A Cost Model Based on Software Maintainability

Tibor Bakota, Péter Hegedűs, Gergely Ladányi, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy
University of Szeged
Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{bakotat,hpeter,lgergely,kortve,ferenc,gyimothy}@inf.u-szeged.hu

***Abstract*—In this paper we present a maintainability based model for estimating the costs of developing source code in its evolution phase. Our model adopts the concept of entropy in thermodynamics, which is used to measure the disorder of a system. In our model, we use maintainability for measuring disorder (i.e. entropy) of the source code of a software system. We evaluated our model on three proprietary and two open source real world software systems implemented in Java, and found that the maintainability of these evolving software is decreasing over time. Furthermore, maintainability and development costs are in exponential relationship with each other. We also found that our model is able to predict future development costs with high accuracy in these systems.**

***Index Terms*—Software maintainability, development cost estimation, cost prediction model, ISO/IEC 9126, ISO/IEC 25000**

I. Introduction

Maintainability is generally defined as the effort (i.e. cost) required to perform specific modifications in a software [1]. Being in direct connection with cost, maintainability is one of the most important quality characteristics of a software system. Exploring the relationship of software maintainability and development cost is one of the central issues in software maintenance activities today. However, because of the nonexistence of formal definitions and the subjectiveness of the notion of maintainability, there is currently no common understanding among researchers about its relation to development costs.

Thanks to the abundance of existing software quality models [2]–[6], estimating software maintainability is not the most important challenge nowadays. Although these models may differ a lot in the approach how maintainability is computed, they all rest on capturing low level attributes of source code (e.g. metrics) and aggregating this information to obtain a single measure for maintainability. Very little is known about how these models correlate with each other and actually, answering this question is beyond the scope of this paper. However, software quality models play a crucial role in our current work.

In this paper we present a very simple model for relating development costs to the maintainability of the source code. In our approach, we adopt the concept of entropy in thermodynamics, which is used to measure the disorder of a system. In the case of software systems, maintainability is an appropriate candidate for measuring disorder (i.e. entropy).

Our model is based on two simple assumptions:

1) When making changes to a software system without explicitly aiming to improve it (like e.g. adding new functionalities), its maintainability will decrease (i.e. its disorder will increase), or at least it will remain unchanged.
2) Performing changes in a software system with lower maintainability (i.e. higher disorder) is more expensive.

Only these two assumptions were used to derive a system of equations which serve as a model for relating maintainability to development cost. We introduce the notion of *erosion factor*, which is a vital parameter of the model, that measures the amount of "damage" caused by changing source code lines of a software. As we will show later in Section III, the erosion factor may also serve as a measure for process quality. Model parameters can be computed from historical data, like development costs in the past. After the estimates for the parameters are available, predictions for the future can be obtained from the model.

We evaluated the model on five software systems implemented in Java programming language. Three of these are commercial closed-source systems. In order to facilitate the repeatability of the experiments, we performed the analysis on two open source systems as well. All data are available as an online appendix to this paper at: *http://www.inf.u-szeged.hu/∼ferenc/papers/ICSM2012*

Our findings can be summarized as follows:

- The maintainability of evolving software is decreasing over time, which is also in accordance with Lehman's laws [7].
- Maintainability and development costs are in exponential relationship with each other.
- The presented model is able to predict future development costs based on estimated change rate of the code, with high accuracy.

The paper is organized as follows: in the next section we give a brief overview of other studies which are related to ours. Next, in Section III we present the details of our approach. In Section IV we present the results of the evaluation of our model on real world systems. Afterwards, we give a brief overview regarding the threats to validity in Section V. Finally, we round up with conclusions and future work in Section VI.

II. Related Work

Modeling software development costs has been an intensive research area for a long time [8]–[12]. Effort estimation is important not just for software developers [13] but for system operators, as well [14]. While summaries of research results achieved in the last thirty years are available [15], [16], the field still shows very promising undiscovered areas [17]. Relevant comparison studies of different techniques [18] in various domains [19]–[23] also exist.

As an analogy to the notion of entropy in thermodynamics, *software entropy* was first introduced by I. Jacobson et al. [24] for measuring the disorder of a software system. The second law of thermodynamics, in principle, states that a closed system's disorder cannot be reduced, it can only remain unchanged or increase. This law also seems plausible for software systems; as a system is modified, its disorder or entropy always increases. In accordance with that, Lehman stated that software, which is being used needs to be changed, and the changes result in an increased complexity and decrease of quality [25].

Canfora et al. [26] used software entropy when examining changes in ArgoUML[1] and Eclipse.[2] First, the authors computed source code metrics, various design patterns and different process metrics. They approximated the cost by counting the number of contributors to file changes. They found that different types of changes may contribute either negatively or positively to the entropy. Namely, refactoring decreases entropy, while feature development usually increases it. They also showed that entropy tends to increase with the number of contributors to file changes.

Bianchi et al. [27] investigated software entropy using bug information. They showed that constantly changing software systems are affected by degradation. The authors collected a dataset from individual groups of university students responsible for developing systems with a predefined functionality. The dataset consisted of bug reports from various development stages, which contained the number of found and slipped bugs as well as the amount of time spent on fixing them. They found that the more time was spent on bug correction, the harder it was to correct newly appeared defects. This study correlates to ours; although we used product metrics instead of process ones. Moreover, they found that there was high correlation between the decrease of source code maintainability and the number of faults in the analysis and design phases.

Hanssen et al. [28] examined software entropy in agile product development. An industrial study was conducted at a company with 60 developers. The code being developed was continuously monitored by a third party consultant using an automated toolchain. They found that code entropy highly affected agile processes, and development tasks took longer time when code complexity increased due to entropy. Moreover, in the short time iterations of the agile model, the resources needed to detect and resolve coding issues were insufficient. Instead of lengthening the iterations, the authors gave a viable solution to overcome code entropy. The authors stated that refactoring helps to overcome entropy problems in an agile environment. Our findings are similar, though we stated them generally for all kinds of development methodologies as the processes were not taken into account during our experiments. They suggested using continuous and automated code smell detection and refactoring to preserve maintainability through the iterations.

The ways of conducting development effort estimation [29] range from experience [18] through benchmark [30] to model based approaches. Several methods using a mixture of the above techniques also exist [31], [32]. Many researches are aimed at comparing the different approaches.

In a recent study, Dubey at al. [33] defined a model based on object-oriented metrics for maintainability analysis. The authors used the maintainability definition derived from the ISO/IEC 9126 standard [1]. They proposed high level properties (e.g. fault proneness, defect density, etc.) of a software that affect maintainability. An extensive review of existing methodologies regarding high level characteristics was performed by the authors. They found that source code metrics are highly usable for quantifying software maintainability based on ISO/IEC 9126 standard. They created a non-benchmark based approach by using the Chidamber & Kemerer object-oriented metrics suite [34]. No experimental validation of the model was conducted. We used our previously published model [4] based on source code metrics to quantify maintainability. Our model is using a benchmark database of the source code measurement data of many systems from various domains.

Radlinski and Hoffman [35] compared 23 machine learning algorithms using local data as a benchmark for development effort prediction. The four datasets used for the experiments contained both process and product information, but lacked source code metrics (except KLOC, measuring the lines of code). The accuracy of the algorithms was highly dependent on the set of predictors, therefore no universal machine learning algorithm was found by the authors. Instead, they found that running several algorithms using arbitrary predictors could serve as a good starting point for project managers on estimating development costs.

Several comprehensive studies summarize the effectiveness of various cost prediction methodologies.

Riaz et al. [36] examined 710 studies and selected 15 for an in-depth analysis. Their aim was to evaluate the ability of existing methods to measure maintainability. The paper gives a thorough overview of the interpretations of maintainability and summarizes the most commonly used techniques for defining it. The authors analyzed the measures used by these approaches and also the accuracy of the prediction results. A systematic review of research questions found in the papers was performed. The answers of these questions were used to grade the effectiveness of the studies based on a system of criterions. They found that there is little evidence on the effectiveness of maintainability prediction models.

[1] http://argouml.tigris.org/
[2] http://www.eclipse.org/

Mair et al. [37] examined 171 papers regarding analogy and regression based techniques for cost estimation. They proposed a broad selection of criterions for defining the effectiveness of the different approaches. They found that regression models performed poorly, while analogy based methods were much superior. In many cases, the two different methods provided conflicting results on the same dataset. Case based reasoning using a benchmark database gave fine results in general, but exceptions were also found. Due to the lack of standardization in software quality assurance, no universally good method was found by the authors for conducting software cost estimation.

III. Approach

Our approach presented in this section is based on software entropy. In thermodynamics, entropy is a measure of the disorder of a system. According to the second law of thermodynamics, the entropy of a closed system cannot be reduced; it can only remain unchanged or increase. The only way to decrease entropy (disorder) of a system is to apply external forces, i.e. to put energy into making order.

We will apply the notion of entropy in a very similar way for software systems. Maintainability of a source code is usually defined as a measure of the effort required to perform specific modifications in it. Assuming that the higher the disorder is, the more effort is needed to perform the modifications, maintainability can be interpreted as a measure of the disorder, i.e. entropy of the source code.

Our approach lays on two basic assumptions, which we mentioned already in the Introduction section:

1) Making changes in a source code does not decrease the disorder of it, provided that one does not work actively against this. In other words, when making changes to a software system without explicitly aiming to improve it, its maintainability will decrease, or at least it will remain unchanged.
2) The amount of changes applied to the source code is proportional to the effort invested, and to the maintainability of the code. In other words, if one applies more effort, the code will change faster. Additionally, a more maintainable code will change faster, even if the applied effort is the same. Another interpretation is that the effort aiming on code change is inversely proportional to the maintainability at a particular time t.

Before formalizing these assumptions, we introduce the following notions:

- $\mathcal{S}(t)$ - the size of the source code at time *t*, measured in lines of code.
- $\lambda(t)$ - the change rate of the source code at time *t*, i.e. the probability of changing any line independently. $\mathcal{S}(t)\lambda(t)$ equals the number of lines changed at time *t*.
- k - a constant for the conversion between different units of measure. Our approach deals with two scalar measures: maintainability and cost. We do not fix particular units of measure for each, instead we introduce the conversion constant k. In the sequel, we may assume without the loss of generality, that cost is expressed by any measure of effort, e.g. salary, person month, time, etc., while maintainability may have any other scalar measure. In practice, after fixing the measures of unit for each, k can be estimated from historical project data.
- $\mathcal{C}(t)$ - the cost invested into changing the system until time *t*, measured from an initial time $t = 0$. Obviously, $\mathcal{C}(0) = 0$.
- $\mathcal{M}(t)$ - maintainability (i.e. disorder) of the system at time *t*.

In the following, we assume that modifications do not explicitly aim on code improvement, meaning that only new functionality is being added to the system and no refactoring or other explicit improvements are done. In this case, the first assumption above can be formalized as follows:

$$\frac{d\mathcal{M}(t)}{dt} = -q\mathcal{S}(t)\lambda(t) \qquad (q \geq 0), \tag{1}$$

meaning that the decrease rate of maintainability is proportional to the number of lines changed at time *t*. The constant factor q is called the *erosion factor* which represents the amount of "damage" (decrease in maintainability) caused by changing one line of the code. The erosion factor depends on many internal and external factors like the experience and knowledge of the developers, maturity of development processes, quality insurance processes used, tools and development environments, the programming language, the application domain, etc. The $q \geq 0$ assumption makes it impossible for the code to improve by itself just by adding new functionality. The assumption is in accordance with Lehman's laws of software evolution, which state that the complexity of evolving software is increasing, while its quality is decreasing at the same time.

Formalizing the second assumption, leads us to the following equation:

$$\frac{d\mathcal{C}(t)}{dt} = k\frac{\mathcal{S}(t)\lambda(t)}{\mathcal{M}(t)}. \tag{2}$$

The nominator represents the amount of change introduced at time t. The formula states that the utilization of the cost invested at time t for changing the code is inversely proportional to maintainability.

Solving the above system of ordinary differential equations, yields the following result:

$$\mathcal{C}(t_1) - \mathcal{C}(t_0) = \int_{t_0}^{t_1} k\frac{\mathcal{S}(t)\lambda(t)}{\mathcal{M}(t)}dt = -\frac{k}{q}\int_{t_0}^{t_1}\frac{\dot{\mathcal{M}}(t)}{\mathcal{M}(t)}dt =$$

$$= -\frac{k}{q}\left[\ln\mathcal{M}(t_1) - \ln\mathcal{M}(t_0)\right] = -\frac{k}{q}\ln\frac{\mathcal{M}(t_1)}{\mathcal{M}(t_0)}. \tag{3}$$

By expressing $\mathcal{M}(t)$ from the above equation, we get to the main result:

$$\mathcal{M}(t_1) = \mathcal{M}(t_0)\, e^{-\frac{q}{k}(\mathcal{C}(t_1)-\mathcal{C}(t_0))}, \tag{4}$$

which suggests that the maintainability of a system decreases exponentially with the invested cost to change the system.

The erosion factor q determines the decrease rate of maintainability. It is obvious that for a higher erosion factor the decrease rate will be higher as well. It is crucial for software development companies to push the erosion factor as low as possible, for instance by training the employees, improving processes, utilizing sophisticated quality assurance technologies.

Although, the formula does not provide a way of having an absolute measure for maintainability, one can easily define a *relative maintainability* for the system. Indeed, by letting $t_0 = 0$, and defining $\mathcal{M}(0) = 1$, we get to the following function for maintainability:

$$\mathcal{M}(t) = e^{-\frac{q}{k}\mathcal{C}(t)} \tag{5}$$

For the interpretation, let us consider two artificial scenarios. Figure 1 shows the case, when the invested effort is constant over the time. In this case, both the maintainability $\mathcal{M}(t)$ and the change rate $\lambda(t)$ decrease exponentially.

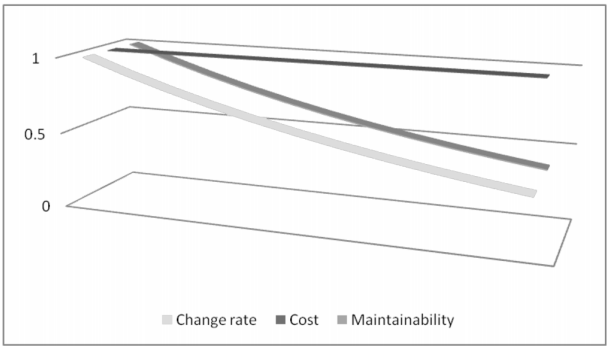

Figure 1. Changes of *Change rate* ($\lambda(t)$) and *Maintainability* ($\mathcal{M}(t)$) when the cost of the development ($\mathcal{C}(t)$) is constant over time.

In the other case, let us suppose that one intentionally wants to keep the change rate of the system constant. Figure 2 shows how the maintainability $\mathcal{M}(t)$ and the overall cost $\mathcal{C}(t)$ change over time. Now, the maintainability decreases linearly until it reaches zero, while the cost is increasing faster than an exponential rate. The cost will reach infinity in finite time, exactly when maintainability reaches zero, meaning that any further change would require infinite amount of effort. This is, of course, just a theoretical possibility, as no one disposes an infinite amount of resources required to degrade the maintainability of a system to absolute zero.

The problem with applying the model to real-world software systems lies in the erosion factor q. While the other model parameters (k and $\mathcal{C}(t)$) can be computed easily, the erosion factor, which measures the "damage" caused by changing one line, is challenging. Contrarily, if there was an absolute measure of maintainability, the constant, project specific erosion factor q could easily be computed by expressing it from Equation 5. Furthermore, by having an absolute measure for q as well, the erosion factors of different projects, organizations could be compared. The analysis of the causes of

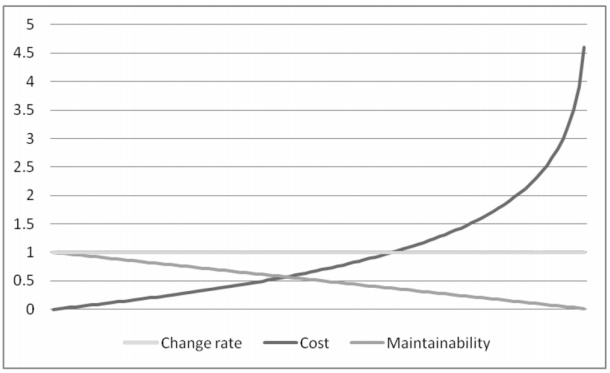

Figure 2. Changes of *Cost* ($\mathcal{C}(t)$) and *Maintainability* ($\mathcal{M}(t)$) when the *Change rate* ($\lambda(t)$) is constant over time.

the differences would make it possible to lower the erosion factor, e.g. by improving the processes, training people, etc. In addition, the overall cost of development could also be expressed explicitly from the model:

$$\mathcal{C}(t) = -\frac{k}{q}\ln\left|1 - \frac{q}{\mathcal{M}(0)}\int_0^t \mathcal{S}(s)\,\lambda(s)\,ds\right|. \tag{6}$$

For computing future development costs, it would just be required to have an estimate for the change rate $\lambda(t)$ over a time period.

In one of our earlier papers [4] we presented an approach for obtaining an absolute measure of maintainability for software systems. We used source code metrics and benchmarks to probabilistically approximate a benchmark independent measure of maintainability. In this paper, we use this approach for computing absolute maintainability, to obtain an absolute erosion factor q, which can be used to estimate further development costs and to compare the erosion factors of different projects and organizations.

IV. Validation Results

In order to evaluate the presented cost model, we analyzed a large number of consecutive versions of five different Java projects. Three of these are commercial, closed source systems, which will be referred to as *System-1*, *System-2* and *System-3*. To facilitate the repeatability of the experiments, we performed the analysis of two open source systems as well. Some of the relevant details about the analyzed systems are listed in Table I.

Table I
Properties of the analyzed systems

System	Nr. of revisions	First date	Last date	System size[3] interval	Nr. of authors
System-1	149	06/03/2011	01/31/2012	14175-24861	7
System-2	357	05/09/2008	03/09/2010	53262-143017	21
System-3	641	11/05/2010	10/12/2010	128653-148903	12
jEdit[4]	1370	09/02/2001	07/25/2006	30986-96203	18
log4j[5]	1889	12/14/2000	08/15/2007	1464-25642	17

[3]Measured by the total of non-empty non-comment lines of code (TLLOC)
[4]https://jedit.svn.sourceforge.net/svnroot/jedit/jEdit/branches/4.5.x
[5]http://svn.apache.org/repos/asf/logging/log4j/branches/BRANCH_1_3

The reader may have noticed that the model presented in this paper deals with parameters which must be approximated in order to work. For computing the k and q parameters of the model, one needs to know $\mathcal{C}(T_0)$ for some $T_0 > 0$, i.e. the cost of development until time T_0. This can usually be estimated by using historical project records, but can also be approximated in other ways. After k and q are computed for some time T_0 (i.e. the model is trained), the model can be used to make predictions for $\mathcal{C}(t), t > T_0$. Unfortunately, historical records regarding the development costs were not available in any of the cases. Therefore, in order to conduct the evaluation, we were forced to make assumptions regarding the costs: we assumed that the costs of the development are proportional to the elapsed time. Provided that, in case of industrial systems, fixed teams are usually working on a project with relatively little variations in their size, the assumption does not seem too restricting. Unfortunately, this might not be the case with open source systems: there is usually no stakeholder enforcing steady expectations regarding the invested effort. We treat the case of open source systems as a threat to validity because of this assumption.

We performed the evaluation according to these steps:

1) First, we checked out every revision of the source code of each system from their configuration management systems.
2) We calculated the maintainability of each source code revision by using our probabilistic software quality model [4]. We used this number as an approximation of $\mathcal{M}(t)$.
3) For each source code revision, we computed the number of changed source code lines (added, deleted and modified), compared to the previous revision. The number obtained in this way is exactly equal to $\mathcal{S}(t)\lambda(t)$, therefore computing $\mathcal{S}(t)$ explicitly is not necessary.
4) We computed estimates for k and q from Equation 2 and Equation 5, respectively, at some time $T_0 > 0$. The estimates for k and q are the following:

$$k = \mathcal{C}(T_0)\left(1/\int_0^{T_0}\frac{\mathcal{S}(t)\lambda(t)}{\mathcal{M}(t)}dt\right), \tag{7}$$

and

$$q = -\frac{k}{\mathcal{C}(T_0)}\ln\frac{\mathcal{M}(T_0)}{\mathcal{M}(0)}. \tag{8}$$

5) These estimates, being constants according to our model, are valid for time $t > T_0$, and can be used to make predictions by using Equation 6. The predicted costs will be denoted by $\tilde{\mathcal{C}}(t)$.

For computing the number of modified lines of code, in step three, we used a heuristic algorithm that combines *diff*s returned by the SVN client. We consider it as a threat to validity as well.

Different aspects of our findings are summarized in the following subsections.

A. The maintainability of evolving software is decreasing over time

The dark lines on the right hand side diagrams in Figure 3 show how maintainability $\mathcal{M}(t)$ changes as a function of time t, measured in number of revisions. All of the charts show a decreasing tendency of maintainability as more effort is put in the development of these systems. To confirm the intuitive feeling of decreasing functions, the linear regression lines and their equations are also visible on the diagrams. All the coefficients of x being negative, the average decrease of maintainability follows in every case, which is in accordance with Lehman's laws [7].

B. Maintainability and development costs are in exponential relationship with each other

Let $\tilde{\mathcal{M}}(t)$ denote the predicted maintainability computed by using Equation 5 and $\tilde{\mathcal{C}}(t)$ (the cost function predicted by the model). Clearly, $\tilde{\mathcal{M}}(t)$ decreases exponentially as a function of $\tilde{\mathcal{C}}(t)$. It is sufficient to show that the real cost $\mathcal{C}(t)$ highly correlates with the predicted cost $\tilde{\mathcal{C}}(t)$ and real maintainability $\mathcal{M}(t)$ with the predicted maintainability $\tilde{\mathcal{M}}(t)$ for some k and q constants. It would mean, that for some parameters, the model describes the real world relatively well. Consequently, measured maintainability would be decreasing exponentially as a function of real costs (at least with high correlation).

We may compute estimates for any time $T_0 > 0$ as suggested in step 4 above. Obviously, for larger T_0 values, the estimates are better, provided that more historical data is available for training the model. By taking the last revisions, i.e. the biggest possible T_0, we obtain the best estimates for k and q. These constants are then used to compute $\tilde{\mathcal{C}}(t)$ and $\tilde{\mathcal{M}}(t)$ for any $t \geq 0$.

The left-hand side diagrams in Figure 3 show both $\mathcal{C}(t)$ (dark) and $\tilde{\mathcal{C}}(t)$ (light) functions. On the right-hand side, the dark lines visualize the changes of $\mathcal{M}(t)$, while the light ones show $\tilde{\mathcal{M}}(t)$. The diagrams also show the Pearson's correlations between the real and the predicted curves. The high correlations indicate, that both costs and maintainability are well described by the model, at the same time. It follows, that maintainability and cost are in exponential relationship with each other in a way prescribed by the model, with a high correlation. In case of *System-3* and *log4j* the correlations are slightly worse than in the other cases. The reason of that might be that the time period of the analysis was relatively short, and lots of refactoring work was done, according to the SVN logs.

C. The presented model is able to predict future development costs based on change rate of the code, with high accuracy

In the previous subsection we showed that the model parameters k and q can be chosen such that the model describes real world costs and maintainability with high correlation. Figure 4 shows the estimated k, q and q/k values for every system.

Based on the diagram, the most damage is caused in *System-1* when changing one line, as the erosion factor q is the largest in this case. This might be due to the rapid and intensive development of *System-1* during the analyzed period.

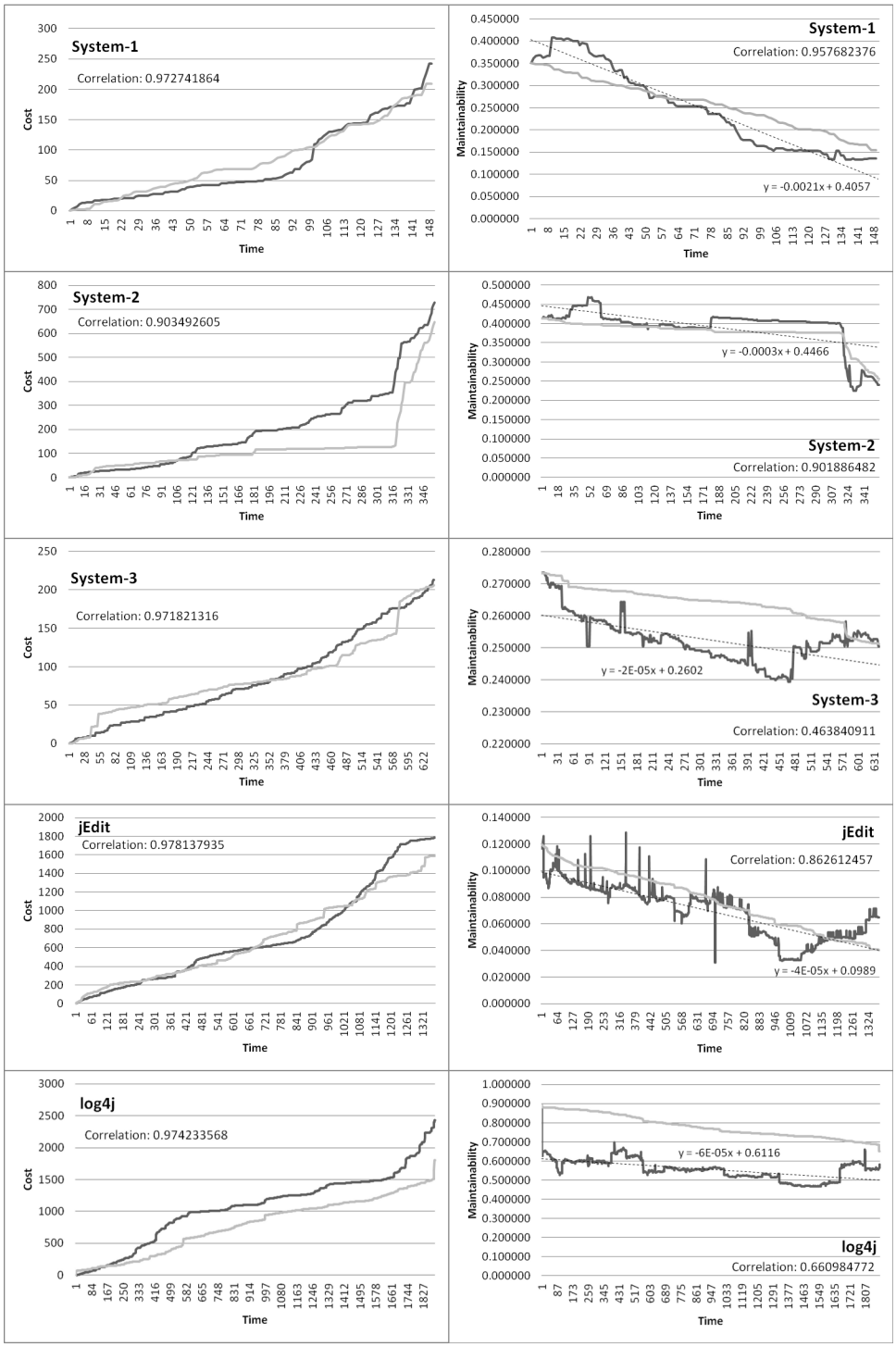

Figure 3. Estimated and real costs and maintainability as functions of time

This is also the system whose maintainability decreases the most, provided that the applied effort is the same, because the q/k is also the largest in this case. The conversion constant k is the largest for *log4j*, meaning that same amount of effort induces less amount of changes compared to the other systems.

For estimating the cost of a new development, Equation 2 of the model requires to have an estimation of the total amount of lines that will change, and the function describing maintainability change in the future. Although the total number of changes can be estimated in advance, based on requirement

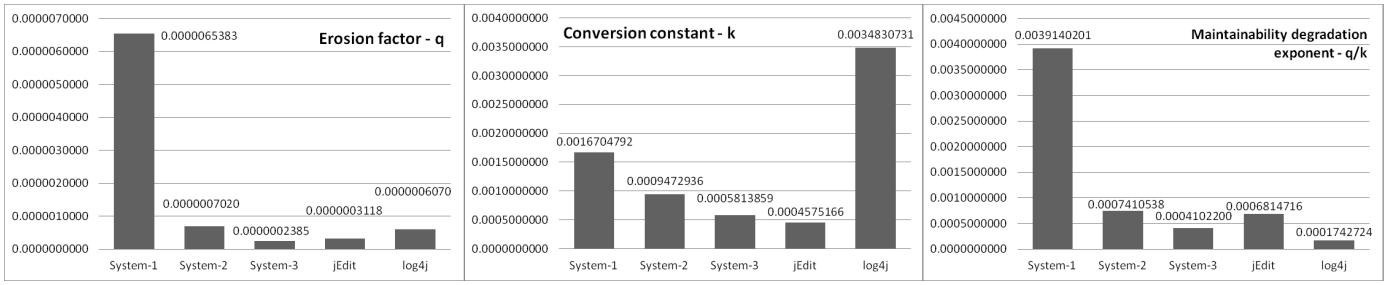

Figure 4. The calculated constant values for the different systems

and impact analysis [38], the maintainability is obviously unavailable before the changes would have been committed, and maintainability would have been measured. Fortunately, the erosion factor introduced by Equation 1 in Section III makes it possible to approximate future maintainability based on estimated change rates. Future development costs can be computed using Equation 6, without having to know the change of maintainability in advance.

To validate the prediction power of our model, we performed future estimations with different window sizes measured in time. For a particular window size $n > 0$, we used the model to compute the estimated cost at time t, based on the already known cost at $t - n\,(\geq 0)$ and the planned amount of changes between $t - n$ and t. In other words, at time $t-n$ we are trying to estimate the overall cost at time t, by knowing the overall cost until time $t - n$ and the planned amount of future changes. In this way, for a particular window of size n we obtain a sequence of predicted costs, for time $n + 1, n + 2$, etc. Window sizes vary form 1 to the largest possible ones, i.e. the number of revisions available. When the window size is 1, it means that the development cost of a revision is being approximated based on the previous revision, and the changes between them. In the case of the largest possible window, the overall development cost of the whole period is estimated based on the initial cost (which is zero), and the future changes. For every window size, we computed both the *mean squared error* [39] and *Pearson's correlation* between the real costs and the ones predicted by the model.

For comparability reasons, we also performed another, classical type of cost estimation. Namely, for estimating future costs, we computed the average cost of a change up to time $t - n$, then interpolated the future cost by multiplying the average change cost with the amount of overall change till time t. In other words, we computed the average change cost based on historical data, and expected it to remain the same in the future. This classical model, differs from ours, as it does not take the change of maintainability over time into account, which makes the changes more and more expensive. We will refer to this classical type of cost estimation as *linear prediction*.

The left-hand side diagrams in Figure 5 show how the mean squared errors (*MSE*) behave for various window sizes, while on the right-hand side the correlations of the predicted and real costs are visible. In both cases, the x-axis stands for the size of the window, measured in number of revisions, i.e. time, and the y-axis shows the MSE and Pearson's correlation values, respectively.

Seemingly, both models become more and more precise for larger window sizes, but this happens only because the prediction sequences are getting shorter. For example, in the case of the largest possible window size, only one cost value is predicted, the last one.

It can be seen that the predictions made by the presented model outperform the classical linear model, which does not take the changes of maintainability into account. The differences are especially noticeable for larger window sizes, i.e. long term predictions. Actually, it is a natural phenomenon, because changes of maintainability are more significant in longer periods of time.

V. Threats to validity, limitations

First of all, our cost model is based on two assumptions described in Section I. If these assumptions do not hold, our cost model may be invalid. However, our practical experience and the feedbacks from our industrial and research partners show that these assumptions are reasonable.

Another threat to the validity is that we assume that the k conversion and q erosion factors in the model are constants. It might be possible that these factors are changing over time in reality. But even if this is the case, it only means that further improvements in the prediction model can be achieved. Our aim was to validate a new approach for software maintainability based cost estimation and not to model it in full detail. The presented model is a simplistic cost model that appears to be very expressive in its current form according to the empirical results shown in Section IV.

Due to the lack of real data, we had to apply heuristics several times in the work. For calculating the total amount of changed lines between two revisions of the system, we used the SVN diff command that returns only added and removed lines. Modified lines are presented by consecutive inserted and deleted lines. Although our algorithm for calculating modified lines might not be totally precise, it does not affect the achieved results too much. Our experiments show that we get

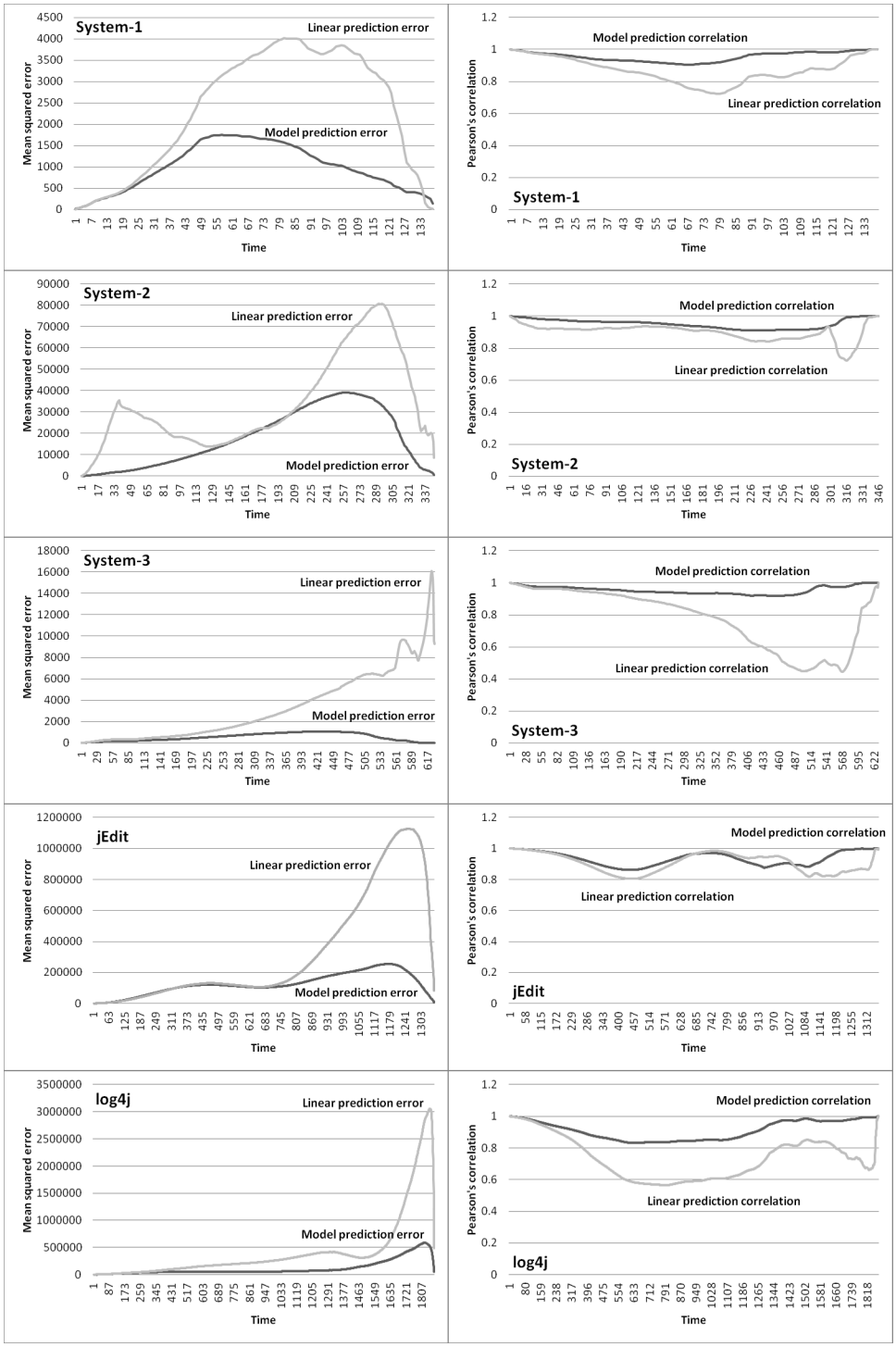

Figure 5. The mean squared errors and correlations of the linear and model predictions

similarly good results using only the number of inserted lines as a measure of total changes.

Not being able to collect real efforts from tracking systems, we assumed that the amount of invested cost to the development is proportional to the elapsed time. The reason behind this is that usually there is a constant team developing the software, putting a constant amount of effort into the development. This was the case for the three proprietary systems that we analyzed but a possible threat to validity might be that this assumption is not valid for open source systems.

Another major threat to validity is using our previously published model for calculating maintainability values. Although we have done some empirical validation on our probabilistic quality model in our previous work, we cannot state that the used maintainability model is fully validated. Moreover, as the ISO/IEC 9126 standard is not defining the low-level metrics, the results can vary depending on the quality model's settings (chosen metrics and weights given by professionals). It is also very important to have a source code metrics repository with a large enough number of systems to get an objective absolute measure for maintainability. The maintainability model uses only source code metrics and no process metrics at all. All these factors are possible threats to validity, but our first results and continuous empirical validation of the maintainability model proves its applicability and usefulness.

Seemingly, the model cannot deal with refactoring and other improvements, as it assumes that only pure feature developments are allowed. Fortunately, considering these activities as part of the development or quality assurance processes, which are meant to moderate quality degradation, they are implicitly encoded in the erosion factor q. In particular, q is smaller in cases when refactoring and other improvements are performed regularly or even occasionally. Therefore, we do not consider this as a threat to validity.

A major limitation of the approach is that the predictions are made based on the amount of changes of lines in the system, which makes the model less useful in practice. This restriction follows from the simplicity of the model. However, the model can be easily altered to use function points instead of line changes, yielding a more practical prediction model. We chose to use line changes, because they can easily be extracted from a configuration management system, which is not the case with function points.

VI. Conclusions and Future Work

In this paper we presented a cost model based on software maintainability. The model describes the relationship between the change rate of the source code, the maintainability of the system and the costs of development. Measuring software maintainability is one of the core parts of our cost model. For measuring the maintainability we applied the probabilistic quality model introduced in our previous work [4].

To validate our approach we analyzed five Java systems (three proprietary and two open source) and collected a vast amount of data. Altogether 4,396 different revisions of the systems and more than 1 million lines of code changes were examined. We also checked all the change set logs of those commits that caused a larger improvement in the maintainability of the system manually. In most of the cases some kind of refactoring (e.g. component replacing) caused the improvement, which further strengthens our theory that pure development does not improve the maintainability of the software. The analysis of the empirical data shed light to the following important findings:

- The maintainability of an evolving software is decreasing over time.
- Maintainability and development costs are in exponential relationship with each other.
- The presented model is able to predict future development costs based on estimated change rate of the code, with high accuracy.

Although our model and analysis process contains some threats to validity, we believe that the results are very valuable, and reflect the theoretical connection between development costs and software maintainability.

Of course, our presented work is just a small step in the direction of connecting software maintainability with development costs, but we have lots of plans for future work to further improve our cost and maintainability models. First of all, for making our results more general, we would like to repeat the empirical analysis on more proprietary and open source systems.

In the current approach we consider the q erosion and k conversion factors as constants. Although this simplifies our model, in reality these numbers might change over time (e.g. by changing development processes or changes in the economy). Further investigation and modeling of these values is planned to improve the precision of our cost model.

It would also be interesting to compare the erosion factors of software systems developed by different companies. The question is how these factors are related to different company-specific processes and regulations, the nature of the development, the domain of the developing systems, etc. Based on the findings we could introduce cost predictors based on processes of the developers (according to CMM [40] or CMMI) based on the early work of Knox [41] and taking into account governance, based on the work of Capra et al. [42]. We could also examine the correlation of the invested effort with the applied process maturity of the company.

As the model for estimating software maintainability plays an important role in our approach, improving it may also help enhancing the cost model. A possible improvement to the maintainability model is to make it compatible with the ISO/IEC 25000 standard, the successor of ISO/IEC 9126. Another possibility is to extend the input of the model with process metrics and other important aspects of software maintainability.

Acknowledgements

This research was supported by the Hungarian national grants GOP-1.1.1-11-2011-0038, GOP-1.1.1-11-2011-0006, and OTKA K-73688.

References

[1] ISO/IEC, *ISO/IEC 9126. Software Engineering – Product quality*. ISO/IEC, 2001.

[2] T. Kuipers and J. Visser, "Maintainability Index Revisited - position paper," in *System Quality and Maintainability, satellite of CSMR 2007*. IEEE Computer Society Press, 2007.

[3] J. P. Correia and J. Visser, "Certification of Technical Quality of Software Products," in *Proc. of the Int'l Workshop on Foundations and Techniques for Open Source Software Certification*, 2008, pp. 35–51.

[4] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy, "A probabilistic software quality model," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, sept. 2011, pp. 243 –252.

[5] E. Georgiadou, "Gequamo - a generic, multilayered, customisable, software quality model," *Software Quality Control*, vol. 11, no. 4, pp. 313–323, Nov. 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1025817312035

[6] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard, "An activity-based quality model for maintainability," in *Proceedings of the 23rd International Conference on Software Maintenance (ICSM 2007). IEEE Computer Society*. Society Press, 2007.

[7] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution - the nineties view," in *Proceedings of the 4th International Symposium on Software Metrics*, ser. METRICS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 20–. [Online]. Available: http://dl.acm.org/citation.cfm?id=823454.823901

[8] A. Albrecht and J. Gaffney, J.E., "Software function, source lines of code, and development effort prediction: A software science validation," *Software Engineering, IEEE Transactions on*, vol. SE-9, no. 6, pp. 639 – 648, nov. 1983.

[9] Y. Miyazaki and K. Mori, "Cocomo evaluation and tailoring," in *Proceedings of the 8th international conference on Software engineering*, ser. ICSE '85. Los Alamitos, CA, USA: IEEE Computer Society Press, 1985, pp. 292–299. [Online]. Available: http://dl.acm.org/citation.cfm?id=319568.319657

[10] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, pp. 416–429, May 1987. [Online]. Available: http://doi.acm.org/10.1145/22899.22906

[11] A. Cuelenaere, M. van Genuchten, and F. Heemstra, "Calibrating a software cost estimation model: why and how," *Information and Software Technology*, vol. 29, no. 10, pp. 558 – 567, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0950584987900899

[12] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: Cocomo 2.0," *Annals of Software Engineering*, vol. 1, pp. 57–94, 1995, 10.1007/BF02249046. [Online]. Available: http://dx.doi.org/10.1007/BF02249046

[13] N. Nan and D. Harter, "Impact of budget and schedule pressure on software development cycle time and effort," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 624 –637, sept.-oct. 2009.

[14] B. W. Boehm, "Software engineering economics," *Software Engineering, IEEE Transactions on*, vol. SE-10, no. 1, pp. 4 –21, jan. 1984.

[15] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, "Software quality models: Purposes, usage scenarios and requirements," in *Software Quality, 2009. WOSQ '09. ICSE Workshop on*, may 2009, pp. 9 –14.

[16] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 33 –53, jan. 2007.

[17] M. Shepperd, "Software project economics: a roadmap," in *Future of Software Engineering, 2007. FOSE '07*, may 2007, pp. 304 –315.

[18] M. Jorgensen, B. Boehm, and S. Rifkin, "Software development effort estimation: Formal models or expert judgment?" *Software, IEEE*, vol. 26, no. 2, pp. 14 –19, march-april 2009.

[19] L. C. Briand, K. El Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *Proceedings of the 21st international conference on Software engineering*, ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 313–322. [Online]. Available: http://doi.acm.org/10.1145/302405.302647

[20] L. C. Briand, T. Langley, and I. Wieczorek, "A replicated assessment and comparison of common software cost modeling techniques," in *Proceedings of the 22nd international conference on Software engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 377–386. [Online]. Available: http://doi.acm.org/10.1145/337180.337223

[21] T. Lee, D. Choi, and J. Baik, "Empirical study on enhancing the accuracy of software cost estimation model for defense software development project applications," in *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol. 2, feb. 2010, pp. 1117 –1122.

[22] P. Nesi and T. Querci, "Effort estimation and prediction of object-oriented systems," *Journal of Systems and Software*, vol. 42, no. 1, pp. 89 – 102, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121297100218

[23] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *Software Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 751 –761, oct 1996.

[24] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[25] M. M. Lehman and L. A. Belady, Eds., *Program evolution: processes of software change*. San Diego, CA, USA: Academic Press Professional, Inc., 1985.

[26] G. Canfora, L. Cerulo, M. Di Penta, and F. Pacilio, "An exploratory study of factors influencing change entropy," in *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, 30 2010-july 2 2010, pp. 134 –143.

[27] A. Bianchi, D. Caivano, F. Lanubile, and G. Visaggio, "Evaluating software degradation through entropy," in *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, 2001, pp. 210 –219.

[28] G. Hanssen, A. Yamashita, R. Conradi, and L. Moonen, "Software entropy in agile product evolution," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, jan. 2010, pp. 1 –10.

[29] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches Ů a survey," *Annals of Software Engineering*, vol. 10, pp. 177–205, 2000, 10.1023/A:1018991717352. [Online]. Available: http://dx.doi.org/10.1023/A:1018991717352

[30] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *Software Engineering, IEEE Transactions on*, vol. 21, no. 2, pp. 126 –137, feb 1995.

[31] P. Pendharkar, G. Subramanian, and J. Rodger, "A probabilistic model for predicting software development effort," *Software Engineering, IEEE Transactions on*, vol. 31, no. 7, pp. 615 – 624, july 2005.

[32] M. Klas, A. Trendowicz, Y. Ishigai, and H. Nakao, "Handling estimation uncertainty with bootstrapping: Empirical evaluation in the context of hybrid prediction methods," in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, sept. 2011, pp. 245 –254.

[33] S. K. Dubey and A. Rana, "Assessment of maintainability metrics for object-oriented software system," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 1–7, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2020976.2020983

[34] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476 –493, jun 1994.

[35] L. Radlinski and W. Hoffmann, "On predicting software development effort using machine learning techniques and local data," in *International Jounral of Software Engineering and Computing*, ser. Vol.2, No.2. International Science Press, 2010.

[36] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 367–377. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2009.5314233

[37] C. Mair and M. Shepperd, "The consistency of empirical comparisons of regression and analogy-based software project cost prediction," in *Empirical Software Engineering, 2005. 2005 International Symposium on*, nov. 2005, p. 10 pp.

[38] M. Lee, A. J. Outt, and R. T. Alexander, "Algorithmic analysis of the impacts of changes to object-oriented software," in *In Proceedings of the International Conference on Software Maintenance*. IEEE, 2000, pp. 171–184.

[39] D. M. Allen, "Mean square error of prediction as a criterion for selecting variables," *Technometrics*, vol. 13, no. 3, pp. 469–475, 1971. [Online]. Available: http://www.jstor.org/stable/1267161?origin=crossref

[40] M. Agrawal and K. Chari, "Software effort, quality, and cycle time: A study of cmm level 5 projects," *Software Engineering, IEEE Transactions on*, vol. 33, no. 3, pp. 145 –156, march 2007.

[41] S. T. Knox, "Modeling the Cost of Sofware quality," in *Digital Technical Journal*, ser. Vol.5 No.4, 1993.

[42] E. Capra, C. Francalanci, and F. Merlo, "An empirical study on the relationship between software design quality, development effort and governance in open source projects," *Software Engineering, IEEE Transactions on*, vol. 34, no. 6, pp. 765 –782, nov.-dec. 2008.