is that the same sections of code are usually modified more than once in the series of (sometimes imperfect) atomic changes. (Note, that this can be observed in any type of creative work, not only software development.)

This phenomenon is undoubtedly found in most development or maintenance projects, moreover, it seems to have a specific *profile* that depends on the type of the system, the development model, the composition of the team, and other factors, but it has to be typical for the current project. In this work, we propose to use this profile – *Micro-Productivity Profile (MPP)* – to adjust the estimation values got for atomic change sequences in order to get more accurate values at any desired change task size. Essentially, MPP shows how much repeated modifications to distinct lines of code are to be expected – in other words, how much the micro-productivity is decreasing by the length of the atomic change sequence. We found in our subject projects that, on average, after merely 8 atomic modifications the micro-productivity halves.

We determined a method to formulate MPP, then applied it to our prediction models, which are based on machine learning, and can predict the net developer time for modifying a single code line. In our machine learning model [13], we employ various product and process metrics as predictors, and in the present work we extend it with statistical values to be able to predict change sequences, and not only an atomic change as in the previous version. We combine this model with MPP by taking explicitly the decrease in micro-productivity into account, and verify the effect of this enhancement to the prediction accuracy in an empirical study. We make the following contributions in this paper:

1. We apply our machine learning based effort estimation method from previous work on an extended data set.

2. We introduce the notion of Micro-Productivity Profile, and present the way we have calculated it for our subject projects.

3. Finally, we compare the different prediction approaches – learning models with and without MPP – and check which one is the closest to the actual efforts.

The rest of the paper is organized as follows. Section 2 presents the micro-productivity decrease phenomenon through an example. Section 3 defines the MPP and describes its use in estimation models. Section 4 presents the machine learning that we used to predict the modification effort. In Section 5, we describe our experimental study using machine learning method and MPP. Results are reported in Section 6. Reviewing related work in Section 7, we conclude and list our future plans in Section 8.

2 Motivating Example

To understand the motivation for our work, let us consider the following example, which has been taken from one of the real projects we used in our experiment. The example represents two changes. The initial code can be seen in Figure 1 (a), while Figure 1 (b) and Figure 1 (c) shows the modified parts of the first and the second changes.

[1]We use the term micro-productivity related to atomic developer changes as opposed to what we call macro-productivity, which refers to the investigation of productivity on a larger scale, for a whole organization, its processes, people, a project or a technology, for instance.

```
...
83: if (!dataProvider.getDataMap().containsKey(tagId)) {
84:   dataProvider.getDataMap().put(tagId,
                    new ArrayList<DataPoint>());
85: }
86:
87: dataProvider.getDataListForTag(tagId).
                 add(new DataPoint(xValue, yValue));
```

(a)

```
...
83:   initDataMap();
84:
85:   dataProvider.getDataListForTag(tagId).
                   add(new DataPoint(xValue, yValue));
```

(b)

```
...
83:   dataProvider.addDataForTag(tagId,
                   new DataPoint(xValue, yValue));
```

(c)

Figure 1: The (a) original and the changed code after (b) the first and (c) the second modifications.

The first change moved the `if` statement (lines 83-85) to the `initDataMap()` method, so the `if` statement was changed to the method call. The aim of this solution was to decrease the number of duplicated code instances in the code. The second change merged the initialization and the addition as a refactoring operation. This operation increases the maintainability, as the developers do not have to care about calling the initialization method manually. A new method was declared with the name of `addDataForTag()`, which performs the `initDataMap()` method call in case of necessity.

During the first change 3 lines were removed and 1 line was added, while during the second change 2 lines were removed and 1 line was modified. It could be counted as 7 line modifications, however, there were lines which were modified twice. If the duplicated code elimination and the refactoring operation were executed in one step, only 4 lines would be removed and 1 line would be modified. In other words, we could speculate that higher programmer productivity can be obtained in short periods (7 lines) than in a longer period (5 lines). This is due to the natural process of incrementally creating something new, sometimes performing atomic changes superfluously. This, of course, cannot be avoided, since there is no such thing as *perfect* development, but we argue that this phenomenon should not be neglected in software cost estimation.

In the following, we will refer to this observation as *micro-productivity decrease* We will define the modification effort, the Modification Complexity (**MC**) metric as the ratio of the development time and the code churn size (the number of modified, added, and deleted lines), and with the help of this metric, the Micro-Productivity Profile (**MPP**) will be defined as well. Note when investigating micro-productivity decrease, the time required for individual modifications will always sum up, while the net amount of modifications will be subjected to the effect above (the total number of changed lines being less than or equal to the sum of the atomic changed lines), hence it will affect LMC similarly.

3 Micro-Productivity Profile and its use in effort estimation

3.1 Definitions

First, we introduce the modification effort metric, which will be our representation of the effort (developer time) required to perform a single or multiple changes on a program element such as a file, *i.e.*, to modify one line in a given subsequent revision (modification) of that element. This metric, the Modification Complexity (**MC**) is the ratio of the net development time and code churn size for a subsequent change of a file.[2] More formally, MC is defined for the i^{th} modification of file F as:

$$MC(F,i,x) = \frac{time(F,i,i+x)}{diff(F,i,i+x)},$$

[2]Note, that this metric does not take the time spent on other activities into account, like reading documentation, examining dependencies with other files, which may contribute to the overall cost.

where

- x is the number of subsequent atomic changes,
- $time(F,i,i+x)$ is the net development time (**DT**) spent on F between the i^{th} and $(i+x)^{th}$ versions of F in minutes,
- $diff(F,i,i+x)$ is the size of the **code churn** between the i^{th} and $(i+x)^{th}$ versions of F in lines.

Now we define Micro-Productivity Profile (**MPP**) – computed as a median of all different modifications with the same length in a project – as a function of x, which is the number of subsequent atomic changes to perform a bigger change task, and it shows the ratio of decrease in micro-productivity (increase of MC) with respect to the case when the modification is done in one step. In other words, MPP shows how much longer the modification of a single line takes – given the atomic change sequence length as a parameter – than if the change sequence had been done in one single step. More formally:

$$MPP(x) = median_{\forall F, \forall i} \frac{MC(F,i,x)}{MC(F,i,1)},$$

where

- x is the number of subsequent atomic changes,
- the median values (to avoid over-influence of outliers) are calculated from MC of every modification of every file,
- $1 \leq i \leq max(F) - x$, where $max(F)$ is the last modification of file F.

3.2 Using MPP in effort estimation

Here, we will briefly describe a possible approach to use MPP in effort estimation of a larger development or maintenance task. Suppose that a project manager needs to assess the amount of developer time for completely implementing a specific change request (rather than being interested in the effort of a single modification only) – this is what we called 'Change task estimation'. In this situation, after concept location, the next task is to perform an impact analysis to determine the extent of the required modification [12].

Often only the prediction of the cost performing a smaller modification or an atomic change is reliable, so the manager get the estimation of the whole change task by simply multiplying the effort of an atomic change with the estimated number of the necessary modifications, which relates to the size of the change impact set and depends on the development process model. But the problem is that atomic modifications do not sum up due to the MPP effects.

It is important to stress that in this scenario the amount of atomic changes required (*i.e.*, the size of the change impact and the required steps to perform the change task) has to be determined by the manager separately based on program analysis. We use this length as a parameter in our estimation framework. In a situation when the manager has access to more complex project data (like a series of past change requests) that can be used for predicting the effort of the change as well, the method of adjusting atomic estimation using MPP has no function, but a complex change task-based estimation would be used.

MPP may be applicable for a number of other purposes as well, such as to define customized developer profiles that can be helpful in project resource planning/allocation. Customized profiles could also be defined for different project types, process models, or organizations. We do not deal with these options in this paper, but plan to investigate them in the future.

4 Machine learning based estimation

4.1 Estimation models

In previous work [13], we presented a framework and an effort estimation model based on a complex set of product and process metrics as predictors. With that approach we could answer the question "how many minutes will a single line change take in the *next* modification of a file on the average?"

However, if the effort of a complex modification is to be estimated, this model is not suitable. If the manager estimates the possible time of the modification, due to the sequence of imperfect modifications, the real modification time is usually more than the estimated one. In order to get an accurate cost estimation by aggregating of the atomic changes cost, micro-productivity decrease should be taken into account. In order to investigate this phenomenon, we construct an extended estimation model, which can estimate the effort not only for the *next* modification, but for the *next several* modifications.

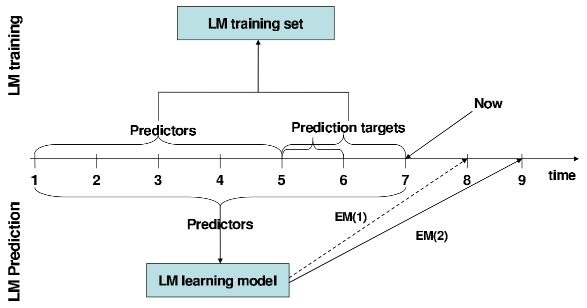

Figure 2: Learning model (LM) illustrated chronologically.

Figure 2 demonstrates the machine learning of the modification effort in the case of a file. The x axis represents the time, with nine modification points of the file. The figure depicts a snapshot at measurement point 7, previous measurement points represent the past, while subsequent measurement points represent the future. The training data is always calculated from the past (on interval 1-7), e.g., predictor values are generated using the information available between points 1 and 5 and target values are obtained from points 5 to 7. The predictors are calculated from the first versions of the file (on interval 1-5), while the targets are calculated from the next modifications based on interval 5-7.

This way, the learning model (which is based on the training set) will be able to forecast the modification complexity for the next modification. In the current example, it means that the available information from interval 1-7 can be considered as a predictor set to predict the modification effort of the next change, which gives a forecast to future point 8.

Furthermore, this learning model is extended with a statistical function to give a forecast to not only the next one but the subsequent modifications. So, we introduce 2 kinds of *estimation methods*:

1. the naive atomic change based estimation method (referred to as **EM**): we assume constant effort during a series of atomic changes, where the effort is estimated by the learning model,

2. the improved atomic change based estimation method (referred to as **EM(x)**): the naive atomic change based estimation method is improved with $MPP(x)$ statistical function to be able to consider the productivity decrease ($EM(x) = EM * MPP(x)$).

4.2 Predictors

In our previous paper [13] we introduced a predictor set for machine learning to estimate modification complexity. Now, we use the same predictors for effort estimation. Figure 3 shows the used product and process metrics.

	Product metrics:
LLOC	Logical Lines Of Code: Total non-empty and non-comment lines of the class.
C&K metrics [4]	**DIT** (Depth of Inheritance Tree), **NOC** (Number Of Children), **CBO** (Coupling Between Object Classes), **RFC** (Response set For a Class), **WMC** (Weighted Methods per Class), **LCOM** (Lack of COhesion in Methods).
Code Churn	The number of added, deleted or modified lines, calculated from the SVN by comparing the previous version of the class with the current version.

	Process metrics:
TT	Task Time: Estimated development time of a task by the project manager in hours (in order to consider the difficulty of the task in prediction). It is aggregated into 4 groups: very short, short, medium, long.
DEP	Developer's Experience in Programming: The level of experience of the developer according to the project manager, aggregated into 4 groups: beginner, junior programmer, senior programmer, leader programmer.
NFA	Number of File Accesses: Shows how many times a developer accessed (got back to) a file during a given modification.
NDF	Number of Developers of the File: Shows how many developers have modified a file before.
DT	Development Time: The net development time of a file – it is calculated by monitoring the development of the file –, it shows how many minutes a modification of a file actually takes between two revisions.

Figure 3: The used product and process metrics.

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 shows the number of the revision when the file was modified, essentially the representation of the time. We have 3 projects with 3 sequences of revision numbers in different repositories, and they were standardized by using sequential numbering starting with 1.

Finally, we define the metric Modification Complexity (**MC**) as the ratio of development time and size of code churn for the next changes of the class. MC means how many minutes one line modification takes on the average during the next changes of the class. The target of our prediction model is the LMC: we defined the levels of modification complexity (**LMC**) by grouping the MC values into three classes: low, medium, high. We used Weka [14] to estimate the LMC from the predictors. Three kinds of learning methods were used: *J48* (C4.5 algorithm), *RBFNetwork* (a normalized Gaussian radial basis function network), and *ClassificationViaRegression* (a classification using regression methods).

5 The Experimental Study

5.1 The Subject Projects

We gathered data from 3 industrial R&D projects (a small, a medium and a large) in the area of agriculture, an off the shelf chart viewer component, and telemedicine. *SCM-map* is a small map viewer portlet project under GateIn Portal[3] which also integrates Geoserver services, an open source map server implementation. *Chart* is a Flot based JSF implementation, and *EE-Oryx* is a visual process template editor for the web which is based on Oryx[4]. We used these three projects together as a whole for our experiments to have more data from which to predict.

The initial version of the project files was generated, this included default implementations of most of the methods. For this reason, during development the insertion of completely new sections of code

[3]http://www.jboss.org/gatein

[4]http://bpt.hpi.uni-potsdam.de/Oryx/

were relatively rare, most of the file modifications involved editing and restructuring existing code. In Table 1, the main characteristics of the projects can be seen.

Project	Classes	Nr. of developers	Work Days	Nr. of revisions	Nr. of file modifications
SCM-map	14 → 40	2	21	140	25
Chart	93 → 107	2	20	72	201
EE-Oryx	1380 → 1418	2	16	285	145
Total				497	371

Table 1: Metrics representing the main characteristics of the projects.

The projects were managed at the same company and each project had two different developers allocated to them. Note, that during the development and maintenance the SCM-map project, there were only 25 usable file modifications, since the source code base contained a great amount of XML, HTML and JavaScript code, from which we were unable to extract product metrics (only from Java).

5.2 The Framework

In this experiment we extend the framework introduced in our previous work [13]. Our framework is a general purpose framework, which can be applied for prediction of different kinds of maintenance properties from different kinds of metrics. Figure 4 shows the extended framework.

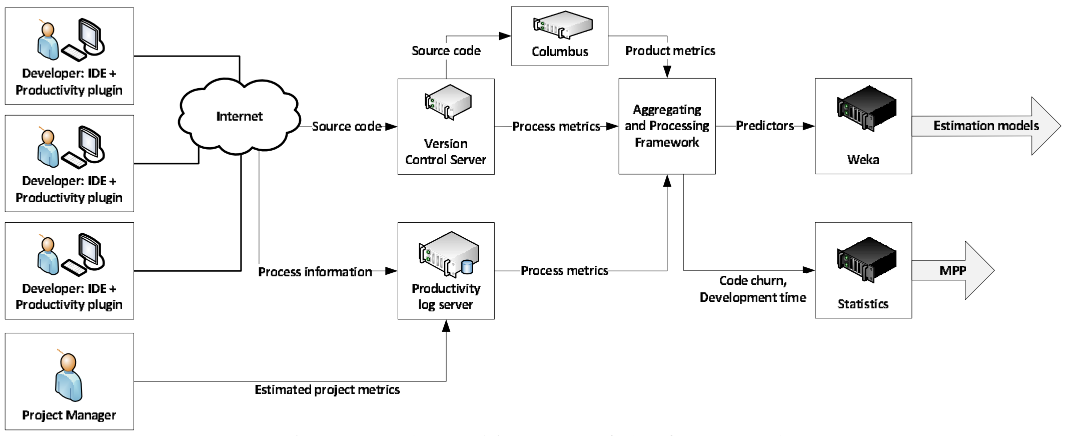

Figure 4: The architecture of the framework.

On each workstation, a Productivity Plug-in was installed into the Eclipse 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. Before starting the development of the project, 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 [6], which analyzes the source code retrieved from the version control server. The framework collects, processes, and aggregates the metrics which serve as input to the Weka [14] machine learning tool, which finally gives estimation models to predict the maintenance effort. We determined the MPP from the collected code churn size and the development time in case of each file in each modification.

During the examined commits to the SVN repository, 371 Java file modifications were made. Adding or removing of a whole file was not considered, only file modifications. There were a lot of file modifications during the experiment, but we can measure static product metrics for only Java files. For each Java file modification, we collected the C&K and LLOC product metrics calculated by Columbus. These metrics have proved to be useful for prediction in software development area [8]. They were processed in three ways to be able to follow their change scale, namely: 1) the value of the metric in the given revision, 2) the relative value of the metric in the given revision compared to its first version, 3) the relative value of the metric in the given revision compared to its previous version.

We determined the code churns from the SVN repository, then the process metrics from the Productivity Log Server were related to the SVN repository changes: this way the file modifications were connected to both the product and the process metrics. Finally, we calculated the MC values from a product and a process metric (DT/code churn) for the training set of the machine learner. However, since there were file modifications among the 371 cases where the file was not modified again at all, 254 file modifications remained to predict LMC.

We have extended the framework with a statistical component, which collects the code churn and the DT values from the past to determine the Micro-Productivity Profile for different modification lengths.

5.3 Training and testing data

The C4.5 algorithm, a normalized Gaussian radial basis function network, and a classification using regression methods were used as prediction algorithms for the effort of the next change, which were validated with 10-fold cross validation by their precision, recall and F-measure values.

However, to test our extended prediction method, we took a sequence of modifications belonging to a randomly selected task called `chart.implementation.java.connector`. This task modified 17 files during 9 revisions. A total of 44 file modifications belong to this task, so we partitioned the 254 prediction data into two groups: one group contained 210 training data instances and the other one contained 44 testing data instances. In the following, we will refer to the testing data as the *actual* value.

As the MC metric is a nominal attribute, we aggregated it into 3 groups (called Levels of Modification Complexity (LMC)) to predict only 3 classes by the learner: we sorted ascending the MC values and composed 3 sets with the same number of elements (low, medium, high).

6 Results

In this section the estimation models (EM and $EM(x)$) are applied to estimate the effort of the next modification of the files. We also quantify the Micro-Productivity Profile for our projects and compare the actual effort to the predicted data by the constant EM model and by the improved $EM(x)$ model.

6.1 EM learning model

EM is the original learning model we used in our previous experiment [13]. In that paper we showed that the combination of the product and process metrics resulted a significantly higher accuracy in the case of our effort prediction. Since we have another, larger training set, we re-evaluated the precision and recall of this learning model using the same 10-fold cross validation. The precision and recall values determined by Weka are shown in Table 2. The result of our previous experiment can be seen in parentheses. On the whole, the larger data set shows a bit better accuracy in modification complexity prediction, except for RBFNetwork.

Model	Precision(%)	Recall(%)	F-Measure
J48	50.1 (47.5)	49.8 (47.3)	49.9
RBFNetwork	37.9 (39.7)	40.3 (39.6)	38.4
ClassificationViaRegression	46.6 (42.7)	47.4 (45.1)	44.9

Table 2: Precision and recall values of EM.

6.2 Micro-Productivity Profile

We collected the modification complexity values for 1, 2, 3, etc. modification lengths. Among the 254 file modifications, the most frequently modified file was changed 17 times, so we had modification complexity statistics up to the next 16 modifications. Because of the asymmetric distribution of the complexity values, their average would not characterize them well, so the median was calculated for each modification length instead. The Micro-Productivity Profile defined as the rate of effort can be seen for our projects in Figure 5.

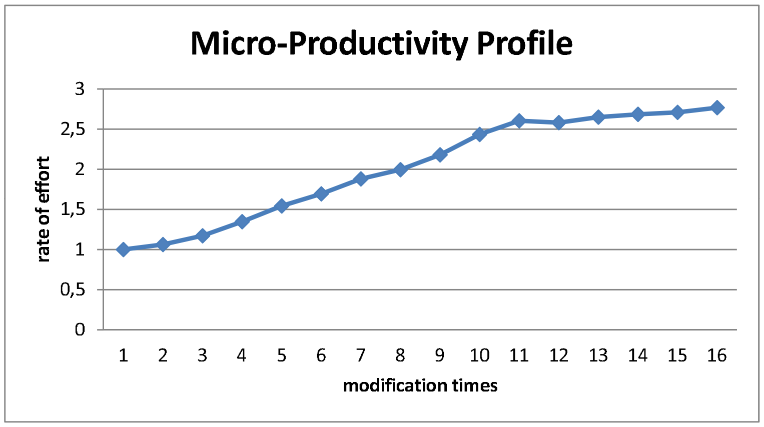

Figure 5: Micro-Productivity Profile.

The Micro-Productivity Profile of the projects is an increasing curve. For shorter sequence lengths the rate of effort increases fast, then for longer modification sequences the growth slows down. Note in particular that the profile relates the number 2 to number 8. For these projects this means that if a file were modified 8 times, the cost of the 8 modification would be twice as much as the cost if the modification would be done in a single step.

6.3 Comparison of estimation methods and the actual data

In the following, we present some examples on how the 3 effort (EM, $EM(x)$ and the actual effort) values are related to each other for certain files. For the measurements we used a task with 42 modifications of 17 files as the basis of the testing data. The most frequently modified files from the 17 files of the testing data are the `SettingProvider.java` and `DataProvider` files with 7 modifications. The predicted and the actual effort values of these files are given in Figure 6. The y axis represents the net time of a change of a line in minutes, while the x axis shows the length of the modification sequence.

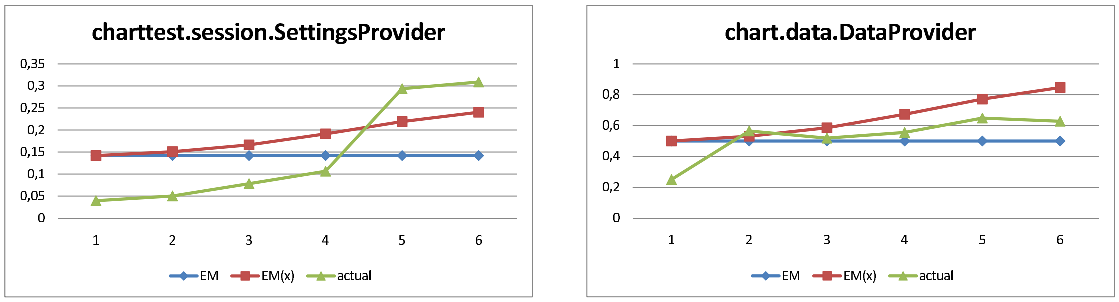

Figure 6: Predicted and actual modification efforts in `SettingsProvider` and `DataProvider` classes.

The naive atomic change based method (EM) assumes constant productivity, while the improved atomic change based method ($EM(x)$) improves the naive approach with MPP. Although EM overestimated the effort of the first modifications in the cases of `SettingsProvider` and `DataProvider`, it gives an average value of all of the actual ones, which is overall a good estimation. By improving the estimation model, the estimation accuracy seems to be the same, however, because of its growing tendency the enhanced model seems to be much closer to the actual values. EM and $EM(x)$ approximate the actual efforts pretty well in these cases.

Using prediction values of all 17 files we determined the distance between the different estimation models and the actual values. While the Euclidean distance between the EM and the actual values is 1.3669, the distance between the EM(x) and the actual values is 1.3289. The lower distance concludes higher estimation accuracy.

6.4 Discussion

In this section we summarize some observations about the results we obtained.

Atomic change estimation: We repeated our previous experiment [13] in a larger data set, and we found that the accuracy of the estimation models (see Table 2) is a bit higher than in the case of our earlier research. The relatively high precision and recall values proved the relevance of the predictor set. Learning precision is over 50%, nearly 60% for the classes which is notable compared to the random classification (33%).

Micro-Productivity: The findings of this research statistically confirm our observation that the cumulative effort to implement a series of changes is larger than the effort that would be needed if we made the same modification in one step.

Micro-Productivity Profile: We quantified the micro-productivity decrease phenomenon by collecting data representing the effort, and determined the measure of effort increase if the modification is done by a certain number of changes (see Figure 5). We observed that for shorter sequence lengths the rate of effort increases fast, then for longer modification sequences the effort growth slows down.

Comparison of atomic change sequence estimations: Finally, we compared the estimated targets of the two kinds of prediction models to the actual efforts. We found that the atomic change based estimation model improved by MPP hence giving more accurate prediction than the naive one.

6.5 Threats to Validity

Undoubtedly, the limited amount of training data and the group of testing data is a risk: we had only 3 projects to retrieve training data and we chose only 1 task to get the testing data. We intend to perform similar experiments in the future on more data.

Drawing general conclusions from this experiment is difficult, because any method depends on a potentially large number of relevant modifications. For this reason, we cannot assume a priori that the results of this study generalize beyond the specific environment, in which it was conducted. However, the results are in line with our assumption.

We are confident that our predictor set (product and process metrics) has predictive power for the maintenance effort in other projects, but there can be other metrics which are not considered in formulating our models which could improve the prediction models.

To predict the modification complexity, we separated the values on which we base our prediction into 3 groups, but this way we got extreme values inside the groups. To avoid this, we could use more than 3 groups, but for this to be possible more training data would be needed.

Managers can estimate the maintenance effort in top-down and bottom-up approaches. The top-down effort estimation is based on the known effort of previous projects, while the bottom-up effort estimation is based on aggregating distinct modification tasks. Our approach is a form of the latter approach, where the effort can be estimated as the sequence of smaller modifications (which also relates to the so-called *decomposition* technique [11]).

7 Related work

Note that the discussion revolves around two topics: productivity decrease and the applicability of machine learning to derive cost models.

Traditional, the well-understood productivity measuring approaches include Putnam's SLIM, Albrecht's FP method of estimation, COCOMO and COCOMO II [3]. Besides these traditional approaches, various machine learning techniques have been evolved. These methods are based on a set of cost drivers, like process (e.g. process maturity), project (e.g. reuse, platform), personnel (e.g. personnel experience), and product measures (e.g. size). An effort estimation model was introduced by Mockus *et al.* [9], which predicts the amount and the distribution of maintenance effort over time. They examined several factors, including the developer, and properties of the change: type, status, size, rate of size and complexity. They were focused on what changes were made and how they were made, not on the properties of the code. Our most important cost drivers are also the net development time and the code churn size.

Most of the approaches deal with macro-productivity, i.e., they try to reason about the (future) behaviour of a system by considering coarse granularity data that is available at higher levels of observation. Premraj *et al.* [10] examined the productivity trends over time with long term objective. They had evidence of productivity improvement over the years and stated that there are significant differences in size and effort between development and maintenance projects. Contrarily, we examined productivity trends with short term objective and to distinguish development and maintenance is only our plan.

Our approach aims at micro-productivity, where fine granularity data is collected and used from lower levels of operation in the projects. Donzelli [5] used data from a real project to show that using a combination of different maintenance practices are needed to maximize maintenance performance. In the area of micro-productivity Junio *et al.* [7] applied k-means clustering algorithm for partitioning and grouping the maintenance requests. By their PASM process the grouping maintenance requests helps to reduce maintenance costs. Although we grouped the maintenance requests not by automatic clustering but continuously coming maintenance requests of a file, we also observed the micro-productivity growth. While in the former case the cost can be reduced by grouping the maintenance requests, our approach shows an inevitable extra cost in maintenance.

According to Abdel-Hamid [1] the actual productivity is the potential productivity decreased with losses due to faulty processes, where the potential productivity is the level of productivity attained when an individual or group makes the best possible use of their available resources, and the faulty processes are typically due to dynamic motivation factors and communication overhead. Imperfect or not deliberating changes are not referred, which also can cause loss.

8 Conclusion

We believe that micro-productivity decrease is a general phenomenon (not even specific to software engineering), which deserves much more attention when software estimation is concerned. The MPP profile computed for our subject development projects shows an eye-opening tendency that, on average, after only 8 atomic modifications the micro-productivity halves. This means that an effort prediction approach based on atomic changes cannot be accurately used to predict larger modifications by simple summing, but the inclusion of MPP profiles is recommended.

We concentrated on what we called atomic change sequence estimation as opposed to change task estimation. Although the available experimental data did not allow us to try learning on task based changes, we took an alternative route with MPP profiles and atomic change sequences, which turned out to be well applicable. However, in the future when we will have significantly more data about change requests we will concentrate on this topic as well.

We presented our experimental approach and framework for effort estimation, which employs machine learning techniques based on various process and product metrics. Despite the relatively good learning accuracy of more than 50% (with a 3-class classification), surely, we have to perform additional experiments on larger data sets in order to further enhance the basic methods and get even better learning results. Our initial model could predict the effort for atomic changes, and after we have observed that the prediction can be made better when incorporating the expected length of the modification, we started to work on incorporating the micro-productivity decrease. Our experiments with a real industrial project showed that a prediction model based on predicting atomic change adjusted by MPP profile shows better approximation to the actually measured data than without MPP profile.

Further research is required to validate the approach in different situations and environments, and there are several ways to improve or extend our current results. We are interested in how these MPP profiles look like on different projects, with different development models (e.g. linear, agile, evolutionary, open or closed source). We also expect to find different profiles for different phases of a project, like initial development, evolution, or maintenance. Finally, it would also be interesting to see whether individual developers have their own profiles based on their experience and skill.

9 Acknowledgments

The work presented in this paper was partly funded by the National Innovation Office, Hungary (project No. OM-00191/2008-AALAMSRK 'ProSeniis'), Telenor 19/55 1I066, Nokia Komarom 19/55 1C117, GOP-1.1.2-07/1-2008-0007 and OTKA K-73688.

References

[1] Tarek K. Abdel-Hamid. The slippery path to productivity improvement. *IEEE Softw.*, 13:43–52, July 1996.

[2] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches - a survey. *Ann. Softw. Eng.*, 10:177–205, January 2000.

[3] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

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

[5] Paolo Donzelli. Tailoring the software maintenance process to better support complex systems evolution projects. *Journal of Software Maintenance*, 15:27–40, January 2003.

[6] Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus – reverse engineering tool and schema for C++. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)*, pages 172–181. IEEE Computer Society, October 2002.

[7] Gladston Aparecido Junio, Marcelo Nassau Malta, Humberto de Almeida Mossri, Humberto T. Marques-Neto, and Marco Tulio Valente. On the benefits of planning and grouping software maintenance requests. CSMR '11, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, 23:111–122, November 1993.

[9] Audris Mockus, David M. Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *In 2003 International Conference on Software Engineering*, pages 274–284. ACM Press, 2002.

[10] Rahul Premraj, Martin Shepperd, Barbara Kitchenham, and Pekka Forselius. An empirical analysis of software productivity over time. In *Proceedings of the 11th IEEE International Software Metrics Symposium*, pages 37–, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Roger S. Pressman. *Software Engineering, A Practitioner's Approach.* McGraw Hill, 7th edition, 2010.

[12] Vaclav Rajlich and Prashant Gosavi. Incremental change in object-oriented programming. *IEEE Softw.*, 21:62–69, July 2004.

[13] Gabriella Tóth, Ádám Zoltán Végh, Árpád Beszédes, and Tibor Gyimóthy. Adding process metrics to enhance modification complexity prediction. In *Proceedings of the IEEE International Conference on Program Comprehension (ICPC 2011)*, pages 201–204.

[14] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.