Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models

Pekka Abrahamsson[A], Raimund Moser[B], Witold Pedrycz[C], Alberto Sillitti[B], Giancarlo Succi[B]

[A]VTT Electronics, Oulu, Finland
T P.O. Box 1100
FIN-90571 Oulu, Finland
+358 40 541 5929

[B]Center for Applied Software Engineering, Free University of Bolzano-Bozen, Italy
Piazza Domenicani 3
I-39100 Bolzano, Italy
+39 0471 016130

[C]Department of Electrical and Computer Engineering, University of Alberta, Canada
Civil/Electrical Engineering Building, 238, Edmonton, Canada
+1-7804923332

pekka.abrahamsson@vtt.fi, rmoser@unibz.it, pedrycz@ee.ualberta.ca, asillitti@unibz.it, gsucci@unibz.it

Abstract

Estimation of development effort without imposing overhead on the project and the development team is of paramount importance for any software company. This study proposes a new effort estimation methodology aimed at agile and iterative development environments not suitable for description by traditional prediction methods. We propose a detailed development methodology, discuss a number of architectures of such models (including a wealth of augmented regression models and neural networks) and include a thorough case study of Extreme Programming (XP) in two semi-industrial projects. The results of this research evidences that in the XP environment under study the proposed incremental model outperforms traditional estimation techniques most notably in early phases of development. Moreover, when dealing with new projects, the incremental model can be developed from scratch without resorting itself to historic data.

1. Introduction

Given its importance and practical relevance, effort estimation has been for a long time a major focus of research in software engineering. Since the pioneering work by Putnam (1978), Boehm (1981) and Albrecht (1983), there have been many attempts to construct prediction models of software cost determination. An overview of past and current effort estimation techniques, their application in industry, and their drawbacks regarding accuracy and applicability can be found for example in [14], [6], [8], [19]. Nowadays there exist a few theories and pertinent models that help quantify effort prediction. The most prominent one comes in the form of the so-called COCOMO family of cost models [5]. While capturing the essence of project cost estimation in many instances, they are not the most suitable when we are faced with more recent technologies and processes of software development such as agile approaches. Moreover, models such as COCOMO II depend quite heavily on many project-specific settings and adjustments, whose impact is difficult to assess, collect, and quantify. What makes the situation even worse, is the fact that in agile processes an effective collection of such metrics and the ensuing tedious calibration of the models could be quite unrealistic.

As far as we know, no specific models have been developed for agile and iterative development processes. Only a few studies deal with the idea of refining prediction models during project execution and enhancing them with effort information of previous phases. Closest to our work is a recent study by Trendowicz *et al.* [27] who incorporate into a hybrid cost estimation model feedback cycles and possibility for iterative refinement. MacDonell and Shepperd [15] use project effort of previous phases of development as predictor for a regression model and show that it yields better results than expert opinion. However, both studies do not address the peculiarities of agile processes, use a different approach for model building and do not provide any comparative studies with traditional models. Finally, Alshayeb and Li [1] investigate correlation between object-oriented design metrics and effort in agile projects, but do not consider iterative model building or refining.

In general, traditional effort estimation models work as follows. Some predictors are collected or estimated at the beginning of a project and fed into a model. The model, which is usually built upon historic data using similar projects, predicts the total development effort. While this approach is reasonable for traditional, waterfall-like development processes where common predictors such as function or feature points, software size, formal design specifications, design documents, etc. are known at the beginning of a project and typically do not change too much throughout the overall project this is not the case for agile development processes. In agile development, a project is realized in iterations and requirements usually change from one iteration to the next. At the end of each iteration developers release the software to the customer who will eventually require new features, change or removal of already implemented functionalities. At the beginning of a new iteration developers will negotiate with the customer about requirements and develop a plan (design document) for the next iteration (in XP this process is referred to as planning game [3]). Therefore, standard predictors proposed in the literature, in particular the ones derived from design documents, are only known at the beginning of the next development iteration and not *a priori* for the whole project.

Being cognizant of the existing challenges as outlined above, the key objectives of our study can be outlined as follows:

- We consider a proposal for a new, incremental effort prediction model for iterative software development processes.
- We carry out a thorough experimental validation of the incremental model and offer a comparative analysis with the existing monolithic (global[1]) prediction models.

Overall, the underlying research hypothesis is formed as follows: When using iterative software development processes, incremental effort prediction models are more efficient than global models.

The paper is organized as follows. Section 2 elaborates on the concept of global and incremental prediction models. Section 3 discusses the models we use to realize effort prediction. In Section 4, we present a case study and in Section 5 we provide the results of the case study. Section 6 addresses limitations of the study and identifies future research needs; and finally, in Section 7 we draw the conclusions.

[1] To emphasize their operation mode we call traditional effort estimation models “global” models.

2. Global versus incremental prediction models

In this Section we explain the essence of global and incremental prediction models. Realizations of both types of approaches with concrete *mathematical* models are presented in Section 3.

2.1. Global prediction models

Let us highlight the essence of traditional, global prediction models in software engineering [8] [14] [6] [21]. At the beginning of a software development project, we identify several meaningful predictor variables and collect or estimate their values. A list of predictor variables could involve size, design metrics, qualitative and quantitative development factors (including type of development environment, application domain, experience of developers, organizational structure, development process, etc.). Models are derived from historic data and used to predict development effort for similar projects. There are crucial model development and utilization issues that have to be clearly underlined:

- The choice of the predictor variables. This task is highly demanding as at the beginning of the project we may not know which variables of the project could have a high impact on the assessment of the development effort.
- While we might be tempted to collect a lot of variables (in anticipation of a proper compensation for the shortcoming identified so far), the process could be time consuming, costly, and at the end lead to overly complicated models whose estimation could be quite complex and inefficient.

In the construction of global models we rely on historic data or/and expert opinion. This requires that first one has to gain experience and collect data for at least one project and afterwards construct the model and apply it to similar projects. This is not only a long-term process, but comes with some risk that given the unstable and highly non-stationary environment of software development, it may lead to models whose predictive capabilities are questionable. Moreover, software industry is moving into a direction where projects continuously evolve with new updates and deliveries in response to market demand. In such scenario it is not obvious when to freeze the project for model building purposes.

Agile software development introduces new challenges to effort estimation. Predictor variables usually are not known at the beginning of a project, but become available at the beginning of each iteration as requirements change often and fast. Under these circumstances a long-term model of effort estimation that naturally relies on information available at the beginning of a project seems to be of limited applicability. However, one could still contemplate the use of the long-term, global cost estimation model and use it for reference purposes.

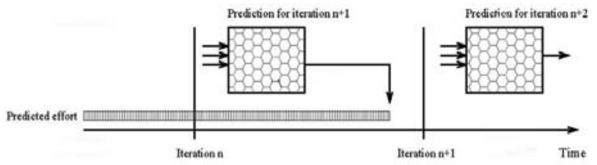

Figure 1: Global model for iterative effort prediction.

The essence of the global model (Figure 1) can be highlighted as follows:

- *Model building*: The global model is constructed **after** project completion. It uses predictor variables collected at beginning of the project. The model predicts the *total* development effort.
- *Iterative effort prediction*: Such model is used for predicting the development effort of a new, future project of similar characteristics to those already realized. Usually at the start of the project predictor values are collected (or estimated) and fed into the model, which in turn predicts the total development effort. However, in agile development it may happen that predictors are only known at the **beginning of each iteration**. In such cases we have to use the global model in an iterative way. The straightforward approach we use is the following: At the beginning of iteration *N* predictor values are collected, fed into the global model and an estimate of the total development effort for iteration *1* to *N+1* is obtained.

Following the rationale outlined above, we can identify three major reasons that may suggest limitations of the construction and use of global models when dealing with agile, iterative development environments:

- First, the model building process takes a long time (the completion of one or more projects) and in the meantime technology, developers, processes, and other factors may change. Thus, it is unlikely that the extracted model will work very well for new projects. Mohanty [19] pointed out that effort estimation models are calibrated and developed using data collected in unique environments. Therefore, the prediction of such models will be based on underlying assumptions, which are not present in the data. This can have a dramatic impact on the accuracy of such models in different environments.
- Second, in agile development, requirements change frequently and fast when moving from one iteration to another; this may imply, that also the design of the software, technology, or other factors have to be changed to meet requirements and implementation effort may vary significantly for different iterations. A global model captures only the overall contribution of predictor variables to development effort and is unaware of their changes within single iterations.
- Last, if a company does not have historic data it cannot build or calibrate any “customized” global prediction model, and thus it has to rely on subjective estimates or models published by other companies or being available in the literature.

Being prompted by these limitations, we assume a different development position by focusing on the incremental mode of model development.

2.2. Incremental prediction models

The main idea of an incremental prediction model is that it is built **after each iteration** instead of at the end of a project to estimate effort for the next iteration. Thus, it is able to accommodate to any changes during development in a much smoother way than a global model. Moreover, we endow the incremental model with a *dynamic character* by using effort of previous iterations that is treated as an additional input. In this way, effort prediction does not only depend on the usual predictor variables but also on the past effort distribution itself, which may make a significant contribution for explaining future effort variation. The incremental model operates only for iterative effort prediction as it cannot be used to predict total development effort at the beginning of a project.

Figure 2 explains in some schematic way the operating mode of an incremental effort prediction model.

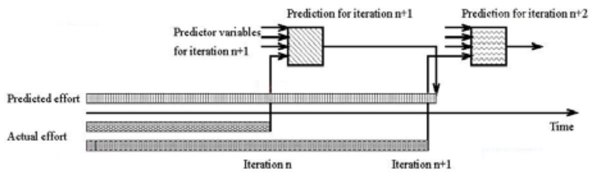

Figure 2: Incremental model of iterative effort prediction.

The essence of the incremental model (Figure 2) can be highlighted as follows:

- *Model building:* At the **end of each iteration** a new model is built using as input the predictor values estimated for that iteration and development effort of previous iterations. Thus, the main difference to the global model is that this model is dynamic in the sense that it depends on the phase of development it is built and past effort distribution.
- *Iterative effort prediction:* At the beginning of a new iteration predictor variables are collected and fed together with past effort into the newly built incremental model. The output of the model is an estimation of the total development effort for the next iteration.

When comparing incremental models versus global ones, we stress the following essential differences:

- An incremental model is constantly adjusted during development and evolves with the project; therefore, it can much better accommodate to any changes in technology, requirements, personnel, or other occurring during project evolution.
- The relative impact of different predictor metrics on development effort is likely to change during project evolution. An incremental model can select the best features for each single iteration; hence, it should give more accurate and reliable estimations.
- The incremental model is useful and applicable right from second iteration of development. There is no need for historic data and also no risk that the model is outdated. In particular, this is very valuable in highly volatile domains where it is difficult to find stable and general models.
- Medium and small companies usually get development projects for very different business environments. They often face the problem to have no expertise in the project's problems or solution domains. An incremental effort estimation model can help produce more reliable estimates and raise confidence in their personal judgments after only a few development iterations.

A comparison of the properties of the incremental and global models is presented in Table 1.

Table 1: Comparison between incremental and global models.

Feature	Incremental model	Global model
Continuous model evolution and refinement	✓	--
Historic data is needed for model building	--	✓
Automatic adaptation to new business environments	✓	--
Estimation of total project effort	--	✓
Estimation of effort for single development iterations	✓	✓

It is clear that the incremental model requires more data for model building; in particular it needs fine-grained effort data for all development iterations. This may be seen as limitation to its practical applicability. However, in Section 4 we show how to overcome these shortcomings by using appropriate measurement tools that enable automatic and non-invasive data collection. Another weaknesses of incremental models is their inability to help people answer questions up front about how much time and money a proposed software project will take. This may be an important issue for a company for deciding whether or not to take on a given software project. However, in agile development companies negotiate with customers which and how many features have to be developed within a short period of time (one development iteration). In such scenarios an incremental model may help a software firm to get a reliable estimate of the number of features that can be implemented within that period. A realistic estimate is important for a strong marketing strategy and for being at the same time competitive and keeping business terms.

3. Models of effort prediction

Given the two fundamental modes of incremental and global modeling, we contemplate several detailed architectures of the models considering various predictor variables and forms of mapping from these variables to the predicted development effort.

In software engineering we can encounter many forms of effort prediction models. We can recall here empirical models based on subjective judgment [2], [23], regression models [17], regression trees, neural networks [26], theory based models such as COCOMO II [5], so-called hybrid models [27], and Bayesian nets [20].

In this research, we consider only algorithmic models that can be effectively constructed (without human interaction) from quantitative data. Considering the constraints imposed by data collection in agile environments we restrict our analysis to two different families of prediction models, which are representative for common types of approaches in software engineering and require a limited amount of resources for both model building and operation:

- Regression models that are easy to understand and develop; we can use them as a reference model to compare with other, more advanced models. Most effort estimation methods considered in the past are based on regression models [14].
- The use of neural networks.

At present there is no agreement of what kind of models are most accurate and powerful for effort prediction [18]. The simplest model, ordinary least squares regression, seems to perform just as well as many other proposed prediction models [8]. Therefore, we use it as a benchmark in this study.

A far more challenging problem in software engineering concerns scarcity of data, which implicates all kind of problems for model validation and generalization. Given the very limited size of experimental data available in the problem, we consider the use of a so-called leave-one-out (LOOV) cross validation procedure [4]. It works as follows: All but one data points are used for building the model, which in turn is used to predict the left out point. This is done for each single data point in the set and the average mean squared error for all left-out points (LOOV-RMSE) is used as a performance index for the efficiency (generalizability) of the model.

3.1. Regression models

While regression models are often encountered in software engineering, we have to be aware that they could exhibit several shortcomings:

- They are not capable of capturing non-linear types relationship between input and output variables.
- Software engineering data are often messy due to cost and inherent difficulties of gathering data; therefore, in general we cannot expect that all standard assumptions encountered in regression analysis are fully satisfied.

In this study, we endorse a holistic view at the modeling pursuits by considering a suite of refined regression models (Table 2).

Table 2: Selected classes of regression models.

Type of regression	Advantages over simple regression
Stepwise	Removes in part collinearity between input features
Robust	Lessens the influence of outliers on regression coefficients
Quadratic	Takes into account higher order components

Furthermore, we consider some combinations of the methods shown above; for example, stepwise and robust regression to limit both the influence of outliers and at the same time cope with the problem of collinearity.

3.2. Neural networks

Neural networks can potentially offer several advantages in analysis of software data given the fact that they do not rely on assumptions of linear regression models and come with interesting learning capabilities. There have been a number of studies along these lines [24] [26] [16] [7]. We consider two different types of neural networks: a classical feed-forward Multilayer Perceptron (MLP) and Radial Basis Functions (RBF) neural networks [4].

The performance of both types depends highly on the chosen network architecture (number of layers, number of neurons, activation function, number of receptive fields, spread, etc.). While there is no general theory behind the structural optimization of the topology of the networks, they are developed as a result of some trial and error: For each set of parameters that specify the model completely we compute the cross validation error. We keep the set that produces the lowest error and this topology of the network is deemed optimal. For the MLP we start with 2 neurons in the hidden layer and keep adding one neuron at a time until the leave-one-out cross validation error (LOOV-RMSE) stops decreasing. Its minimum value determines the optimum number of neurons to choose for the hidden layer. For RBF we do exactly the same: We start with one receptive field and add one at a time until the cross-validation error stops decreasing. We repeat this procedure for a range of spread parameters and keep the spread and number of receptive fields that return the absolute smallest cross-validation error.

For comparing and evaluating the predictive performance of the various models we use common criteria for evaluation of cost estimation models in software engineering: the Magnitude of Relative Error (MRE), the prediction at level k (PRED(k)) [10], the Magnitude of Error Relative to the estimate (MER), and the usual standard deviation SD [13].

4. Case study

In this Section, we present two case studies conducted in a semi-industrial environment. Semi-industrial environment refers to a development endeavor where a mix of professionals and students are developing real software product in a controlled development setting. Such type of a research setting is depicted in detail in [22]. First, we describe the characteristics of the projects under scrutiny (environment, development process etc.); then, we discuss the data collection process and the choice of the input and output features we use for model building. Finally, we provide a descriptive statistic of the data sets used for analysis.

4.1. Characteristics of the projects

The study concerns two commercial software projects – we refer to them as project *A* and project *B* - developed at VTT Technical Research Centre of Finland in Oulu, Finland. The programming languages used in both projects were Java and the IDE was Eclipse 3.x (www.eclipse.org). Project *A* delivered a production monitoring application for mobile, Java enabled devices. The software is a client-server solution using J2ME and Oracle database. Project *B* delivered a project management tool for agile projects. The software is programmed with Java and Flash and is a standard web application for desktop computers.

For both projects the development process followed a tailored version of the Extreme Programming practices: in project *A* two pairs of programmers (four people) and in project *B* three pairs of programmers (six people) have worked for a total of eight weeks. The projects were divided into five iterations, starting with a 1-week iteration, which was followed by three 2-week iterations, with the project concluding in a final 1-week iteration. The working time for the dedicated project was 6 hours per day, 4 days a week. Beside the mentioned XP practices (pair programming, small releases, continuous integration, planning game) also the practices of refactoring and in part test-driven development have been adopted.

Throughout the project mentoring was provided on XP and other programming issues according to the XP approach. For project *A* three of the four developers had an education equivalent to a BSc and limited industrial experience. The fourth developer was an experienced industrial software engineer. The team worked in a collocated environment. Since it was exposed for the first time to the XP process a brief training of the XP practices, in particular of the test-first method was provided prior to the beginning of the project. For project *B* four developers were 5 - 6th year university students and the two remaining employees of VTT and as such experienced industrial software engineers.

Both projects were completed successfully. At the end of the project *A* the team not only released on time the software with the required functionalities but it exceeded the original requirements so far that the customer ran out of requirements to be done.

4.2. Selection of predictor and output variables

Many different metrics have been proposed as effort predictors: The most common one is for sure software size (lines of code, function or feature points, number of requirement documents, etc.). COCOMO/COCOMO II, one of the most popular cost estimation models uses in addition to size a wide range of so called cost drivers (mostly technology and human factors). In this study we use the Chidamber and Kemerer (CK) set of object-oriented design metrics [9] for effort prediction.

The CK metrics have some nice properties, which make them in particular attractive for the kind of prediction model we propose:

- They are widely known by practitioners and in the research community and have been at least partially validated by several other researchers.
- For the purpose of model building the CK metrics can be extracted automatically from source code.
- In agile development documentation, requirements elicitation or design plans are minimalist and only produced as far as they are really needed. Therefore, we cannot expect to collect a lot of data before starting an iteration that could be used for effort prediction. However, even in the most extreme case of agile development, XP, at the beginning of a new iteration there are some planning activities that produce at least some sort of design documents (CRC cards). These documents can be used to estimate fairly well the CK design metrics for the new iteration, which will be used as input values for effort prediction.

We do not use all 6 CK metrics as predictors but exclude the NOC (number of children) and LCOM (lack of cohesion of methods) metrics. We exclude NOC because in both projects it is almost 0 for all classes and hence does not contribute significantly to development effort. As for LCOM several researchers have questioned its meaning and the way it is defined by Chidamber and Kemerer [11]; the impact of LCOM on development effort and other metrics such as maintainability is little understood by today and therefore we decide to exclude it from this analysis.

As output variable we use the total coding effort per class in hours at any point in time, i.e. for example at the end of the project for the global model or at the end of each iteration for the incremental model.

4.3. Data collection process

We used in both case studies our in-house developed tool PROM [25] for automatic and non-invasive data collection. To collect the product and process metrics we use for effort estimation with the PROM tool we adopt the following data collection procedure:

- Each night at midnight various source code metrics (among them are the CK metrics) are extracted from a CVS repository.
- A plug-in for Eclipse (the IDE used by developers) collects automatically the time spent for coding on individual classes and methods.

These measures are integrated and stored in a data warehouse. An analysis tool can extract them easily from the data warehouse and use for model building and statistical analysis.

4.4. Descriptive statistics of the data sets

For project *A* the total coding effort recorded by the PROM tool was about 305 hours. Project *A* has 1776 lines of code (counted as Java statements in the source code) divided in 30 classes. Table 3 shows a summary statistics of the collected design metrics and coding effort at the end of the project.

Table 3: Descriptive statistics for project *A*.

Metric	CBO	WMC	RFC	DIT	Effort (h)
Mean Std	9.8 ±7.1	17.8 ±22.9	25.7 ±19.7	2.3 ±1.2	10.1 ±16.1

Project *B* has 3426 lines of code (Java statements in source code) and 52 classes. The total coding effort for project *B* is about 664 h. Table 4 reports the descriptive statistics for project *B*.

Table 4: Descriptive statistics for project *B*.

Metric	CBO	WMC	RFC	DIT	Effort (h)
Mean Std	14.9 ±10	14.6 ±15.7	36.6 ±30	2.5 ±1.1	12.7 ±15.2

Due to space constraints we do not show box plots of the data sets. An inspection of them reveals that almost all metrics have some outliers. Outliers may bias significantly regression coefficients using standard least square regression methods. We avoid this problem to some extent by using robust regression techniques.

5. Results

First, we report the results we obtain by constructing a global model using CK metrics extracted from the final release of the software as input and the total coding effort as output variable. Table 5 reports the LOOV-RMSE error for the training and test data set. The 4th column reports the input variables used by the model, in particular the ones selected by the stepwise regression procedure, and the architecture of the neural networks used.

Table 5: Regression and neural networks for the global approach.

Model	RMSE training	LOOV-RMSE test	Input variables
Regression models for project *A*			
L	2.12 ±0.19	2.10 ±2.71	All
S-L	2.16 ±0.16	1.76 ±2.43	CBO, WMC
S-QR	2.94 ±0.29	1.64 ±2.48	WMC, RFC, CBO^2
R	2.74 ±0.25	1.71 ±2.89	All
S-R	3.26 ±0.29	1.62 ±2.92	CBO, WMC
Regression models for project *B*			
L	3.26 ±0.06	2.72 ±2.68	All
S-L	3.27 ±0.06	2.58 ±2.57	WMC, RFC
S-QR	3.33 ±0.12	2.97 ±3.60	WMC, RFC, WMC^2
R	3.63 ±0.20	3.40 ±5.47	All
S-R	3.65 ±0.10	2.85 ±3.78	WMC, RFC
Neural networks - project *A*			
MLP	1.28 ±0.81	3.09 ±4.89	All 2-1 layer
RBF	0.74 ±0.03	1.26 ±1.10	All, spread=10, 12 neurons
Neural network - project *B*			
MLP	1.20 ±0.57	2.52 ±2.56	All 4-1 layer
RBF	1.12 ±0.03	1.77 ±2.53	All, spread=1.8, 17 neurons

Legend: L - multi-linear regression, S-L - stepwise, multi-linear regression, S-QR - stepwise quadratic robust regression, R - robust regression, and S-R - stepwise robust regression. MLP - multilayer perceptron neural network and RBF - radial basis functions network.

For computing confidence intervals for regression coefficients and as p-value for a predictor to be recommended for adding to the model in the stepwise regression procedure we use $\alpha = 0.05$. For quadratic regression we consider only pure quadratic terms as otherwise the input space is too high dimensional for the data sets used and we run into risk of the *curse of dimensionality* [4]. Among all different regression models stepwise robust regression for project *A* and ordinary stepwise regression for project *B* give the lowest LOOV-RMSE error. Moreover, both models select WMC as input feature suggesting that it is a good effort predictor.

It is interesting to observe that any neural network *per se* is not better for prediction than simple regression. The MLP network for example performs worse than most regression models. However, the RBF network, which is able to capture better local properties of single data points works best for both projects. Thus, we confirm the findings of [24] and agree that RBF networks are a promising approach for modeling software engineering data. Based on these findings we choose stepwise regression and RBF networks for building corresponding incremental models.

As next step we perform iterative effort prediction using the incremental and global approach for both regression and RBF models. For global models we use – as in a real-world scenario – models fitted on data collected in project *A* for predicting effort in project *B* and vice versa. In practice global models aim at predicting effort for “similar” (similar to the ones used for model building) projects: This is the case for the two projects under scrutiny as they are similar in size, development environment and methodology, programming language/technology and tools used. The prediction errors for estimation of total coding effort per iteration (obtained as sum over the effort predicted per class) are presented in Table 6. We report only MRE values as they are reported in many studies and thus are useful for comparison; the other performance criteria we compute confirm unanimously MRE values.

Comparing the relative performance of global and incremental approach we find clear evidence of the superiority of the incremental model. Due to space constraints we only report relative errors for the **total** development effort per iteration; a closer look reveals that effort prediction at a class level is very unreliable and – if at all – works only in the incremental mode and using RBF models (but only few estimates have relative errors lower than 25%). We explain this by the fact that data under scrutiny have outliers and skewed distributions and models - in particular global models - are not able to adapt to single data points but only to approximate average values. The results in Table 6 emphasize that an incremental model makes (**a**) more accurate effort predictions (moreover, the accuracy of global models is unsatisfactory) and (**b**) can be used in a meaningful and reliable way much earlier in development than the global model.

Table 6: Prediction of effort per iteration using incremental and global models.

Iteration	MRE SR - *A*	MRE RBF - *A*	MRE SR - B	MRE RBF - *B*
3 (G)	154%	412%	84%	63%
3 (I)	**51 %**	**71%**	**57%**	**37%**
4 (G)	34%	12%	105%	32%
4 (I)	**32%**	**33%**	**25%**	**19%**
5 (G)	57%	90%	47%	17%
5 (I)	**11 %**	**6%**	**16%**	**21%**

SR – stepwise linear regression, RBF Radial Basis Functions network, (I) – incremental model, (G) global model.

For example for iteration 3 a global model gives a relative error of 154% in project *A* and 63% in project *B*, while an incremental model reduces the relative errors to 51% respective 37%. Moreover, for project *B* from iteration 4 onwards (for project *A* only for iteration 5) the relative error for incremental effort prediction falls under 25%, which is considered a good result in software effort estimation [10]. Since these results are confirmed both by regression models and neural networks we strongly believe that they are model independent. Furthermore, incremental models reduce the relative error from one iteration to the next indicating that they stabilize and adjust to project characteristics as more data become available and are used for model construction. It is interesting to note that predictions are more accurate for project *B* than for project *A*. This could be explained by the fact that project *B* has more data points (its effort distribution is smoother than the one for project *A*) which enables better training and model adaptation. In general we expect the performance of incremental models to improve as project size grows and more development iterations have been concluded. In both cases sample size increases, which leads to better model training and generalization.

Altogether we can state that:

- In general, incremental models outperform global ones for iterative effort prediction (this is even more manifest in early iterations of development).
- Unlike global, incremental models improve continuously prediction accuracy during project evolution reaching a satisfactory level (less than 25% MRE) and providing more confidence to stakeholders (developers, managers, customers) due to their convergent nature.

6. Limitations of this work

This work is a first step towards understanding how to develop and use effort prediction models in iterative, agile environments. It is needless to say that in order to consolidate the findings of this study and transform them into actual “knowledge” and recommendations to developers and managers several replications are needed. We use a novel approach for data collection, which does not require an active involvement of developers, and thus is well suited for agile environments. This approach is able to provide a high granularity and a large amount of effort data but a quantification of the improvement compared to manually collected data has to be carried out in a future experiment. All models for effort prediction proposed and validated in this research have been built using two software projects in a particular XP environment and all possible threats to external and internal validity have to be considered carefully [28]. In particular, we are faced with the following threats:

- The participants of the case studies are in part master students; it is questionable whether or not they represent the average software developer in industry. Thus, further investigation with a randomized sample of developers is needed to analyze to which degree our findings are biased by the selection of the subjects.
- Most of the participants of the study have been exposed to XP for the first time. We do not control the impact of a learning curve on the results. It is referred to a future study if experienced XP developers would have performed in the same way.
- We develop regression models and neural networks for comparing global with incremental approaches. However, there are other promising modeling techniques for effort estimation such as Bayesian networks, classification and regression trees or hybrid models. In the future we plan to investigate whether different modeling choices sustain our results or limit the validity of the findings to the subset of analyzed models.
- The choice of CK design metrics as predictor variables may also impact the findings of this research. Other choices could favor global models. Moreover, CK metrics are extracted from source code, but in practice they have to be estimated from design documents. Further analysis is needed to investigate the impact of variability in the estimation of CK metrics on performance of incremental models.

Finally, to do justice to global models we have to note that we use only one project for model building and use such model for effort prediction on a second project. In real life a software company would possibly use and combine several past projects for the purpose of model building. Probably such model is more reliable and stable as it averages data from several projects and becomes less influenced by characteristics of one particular data set. Shepperd and Schofield argue that in order to calibrate and stabilize properly an estimation model they need at least 15 projects [23]. It remains to future experiments to determine whether a global model derived from a bunch of historic projects would be competitive to the proposed incremental approach.

7. Conclusions

In this study we propose a new approach to iterative effort prediction. We have identified a number of reasons for which the suitability of the monolithic prediction models is limited when dealing with agile software development:

- Most companies using agile development are rather small (94% of software companies in US have less than 20 people [12]). Thus, it is very likely that they may not have historic data to build traditional effort prediction models.
- Agile projects change rapidly and it is difficult to anticipate that a model constructed with the use of data for one project would be valid for another project.
- In general, predictor variables may not be known at the start of the project but become available at the beginning of each iteration. Therefore, early estimation of total development effort is impossible and should be replaced by an iterative estimation of development effort for the next future iteration.

Considering these arguments we develop an incremental, iteration-based prediction model. It has the following advantages over a monolithic model:

- There is no need for historic data.
- It fits naturally into iterative development cycles, as it evolves and accommodates to changes from one iteration to the next.
- There is no risk that the model is outdated; this is in particular important for companies that accept projects in new application domains or dealing with new technologies. In such cases models based on historic data may completely fall short.
- An incremental model gives early and more frequent feedback to developers and managers. A short feedback cycle is important to detect early aberrations from preset targets. In such cases a company may investigate why predicted effort goals are not met or eventually re-calibrate the prediction model.

We apply the incremental approach to two agile, semi-industrial development projects and find evidence that it is superior to a global, monolithic prediction model. Incremental models are stable and convergent in the sense that their prediction error decreases from iteration to iteration. They can be used right from the start of development and improve their accuracy throughout project evolution due to their iterative nature. Intelligent data collection and analysis tools allow easy automation of the model construction process. At the beginning of a development iteration they could be integrated in a planning game where customers, developers, and managers develop a first objective cost estimation based on current project data.

8. References

[1] M. Alshayeb, W. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", *IEEE Transactions on Software Engineering*, November 2003, **29**(11): 1043-1048.

[2] L. Angelis, I. Stamelos, M. Morisio, "Building A Software Cost Estimation Model Based On Categorical Data", *Proceedings of the 7th International Symposium on Software Metrics*, London, 2001.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[4] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1994.

[5] B.W. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby, C. Westland, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", *Annals of Software Engineering*, 1995.

[6] B.W. Boehm, C. Abts, S. Chulani, "Software development cost estimation approaches - A survey", *Ann. Software Eng.*, 2000, 10: 177-205.

[7] G. Boetticher, "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains", *Model Based Requirements Workshop*, San Diego, 2001, pp. 17 – 24.

[8] L.C. Briand, I. Wieczorek, "Resource Estimation in Software Engineering", *ISERN-00-05 Technical Report*, Fraunhofer Institute for Experimental Software Engineering, Germany, 2000.

[9] S. Chidamber, C.F. Kemerer, "A metrics suite for object-oriented design", *IEEE Transactions on Software Engineering*, June 1994**, 20**(6): 476-493.

[10] S. Conte, D.H.E. Dunsmore, V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings Publishing Company, Inc., 1986.

[11] S. Counsell, S. Swift, J. Crampton, "The interpretation and utility of three cohesion metrics for object-oriented design", *ACM Trans. Softw. Eng. Methodol.*, 2006, **15**(2): 123-149.

[12] M. E. Fayad, M. Laitinen, and R. P. Ward, "Software Engineering in the Small", *Communications of the ACM*, March 2000, vol. 43, no. 3.

[13] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE", *IEEE Transactions on Software Engineering*, 2003, **29**(11): 985-995.

[14] M. Jørgensen, and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies Document Actions", IEEE Transactions on Software Engineering, 2006, **33**(1): 33-53.

[15] S. MacDonell, M.J. Shepperd, "Using Prior-Phase Effort Records for Re-estimation During Software Projects", *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*, Sydney, Australia, 2003.

[16] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, S. Webster, "An Investigation of Machine Learning Based Prediction Systems", *Journal of Systems and Software*, 53, 2000, pp. 23-29.

[17] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust regression for developing software estimation models", *Journal of Systems and Software*, 1994, **27**(1): 3-16.

[18] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models", *IEEE Transactions on Software Engineering*, 2005, **31**(5): 380-391.

[19] S. Mohanty, "Software cost estimation: Present and future", *Software-Practice and Experience***,** 1981**,** vol. 11, pp. 103-121.

[20] C. P. C. Pendharkar, G. H. Subramanian, J. A. Rodger, "A Probabilistic Model for Predicting Software Development Effort", *IEEE Transactions on Software Engineering*, July 2005, **7**(31): 615-624.

[21] L.H.A. Putnam, "A general empirical solution to the macro software sizing and estimation problem", *IEEE Transactions on Software Engineering*, 1978, **4**(4): 345-381.

[22] O. Salo and P. Abrahamsson, "Empirical Evaluation of Agile Software Development: The Controlled Case Study Approach", *PROFES 2004*, Keihanna-Plaza, Kansai Science City in Kyoto-Nara area, Japan, 2004.

[23] M.C. Shepperd, and C. Schofield, "Estimating software project effort using analogies", *IEEE Transactions on Software Engineering*, 1997, **23**(11): 736-743.

[24] M. Shin and A.L. Goel, "Empirical Data Modeling in Software Engineering Using Radial Basis Functions", *IEEE Transactions on Software Engineering*, Jun. 2000, **26**(6): 567-576.

[25] A. Sillitti, A. Janes, G. Succi, T. Vernazza, "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data", *Proceedings of the EUROMICRO 2003*, Cyprus, 2003.

[26] K. Srinivasan, D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort", *IEEE Transactions on Software Engineering*, Feb. 1995, **21**(2): 126-137.

[27] A. Trendowicz, J. Heidrich, J. Münch, Y. Ishigai, K. Yokoyama, N. Kikuchi, "Development of a hybrid cost estimation model in an iterative manner", *Proceeding of the 28th International Conference on Software Engineering*, China, 2006.

[28] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlson, B. Regnell, A. Wesslén, *Experimentation in Software Engineering An Introduction*, Kluwer Academic Publishers, 2000.