UNIVERSITY OF OSLO
Department of informatics

Construction and evaluation of a tool for quantifying uncertainty of software cost estimates

Master thesis
60 credits

Magnus Holm

01.05 2011

Table of Contents

1 Abstract

Software development effort estimation is a continuous challenge in the software industry. The inherent uncertainty of effort estimates, which is due to factors such as evolving technology and significant elements of creativity in software development, is an important challenge for software project management. The specific management challenge addressed in this thesis is to assess the uncertainty of effort required for a new software release in the context of incremental software development. The evaluated approach combines task-level estimates with historical data on the estimation accuracy of past tasks for this assessment, by creating effort prediction intervals. The approach was implemented in a web-based tool, and evaluated in the context of a large Norwegian software project with estimation data from three contracted software development companies. In the evaluation we compared the approach to a simpler baseline method, and we found that our suggested approach more consistently produced reasonably accurate prediction intervals. Several variants of the basic approach were investigated. Fitting the historical data to a parametric distribution consistently improved the efficiency of the produced prediction intervals, but the accuracy suffered in cases where the parametric distribution could not reflect the historical distribution of estimation accuracy. Clustering tasks based on size had a positive effect on the produced effort intervals, both in terms of accuracy and efficiency. We believe the suggested approach and tool can be useful in software development project planning and estimation processes providing useful information to support planning, budgeting and resource allocation.

2 Introduction

2.1 Problem

Estimation of effort needed for software development is an important part of project planning, budgeting, and pricing/bidding in the software industry. The benefits of accurate effort estimates, or rather the problems with inaccurate ones are clear. Low effort estimates can lead to delayed deliveries, cost overruns and even low quality software. Too high effort estimates can lead to sub-optimal use of resources or even lost profit by losing software bidding rounds, because a high bid is less likely to get selected by the customer.

The most commonly used estimation method used in the software industry is based on expert judgment. Although research into cost estimation has focused on model-based estimation, research has shown that such models do not produce more accurate effort estimates than expert based estimation [1]. Still, studies have shown that there is a trend of overoptimistic effort estimates in the software industry, with as high as 30-40% overruns on average [2].

While a certain degree of inaccuracy in software effort estimates has to be accepted, accounting for uncertainty helps project planning by making managers able to put in place

proper contingency buffers, and by protecting the estimator from being blamed when an effort estimate is not met.

The challenging problem that is addressed in this thesis is that of designing sound and easy-to-use practices and tools that help software estimators quantify the uncertainty associated with estimates. The specific situation addressed and evaluated is that of assessing the uncertainty of effort required for a new software release in the context of incremental software development. However, the proposed approach and tool can be used in any situation where historical accuracy data is available.

2.2 Motivation

Despite all the research on modeling and improving effort estimation, the effort estimates in the software industry do not seem to become more accurate. By accepting that software effort estimates are inherently uncertain, we wish to better assess *how* uncertain estimates are. With usable practices and tools for quantifying uncertainty in place, software projects would be easier both to plan and manage. Armed with information about the uncertainty of a project's estimate(s), a project manager could be more confident when scheduling releases and other milestones throughout a project. Information about uncertainty also helps risk management, because risk can be quantified as the probability of an event occurring multiplied by the impact the event will have. Reliable information about the probability for different magnitudes of project overruns would be of help to a risk manager.

Several studies suggest that the predominant approach for uncertainty assessment in the software industry – expert judgment – is unreliable [3]. There is strong evidence towards overconfidence in uncertainty assessments based on expert judgment. Experts typically give a too narrow interval when asked to provide an interval that the actual value would most probably fall in. More structured approaches to uncertainty assessments are not commonly used in the software industry. It is therefore of significant value to design and evaluate new practices and tools to provide an alternative to the ad-hoc and intuition-based uncertainty assessment currently in common use.

With the availability of a tool for predicting uncertainty, structured uncertainty assessment could easily be included in project planning and management.

2.3 Goals

The goal of this thesis is to create and evaluate a tool to assess estimation uncertainty by using empirical data about the estimation accuracy in previously completed software projects or tasks. The basic approach was proposed and initially evaluated by Jørgensen and Sjøberg [4]. This thesis proposes and evaluates extensions and variants of the basic approach, and contributes with tool support for the basic approach, the extensions and the variants. The

ultimate goal is to make uncertainty assessment easier, unbiased and more accurate than expert judgment alone.

The tool should be easy to use, and not rely on any other estimation tools or methods. By creating an easily accessible and usable tool, it will be easier to conduct studies in the field of software development. The tool will also simplify evaluation of the suggested approach on existing data sets. The ability to more easily conduct such studies should improve the possibility of evaluating the approach in real life situations and give an indication on how effective the approach is compared to other methods of uncertainty assessment.

The tool should be easily accessible through the web, provide good usability, allow for integration with other software packages and be developed in a way that makes future improvements and extensions easy.

2.4 Basic concepts

2.4.1 Probability distributions and prediction intervals

Because the effort required to construct software is affected by uncertain events (e.g., threats and opportunities) that might or might not materialize, an estimate is better represented as a probabilistic distribution. A probability distribution reflects the inherent uncertainty of an effort estimate. A key question is how to create a realistic probability distribution. One approach is to let the expert estimate individual points on the probability curve, for example an optimistic, a most likely and a pessimistic estimate. A pessimistic estimate could be defined as the effort that would be sufficient in more than, for example nine times, in the hypothetical case that the project or tasks was repeated ten times. One problem when basing uncertainty estimates on human judgment is that such estimates have been shown to suffer from overconfidence, especially when dealing with high confidence levels [5].

In Figure 1 we can see an example of a probability distribution, the normal distribution. Imagine the peak of the curve as the most likely effort and the probability of other results decrease as you move further from this point. As we can see, this probability distribution is symmetric. This is often not the case for the distribution of effort. Any actual effort is lower bound at 0, while there is no such limitation when it comes to overruns. Figure 2 illustrates a *gamma distribution*, which probably gives a better picture of the probability distribution for development effort.

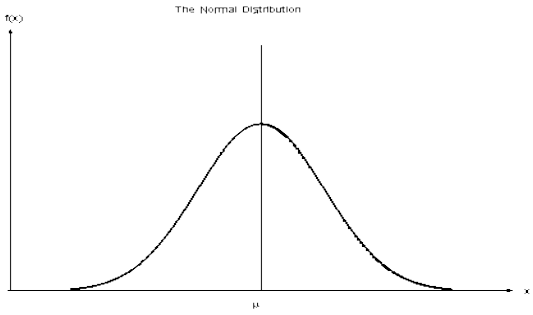

Figure 1 - The normal distribution

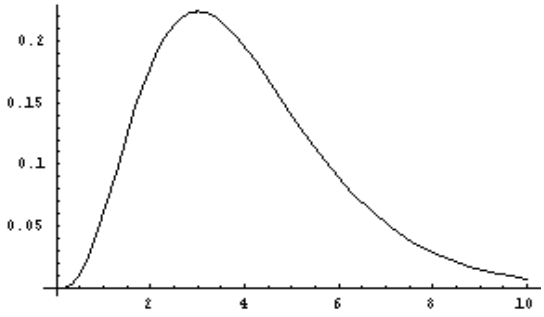

Figure 2 - The gamma distribution

A *prediction interval* (abbreviated *PI* later in the text) is an estimate of an interval where a future observation will fall (the actual outcome), with a certain probability, given previous observations. Given an effort PI with a minimum effort of 400 hours and a maximum effort of 600 hours with 90% certainty, then the actual effort should be between 400 and 600 hours nine out of ten times, given ten hypothetical repetitions of the project.

2.4.2 Common terminology

Magnitude of Relative Error (MRE) is a measure of accuracy used in many software cost estimation studies.

$$MRE = \frac{|Actual - Estimate|}{Actual}$$

Hit rate is a way of evaluating the performance of PIs in terms of accuracy, giving information about how often actual observations fall within the PI. Ideally, the hit rate should correspond to the requested confidence level that was used when generating the PI. For example, if a 90% confidence PI is requested, then hit rate should, in the long run, be around 90%.

It is easy to achieve almost 100% hit rate by using very wide PIs. For project planning however, very wide intervals have little value. The width of the PI should therefore be used in addition to hit rate when evaluating a PI. *Prediction interval width* (PIWidth) is a measure of the width of a PI, and is defined as

$$PIWidth = \frac{Maximum\ Value - Minimum\ Value}{Estimate}$$

2.4.3 Overall approach

The approach used to generate effort PIs in this thesis is based on the one suggested in [4]. The approach uses the accuracy values of completed software project's estimates to generate PIs, based on a requested confidence level.

The idea is to select a set of projects or tasks with the same expected degree of estimation uncertainty, and use the distribution of estimation accuracy values from these projects to calculate PIs for our estimate(s). When first studied, results showed that this approach was able to more accurately find a PI which corresponds with the confidence level requested, than human judgment [4].

2.4.4 Organization of the rest of this thesis

Section 3 reviews previous research on software cost estimation and previous approaches to uncertainty assessment. Section 4 details the approach to estimating project effort based on estimation data from previously completed projects. Section 5 details the technologies utilized and the implementation of the approach in a web application. In Section 6, the results of evaluating the application against a data set from a large Norwegian software project, with data from three different companies delivering development services to that project, are presented. Section 7 discusses the results of the evaluation and the potential for further research. Section 9 will conclude.

3 Related work

A lot of research has been conducted on software development cost and effort estimation. In a 2004 review of software cost estimation studies [6], Jørgensen and Shepperd searched more than a hundred potentially relevant, peer-reviewed journals for papers on software cost estimation. They found 304 relevant papers in 76 journals. These papers were classified based on research topic, estimation approach, research approach, study context and data sets. Regression-based estimation is the estimation approach most discussed, with about half of the papers focusing on this approach. With expert judgment-based estimation being the approach most commonly used in the software industry, it might come as a surprise that only 15% of the papers discussed this estimation approach. The amount of research into formal models

suggests that many researchers have focused on replacing expert-judgment as an estimation method altogether instead of improving it or complementing it. The proportion of papers focusing on expert judgment- and analogy-based estimation approaches is increasing, from respectively 7% and 2% of articles focusing on these approaches before 1990 to 21% and 15% since 2000. Only 25 papers were found to discuss uncertainty assessments, but the number of papers on uncertainty assessment has been increasing since the year 2000. The authors identified only 12 articles on uncertainty written before 2000, while there were already 13 papers discussing uncertainty assessment between 2000 and 2004.

Among the papers on uncertainty assessment there are several suggestions on approaches to assess risk in software development cost and effort estimation. In the next sections we will review and discuss some of the reasons for the inherent uncertainty in software development effort estimation, and some of the approaches and tools that have been suggested as a means to quantifying this uncertainty.

3.1 Uncertainty in software development estimates

In 1997 an article was published by Kitchenham and Linkman [7] discussing the sources of uncertainty inherent in software estimates, and how risk and uncertainty can be managed in an organization. The authors argue that most people who are involved in software estimation agree on one thing; good estimates require estimation methods and models that are based on your organization's performance, working practices and software experience. Because an estimate is a probabilistic assessment of a future outcome, a point estimate cannot be expected to be precise. A point estimate might be the most likely effort, but the uncertainty and risk inherent in development effort makes a probability distribution a better representation of reality.

In [7], the sources of estimate uncertainty are divided into four categories; *measurement error*, *model error*, *assumption error* and *scope error*. These categories describe sources of uncertainty present in an effort estimate. The model is the method used by the estimator to generate an estimate, and it could be based on human judgment or a mathematical formula. Whatever the estimation model is, it can be a source of uncertainty. *Model error* is inevitable because the model is not an exact representation of reality. It is impossible to include all the factors affecting the actual effort of a real life software development project in an estimation model. Even if estimation models are based on results from previously completed projects, future projects and their environment are not likely to have the exact same properties. This does not mean that the model will perform poorly; it means that even good estimation models, which perform well on average, will not be able to be perfectly accurate. *Measurement error* occurs if input to the model has accuracy limitations. An example of this could be the inability to accurately assess the size of a project in cases where this is input to your estimation model.

Assumption error occurs when the wrong assumptions are made about the model's input parameters, such as having faulty assumptions about a project's requirements. *Scope error* is the type of assumption error when the project is outside of the estimation model's domain. If your project is not in the estimation model's domain, it is difficult to provide a meaningful estimate. The authors show examples of calculating risk exposure based on estimated effort and the potential magnitude of the error sources. In conclusion they suggest that risk management should be done on an organizational level, not project level.

3.2 Modeling uncertainty in effort estimates

Jørgensen [8] describes models attempting to explain the accuracy and bias variation of an organization's estimates of software development effort through regression analysis. The models include factors important for predicting accuracy as well as most likely effort. The study was performed on data from a Norwegian web-development company, spanning 49 tasks estimated to be bigger than one work-day. All independent variables were binary and those included in the analysis were: company role (project manager/developer), task complexity, contract type (fixed price/hourly rate), importance (to the customer), customer priority (cost or time to delivery), level of knowledge, participation in completion of the work, previous accuracy for similar tasks, estimated effort (log-transformed), actual effort and estimation accuracy.

Regression models using the accuracy measures Magnitude of Relative Errors (MRE) and Relative Error (RE) as dependent variables were developed by applying stepwise regression with backward elimination. Looking at the results, the MRE model suggested that estimation error increases when a developer estimates a task as opposed to a project leader, when the estimator is not involved in the completion of the task and when customer prioritizes time-to-delivery rather than quality or cost. The RE model suggests that project leaders are more likely to underestimate than software developers. Project leaders explained this by saying that slightly low estimates would push project members to work more efficiently.

The analysis does not exclude the importance of the other factors mentioned above, and the author mentions that they might have been included if there were more observations. The low number of observations for different combinations of variables also means that the results should be interpreted with care. The majority of variance in estimation accuracy was not explained by either model.

Analysis of model residuals and the estimators' own descriptions of reasons for high/low estimates leads the author to conclude that formal models cannot be expected to explain most of estimation accuracy and bias variation, unless the amount of observations is unrealistically high. Also, some important causes for low estimation accuracy are related to rare events, project management and execution issues. The article suggests that the advantages of formal

models should be combined with the advantages of expert judgment in order to improve estimation.

3.3 Expert judgment and uncertainty assessment

Among the 304 studies on cost estimation identified in [6], 184 papers introduced new variants of formal models for software cost/effort estimation. Still, the most common estimation methods used in the software industry are entirely judgment-based. In [9], Jørgensen reviewed the evidence on use of expert judgment and formal models in software effort estimation. The review found that the accuracy of expert judgment-based estimates outperformed formal model-based estimates on average. In addition, expert judgment-based estimation is a simpler approach, when compared to the use of formal models which add significant complexity to the estimation process. Despite expert judgment being a sound approach to effort estimation, an expert judgment-based approach to uncertainty assessment might not be ideal. Several studies suggest expert judgment leads to overconfident uncertainty assessments [3].

The uncertainty of a software effort estimate is often indicated through the use of a PI (min and max values) corresponding to a stated confidence level, e.g., 90%. Jørgensen et al. [3] presents the results of four studies on prediction intervals based on human judgment. The first study found that the prediction intervals were too narrow. Three industrial projects and 13 student projects were included in the study. From a total of 100 software projects, only 30% used effort PIs and only three had recorded actual effort. The industrial projects' PIs had no stated confidence level, but were described as the minimum and maximum value and that it should be "very unlikely" that the actual effort fell outside of the interval. The hit rates of the three industrial projects were 33%, 44% and 21%. When the students were asked to estimate a PI with a confidence level of 90%, the resulting average hit rate was 62%.

The second study investigated the impact on effort PIs of project role and experience. The study was conducted in the Norwegian branch of a large international e-commerce organization where the company was paid for participating in the study. Twenty professionals estimated most likely effort and a 90% effort PI, first individually, then in teams. Each team consisted of an engagement manager (customer interaction), a project manager, a software developer and a GUI designer. The study found that the people with the least technical background had the most realistic effort PIs. One reason could be their need to take an "outside view" approach to estimating the project effort. From an outside view of the project it is perhaps easier to take a statistical approach to the problem and use previous experience with similar projects and tasks as the basis for estimation. Looking at the results of the individual estimation, only two of the people participating in the study, two engagement managers, managed to include the actual effort in their PI, making the overall hit rate 10%. After individual estimation, the participants were divided into teams, and agreed on a common team estimate. Two of the teams managed

to include the actual effort in their effort PI. The team with the best results used a most likely effort +- 50% approach. This suggests that it might be useful to use another approach than human judgment when assessing uncertainty. This study shows the same strong bias towards overconfidence, resulting in much too narrow effort PIs, as the first study.

The third study investigated the impact of the requested confidence level on the produced effort PI. Interviews from the first two studies suggested that participants did not put much thought into the requested confidence level when producing an effort PI. Based on this information, a study was conducted on four teams of students asking for effort PIs on a programming task/project with respectively 50%, 75%, 90% and 99% confidence. The results showed that the median width of the PIs in each group was almost equal. Rationally speaking, the higher confidence effort PIs should have been much wider than the lower confidence ones.

The fourth study focused on effort PI interpretations. Interviews indicated an unwillingness to provide sufficiently wide effort PIs due to not wanting to provide "useless, much too wide effort PIs" to the project manager, making them seem like they lacked competence. The study involved 37 software professionals with management experience. The participants were presented with two scenarios concerning effort estimation and asked to answer questions regarding each scenario. Both scenarios involved two software developers providing most likely effort and effort PIs for the same development tasks. In the first scenario the two developers have the same most likely effort and confidence level, but one has much wider PIs, leading to a better hit rate. Still almost all participants preferred the developer with narrower, less realistic PIs. They also thought he knew more about the tasks he estimated, while they thought the developer with wide effort PIs knew more about uncertainty. In the second scenario, the two developers had the same hit rate (60%), actual effort, estimated effort and effort intervals. The only difference was that the PIs had different confidence levels, 60% and 90%. There was no preference for one over the other, but many thought the developer who produced the 90% confidence PIs had more knowledge of the development tasks. Most subjects thought the developer with the 60% confidence intervals had more knowledge about uncertainty. These results related to perceptions of skills, could explain why too narrow PIs are generally produced by expert judgment. The results correspond to the result from the second study, where more knowledge about how to perform tasks led to narrower effort PIs.

These studies suggest that there are some problems with judgment-based effort PIs. The results show that there may be other factors than accuracy that are dominant when software professionals produce effort PIs, such as the goal of appearing skilled and providing "useful information" to the project manager. Also, many software projects do not record actual effort. This means that estimators receive little or no feedback on the accuracy of their effort

estimates and their uncertainty assessments. The authors suggest that this might also be one of the reasons for the inaccurate effort PIs.

Another study [10] investigated the effect of introducing lessons-learned sessions on estimation accuracy. Learning from experience is a commonly proposed strategy for process improvement in software development. Twenty software professionals participated in a study where they estimated and completed five software development tasks. They were divided into a learning group and a control group. The learning group was told to spend 30 minutes on analyzing and reflecting over their effort estimates and uncertainty assessments after completing each task. The study found that the lessons-learned sessions had no effect on the accuracy of the effort estimates or the realism of the uncertainty assessments in the learning group compared to the control group.

In [11], Jørgensen suggested that in order to get better uncertainty assessments from expert judgment, we may need to reframe the way we ask for uncertainty assessments. When asking for 90% confidence intervals, the produced intervals usually end up having a hit rate much lower. In his study in [11], Jørgensen frames the uncertainty assessment question differently, by asking "how likely is it that the project will exceed 50% (half) of the estimate?" and "how likely is it that the project will exceed 200% (double) of the estimate?", and compared it to the traditional "provide an interval which is 90% likely to include the actual effort". The results indicated that the alternative framing performed better in terms of hit rate corresponding to confidence level.

In [12], the viability of the concept called *the cone of uncertainty* is investigated. This concept reflects the idea that estimation accuracy improves and uncertainty decreases as a project progresses. Assuming that estimation accuracy improves as a project progresses is a reasonable assumption. More information about the project will be known as it gets closer to completion, and it is reasonable to assume that estimates can be adjusted accordingly, making them more accurate than they were at project conception. The author analyses project data from 106 software development projects at Landmark Graphics spanning a period of three years (1999-2002). Each project's project manager recorded several aspects of the project weekly, and the estimates of the projects were updated regularly, eight times during an average project. The estimation method used was expert judgment. Results of the analysis showed that there was no significant reduction in uncertainty as a project progressed, but that uncertainty was constant through the project lifetime (relative to remaining effort). In conclusion the author acknowledges that it might be better to successfully manage uncertainty instead of trying to reduce it.

Studies [3, 10] showed significant issues related to judgment-based uncertainty assessment. Even in [10] where feedback was immediate, and the motivation for improving should have

been high, lessons-learned sessions seemed to have no effect. This could lead to the conclusion that learning-based improvement in a normal on-the-job environment is not very effective. Based on the research done in [12], it would also seem that even as projects progress, and more information is available, estimate uncertainty does not decrease. These insights motivate the search for other approaches to uncertainty assessment than expert judgment. Over the years, several methods for uncertainty assessments have been suggested, and tools have been created. The downside of many of these is that they are dependent on using input and output from formal estimation models that are not commonly used in the software industry, and are therefore to a lesser extent suited for a context of judgment-based effort estimation.

3.4 Proposed approaches and existing tools

In the 1990s, when most studies on cost estimation focused on regression-based estimation models, several approaches to uncertainty assessment were suggested, and some tools were created for the purpose of uncertainty assessment [13, 14]. These tools were based on regression-based models and on the assumption that existing knowledge gained from previously completed projects could be useful. The next sections review and discuss previously suggested approaches to uncertainty analysis and quantification.

3.4.1 Expert Cocomo II

Expert Cocomo II was a tool developed to identify risks during cost estimation [13]. The author argues that software risk management is often difficult to implement because of unique characteristics of single software projects and the need to make use of related expert knowledge, which is not always accessible. He suggests leveraging on existing knowledge and expertise during cost estimation by using cost factors to detect patterns of project risk. The method is based on analyzing patterns of cost-driver ratings submitted for a Cocomo [15, 16] estimate to find sources of project risk, as further described below. The Cocomo model uses regression with parameters derived from historical project data, and current project characteristics to estimate effort based on project size and adjusting the estimate based on cost driver ratings.

The uncertainty assessment approach behind Expert Cocomo II is primarily for project analysis and input anomaly detection. The approach uses the cost drivers in the Cocomo model as a complete set of attributes for risk diagnosis. The method is based on identifying risk situations by looking for specific combinations of extreme cost driver values. These combinations were identified through the use of expert knowledge and cost estimation and risk management studies. The risk situations were formulated as a set of rules. Based on these rules the method identifies risk situations and input anomalies.

An example of an identifiable risk situation taken from the article would be that the input suggests a *tight schedule* and *low application experience* amongst staff. This could trigger a risk

situation, because low application experience could mean using time understanding the application domain, and with a tight schedule, this might not have been planned for. An example of an input anomaly would be combining project characteristics such as *a large application*, *not been done before* and *low complexity.* A large application that's never been developed before often suggests complexity, suggesting that the input is inconsistent.

Risk situations are categorized into categories. A weighting scheme is used to calculate overall risk for the project based on assigned risk levels and cost multiplier data. The tool quantifies overall project risk on a scale from 0-100 where 100 is a project where all cost factors are rated at their most expensive. In addition to providing an assessment of overall project risk, the *Expert Cocomo II* tool provides an effort estimate (person months), a schedule (project duration), an overview of the risk categories and their risk levels, and a ranked list of risk items which shows the user what items the tool identifies as having the most risk. Statistical testing done using historical data showed correlation between the calculated overall risk level of the project and the number of person-months per thousand lines of code produced.

3.4.2 **RiskMethod**

In 1991 Kari Känsälä specified an integrated quantitative method, RiskMethod [14], to combine cost estimates and risk assessment. A tool called RiskTool was developed to support the method. The method assesses risk based on the probability and magnitude of risk items, where risk identification is done manually by using a risk item checklist. The method has to be supported by a cost estimation method and the risk items have to have corresponding cost drivers in the estimation model. In the paper, risk items were identified by collecting large amounts of risk data from 14 major Finnish software companies. With the help of these companies a list of the top fifty risk items was compiled. The same software companies weighted the top fifty items according to probability and impact magnitude in a later questionnaire. These risk items were then integrated into RiskTool. By using the information gathered from the software companies, generic values for probability and impact magnitude were generated. An impact rating for each cost driver is then calculated using regression. This is done under the assumption that impact rating is proportional to cost driver values and that cost driver values are proportional to estimated effort. With these parameters configured, risk exposure can be calculated based on the probability of a risk event and the impact magnitude should the risk event occur. Simply put, the user identifies potential risk items in the project and the probability that they will occur. The tool then calculates risk exposure and adjusts the estimate.

In the study described in the previous paragraph, RiskTool was integrated with Laturi, a Finnish function point analysis tool, and tested in selected Finnish software companies. The method was compared to other estimation approaches, such as using Laturi alone and expert judgment.

The results showed improvements over using Laturi alone, with average estimation improving and the amount of projects achieving a +-15% estimation accuracy increasing from 58% to 90% when using Laturi *and* RiskTool. There was, however, a few concerns regarding the complexity of the method, even if the tool simplifies its use. Another possible problem is that the risk items are related to cost drivers which are already calibrated in the estimation model. If the cost driver ratings are already based on data where risk events have occurred, the effect of these will be double-counted. Also, the manual identification of risk and risk probabilities makes the tool subjective to judgment-based uncertainty assessments.

3.4.3 Estimation by Analogy

Estimation by analogy is, at the outset, a reasonable approach to software cost estimation. The approach is based on the reasonable assumption that similar software projects have similar costs, and that using information and knowledge from previously completed projects will improve cost estimation. The technique has four steps; retrieving similar cases, reusing information from these cases, revise the proposed solution and record new experience that could be valuable to future problem solving. The two main advantages of analogy-based estimation are that it is easy to understand, and that it can model a complex set of relationships between the dependent variable (effort or cost) and independent variables (cost drivers). An important part of the approach is to retrieve similar projects, and to do this projects have to be assigned a set of attributes for comparison purposes.

In [17] uncertainty is investigated in an estimation method called Fuzzy Analogy [18]. Fuzzy Analogy uses principles of *soft computing* to produce effort estimates based on analogy. Human intelligence is very good at reasoning by analogy, but human reasoning by analogy uses approximations and not accurate values when doing so. In soft computing the hypothesis is that precision and certainty comes with a cost, and that tolerance of imprecision and uncertainty should be exploited whenever possible. Fuzzy Analogy handles vague information and imprecision in describing and comparing projects, and when producing an effort estimate based on the chosen analogies. By using a graded membership function when describing project properties it is possible to represent imprecise values such as *low*, *young* and *complex.* Fuzzy Analogy produces a single numerical estimate which says nothing about the uncertainty of an estimate, which prompted further research into the uncertainty of estimates generated by Fuzzy Analogy.

The authors of [17] investigate two of Kitchenham and Linkman's four previously mentioned sources of uncertainty and risk in the context of Fuzzy Analogy, measurement error and model error. In Fuzzy Analogy, project attributes are measured by numerical or linguistic values. The authors argue that measurement values should be linguistic rather than numeric to reduce measurement error. They base this argument on the fact that most software projects attributes

are qualitative rather than quantitative. Linguistic values are easy to understand, allow for imprecision and accept that human reasoning is not based on completely precise information. Measurement error when using numerical values is evaluated by the difference between estimated and actual value, while when using linguistic values it is evaluated as the difference between the estimated and the actual degree of membership to the linguistic values. The authors suggest that the effect of measurement error will be higher when using numerical values. By using linguistic measurement values, the authors conclude that they can avoid or at least reduce the effect of measurement errors and as such they do not consider this source of uncertainty in their revised model. The present author notes that this reasoning applies when "numerical values" are interpreted as numerical values on a rational or interval scale. The difference between linguistic and numeric values is less obvious with numerical values on an ordinal (or even nominal) scale.

The authors aim to deal with uncertainty in the model by generating a cost distribution using *possibility theory*. *Possibility theory* was introduced as an extension to fuzzy sets and fuzzy logic back in 1978 [19], and is an alternative to probability theory. The Fuzzy Analogy approach is based on the rationale of case based reasoning that similar projects have similar costs. The article does however conclude that this affirmation is non-deterministic in the *Cocomo'81* data set, i.e. a project can be similar to many projects in the data set but the costs of these projects are different. They have therefore used a non-deterministic approach in order to handle uncertainty in the Fuzzy Analogy approach.

The non-deterministic approach is based on the affirmation that "similar projects have *possibly* similar costs". Based on this affirmation, the article proposes a rule "the more similar project P is to project P_i, the more likely that the cost of P is similar to the cost of P_i". To generate a possibility distribution for the cost of project P, fuzzy sets of possible values are generated from similar projects. They are then combined by a max-based aggregation in order to obtain the fuzzy set with possible values for the cost of P. This combination represents the possibility distribution for the estimated cost, which can be used to generate a point estimate and a risk assessment.

Other approaches to analogy based cost estimation have been studied as well. In [20], a statistical simulation procedure is suggested to generate analogy-based interval estimates. It investigates methods of parameter configuration and ways to generate interval estimates both with and without assumptions about the theoretical distribution of the data set. The approach uses the *bootstrap method* to draw samples from a set of completed projects to estimate effort. A point estimate is generated by looking at the mean of the projects which are most similar to the project being estimated. To generate effort intervals an algorithm generating a large number of independent bootstrap samples is used. The algorithm draws analogous

projects randomly into samples with as many projects in them as the original data set using replacement. This means that the sampled data sets can contain some projects more than once and some projects not at all. Based on the actual results of the randomly drawn projects, the sets represent different simulated estimates. This distribution is used to generate effort intervals.

3.4.4 Assessing uncertainty based on previous estimation accuracy

Jørgensen and Sjøberg describe an approach for generating effort PIs based on empirical project data in [4]. This approach forms the basis for the methods and tool developed as part of the present thesis. Their article argues that the lack of simple models for generating effort PIs lead to the observed situation where human judgment is the base for generating software development effort PIs. The problems related to human judgment-based effort PIs are mentioned earlier in this paper, the main problem being that human judgment based effort PIs are much too narrow, for high confidence levels in particular.

The approach suggested in the paper is based on well-known statistical principles of prediction intervals. The article lists these main steps for the suggested method:

1. Select a measure of estimation accuracy that enables a separate analysis of bias and spread.
2. Find a set of projects with the same expected degree of uncertainty as the project to be estimated.
3. Display the distribution of estimation accuracy for the selected projects. Use the information provided by the historical distribution of estimation accuracy to determine how to best calculate an effort PI.
4. Decide the confidence level, e.g., that it is 90% certain that the actual effort of the project will be within the PI, and then calculate the effort PI.

The approach is based on the assumption that future projects will have similar estimation errors to those selected as a basis for generating effort PIs. Bootstrap and regression models require a formal model describing the variation in use of effort. The approach suggested here can be used on all kinds of estimates, expert judgment or regression based, as long as the selected project estimates have a similar degree of uncertainty. An advantage to the approach is that it is easy to use and understand. In order to find a 90% confidence level effort PI using the empirical distribution of accuracy, simply use the 5% percentile for the minimum value and the 95% percentile for the maximum value. The authors argue that the simplicity of the approach makes it less prone to over-fitting.

The approach used in the article uses the Balanced Relative Error (BRE) as the measure of estimation accuracy. In software development studies, the Magnitude of Relative Error (MRE) is

often used. MRE does not enable a separate analysis of bias and spread as it uses the absolute value of the difference between actual effort and estimated effort. Removing the absolute value would enable this analysis, but the authors chose to uses BRE in their implementation of the approach.

$$BRE = \begin{cases} \frac{Actul - Estimate}{Actual}, Actual \leq Estimate \\ \frac{Actual - Estimate}{Estimate}, Actual > Estimate \end{cases}$$

$$MRE = \frac{|Actual - Estimate|}{Actual}$$

Using MRE as the estimation accuracy measure makes it impossible to separate overestimates from underestimates. For this measure to be effective, the expected estimation errors would have to be normally distributed around the unbiased median.

The article suggests using parametric accuracy distributions if there are reasons to believe that the estimation accuracy can be approximated by a particular distribution. However, given a sufficient number of previous observations the empirical accuracy distribution might be a better choice. The empirical distribution approach simplifies the method, and makes it easier to understand and use.

While effort estimates can be evaluated against actual values, effort PIs have no corresponding actual value. To measure the accuracy of the effort PIs, the hit rate is used. In the long run, hit rate should correspond to confidence level. If the confidence level is 90%, then nine out of ten actual efforts should be within their estimated effort PI. In addition to the accuracy of the effort PIs, the efficiency can be measured by the width of the generated effort PI. Obviously, the more efficient (narrow) an effort PI is, the more useful it is to a project manager as long as it does not affect accuracy.

$$PIWidth = \frac{Maximum\ Value - Minimum\ Value}{Estimate}$$

In the authors' evaluation of the approach, the median PIWidth was used instead of the mean to avoid strong impact from a few very wide or very narrow PIs.

The approach was evaluated against a data set from another study on estimation accuracy [21]. The data set contains information on estimation method and size for each project. The projects were divided in four clusters; a small and large set for expert judgment and a small and large

set for estimates based on tools or structured estimation process. The projects in three of the clusters were biased towards too high effort estimates.

To calculate effort PIs for the different projects, projects from the same cluster was selected for the learning set, and cross-validation was used so that the project itself was not part of its own learning set. Three types of effort PIs were calculated for confidence levels 60% and 90%, two kinds using parametric approaches and one using the empirical distribution of accuracy. In the evaluation, the empirical approach came out on top, generating the most accurate effort PIs and having about the same median PIWidth as the other effort PIs with a similar hit rate.

The empirical approach was then compared with regression and human judgment approaches. The human judgment-based effort PIs were generated by thirteen paid software professionals. The experiment was based on fifteen software development tasks completed in another organization. The software professionals were given five tasks to use as background data, and then estimated most likely effort and 90% confidence effort PIs for ten tasks with feedback on their performance after each task estimated. They had limited relevant estimation experience as the information on each task was very limited. This was not considered a validity issue, since less information should lead to wider effort PIs, and not reduced accuracy (hit rate) in the effort PIs. Only the last five tasks were used in the evaluation, giving the software professionals some chances of learning from their performance when estimating the first five effort PIs.

For the empirical effort PIs, the BRE values of the estimates provided by each of the 13 software professionals on task 6-9 were used to calculate one effort PI for each person for task 10. Then the BRE values of task 6-10 were used for calculating the effort PI for task 11, and so on, until task 15 was estimated.

The regression-based effort PIs were calculated using a log-linear regression model with variables size and effort based on task 1-9. The model was not expected to be very accurate, but the point here was to illustrate important differences between empirical and regression-based PI approaches. Two variants of the regression model based prediction were applied; PIs provided by the regression model and PIs based on the previous estimation accuracy of the regression model. Because only one set of effort PIs is calculated for the regression-based approaches, the evaluation results related to the regression-based approach are likely to be more uncertain than those related to the human judgment and empirical approaches where there are thirteen different sets of effort PIs.

Because the human judgment approach achieved only a 68% hit rate with the 90% confidence level, results of the empirical approach with a confidence level that resulted in the same 68% hit rate was added. The width of the effort PIs produced by human judgment were more efficient than those of the empirical approach for the same hit rate, suggesting more efficient

use of the uncertainty information. Both the empirical approach and the regression-based approaches had a hit rate that was somewhat low compared to the requested 90% confidence level. The effort PIs based purely on the regression-model were much wider than the ones produced by the empirical and the empirical approach using the regression-model accuracy. The approach based on the empirical distribution of the regression-model accuracy had the best performance when it came to the width of the predicted PIs. This suggests that combining a formal model to find the estimate with an empirical model to find the effort PI might be a recommendable approach.

The overconfidence in the human judgment effort PIs came as no surprise as this was one of the motivations behind the BRE-based effort PI approach described in the paper. There are other studies [3] suggesting that it is a problem when using human judgment for producing high confidence level effort PIs. The participants had problems describing their strategies for predicting effort PIs and they were to a large extent intuition-based. Formal approaches and human judgment approaches seem to have complementary advantages, with humans being able to use uncertainty information more efficiently while the formal approaches does not suffer the same overconfidence as the human judgment approaches, i.e. the hit rate of the effort PIs correspond better to the requested level of confidence.

The article concludes that there is a lack of research on the use of effort PIs in the software development projects. Based on the two studies in the article, some insight into when to use this approach, regression-based approaches or human judgment approaches is given. The tradeoff between accuracy and efficiency seems to be one of the major challenges.

3.5 Main principles of uncertainty assessment and software cost estimation

Some important insights have been gained on the basis of previous research on uncertainty assessment and software cost estimation. One is the importance of employing knowledge from similar projects when estimating effort. In [5] Jørgensen argues that experience in knowing how to complete a task differs from knowing how much it will cost, or from knowing the uncertainty of the effort estimates. He found that accurate estimates were often produced when the estimator could draw on experience from very similar projects. In [22] the same principle is suggested as a guideline for uncertainty assessment: Base uncertainty assessments on an outside view of the project, where you can compare the uncertainty properties with previously completed projects, and use data on accuracy from these projects.

Expert judgment is the most effective way of using available uncertainty information, but expert judgment based uncertainty assessments are generally overoptimistic. Formal models have problems explaining the majority of variance in estimation accuracy and compensates by creating PIs that are so wide that they become less useful. Formal models do however have the

advantage that they are not biased towards overconfidence, which is the main problem with expert judgment uncertainty assessments.

In conclusion, by combining expert judgment with a formal model we might be able to use the advantages of both approaches. Expert-based estimation should be supported by tools and checklists to be able to include as much relevant information as possible, and a formal model based on previous estimation accuracy could support the uncertainty assessment. In the next section we describe an approach that is meant to do just that.

4 Approach

In this section we describe an approach to assessing uncertainty for a set of estimates based on the empirical distribution of estimation accuracy. It is based on the idea behind the approach described in [4], but includes the following variants and extensions:

- Employ historical data on accuracy of low-level tasks to produce probability distributions for aggregated effort. This can be useful to make release level PIs on the basis of task-level accuracy data. Monte Carlo simulation is employed to produce such distributions.
- Allow for smoothing of empirical data sets, i.e., to fit empirical data to a parametric distribution, as the fitted data may be a better representation of the underlying uncertainty.
- Let the user decide the extent to which estimation bias in the historical data should be reflected in the PIs.
- Possibility of narrowing the empirical distributions, to take into account possible process improvements.
- Possibility to let the user simulate historical data by using an assumed parametric distribution.
- Possibility to tag historical and future tasks, to facilitate the identification of tasks expected to have similar estimation error.

The approach contains four steps. First step is to select an appropriate set of historical data, with pairs of estimates and actual effort. To produce meaningful results, we need to use the data which is most likely to have the same estimation errors as the task we are estimating. In our approach we enable tasks from previous projects to be classified differently in the empirical data sets. This means that you can separate tasks that are likely to have different estimation accuracy distributions. If tasks that are related to new application features are more volatile than say graphical user interface tasks, our method allows the data sets for the two kinds of tasks to be separated. Tasks that are related to new features will then be estimated based on another subset of data then the GUI tasks.

The next step is to input all task estimates for the project period to be estimated. Our approach is meant to be a complement to whatever estimation method is used. Because it uses the empirical accuracy distribution for assessing uncertainty it does not matter what kind of estimation method is used or how accurate the estimates are. If the estimates are historically very inaccurate, then the resulting effort PIs will be wider, but it should not affect the accuracy (hit rate) of the produced effort PIs. If your data set contains task classification sets, then individual estimates can be tagged to use either of these classified subsets.

The following measure is used for estimate accuracy:

$$Accuracy = \frac{Actual\ effort}{Estimated\ effort}$$

The third step is an automated aggregation of effort by randomly drawing task outcomes from the empirical distribution. For each task, a random task from the selected empirical data set is drawn. The task estimate is then multiplied by the accuracy factor of the drawn task and added to the total effort for an iteration of the simulation. By using Monte Carlo simulation with this approach, a large number of potential outcomes are generated. The set of simulated outcomes represents a probabilistic distribution of actual effort.

The fourth step (also automated) is using the percentiles of the probability distribution to propose a PI based on the requested confidence level. Using the probabilistic distribution from the Monte Carlo simulation we can generate effort PIs. If the requested confidence is 90%, then the minimum value is set to the 5% percentile of the distribution and the maximum effort is set to the 95% percentile of the distribution. This means that when simulating the amount of actual effort one hundred times, and sort the outcomes by size, the fifth effort value would become the lower limit of the PI and the ninety-fifth effort value would be the upper limit.

5 Tool construction

5.1 Functional view

5.1.1 User stories

The following sub-sections summarize the functionality of the developed tool, using *user story format*. User stories are a compact way of expressing the actions, users and goals in a compact and consistent manner.

5.1.1.1 Historical data

These user stories enable historical data to be made available to uncertainty estimators in the organization. Generally, one or a few people in the organization will be responsible for maintaining high quality data sets. See Figure 3, Figure 4.

As an estimation process owner I can easily register historical data sets, give them names, edit and delete them, so that I maintain a high-quality base of historical data.

As an estimation process owner I can generate data set based on a theoretical distribution, so that I can start without historical data or simulate other situations.

As an estimation process owner I can graphically view the accuracy distribution of historical data sets, so that I get a quick view of historical estimation accuracy.

5.1.1.2 Entering estimates and parameters

These user stories enable the user to trigger the uncertainty assessment, based on a set of new estimates, knowledge of which historical data set to use, and which tuning options are relevant in his case. See Figure 5.

As an uncertainty estimator I can assess the uncertainty for the aggregated effort for a set of tasks, for example all the tasks in one development iteration.

As an uncertainty estimator I can take differences in uncertainty between task types into account when assessing uncertainty, so that the uncertainty assessment becomes more accurate.

As an uncertainty estimator I can fit historical data to a theoretical distribution, so that the data set possibly better reflects the underlying process.

As an uncertainty estimator I can narrow the spread in my historical data, so that increased estimation consistency in my organization is reflected.

As an uncertainty estimator I can adjust the bias in the historical data, so that increased estimation accuracy in my organization is reflected.

5.1.1.3 Output

These user stories enable the user to get an easy to understand overview of the uncertainty of sum effort for the tasks given. Single point output, percentile output and graphical output is supported. See Figure 5, Figure 6.

As a project manager I can calculate the probability that my estimate exceeds, or does not exceed, a given effort value, so that more informed decisions about budgeting and bidding can be made.

As a project manager, I can calculate the amount of effort I will not exceed, with a given probability, so that more informed decisions about budgeting and bidding can be made.

As a project manager, I can calculate all percentiles on the probability curve for aggregated effort, so that more informed decisions about budgeting and bidding can be made.

As a project manager, I can print a textual and graphical summary of my calculated results and the assumptions made to support them, so that I can easily present them to other people in my organization.

5.1.2 User interface

Figures 3 to 6 render the user interface of the developed tool.

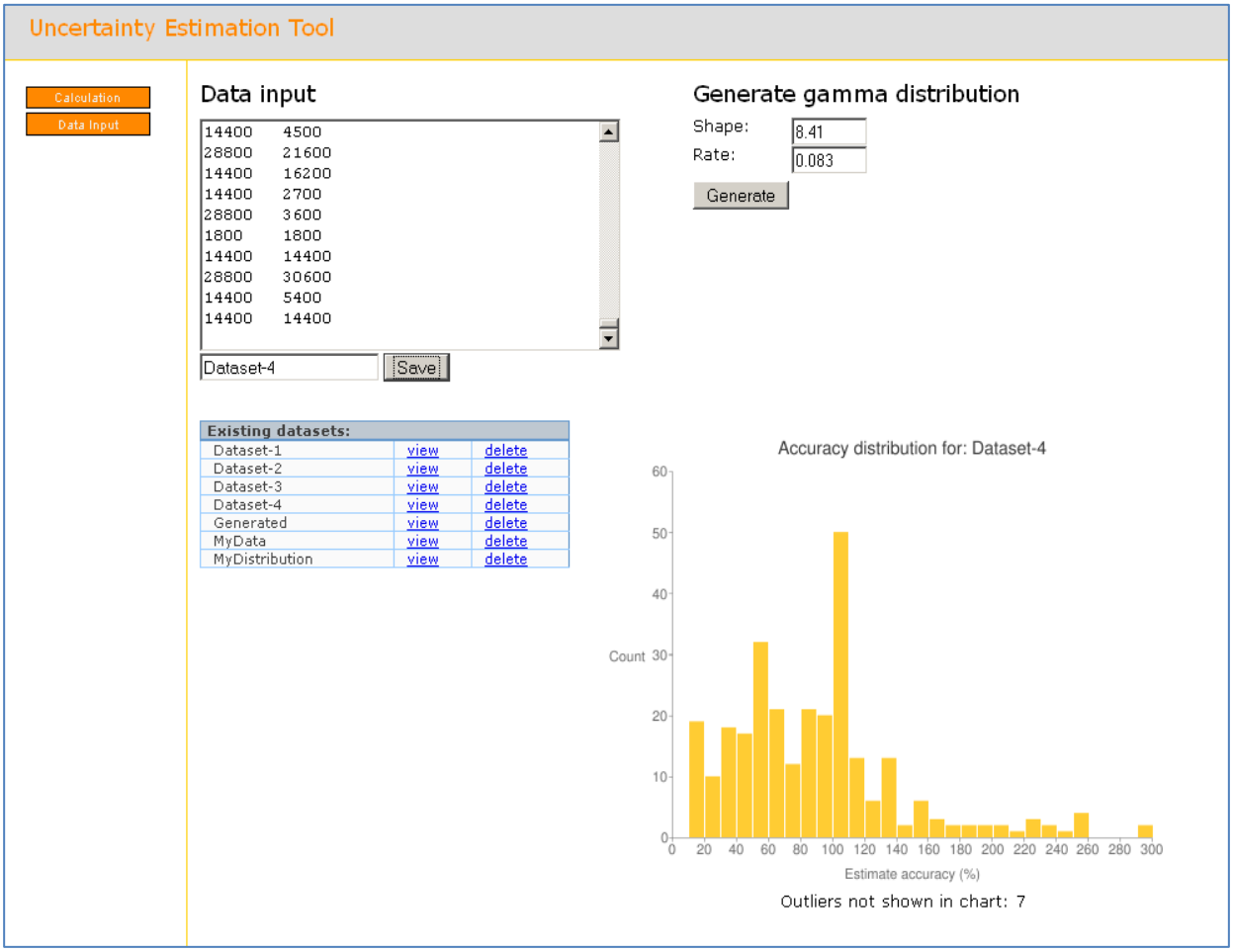

Figure 3 - Input of empirical estimation data

Figure 3 shows the user inserting historical estimation data into the tool by inserting two columns of values, one for estimates, one for actual effort.

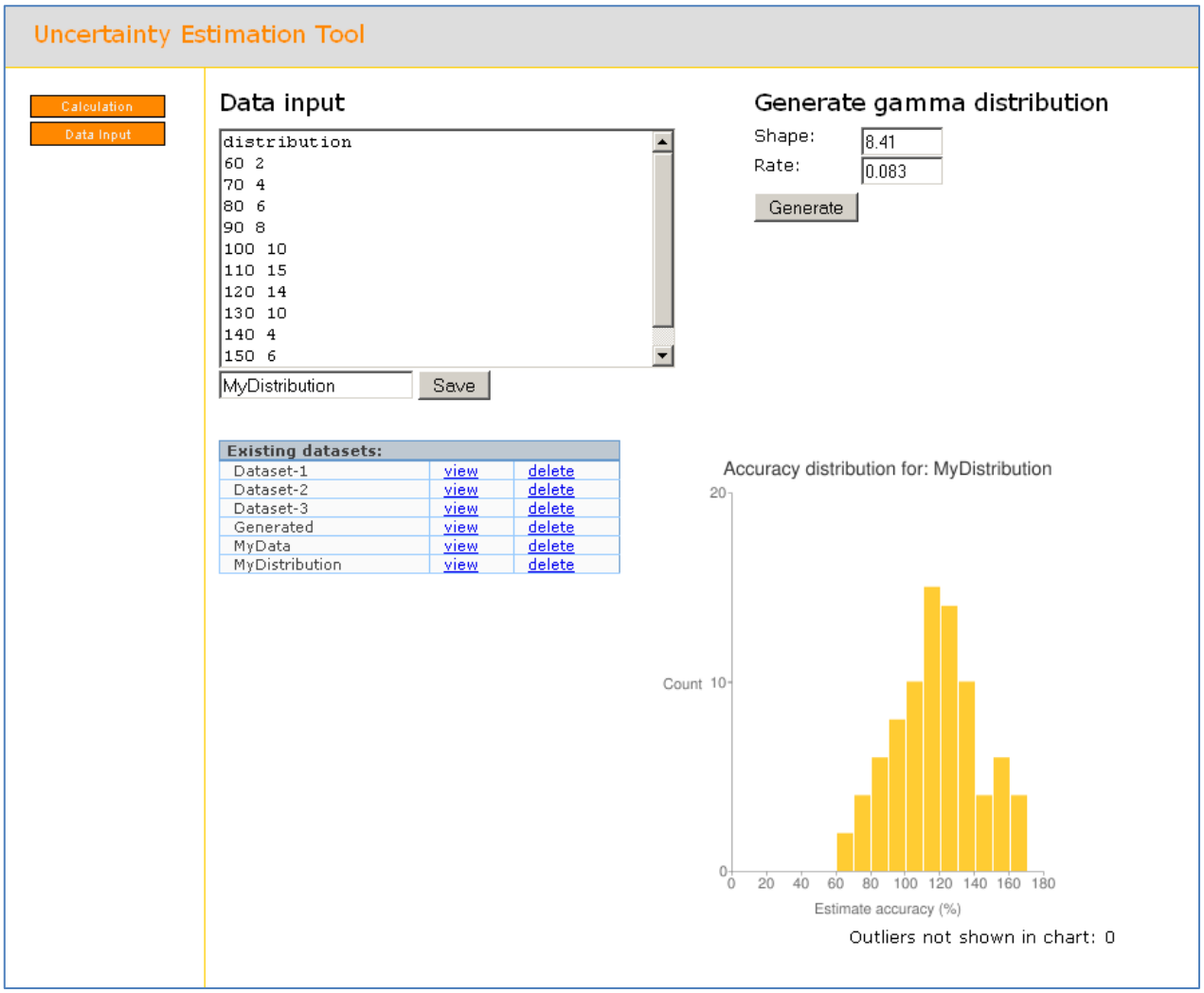

Figure 4 - Input the distribution of accuracy directly

Figure 4 shows the user inserting a distribution of accuracy directly into the tool, without providing any estimated or actual values.

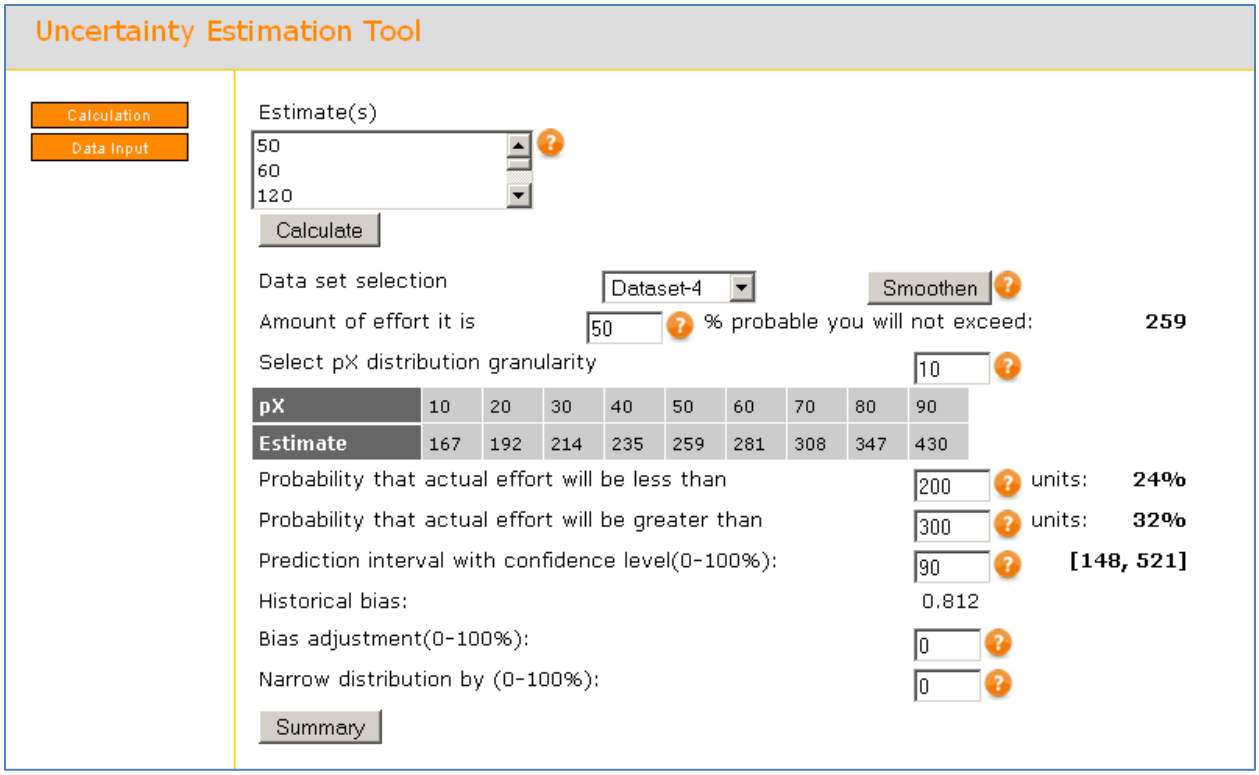

Figure 5 – Input of parameters and calculation - the page also shows the graphical distribution of the selected data set, but it is not included in this screenshot

Figure 5 shows the main calculation page of the application. The user provides estimates and selects the data set he/she wishes to use for calculations. One or more calculation parameters are provided, and the tool calculates and renders the corresponding results.

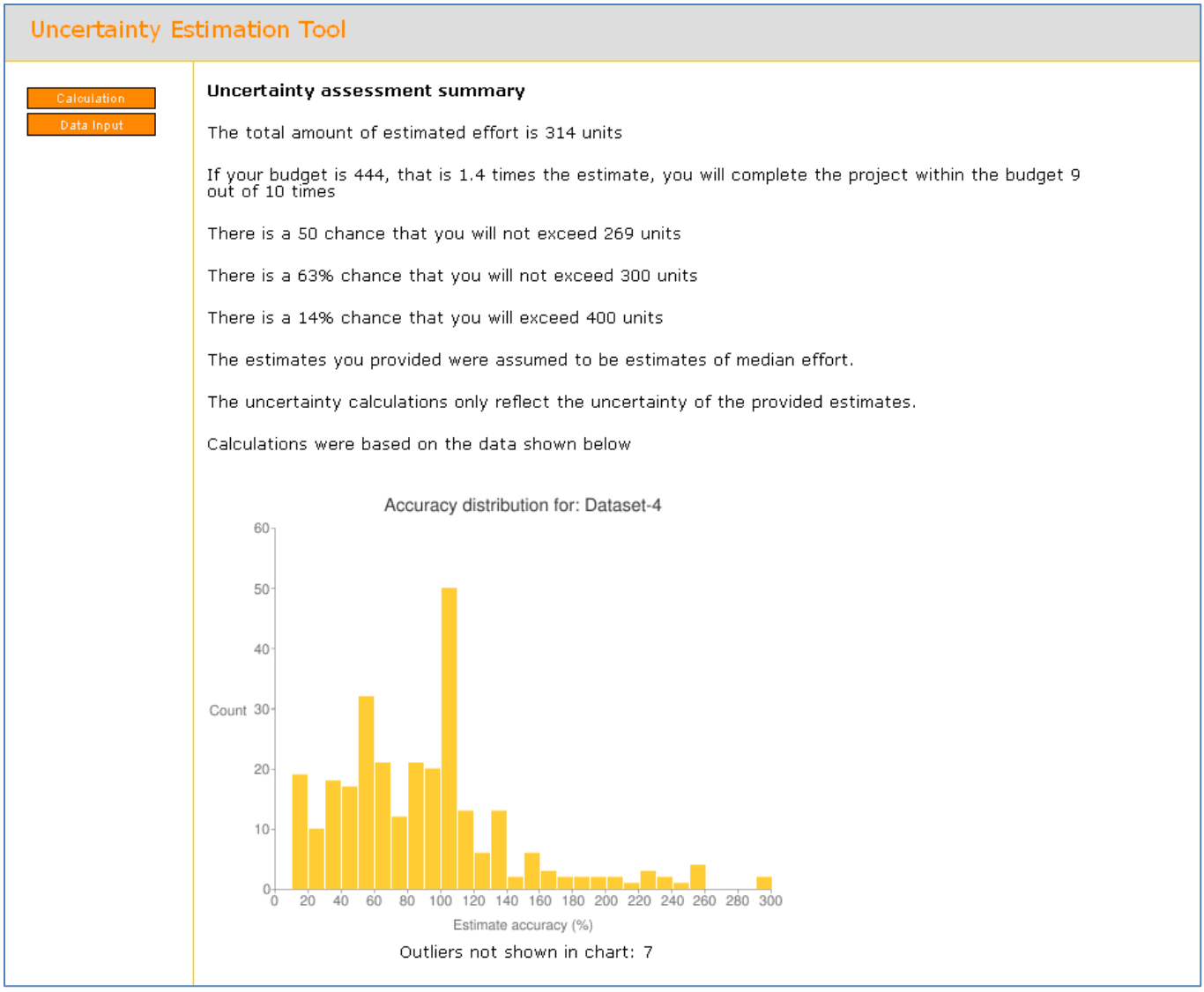

Figure 6 - Summarizing results based on the provided input parameters

Figure 6 shows the effect of creating a summary of the calculated results. What is shown in the summary depends on what kind of input parameters the user has given. It will mention the inclusion of tuning options such as bias adjustments, and will always show the distribution of accuracy for the historical data used, including potential narrowing or bias adjustment.

5.2 Architecture

5.2.1 Development platform

The tool is developed as a Java web application with a database backend for persistence. The tool is developed using *wicket* - an open source, component-based web application framework for Java. *Hibernate* is used to handle the persistence part of the application along with the *Spring* framework, which is also used to handle configuration, inversion of control and transaction management. A java package called *JRI* is used to integrate the tool with R, making it possible to call R from a Java application. In addition, a modified wicket API for the *Google Charts API* is used to graph data.

The project will be built using *Maven*, an open-source software build tool. Maven allows you to automatically download dependencies to the project from publicly available repositories and offers an easy approach to building. The developed code is versioned under *Subversion* (SVN), an open-source revision control system, in order to maintain control of changes and have a reliable backup of the code base.

5.2.1.1 Java

The tool is packaged as a *Java Web Archive*, and has most of its core functionality written in Java. Originally, the tool was to be written in Ruby on Rails, but lack of experience with the technology and issues regarding integration with R prompted a switch to Java.

5.2.1.2 Maven

Maven is a software management tool primarily used to manage a project's build process. In addition to being a build tool, it can assist with testing, run web applications and provide a number of other features through plug-ins. You can create reports on unit tests, dependency lists, IDE files and generate change-log documents directly from the project source.

In order to create a maven project a Project Object Model (POM) file is needed. This file includes basic project properties such as name and version, information about dependencies, repositories and build configuration. All POM files extend a super-POM by default, which contains basic setup of repositories and build information. This means that if you leave out configuration details, maven will use default values. For example, the default packaging type is "jar", but this may be overridden to "war" in the project specific POM file.

Maven dynamically downloads dependencies during building from one or more repositories. The current version's super POM is configured to retrieve dependencies from the Maven 2 Central Repository. Downloaded dependencies are stored in a local cache so the user does not have to download artifacts every time you build. Your projects will also be installed into the user's local repository with the *install* command.

Maven comes with plug-in support for generating project files for several IDE's. This makes it easy to import Maven projects into tools such as Eclipse, without having to include IDE-specific project files in your source repository. Developers can just checkout the source files, including the POM, and generate vendor-specific IDE files locally.

5.2.1.3 Application server

The tool can runs as a web application under any Java Servlet container such as Apache Tomcat, GlassFish, Jetty or JBoss Application server. This makes it easy to deploy on any platform, and does not impose any other platform dependencies than Java does.

5.2.1.4 Subversion

Subversion is an open source version control system. Despite the project only having one developer, a version control system was used to track changes and provide backup of the code base. Having the source code centralized also provides development progress and provides the possibility of checking out the project and running it locally.

5.2.2 External components employed

5.2.2.1 Wicket

Wicket is a lightweight component-based web application framework for Java. It separates logic and presentation into plain Java code and XHTML, a stricter cleaner version of HTML. While many frameworks introduce some kind of special syntax to your XHTML, wicket only uses a single *html attribute*. One page is linked to a single XHTML template. Each component is linked to exactly one element in the XHTML template. Reusable parts of a page, can be extracted into components called panels, also paired with a single XHTML template, and can then be inserted into pages and other panels as a component. It is also possible to manipulate html attributes programmatically, making it possible to for example manipulate an element's classes dynamically, creating visual effects. Wicket also has built-in support for doing Ajax [23] operations, and has several built-in Ajax components.

Traditional MVC [24] frameworks work with whole requests and whole pages, mapping requests to controllers which generate the complete response, usually by pulling data from a model to populate a view. Wicket on the other hand is a component-based, stateful framework. All server side state is automatically managed. The wicket application is built as a tree of stateful components, where all components maintain their state through their own model. The framework does not have knowledge of how the components interact with their models, which are serialized and persisted between requests. Basically each page is a nested hierarchy of stateful components, maintained by Wicket through each user's session.

All components use their own model. A model can be as simple as a wrapper for a String or an Integer, or it can be more complex, retrieving object-trees from a database or through REST

[25], or even creating images. By using flexible repeater components, you can pass a list of objects (or a model that provides one) and create HTML structures such as tables and lists by applying it to table or list elements. Repeaters can be nested the same way other components can be, making it easy to handle advanced data structures.

5.2.2.2 Hibernate

Java 1.5 introduced the Java Persistence API (JPA) for managing (and standardizing) relational data in Java applications. Hibernate is an implementation of this API. Basically, Hibernate is an Object/Relational Mapping solution for Java, implementing the functionality of the JPA. Object/Relational Mapping is the method for mapping data from an object model representation to a relational data model representation, and back. Hibernate can automatically map Java classes to database tables, and Java types to SQL data types. It is highly configurable making for example a switch of database provider easy. It also provides a query API that eliminates the need to write traditional SQL, and eliminates the problems of vendor specific SQL. Hibernate will convert the tabular representation of a query result set to a graph of objects, making the transition between database and application seamless.

5.2.2.3 Spring Framework

Spring framework is an open source java application framework that provides several modules for use with the Java platform. Spring's goal is to foster integration between your application and existing java solutions. Two major features of the Spring framework are used in the solution, *dependency injection* and the *Spring JPA*. There are open source projects for using both Hibernate and Wicket with the Spring framework, so integration with the other technologies will be seamless. We will also use Spring to manage and resolve configuration parameters for our application, by either reading them from a file or from the Java Virtual Machine (JVM) environment.

The fundamental functionality of the Spring framework is dependency injection. Dependency injection is used to decouple Java components from other Java components. The *single responsibility principle* in object oriented programming states that every object should have a single responsibility, and that responsibility should be entirely encapsulated by the class. This makes reuse of code easier, and allows for testing of individual classes. Consequently, a class' dependency to other classes should be managed by another component than the class itself. This means that if a class needs to use functionality from another class, it should not programmatically create or look up this resource. By moving all such control to the outside world, the class offers a very clear set of functionality and dependencies, making it reusable and independently testable using mock-objects (a simulated object that mimics the behavior of real objects in a controlled way). Dependency injection, or Inversion of Control, can be handled by the Spring core container. The container manages the injection of required objects into

other objects based on configuration. This can be configured using XML configuration files or a number of other ways. Wicket-spring uses annotation-configuration for injection into web pages, meaning it is done programmatically. This differs from creating the component programmatically, as the annotation only tells Spring that a component or bean needs to be injected. The components to be injected still have to be configured in XML. This reduces the potential of ripple effects in the Java program when components are replaced or upgraded, and when doing refactoring.

Using Hibernate makes the mapping between Java objects and the database easy, but you still have to manage transactions and database connections manually. This is simplified by using the Spring JPA package where you have support for the Java Persistence API as well as Hibernate. Configuring Spring to use Hibernate's JPA implementation is easy, and it's not problematic to switch to another vendor at a later time. By using Hibernate through the Spring JPA, we can configure automated transaction handling by using Spring's own transaction manager for Hibernate. Instead of manually opening database sessions, starting transactions, changing objects, committing transactions and flushing the session, we will now simply annotate the parts of the code which require a database transaction to be started and Spring will handle it for us.

Even though we use Spring JPA for our application, it is still Hibernate doing all of the communication with the database. The components Spring uses for JPA generally just delegate responsibility to the implementation chosen in the Spring configuration – in our case Hibernate. Using Spring's JPA package, we are also able to gather the entire application configuration in one place – our Spring configuration.

5.2.2.4 JRI – A Java-to-R interface

R is a free software environment for statistical computing and graphics. The specific statistical functions needed in our application are to fit empirical data sets to parametric distributions, and to sample data from the historical data set. To do this, an existing interface for using R directly from Java code is used. It requires that a package called *rJava* is installed in the local R installation. It also requires some environment configuration, depending on which platform the Java application runs on. The interface supports simple calls to R functions and enables you to generate Java data structures using R functionality. A service was built on top of the interface, implementing all the methods we needed for our uncertainty assessment tool. Interfacing to R added complexity to the programming environment, but we were unable to find Java libraries implementing the needed statistical functions. For future extensions to the tool, the required statistical functions will very likely be available through the R interface.

5.2.2.5 Google Charts API

The Google Chart API lets you make a wide array of charts by simply constructing a URL string. You embed your chart configuration and data in a HTTP request and Google returns a *.png* file with your chart.

We will use a Wicket API for Google charts in our application. The API lets you configure your chart, wraps the configuration in a provider class, generates the URL string needed and is then passed to a chart-component which creates a HTML *img* tag which links to the chart created by the Google Charts API. The Wicket API used in our application is based on a blog post and code example released by Daniel Spiewak in December 2007.

The Google Charts API will be used for generating graphical representations of our data sets, by creating histograms showing the frequency of different estimation accuracy values, see Figure 7.

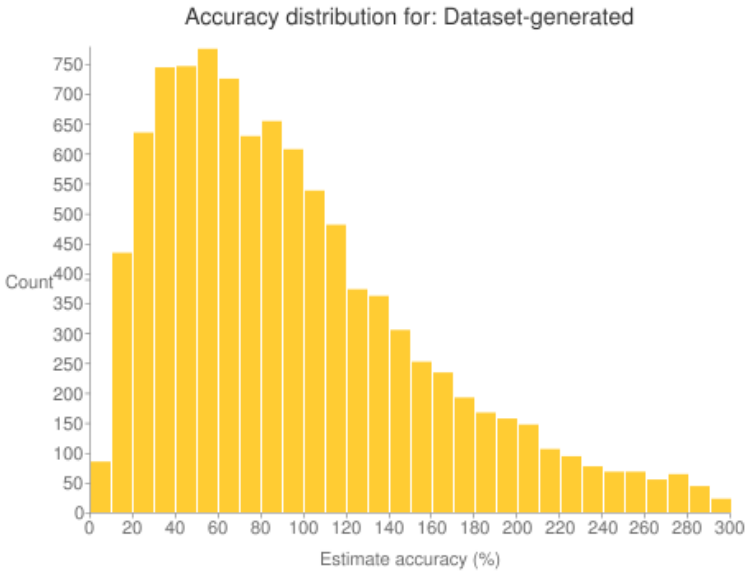

Figure 7 - Histogram generated through the Google Chart API

A couple of modifications had to be done, in order to support data value scaling and remove some minor errors that were present in the code he released. Because the Google Chart API does not support changing the range of the axis's, it does not support a wide array of values automatically. The data provided to the API is encoded using a format of characters a-z, A-Z, 0-9, the punctuation mark and – (64 characters) to represent values. The application uses the extended encoding format, which represents values by using the mentioned characters in pairs of two, giving a range of 0-4095 (64 * 64 = 4096). Using Figure 7 as an example, this range would not be ideal for the y-axis where our max value is 750. To circumvent this we rename the labels on the y-axis, and then scale our values so that they correspond to the underlying range

of values. First we find a max value for the y-axis, then we scale our bar heights using the following formula:

$$Scaled\ value = Unscaled\ value * \frac{Number\ of\ possible\ values}{Max\ value}$$

Assigning frequency values to the correct bins in the histogram is done by making sure the range and increments on the x-axis matches the number of frequencies that is provided. If we look at Figure 7, the number of bins is thirty which fits well with the range of 0-300 and the resulting increment, or bin width, of ten.

5.2.3 Logical architecture

5.2.3.1 Static view

The application is built around the component-oriented web-framework Wicket. Our view is represented by a tree of components, see Figure 8. Many components are already included in the framework, while others were implemented by extending the base classes for components in Wicket. The functionality is separated from the part of the application that uses Wicket. The components use functionality through their own model, which is simply an object that the component uses to retrieve its data. It knows nothing about the underlying functionality of the model, separating the rendering of the model and the view of the application completely from other functionality. While the components tree is managed and populated through Java code, the layout is completely controlled by XHTML and CSS, the language for describing the presentation of web pages.

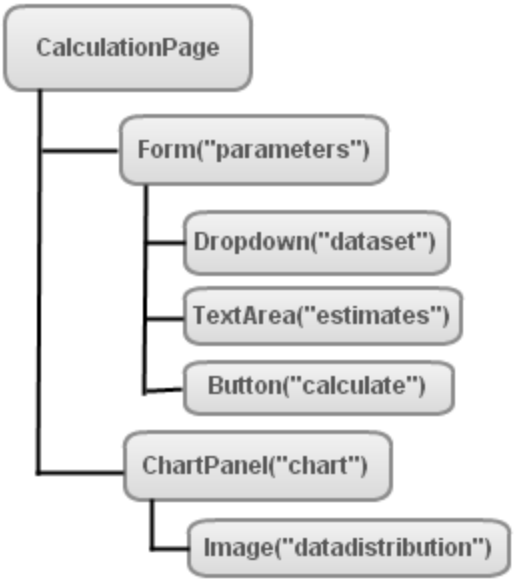

Figure 8 - Part of the component tree for the calculation page

The functionality that is used by our models to populate their components is a separated part of the application. An overview of the most important classes and methods along with their dependencies can be seen in Figure 9. The dependencies between the classes are managed by Spring.

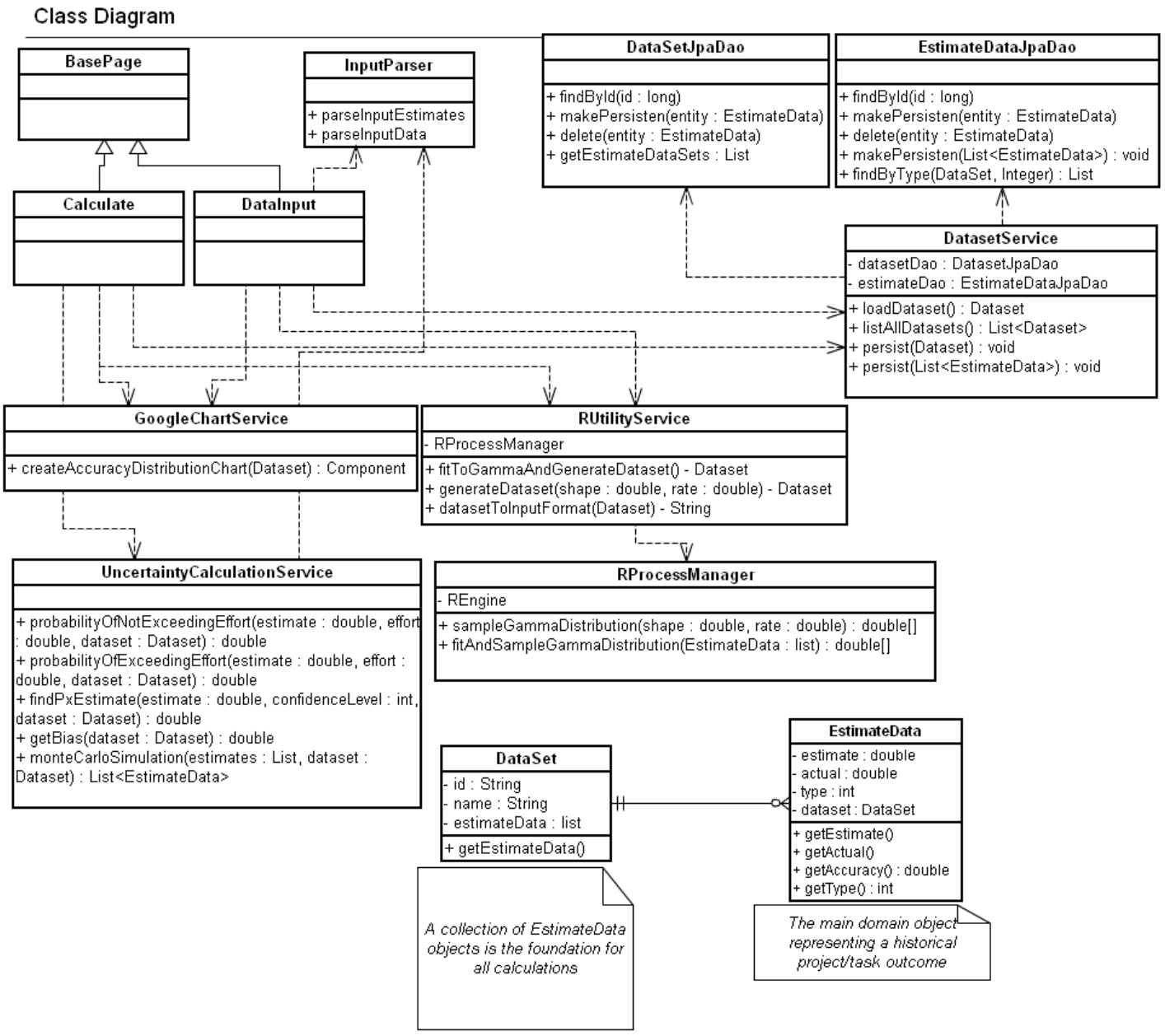

Figure 9 - An overview of the most important classes in the application

The functionality of the application is divided into different services in order to conform to the *single responsibility principle.* The separation of functionality makes the application loosely coupled and makes extensions and refactoring easier. It also makes it easy to test functionality separately.

The functionality of our tool depends on user input. While functionality in the Wicket framework's form components makes it easy to convert input values to common Java objects, we have a few cases where we want to convert the input data from a *String* to for example a

set of objects representing historical estimation data, or a set of estimates. The functionality for parsing input and converting it to our domain objects is segregated in an input parser class.

All functionality that is based on calls to the R engine is located in a service class. The actual interaction with the R engine is done through an R process management class. The R process manager performs call to the R engine which in turn returns R expressions. These are converted to an appropriate Java type through functionality in *JRI* and returned to the R service class.

All charts are generated through calls to the Google Chart Service. This service calculates axis ranges, axis increments, creates the histogram bins and their corresponding frequencies and uses the Wicket Google Charts API to build a provider for a Wicket image component.

Data access is done through the data service which uses data access objects to retrieve and save data. Each domain object has its own data access object (DAO), with functionality inherited from a generic super class. This is made possible with the use of Java generics and the Java Persistence API (JPA). The generic super class makes use of the JPA annotation called *PersistenceContext* which indicates a dependency to an *EntityManager* (hibernate-implementation used in our application), which implements functionality to manage persistent entities' object lifecycles. In addition, all DAO functionality is described through interfaces, meaning that if you wish to move away from the JPA, you can simply make new implementations of the interfaces. The data access model is described in Figure 10.

The generic DAO also uses a Spring annotation called *Transactional*, which indicates that a transaction has to be started in order to call the methods contained in the class. If no such transaction is started when you call the methods in the DAO classes, Spring will automatically start one for you. In addition to the *EntityManager* implementation and configured being handled, the database vendor, the database "dialect" and the driver class is also configured by Spring. This results in nothing cluttering of the Java code. You can change any one JPA component with another implementation without touching the data access objects, making the code highly maintainable.

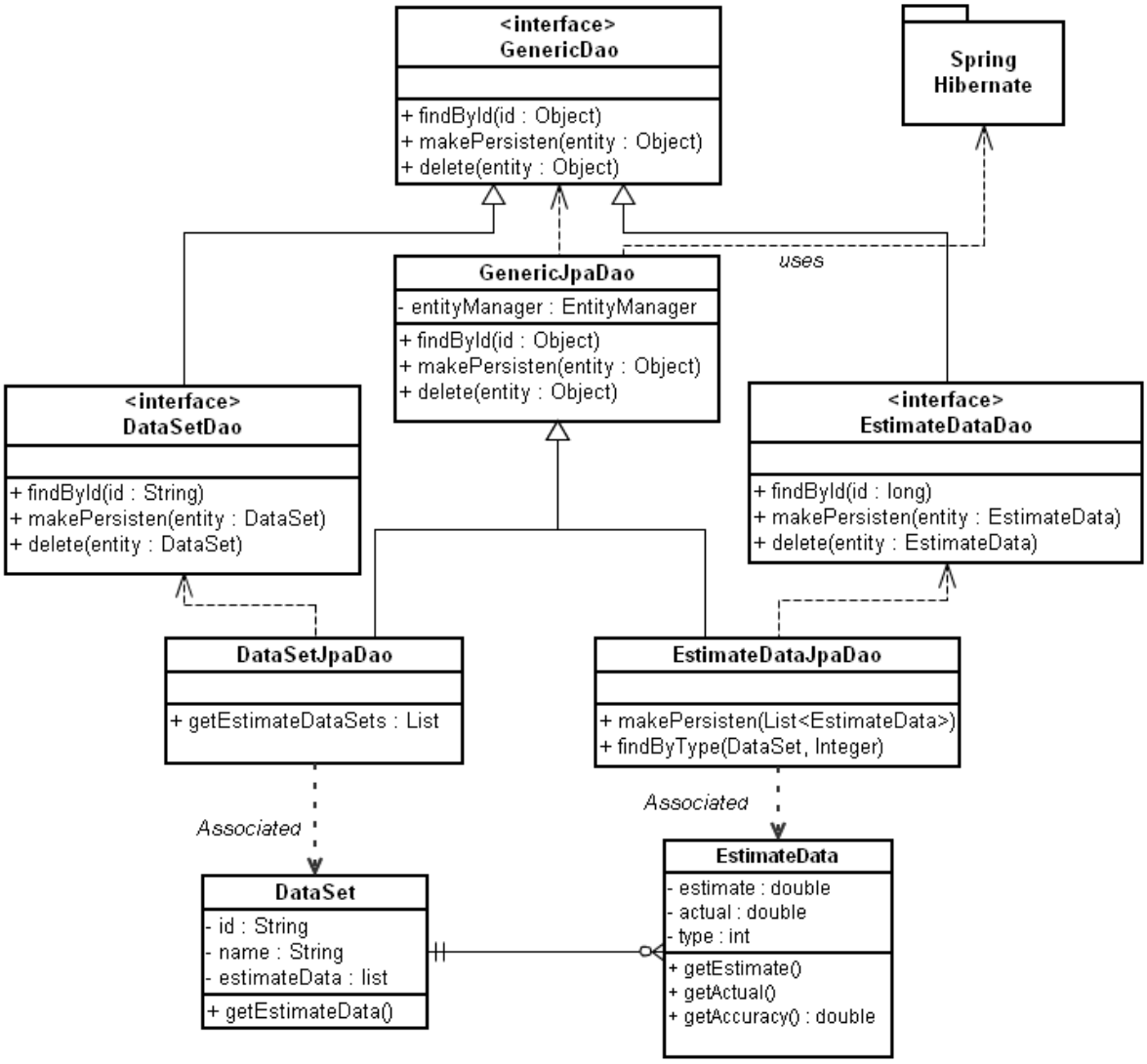

Figure 10 - Data access model

5.2.3.2 Dynamic view

All calculation parameters are wrapped in an input container object that is bound to the form on the calculation page. The input container object is then passed to the calculation class for uncertainty assessments. The calculation class calculates results based on the provided parameters and returns the results wrapped in a result object, which is rendered to the user.

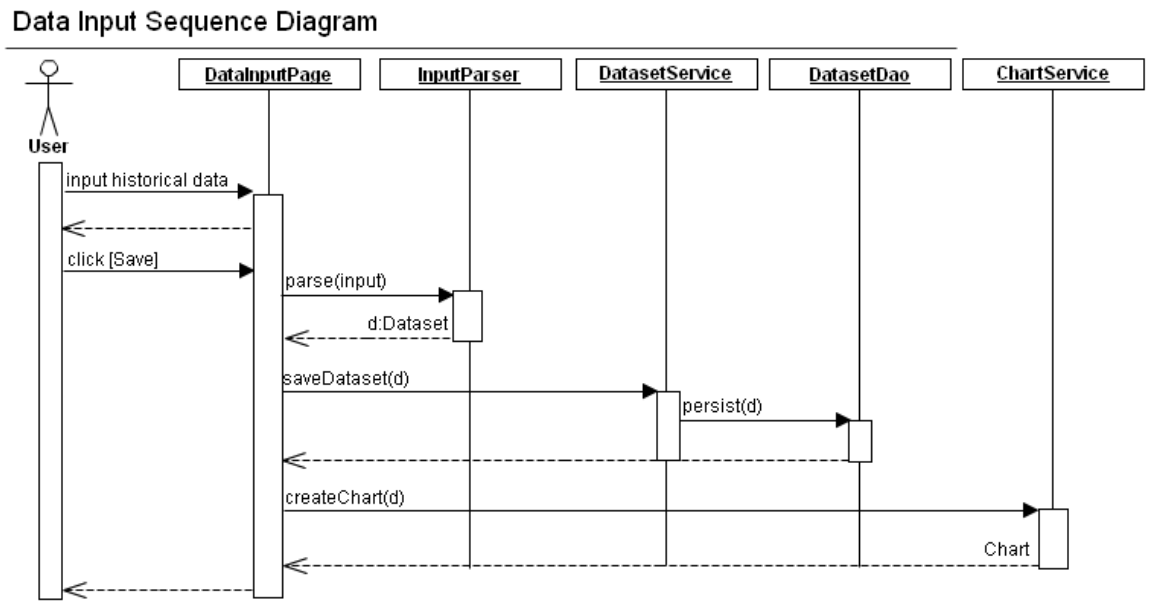

Figure 11 - User inputs historical data

In Figure 11 we see the sequence of events when a user inputs historical estimation data. His textual input is parsed and persisted, before being rendered graphically to give the user an immediate overview over the data set.

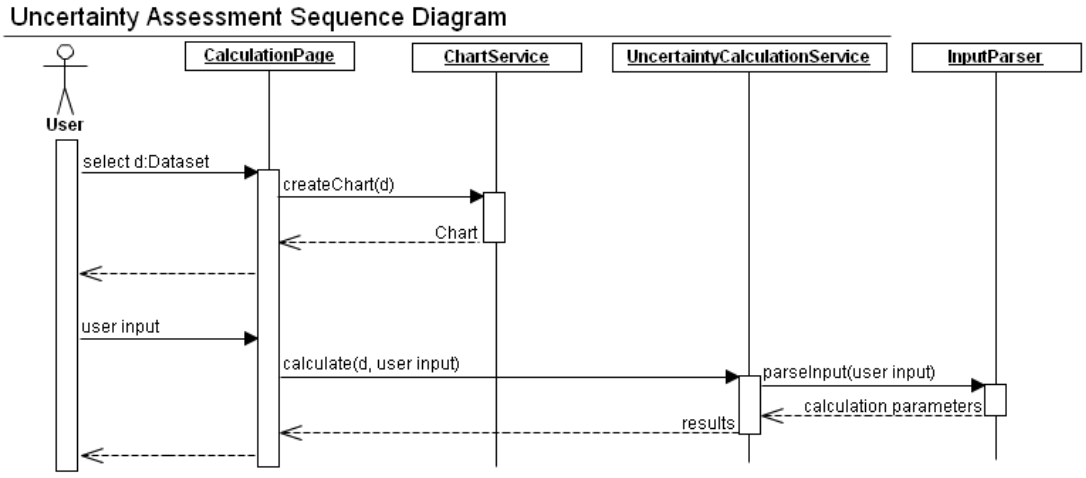

Figure 12 - User does uncertainty assessments

Figure 12 shows the process of calculating uncertainty. The user selects a data set, inputs his estimates and provides calculation parameters for the calculations he/she is interested in. The selected data set and the provided parameters are then passed to the calculation service which returns the results for rendering.

5.3 Process

The application development process was incremental, based on an initial set of requirements and wanted features, described in a Wiki maintained by Simula Research Laboratory. Early development cycles were mostly focused on frameworks and integration to R. New features and their implementation were continuously discussed with the researchers at Simula. As framework and integration issues were resolved, incremental releases and testing was performed. The later development cycles were mostly focused on bug-fixes and validation. Formal acceptance testing was performed by researchers at Simula, but thorough and detailed testing was performed by the candidate through the tool evaluation, described in the next section.

6 Evaluation of method and tool

6.1 Evaluation design

The goal of the evaluation was to assess the accuracy and efficiency of PIs for release-level effort based on task-level historical data, using different variations of the proposed method. The following variations of tool usage were evaluated:

1. Using the tool with its basic configuration. Aggregate task level effort estimates to iteration-level effort through Monte Carlo simulation, using the historical distributions of estimation accuracy for task level effort (actual/estimate).
2. Use a statistically smoothed distribution instead of the empirical one.
3. Using accuracy distributions particular to task type. Task types will be identified by comparing the accuracy dispersion for different candidate task types with overall dispersion.
4. Combining the tuning options in 2 and 3, meaning the use of statistically smoothed distributions for particular task types.
5. Accounting for process improvement – by narrowing historical distributions or reducing bias if it looks like estimation accuracy is improving over time in the empirical data sets.

All of the approaches are supported out-of-the-box by the tool. In principle all variations of the features mentioned in Section 4 could be evaluated. However, we prefer to accumulate settings that produce positive results. Positive accumulation of settings reflects the intended tuning process for the tool in a particular organization, and will give us an indication of the possibility to optimize the results compared to those resulting from a basic configuration.

The analysis will be performed in three replications using real task-level estimation data from three companies A, B and C working within a large Norwegian software project. Potential differences in results between the replications will be briefly discussed. A given data set is

partitioned into two sets of sprints. The first set will be used to retrieve historical distributions (training set) and the second set will be used for the evaluation (evaluation set).

When creating PIs with the tool in this evaluation we will request a 90% confidence level. Accuracy is evaluated by comparing the hit rate of the obtained PIs obtained with the requested confidence level. The accuracy will also be evaluated against a baseline hit rate, obtained by setting PIs to a fixed percentage above and below the estimate, so that the hit rate is 90% in the historical data.

Efficiency is evaluated by comparing the median PI width, with the median PI width for the baseline. PI width is measured as

$$PIWidth = \frac{Maximum\ Value - Minimum\ Value}{Estimate}$$

6.2 Data description

The PREPARE group at Simula Research Laboratoy had an ongoing research collaboration with a large ongoing development project in the Norwegian public sector. Each company had tracked task-level estimates and actual in the projects issue tracker tool. We agreed with the project that we could extract the data from this tool, and use it for the present purpose.

	Company A	**Company B**	**Company C**
Number of tasks	304	712	688
Number of tasks used	291	621	652
Average actual effort	11,7 hours	14,5 hours	6,4 hours
Average actual/estimate	0,93	0,99	1,03
Iterations	10	11	7
Date, from-to	06.10.2008-31.07.2009	09.06.2008-26.01.2009	16.02.2009-25.06.2009

Table 1 – Initial training data

Table 1 shows key properties of the part of the total data set used as the initial historical data in our evaluation. As can be seen in the table, some of the tasks were excluded from the original data set. This is either because there is a lack of data for the task (either estimate or actual missing) or because the accuracy of the task is lower than 5% or higher than 2000%. Exclusion of the outlier accuracy values was done because of the possibility of human error when project members recorded data.

	Company A	Company B	Company C
Number of tasks	791	1 574	1 284
Number of tasks used	775	1 497	1 239
Average actual effort	11,9 hours	12,6 hours	6,2 hours
Average actual/estimate	1,15	0,96	0,90
Iterations	13	21	14
Date, from-to	05.08.2009-16.06.2010	27.11.2008-17.06.2010	30.06.2009-03.06.2010

Table 2 - Evaluation data

Table 2 shows the same properties for the evaluation data as Table 1 did for the training data. The data shows that the mean actual/estimate for company A goes from slight overestimate to slight underestimate, while the mean MMRE is slightly reduced. It also shows that the average accuracy for company C is near 100% in the training data while it's 90% in the evaluation data. We also note that after the predictions for a given sprint have been evaluated, its data is included in the training set, so eventually, the data from all sprints except the last one will belong to the training set.

6.3 Results of the evaluation

The procedures for evaluation according to the variants described in the introduction to this section were automated to limit the chance of human error when handling the large amounts of data. The automation also significantly speeded up the evaluation process. Methods used were the basic approach (1), the usage of smoothed distributions instead of the empirical ones (2), the use of task-type particular distributions (3) and the combination of (2) and (3) as described in (4). A closer look at the data did not show any indication of systematic improvement of estimate accuracy or consistency over time. Because of this there was no use of data set narrowing or bias reduction as proposed in (5).

	Company A	Company B	Company C
Baseline: Hit rate	92%	95%	43%
Baseline: Efficiency	0.49	0.45	0.34
Basic: Hit rate	100%	86%	79%
Basic: Efficiency	0.54	0.52	0.49
Tuning-Smooth: Hit rate	77%	86%	43%
Tuning-Smooth: Efficiency	0.42	0.43	0.38
Tuning-Size: Hit rate	92%	86%	79%
Tuning-Size: Efficiency	0.45	0.38	0.37
Tuning-Size and Smooth: Hit rate	69%	86%	71%
Tuning-Size and Smooth: Efficiency	0.39	0.36	0.33

Table 3 - Evaluation results summary

Table 3 shows some key results of the evaluation; the hit rate and the efficiency of the different approaches. The efficiency is measured as the median PIWidth. More detailed information about the results can be found in Appendix A.

A more detailed look at the approaches shown in Table 3, and the results, will be discussed in the next sections.

6.3.1 Baseline

The baseline effort PIs are produced to have a base for comparison. While effort estimates can be compared to actual effort, an effort PI has no corresponding actual value. By comparing to a simple baseline we will have a way of assessing the possible advantage of a more sophisticated approach. A simple approach for a project to generate a PI at the iteration level would be to sum the task estimates, and subtract/add a factor from/to this sum. For the matter of fair comparison, we identified the factors that resulted in a 90% hit rate for the aggregated iteration effort in the training set. We will then use these factors to create effort PIs for our evaluation data. We note that the baseline approach must be considered a well-founded, empirical approach, and in fact is a simpler application of the principles proposed by Jørgensen in [4], and described in Section 3.4.4.

Company	Evaluation sprints	Low Factor	High factor	Hit rate	PI-width
A	13	0.72	1.21	92% (12/13)	0.49
B	21	0.71	1.15	95% (20/21)	0.45
C	14	0.82	1.16	43% (6/14)	0.34

Table 4 - Results for the baseline approach

Table 4 shows the factors used and the results of using these factors to create effort PIs. The hit rate is good for company A and B while the hit rate for company C is very low compared to the requested confidence level of 90%. This could indicate a weakness in the baseline approach and/or that company C had a change in estimation accuracy or bias. The PI-widths are the same for all the sprints of one company, because the same factors are used to produce the PIs, and the PI-width is calculated in relation to the effort estimate.

6.3.2 Basic approach

We first evaluated the tool without using any of the tuning options. The training data was read into the tool to constitute the empirical accuracy distribution. The rest of the data was used for evaluation by estimating PIs sprint by sprint. After a sprint was estimated, its results are inserted into the training-data before the next sprint is evaluated. This means that when we have twenty sprints, seven of which are designated training data, the first sprint estimated is based on training-data from seven sprints, while the last sprint is estimated based on data from

all of the nineteen previous sprints. This reflects a real-life process, where an increasing amount of historical data will be available as the project moves forward.

For each sprint, all task estimates for that sprint were fed into the tool. The tool performs Monte Carlo simulation and produces a distribution of probable outcomes. A PI with a 90% confidence level was recorded by taking the displayed p5 and p95 values as minimum effort and maximum effort in our PI.

Figure 13, Figure 14 and Figure 15 shows the distribution of accuracy values, for company A, B and C, respectively. The number of outliers for each accuracy level is tabulated to the right of the chart. We denote these data points as *outliers*, even if the extreme outliers (lower than 5% and higher than 2000%) were already removed from the data set. There is a high frequency of 100% accuracy registrations for all companies, and a slight tendency towards high frequency of 50% and 200% accuracy registrations compared to the surrounding accuracy values. This is probably caused by people registering slightly wrong or "rounded" actual efforts. The explanation could be that actual effort values close to the effort estimate are registered with estimated effort as the actual effort while there is actually a small difference. The same could possibly apply to the 50% and 200% registrations, if tasks required roughly half or double the estimated effort, it's easy to pin them down at half or double without further thought. The reason for this might be that the programmer is slightly uncertain as to the exact actual effort, and is easily influenced by the registered effort estimate. This could mean that the registered data is slightly erroneous, but it is probable that the real accuracy values of the tasks registered at 50%, 100% and 200% accuracy are pretty evenly distributed around the peaks we can see in the charts. Because the same peaks are present in the evaluation data as well, this should not be a very influencing factor with regards to the accuracy of the simulated effort PIs.

There are also a number of outliers that could be influential towards underruns and overruns when simulating effort. It is unknown whether any of these outliers are due to erroneous registration. In any case, slightly erroneous registration like this is likely to be present in any real historical data set of this type, and as such adds to the realism of the evaluation and hence the validity of the evaluation results.

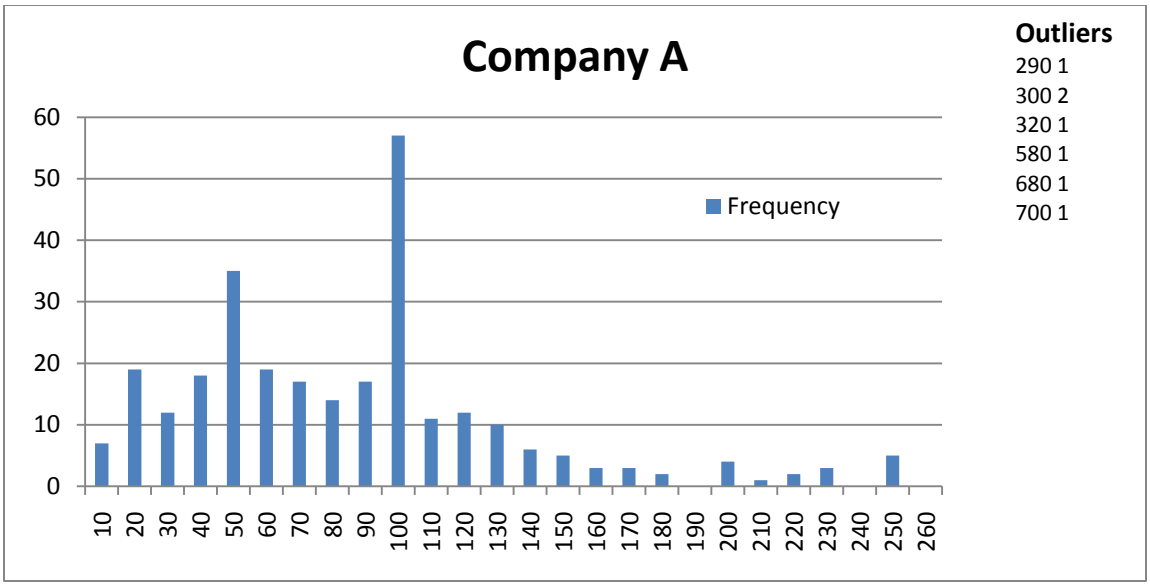

Figure 13 – Training data for company A

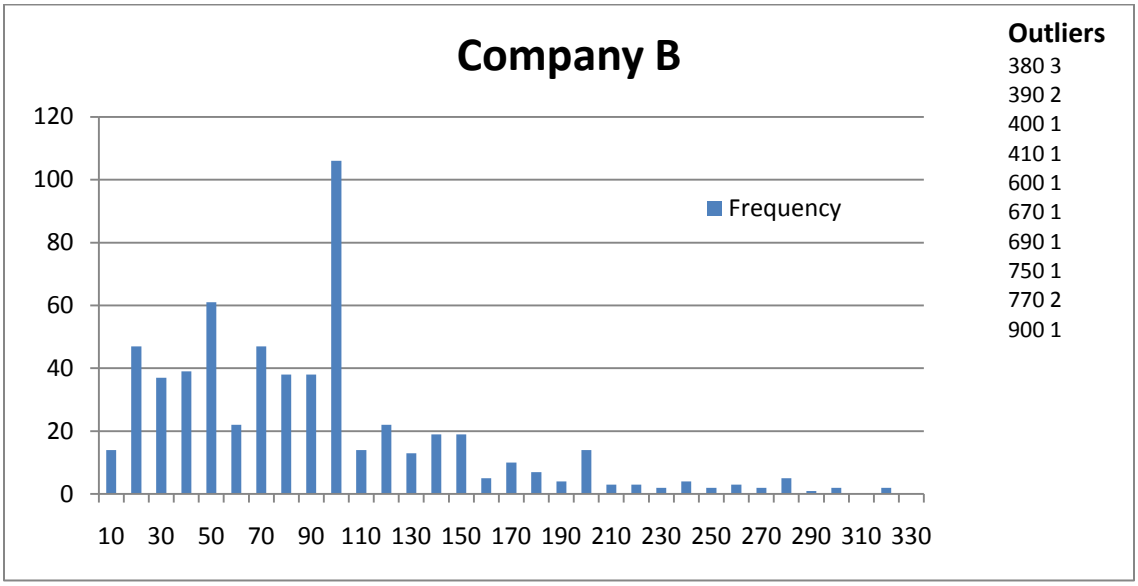

Figure 14 - Training data for company B

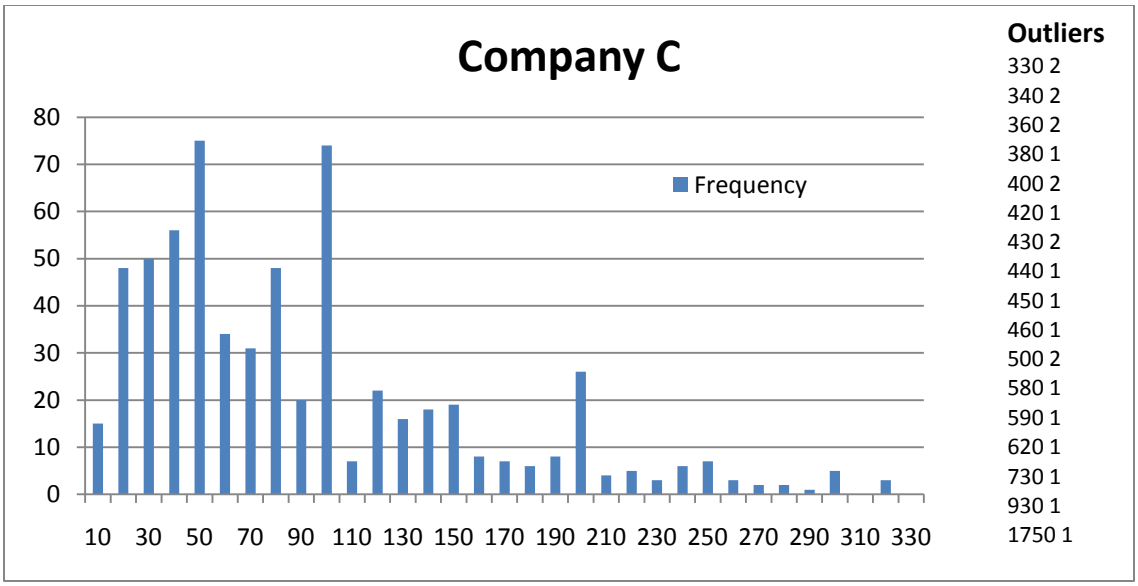

Figure 15 - Training data for company C

As shown in Table 5 the basic calculation produces hit rates that are fairly close to the requested confidence level of 90%, with slightly wider PIs than the baseline approach. The median PI width is used for comparison to the baseline to avoid strong impact from a few very wide or narrow PIs. All the sprints outside of the simulated effort PIs are overestimated, i.e., the actual efforts are lower than the lower limit of the PIs. If we look at the sprints where the baseline effort PI hit while the simulated PI did not, we see that there is a recurring pattern of a high number of tasks, and a simulated PI that is narrower than the baseline PI, see Appendix A.

Company	Sprints	Hit rate	Median PI width
A	13	81%	0.54
B	21	100%	0.52
C	14	79%	0.49

Table 5 - Results of the basic approach

There is a clear tendency of the simulated PIs width being narrower when the amount of tasks is higher. This behavior can be explained by the fact that with few tasks a task simulation underrun or overrun will have greater impact than when there are more tasks, as this leads to higher probability of having underruns or overruns in the total sprint simulation. When there are more tasks the task simulation underruns and overruns will cancel each other out to a higher degree, and the total sprint simulations will not be as volatile as a result. This is expected behavior as the uncertainty of a single task estimate is larger than the total uncertainty for a set of tasks, given that the task efforts are independent.

To gauge the impact the number of tasks in a sprint can have on the tool's simulated effort PIs' width, an additional test was run. The test was conducted by estimating five sets of tasks, with each having a different numbers of tasks. The total estimated efforts for all five of the sets were the same, and all tasks within a set of tasks were of equal size. The provided empirical data was from seven of company B's sprints. The results of the test are shown in Table 6. The more tasks there are in a set of estimates, the narrower the simulated PI will be. As more tasks are simulated, the upper and lower limit of the effort PI moves towards the mean accuracy. This means that a more efficient simulated effort PI will be produced with a large number of tasks, which is sensible under the assumption that the effort required for the tasks are independent. The more simplistic baseline approach is likely to be more robust if this assumption is heavily violated.

Tasks (Tot. est)		PI width
2	(16h)	1.81
4	(16h)	1.59
8	(16h)	1.11
16	(16h)	0.79
32	(16h)	0.55

Table 6 - Impact of number of tasks on PI width

6.3.3 Statistically smoothed distribution approach

As can be seen from Figure 13, Figure 14 and Figure 15, the accuracy of the training data was not smoothly distributed, and there were outliers in the data that potentially could be too influential. In this part of the evaluation we used the tool to generate a statistically smoothed distribution, which was used in the simulation instead of the empirical one used in the first part. The tool is able to fit the empirical accuracy distribution to the gamma distribution, and use data points from this gamma distribution as basis for the effort PI simulations.

When using values from a gamma distribution instead of the empirical accuracy distribution, the effect of outliers is likely to decrease. It could help making the simulated PIs more efficient, without hurting the hit rate significantly.

On the other hand, the gamma distribution has only one peak, which is not true for the empirical accuracy distribution for the data used in this evaluation. This could result in high statistical error when fitting the gamma distribution to the data and the resulting parametric distributions might not be a very good representation of the historical estimation accuracy recorded by the companies.

That said, and with reference to the paragraph in Section 6.3.2 about erroneous registration, there is a possibility that a one-peak, gamma-like distribution better represents the true underlying estimation accuracy in the project, because the historical data (and the evaluation

data might be contaminated by erroneous registration. Therefore, the results of using statistical smoothing might be better in practice than in this evaluation.

Figure 16, Figure 17 and Figure 18 illustrates the result of picking 10,000 random points from the generated gamma distributions. The produced accuracy distributions have eliminated the overrepresentation of 100% accuracy data points. In the original data the number of outliers was also pretty high, and some of the outliers had the potential of being very influential. With the use of gamma distributed data points the presence of outliers is almost eliminated and their influence minimized. The increase in data points should further reduce the influence of very high and very low values, and help reduce the width of the simulated effort PIs as discussed in the previous part of the evaluation.

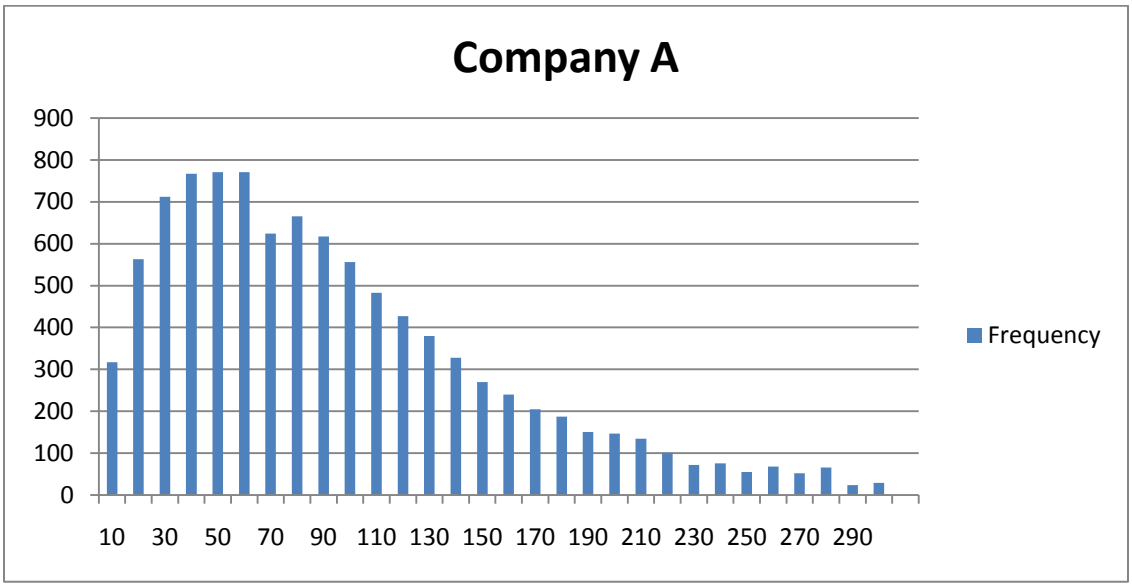

Figure 16 - Values sampled from company A's gamma distribution

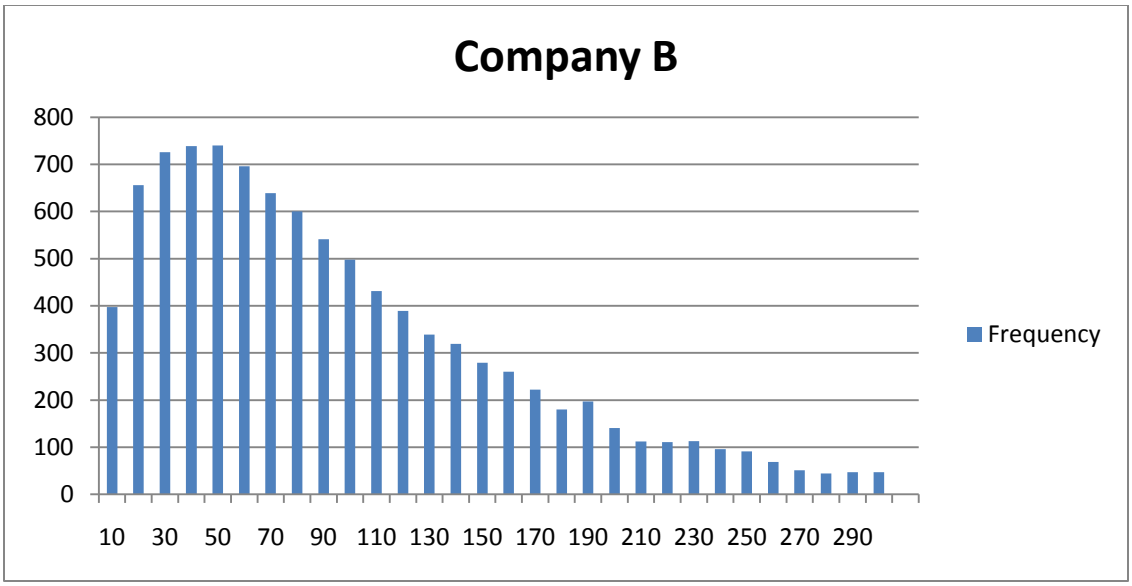

Figure 17 - Values sampled from company B's gamma distribution

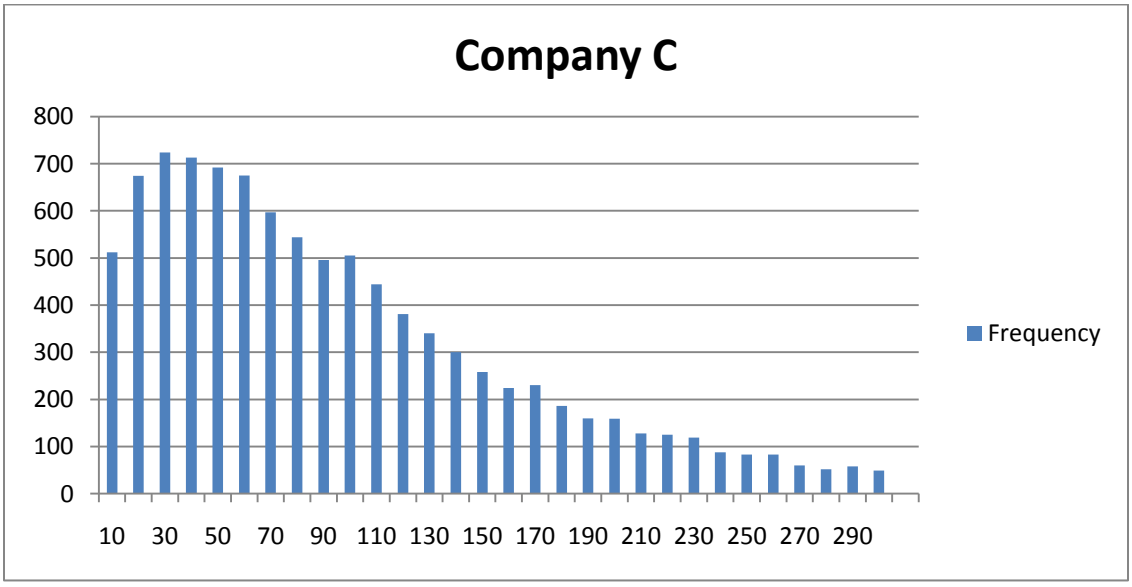

Figure 18 - Values sampled from company C's gamma distribution

We can see from Table 7 that the accuracy of the effort PIs has improved very slightly for company B, bringing the hit rate closer to the confidence level of 90%. The hit rate for company A has been reduced while it for company C has been reduced, and is now further from the requested 90% confidence level than in the basic approach. There could be a couple of reasons for this, either that the gamma distribution is unable to represent the historical estimation accuracy distribution of the company or that there has been a significant change in estimation

accuracy over time, leading to our empirical data being an inaccurate representation of their estimation accuracy.

Company	Sprints	Hit rate	Median PI width
A	13	77%	0.40
B	21	86%	0.40
C	14	43%	0.44

Table 7 - Results of the statistically smoothed data approach

Taking a closer look at the actual efforts that fell outside our simulated effort PIs across all companies we can see that they are not exclusively overestimated as they were in the basic evaluation approach. However, the actual efforts that were outside the simulated effort PIs for company C are still all overestimated. With this being the case for both the baseline and the basic approach as well, this is not very surprising. It might indicate that the company's estimation accuracy has moved towards overestimation as is also indicated by the statistics in Table 1 and Table 2.

The simulated effort PIs are now more efficient than the baseline effort PIs, with the exception of company C, where the baseline approach produced narrower effort PIs than for the other companies. The cause of the narrower effort PIs is probably caused by the more continuous probability distribution, removing influential outliers and the lower relative number of data points towards the "edges" of the distribution. A result of this is that the simulated effort PIs will have less big underruns and overruns in the Monte Carlo distribution, leading to a more compact distribution of simulated sprint results and narrower PIs.

Overall the approach seems to improve our results as long as the historical accuracy distribution can be represented by the gamma distribution. The results showed an improvement in the efficiency of the effort PIs. The hit rate however, was worse for two out of the three companies. With the hit rate being as good as it was in the basic approach, it might not be a good idea to force a parametric distribution on the data set. The less encouraging results for company C implies that it is probably not a good idea to use this approach when the empirical accuracy distribution has several peaks and lots of extreme outliers, as this is surely going to lead to problems fitting a gamma distribution to such a set of data. However, again we mention that our evaluation is conservative, in the sense that we evaluate against recordings of actual effort that might be contaminated by erroneous registration.

6.3.4 Using distributions particular to task type

The tool comes with an option to tag historical tasks and tasks to be predicted, so that a match can be performed with respect to the tag. This means that tasks can be categorized based on properties that affect estimation accuracy. When simulating effort for a task with a category

set, the tool will only use the empirical accuracy distribution based on tasks within the same category.

The candidate categories for using this approach in our evaluation were originally going to be task type, such as “Improvement”, “Bug”, “New feature” etc., but due to extreme imbalance in the number of data points within these categories, this was not possible. Instead we investigated the variation of accuracy based on the amount of estimated effort for the task. The accuracy distribution for tasks that are estimated to be smaller than one day’s work turned out to be very different from the accuracy distribution of the bigger tasks. An example is that all of the outliers in the original training data for company C are tasks that are estimated to be smaller than one day’s work. This may influence the simulated effort PIs when applied to the bigger tasks’ effort estimates that are not nearly as inconsistent in terms of estimate accuracy. The distribution of small versus big tasks’ accuracy distribution is shown in Figure 19, Figure 20 and Figure 21. In this approach, tasks will be divided into two categories, one for tasks that have an estimated effort of one day or less, and one category for tasks that have an effort estimate greater than one day.

The small tasks accuracy distribution contains a lot more outliers than the big tasks accuracy distribution. Not having these outliers in the data when simulating big tasks is likely to produce effort PIs that have a lower minimum and maximum limit. Extreme accuracy values obviously have a bigger impact when paired with bigger tasks, while the impact is less significant when paired with small tasks. The separation of tasks based on size should therefore reduce the number of outliers in the basis for simulation of effort for big tasks, and emphasize the inconsistency in accuracy for small tasks, leading to more accurate and efficient effort PIs.

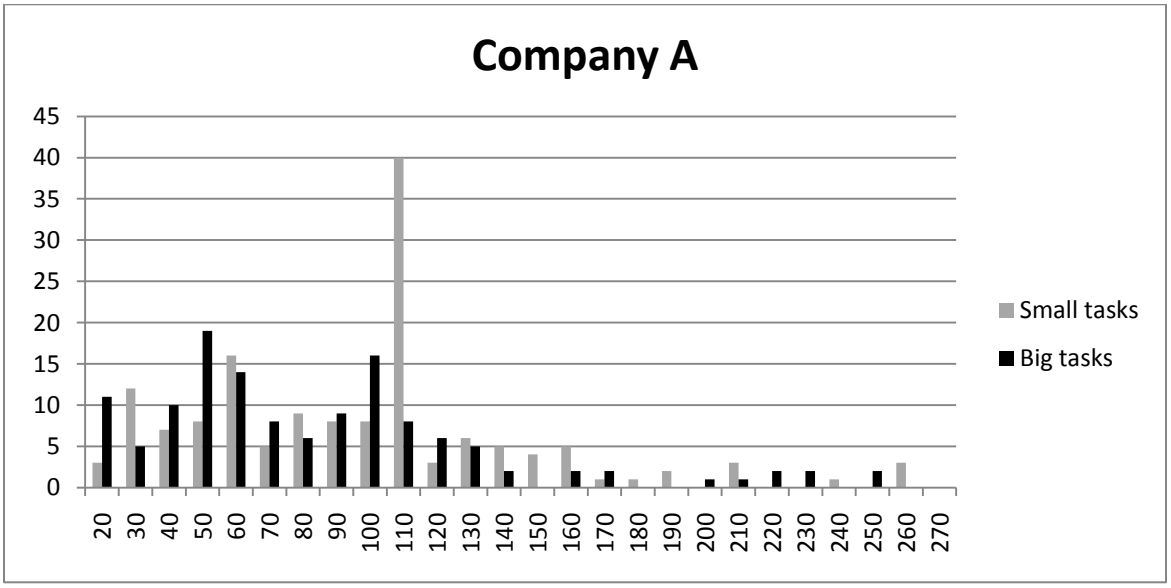

Figure 19 - Distribution of estimation accuracy

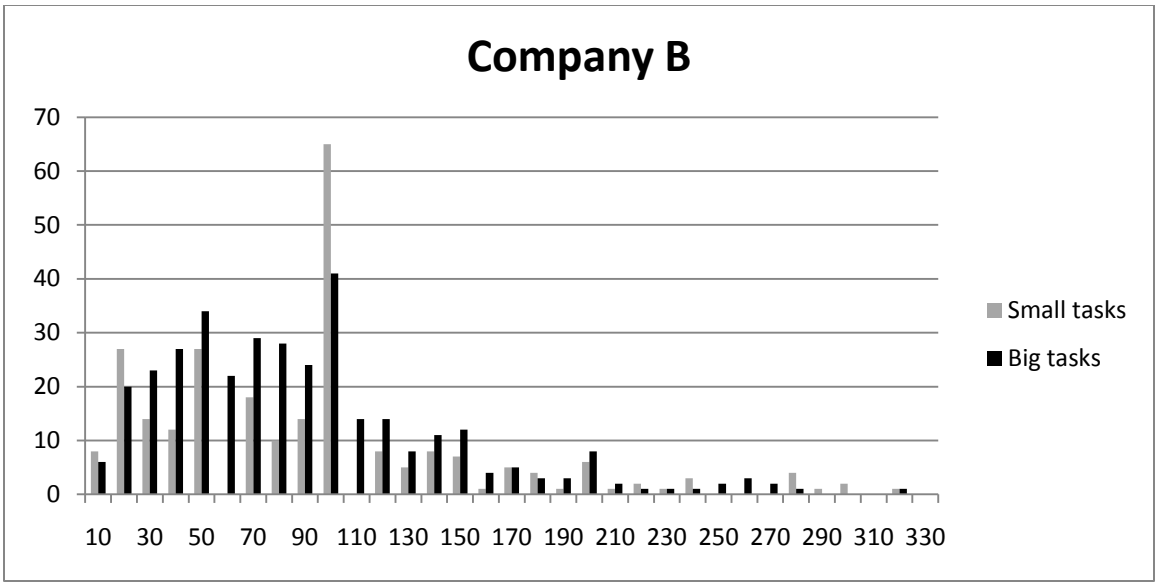

Figure 20 - Distribution of estimation accuracy

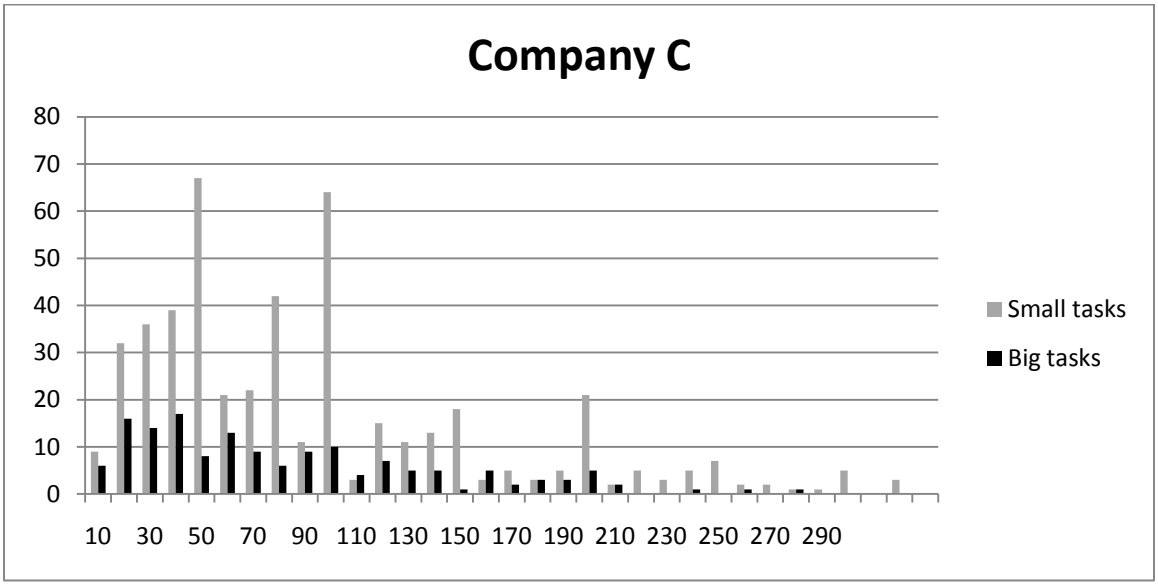

Figure 21 - Distribution of estimation accuracy

As shown in Table 8, dividing the tasks into categories based on the size of their estimated effort has had a positive effect on our results. The hit rate is very close to desired confidence level of 90%, and the hit rate for company C is as good as any other approach we have tried, with the same hit rate as the basic approach and much improved efficiency in the effort PIs. The hit rates for company A and B are as good as or better than they were for any other approach. The width of the simulated PIs also seems to be less affected by the number of tasks in each sprint. This is an indication that the accuracy distribution for the smaller tasks, with its many outliers, had an impact on the bigger more dominant tasks in the sprints where the total number of tasks was low.

Company	Sprints	Hit rate	Median PI width
A	13	92%	0.45
B	21	86%	0.38
C	14	79%	0.37

Table 8 - Results of separating small and big tasks

For company C where there was a tendency towards overestimation in the previous approaches, there is an improvement. The distribution of accuracy in the training data for big tasks looks very different from the distribution used in both the basic approach and the smoothed accuracy distribution approach. Despite the narrower effort PIs, a closer look at the results show that the actual efforts for the sprints that missed the simulated effort PIs are not as far off with this approach as the basic one, see Appendix A.

6.3.5 Combining method (2) and (3)

In this approach we will combine the two previous approaches, where the results have been encouraging.

The training data is split into two categories, one for tasks estimated to less than one day's effort, and one for tasks estimated to more than one day. We will also use the option to smoothen data which causes the tool to fit gamma distributions based on each category's empirical estimation accuracy distribution. The tool will then generate ten thousand data points for each category by sampling from the gamma distributions. These data points will be used as training data when simulating effort.

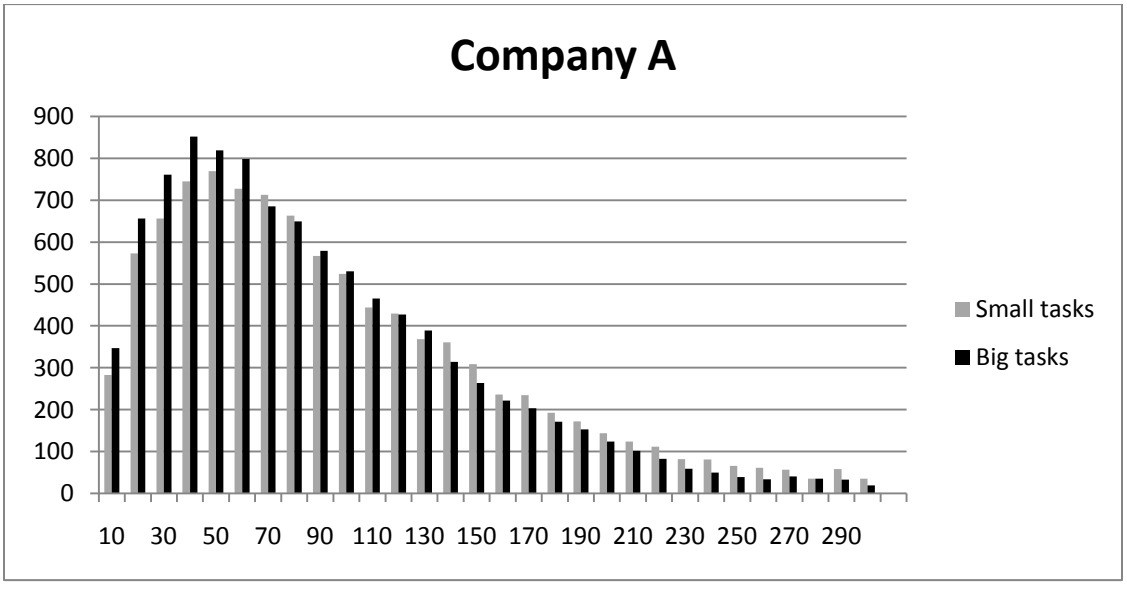

Figure 22 - Gamma distributed by size

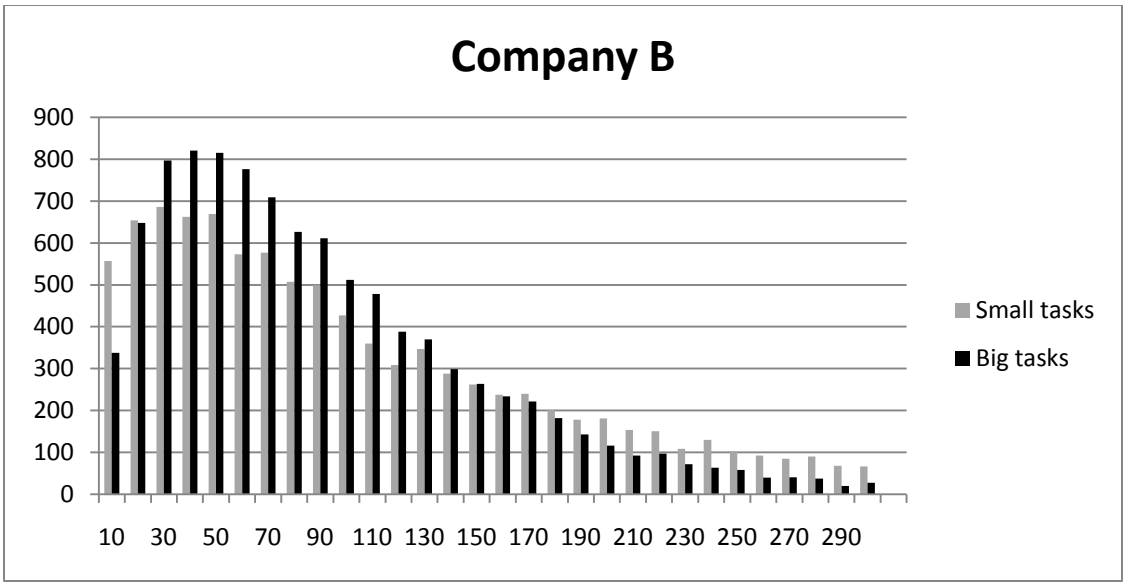

Figure 23 - Gamma distributed by size

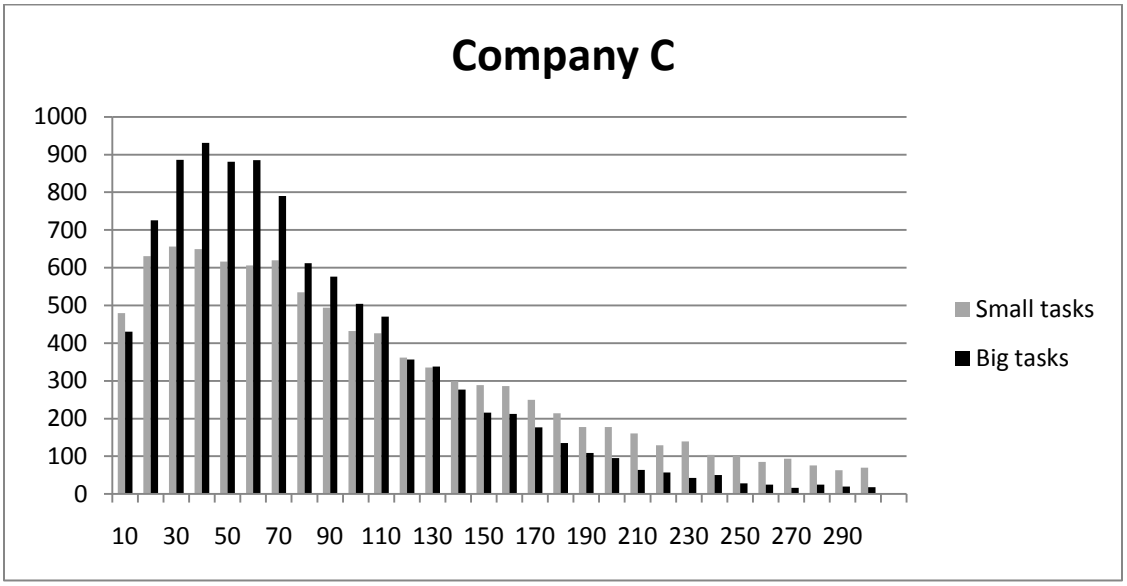

Figure 24 - Gamma distributed by size

Looking at Figure 22 we see that the gamma distributed accuracies for small and big tasks in company A are very similar. When looking at the distribution of estimation accuracy by size in Figure 19, we can see that there is a clear distinction between small and big tasks, especially the amount of tasks around one hundred percent accuracy for small tasks. It might appear that fitting the distribution of accuracy for big tasks was not successful. The distribution of accuracy for small and big tasks for company B does resemble a gamma distribution, if you make the assumption that the large amount of tasks registered at one hundred percent accuracy is actually distributed around this value, see Figure 20. Looking at Figure 23, the smoothing of this distribution also seems to have preserved the difference between estimation accuracy for small

and big tasks. The low accuracy for company C in the other variations has been related to the number of outliers and the big difference in estimation spread between small and big tasks. If we look at Figure 24, we can see that using the separation of small and big tasks in addition to the statistical smoothing has preserved the large difference in accuracy between the tasks in addition to removing the outliers. The accuracy spread of the small tasks is much higher, than the distributed accuracy for the big tasks.

Company	Sprints	Hit rate	Median PI width
A	13	69%	0.39
B	21	86%	0.36
C	14	71%	0.33

Table 9 - Results of combining method (2) and (3)

Comparing to the previously best approach – the division of tasks by size, accuracy is reduced from 12/13 (92%) to 9/13 (69%) for company A, and reduced from 11/14 (79%) to 10/14 (71%) for company C while it remains unchanged for company B (see Table 9). The reduction of accuracy for company A should not come as a surprise. The problem distinguishing between small and big tasks indicates trouble fitting the distribution to the data, and it is probably better to use the empirical data without smoothing, as it did not seem to have any obvious shortcomings or issues related to it. Company B's accuracy when using this method is unchanged at 86%, which is close to the requested confidence level. However, the median PI width has been reduced, showing a potential improvement in efficiency by eliminating outliers and using a smoothed distribution. The results show that the accuracy for company C has declined slightly, despite successful distinction between small and big tasks in addition to the removal of outliers. The median PI width has been reduced here as well.

The results show us that the simulated effort PIs are even narrower than when using statistically smoothed data or the size-category approaches alone. The accuracy has decreased slightly for two of the companies compared to the approach where the size category approach was used without statistically smoothing of the data.

7 Discussion

7.1 Interpretation of results

The results of our study shows that aggregating task level effort estimates to iteration-level effort through Monte Carlo simulation seems to be a feasible approach to reliably estimating effort PIs. Even in its most basic approach, the results of the evaluation were encouraging. Using judgment-based effort estimates combined with uncertainty assessments based on historical data looks like a promising way to go, looking both at research over the past years, current practices in the software industry, and the present study. With studies suggesting that learning from estimation errors is low and lack of feedback on estimation accuracy is common in the software industry, the method presented in this paper could be a step towards better uncertainty assessments.

The results of the evaluation suggest that there was a change in estimation accuracy over time for company C. Their estimates moved systematically towards overestimation over time. None of our approaches performed particularly well on company C's data set.

The evaluation of the baseline approach shows very good results when applied on two out of three companies' data. Accuracy is close to the requested confidence level, and the efficiency is better than it is for the basic variant of the suggested approach. However, the 43% hit rate for company C should be an indication that this simplistic approach is not ideal. If the method fails to be reliable when unforeseen events occur, it should not be used for uncertainty assessments. The basic approach had hit rates that were not quite as good for company A and B, but it performed better on company C's data. The efficiency, or PI width, of the basic approach is not quite as good as the basic approach. This is not necessarily negative, as the method should ideally increase the width to a point where the hit rate reflects the requested confidence level.

In one variant of the evaluation the gamma distribution was fitted to the data. The approach assumes that the underlying distribution of estimation error is close to gamma-distributed. The assumption was that eliminating outliers and smoothing the distribution could improve the results. The gamma distribution has one peak, which in this case equals the most likely accuracy in the simulation. The most likely accuracy values to be drawn are the ones closest to the most likely effort. This seems like a logical assumption. In some of the training data there were peaks at 50%, then at 100%, and then another peak at 200%. This seems illogical, as if there is a high chance of achieving 100% effort, and 200% effort, there should also be a high chance of achieving 130% effort or 160% effort. It was assumed this was because of slightly erroneous registration around these peak values (half effort, accurate effort, double effort). Because of this, the original training data does not look gamma distributed. The results of fitting the gamma distribution to the data were evaluated under the assumption made above. The

approach using gamma distributed data had mixed results. By using this approach, the effort PIs became narrower than in the basic approach. For company A and B, where the data resembled the gamma distribution (if the peaks are evened out a bit), the hit rate of the effort PIs did not suffer much. The hit rate for company A was slightly worse than in the basic approach, while the efficiency of the PI was improved while it remained very close to the requested confidence level for company B. For company C, the hit rate was as bad as in the baseline approach at 43%, suggesting that the approach did not have the desired effect. Looking at the data of company C, we can see that the spread is much higher than in the other data sets and that it has multiple peaks. There are also a lot of outliers in this data set. This could equal difficulties in fitting the distribution to the data. When there is as much empirical data as there was in the evaluation, the use of parametric distributions might not be needed, and when the data does not look gamma distributed it might not be a good idea.

The use of categorizing tasks to separate tasks into groups with the same estimation accuracy distribution looks promising. Unfortunately, we did not have any other candidate properties than size in our data set. However, separating small and big tasks had a significant effect, and it proved to be the best tuning approach overall. The hit rate was close to the requested 90% and the median PI widths were relatively equal to the ones produced by the statistically smoothed approach.

The large amount of small tasks compared to big ones, the differences between their accuracy distributions and the high amount of outliers in the accuracy distribution for small tasks increased the statistical error in the smoothed accuracy distribution, especially when sampling accuracy values for the big tasks. When using separate accuracy distributions for small and big tasks, we eliminate the problem of empirical accuracy of the small tasks impacting simulation of big tasks and vice versa. We get more accurate simulations of tasks in each sprint, and as a result the accuracy and efficiency of the simulated effort PIs improve. While the hit rate only improves slightly compared to the basic approach, the effort PIs are more efficient.

If properties for grouping tasks with similar estimation accuracy distributions can be identified, this tuning option seems to be effective. The challenge is to identify such analogies and maintaining the data. With the high number of tasks in our evaluation set, and the lack of precise information on each task, it was not possible to find any other task separation to use in our evaluation.

After observing the improved results in separating small and big tasks, fitting the gamma distribution to the data sets was evaluated again. The results showed that this made the produced PIs more efficient than any other approach. The hit rate suffered for two companies. In one case the approach failed to preserve the distinction between small and big tasks, the other case was company C, where the estimation accuracy changed over time.

The reduction in accuracy accompanied by the reduction of PI width might suggest that in situations where you have large amounts of empirical data, and a good separation of tasks that have different accuracy distributions, the only effect you get by sampling from a parametric distribution such as the gamma distribution is a narrower effort PI. In cases where the parametric distribution is not well fitted to the data, this has the potential of reducing the hit rate of the produced PIs.

In the description of the evaluation, using bias adjustment and data narrowing to account for process improvement was suggested. The only company that has had an apparent systematic change of estimation accuracy is company C. This could be accounted for by adjusting the historical data. However, the estimation accuracy has gotten worse. Accounting for process deterioration is not supported by the tool.

The goal of this thesis was to provide software professionals with a tool for assessing uncertainty in their effort estimates. Whether this approach is something that is going to be adopted by the software industry remains to be seen, but the simplicity and applicability of the approach makes it interesting. With continued improvement of tool support and further refinement of the approach, such as integration into project tracker frameworks, a formal uncertainty assessment approach would become readily available to the software industry.

7.2 Implications for practice

The introduction of a tool to reliably estimate effort PIs should be of assistance in project planning. We have shown that the approach is applicable to uncertainty assessments for iteration-level effort, and having a reliable effort PIs should make several software project planning aspects easier.

There are many ways effort estimation and estimation uncertainty is managed in software organizations. In many companies uncertainty is not represented in the estimates that are provided by the estimation process. If the estimation process does not clearly state the intent of the estimate, this might lead to mixed assumptions about what an estimate should represent. Some developers might use the most likely effort while others might multiply it by two or three “to be sure”. By having a proven method for iteration-level effort estimation uncertainty quantification, such as the one suggested in this paper, a clear distinction can be made between estimation and uncertainty assessment. Separating effort estimation from uncertainty assessment should make it easier to conform to a common strategy for estimation, and reduce the amount of “hidden strategies” involved in producing estimates.

In the evaluation, estimation data from a project involving three different software companies who used the SCRUM [26] framework for project management was used. SCRUM is an iterative, incremental framework where after each iteration, or sprint, you release a working increment

of the software being developed. As the results of the evaluation showed, the approach suggested in this thesis is applicable to uncertainty assessments in this kind of development environment. At the start of each sprint, development teams choose an appropriate set of tasks for the next iteration, basing the decision on the estimates and the amount of effort they are historically able to put in over the course of a sprint. Using the approach suggested in this thesis and employing relevant clustering of estimation tasks, distinctions between the historical accuracy distributions for different task types is made possible. This makes it possible to avoid taking on a lot of very risky tasks in the same sprint, which could lead to either lots of tasks remaining at the end of a sprint, or a lot of time remaining.

Sometimes in software development, there are important time restrictions on delivering requested software or functionality. A competitor might be releasing a competing system, and delaying release will significantly hurt market share. There could also be other cases, where a company needs to be able to provide system support to a product it is releasing at a specific date. The ability to reliably assess the uncertainty of estimates for such releases is very valuable. Staff can be moved from one project to another, or staff can be hired in order to minimize the chance of not meeting critical deadlines.

Another important aspect of software development management is the ability to take on profitable projects, and dismiss projects that are not. Assessing the uncertainty and related risk of potential projects will help in making decisions related to bidding. In situations where you have more than one potential project to take on, you might want to compare profitability to the associated risk of the project. One project might potentially be very profitable, but if the associated risk is high, it might be desirable to go with a safer project. The suggested approach provides effort PIs and probability assessments that are valuable in an assessment of project profitability versus associated risk.

In big organizations and companies that have their own software development departments, there are also other challenges. Identifying cost efficient projects and dismissing less cost effective ones are important, but the overall strategy of the company is likely to be the deciding factor when it comes to which software development projects to take on. However, risk assessments are still important in order to make sure the needed resources are made available when dealing with release dates that are not flexible. This is common because the software releases are often coordinated with the release of products from other parts of the organization. In many industries delaying releases has the potential of being devastating, not only because an organization's reputation could take a hit, but because time-to-market is often directly related to market shares and profitability.

In a situation where the ability to release software by a specific time is important, it is easy to blame the estimator(s) when the project fails to deliver on time. By quantifying uncertainty at

the time of estimation, you add transparency to the estimation process that protects the estimator(s) from taking all the blame.

Many of the issues mentioned in this section are directly related to budgeting. Being able to accurately assess uncertainty provides a solid basis for setting reasonable budget limits. Appropriate budgeting is important in order to implement proper project risk management. Implementing risk management for a single project is rarely a good idea, because accounting for the worst scenario would be needed, but managing uncertainty across a project portfolio is common. In order to do this you need to have a reasonable contingency buffer for your projects, and a common way of doing this is to rely on the probabilities of failure, or overruns, which is what our tool provides for the estimator.

7.3 Contributions

The research focus in this thesis is the evaluation of the proposed uncertainty assessment strategy. In this thesis we performed empirical evaluation of the approach suggested, and several variations of it. The evaluation was done by estimating effort much like it is done in incremental software development. By estimating sprints and comparing to actual results, we illustrated the feasibility of using the approach for release planning within incremental software development. Through our evaluation process we have successfully evaluated several variants of the approach originally suggested by Jørgensen et al. in [4].

A high quality tool was constructed that implements our suggested approach. The tool has been tested and validated against large, real data sets. The tool enables the proposed approach to be used in practice, and also facilitates further research, as described below.

7.4 Further research

The results of the evaluation done in this thesis look promising, but there is still need for further research. The approach needs to be tested and evaluated on data from other software organizations and projects to ensure generalization of the results. In addition, it would be very productive to test the approach and the tool by performing an experiment where the tool was used in a software organization, by the software professionals themselves. This would require the organization to collect the needed empirical estimation data.

Further research should be done on the applicability of splitting tasks into categories. Some success was observed in this paper by splitting tasks into groups based on size. In other contexts, there could be other ways of categorizing tasks. Some guidelines are needed on the procedures that practitioners should follow to identify useful categories.

Although we did not have the best results when using parametric distributions approximating the empirical ones, it should not be written off as a bad approach. There might be positive

results to gain from using the approach, especially in cases where there is data scarcity. Also, our evaluation needed to be conservative for this approach. More research needs to be conducted on when and how parametric distributions should be used.

The use of bias adjustment and data narrowing in the calculations was not evaluated in this thesis. This tuning option could be evaluated in an environment where an estimation process improvement is taking place, or has taken place. In any case, empirical estimation data would have to be available.

For tool development, it would be useful to create some kind of programmatic API, to allow other tools to pull uncertainty information from the application without any kind of heavy coupling. By creating a REST API for example, it would be possible to retrieve uncertainty information from other estimation and project planning tools.

To allow other project management applications to make use of our approach to uncertainty assessment, functionality should be extracted to a Java library with an easy to use interface. This would allow easy implementation of uncertainty assessments in existing project planning and management tools, where historical data might be readily available.

7.5 Limitations

7.5.1 Measurement accuracy

The data that was used for the evaluation of the approach and tool suggested in this paper was collected from a large Norwegian software project. The data registration and collection was not quality assured, and we can therefore not assess its quality. The large number of registered values that correspond to 50%, 100% and 200% estimation accuracy indicates that the collected data has some quality limitations.

7.5.2 Generalization

The evaluation of the approach was conducted on data from a single software project. There is no way to tell if the results will transfer successfully to other software projects. There were however three different software companies participating in the project and thus in the study. The three companies used the SCRUM approach to project management, which is a commonly used approach in the software industry.

8 Conclusion

In this thesis we evaluate an effort PI approach to uncertainty assessment for software effort estimation that is based on the previously suggested approach in [4]. The approach uses previous estimation accuracy to make uncertainty assessments based on effort estimates and probability theory. The work is motivated by the overconfidence in expert judgment

uncertainty assessments, and the lack of formal models that are able to make efficient use of uncertainty information.

A tool has been developed to support the approach, which also supports several tuning options to the approach including clustering of tasks, historical bias adjustment, distribution narrowing and the use of approximated parametric distributions. By leveraging on historical estimation accuracy data we use Monte Carlo simulation and aggregation of task-level effort to iteration-level effort to calculate effort PIs for development iterations.

The approach was evaluated against a large set of estimation data from an ongoing Norwegian software project, including three different software development companies. Results of the evaluation showed that the approach is applicable to iteration-level effort estimate uncertainty assessment even in its most basic configuration. The results for one of the three companies suggested much lower accuracy than was requested. Analysis did however show that the estimation process seemed to have deteriorated as the project progressed, with a systematic trend of overestimating tasks.

We also investigated the effect of clustering tasks based on size, taking into account the systematic difference in estimation accuracy between small and big tasks. Clustering tasks based on size had a positive effect on both the accuracy and the efficiency of the calculated effort PIs. As this was the approach that had the best results in the evaluation, further research on identifying efficient clustering strategies is suggested.

The use of the gamma distribution instead of the empirical distribution was also investigated. Te results showed a negative effect on the accuracy of the PIs where the underlying distribution of accuracy did not sufficiently resemble the gamma distribution, but increased the efficiency of the produced effort PIs overall. The use of parametric distributions cannot be excluded as a viable approach, but further research is needed on when and how to use them.

References

1. Jørgensen, M., *Estimation of software development work effort: Evidence on expert judgment and formal models.* International Journal of Forecasting, 2007. **23**(3): p. 449-462.
2. Molokken, K. and M. Jorgensen. *A review of software surveys on software effort estimation*. Proceedings of *International Symposium on Empirical Software Engineering*. 2003. Rome, Italy: Simula Res. Lab. Lysaker Norway: p. 223-230.
3. Jørgensen, M., K.H. Teigen, and K. Moløkken, *Better sure than safe? Over-confidence in judgement based software development effort prediction intervals.* Journal of Systems and Software, 2004. **70**(1-2): p. 79-93.
4. Joergensen, M. and D.I.K. Sjoeberg, *An effort prediction interval approach based on the empirical distribution of previous estimation accuracy.* Information and Software Technology, 2003. **45**(3): p. 123-136.
5. Jorgensen, M., *Practical guidelines for expert-judgment-based software effort estimation.* Software, IEEE, 2005. **22**(3): p. 57-63.
6. Jorgensen, M. and M. Shepperd, *A systematic review of software development cost estimation studies.* Software Engineering, IEEE Transactions on, 2007. **33**(1): p. 33-53.
7. Kitchenham, B. and S. Linkman, *Estimates, uncertainty, and risk.* Software, IEEE, 1997. **14**(3): p. 69-74.
8. Jørgensen, M., *Regression models of software development effort estimation accuracy and bias.* Empirical software engineering, 2004. **9**(4): p. 297-314.
9. Jørgensen, M., *Forecasting of software development work effort: Evidence on expert judgement and formal models.* International Journal of Forecasting, 2007. **23**(3): p. 449-462.
10. Jørgensen, M. and T.M. Gruschke, *The impact of lessons-learned sessions on effort estimation and uncertainty assessments.* IEEE Transactions on Software Engineering, 2009: p. 368-383.
11. Jorgensen, M., *Realism in assessment of effort estimation uncertainty: It matters how you ask.* Software Engineering, IEEE Transactions on, 2004. **30**(4): p. 209-217.
12. Little, T., *Schedule estimation and uncertainty surrounding the cone of uncertainty.* Software, IEEE, 2006. **23**(3): p. 48-54.
13. Madachy, R.J., *Heuristic risk assessment using cost factors.* Software, IEEE, 1997. **14**(3): p. 51-59.
14. Kansala, K., *Integrating risk assessment with cost estimation.* Software, IEEE, 1997. **14**(3): p. 61-67.
15. Boehm, B.W., *Software engineering economics.* Software Engineering, IEEE Transactions on, 1984(1): p. 4-21.
16. Boehm, B., et al., *Cost models for future software life cycle processes: COCOMO 2.0.* Annals of software engineering, 1995. **1**(1): p. 57-94.
17. Idri, A., T.M. Khoshgoftaar, and A. Abran, *Investigating soft computing in case-based reasoning for software cost estimation.* Engineering Intelligent Systems for Electrical Engineering and Communications, 2002. **10**(3): p. 147-158.
18. Idri, A., A. Abran, and T. Khoshgoftaar. *Fuzzy analogy: A new approach for software cost estimation*. in *11th International Workshop on Software Measurements*. 2001. Citeseer.
19. Zadeh, L.A., *Fuzzy sets as a basis for a theory of possibility.* Fuzzy sets and systems, 1978. **1**(1): p. 3-28.
20. Angelis, L. and I. Stamelos, *A simulation tool for efficient analogy based cost estimation.* Empirical software engineering, 2000. **5**(1): p. 35-68.
21. Kitchenham, B., et al., *A case study of maintenance estimation accuracy.* Journal of Systems and Software, 2002. **64**(1): p. 57-77.

22. Magne, J., *Evidence-based guidelines for assessment of software development cost uncertainty.* IEEE Transactions on Software Engineering, 2005: p. 942-954.
23. Garrett, J.J., *Ajax: A new approach to web applications.* February, 2005. **18**: p. 2005.
24. Reenskaug, T., *Models-views-controllers.* Technical note, Xerox PARC, 1979.
25. Fielding, R., *Representational state transfer (REST).* Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 2000: p. 120.
26. Schwaber, K. and I. Books24x7, *Agile project management with Scrum*. Vol. 7. 2004: Microsoft Press Redmond (Washington).

Appendix A

The following tables display detailed results of the evaluation performed in the thesis.

Results for the basic approach

Company A – Basic					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI-width**
1	43	2156400	2073960	[1543788, 2658270]	0.52
2	53	2180700	2309400	[1604903, 2746526]	0.52
3	39	1126800	1011600	[832927, 1499379]	0.59
4	42	1513800	1994400	[1134307, 2050077]	0.60
5	45	1854900	2049360	[1468990, 2641441]	0.63
6	43	1472400	1308900	[1157137, 2127709]	0.66
7	72	2645100	2703900	[2134885, 3535338]	0.53
8	68	3193200	3244260	[2628265, 4284379]	0.52
9	61	2179800	2547900	[1768486, 2947476]	0.54
10	64	2896200	3187860	[2463776, 4055716]	0.55
11	77	2563200	2466060	[2257094, 3458121]	0.47
12	89	3081600	3088860	[2737148, 4117769]	0.45
13	45	2541600	3029400	[2130957, 3498270]	0.54

Company B – Basic					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI-width**
1	113	5180400	6039060	[4205981, 6161985]	0.38
2	111	4434300	4708260	[3687512, 5640814]	0.44
3	75	3981600	3560700	[3293073, 5232706]	0.49
4	37	2620800	2529600	[1907512, 3752452]	0.70
5	36	2689200	2203680	[1937016, 3918510]	0.74
6	64	3708000	3505740	[2976900, 4915564]	0.52
7	30	2232000	2280600	[1629360, 3305500]	0.75
8	42	2728800	2038380	[2088538, 3731340]	0.60
9	37	2059200	1898400	[1557696, 2868788]	0.64
10	29	1753200	1677600	[1277773, 2458350]	0.67
11	38	2624400	2154480	[1945566, 3641352]	0.65
12	58	1627200	1573440	[1307607, 2229334]	0.57
13	51	1854000	2058720	[1452963, 2517988]	0.57
14	28	892800	822840	[660789, 1310082]	0.73
15	88	3769200	3099900	[3140758, 4729769]	0.42
16	103	4766400	3950520	[3946262, 5860905]	0.40
17	82	3600000	3183540	[2934177, 4460043]	0.42
18	113	5882400	5190480	[4748608, 7338194]	0.44
19	144	7522200	5574240	[6346743, 8920360]	0.34
20	103	5662800	4198440	[4621467, 6721880]	0.37
21	93	4896000	4386120	[3828266, 6084672]	0.46

Company C – Basic					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	104	2527200	2141280	[2111453, 3308656]	0.47
2	143	3313800	2545560	[2852910, 4129227]	0.39
3	109	2858400	2475120	[2314091, 3717064]	0.49
4	74	2318400	1489020	[1690172, 3103275]	0.61
5	11	273600	207900	[143918, 470225]	1.19
6	106	3535200	2964780	[2698571, 4736061]	0.57
7	75	1980000	1408500	[1489888, 2456357]	0.49
8	90	3088800	2325120	[2314654, 3816612]	0.49
9	57	1521480	1486800	[1056826, 1997633]	0.62
10	107	2440800	2172600	[1857235, 2993917]	0.47
11	135	3790800	3060060	[3005333, 4448992]	0.38
12	119	3664800	3042540	[2844022, 4330260]	0.41
13	53	1454400	1162320	[1055006, 1873266]	0.56
14	60	1890000	1542840	[1309099, 2435499]	0.60

Results for the smoothed training data approach

Company A – Smoothed training data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	43	2156400	2073960	[1574460, 2534436]	0,45
2	53	2180700	2309400	[1638981, 2514069]	0,40
3	39	1126800	1011600	[811368, 1328400]	0,46
4	42	1513800	1994400	[1101150, 1795968]	0,46
5	45	1854900	2049360	[1376991, 2102679]	0,39
6	43	1472400	1308900	[1095480, 1686275]	0,40
7	72	2645100	2703900	[2000042, 3053016]	0,40
8	68	3193200	3244260	[2429064, 3589632]	0,36
9	61	2179800	2547900	[1636002, 2520378]	0,41
10	64	2896200	3187860	[2181150, 3307229]	0,39
11	77	2563200	2466060	[2025540, 2836115]	0,32
12	89	3081600	3088860	[2414088, 3391956]	0,32
13	45	2541600	3029400	[1931760, 2910312]	0,39

Company B – Smoothed training data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	113	5180400	6039060	[4353372, 5848128]	0,29
2	111	4434300	4708260	[3683493, 5197932]	0,34
3	75	3981600	3560700	[3195288, 4654656]	0,37
4	37	2620800	2529600	[1863180, 3401856]	0,59
5	36	2689200	2203680	[1856339, 3733164]	0,70
6	64	3708000	3505740	[2928168, 4395024]	0,40
7	30	2232000	2280600	[1580472, 2849040]	0,57
8	42	2728800	2038380	[1979856, 3485664]	0,55
9	37	2059200	1898400	[1519488, 2580120]	0,52
10	29	1753200	1677600	[1251324, 2240748]	0,56
11	38	2624400	2154480	[1925100, 3377988]	0,55
12	58	1627200	1573440	[1273410, 1999547]	0,45
13	51	1854000	2058720	[1438524, 2296512]	0,46
14	28	892800	822840	[653976, 1147104]	0,55
15	88	3769200	3099900	[3056832, 4435164]	0,37
16	103	4766400	3950520	[3896928, 5677668]	0,37
17	82	3600000	3183540	[2909610, 4214700]	0,36
18	113	5882400	5190480	[4757193, 7049619]	0,39
19	144	7522200	5574240	[6414048, 8453970]	0,27
20	103	5662800	4198440	[4760406, 6454836]	0,30
21	93	4896000	4386120	[3923946, 5887152]	0,40

Company C – Smoothed training data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	104	2527200	2141280	[2175768, 3089160]	0,36
2	143	3313800	2545560	[2908116, 3911256]	0,30
3	109	2858400	2475120	[2456676, 3419316]	0,34
4	74	2318400	1489020	[1851984, 3047616]	0,52
5	11	273600	207900	[156312, 419004]	0,96
6	106	3535200	2964780	[2945664, 4532148]	0,45
7	75	1980000	1408500	[1641384, 2513412]	0,44
8	90	3088800	2325120	[2616264, 3832668]	0,39
9	57	1521480	1486800	[1181242, 1993783]	0,53
10	107	2440800	2172600	[2053835, 2945160]	0,37
11	135	3790800	3060060	[3343626, 4504986]	0,31
12	119	3664800	3042540	[3162168, 4332924]	0,32
13	53	1454400	1162320	[1176084, 1824660]	0,45
14	60	1890000	1542840	[1511352, 2446308]	0,49

Results for the approach with tasks categorized based on size

Company A – Size categorized data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	43	2156400	2073960	[1511116, 2569955]	0.49
2	53	2180700	2309400	[1582958, 2561115]	0.45
3	39	1126800	1011600	[794960, 1444764]	0.58
4	42	1513800	1994400	[1094475, 1936305]	0.56
5	45	1854900	2049360	[1402610, 2306306]	0.49
6	43	1472400	1308900	[1159170, 1941116]	0.53
7	72	2645100	2703900	[2098499, 3229856]	0.43
8	68	3193200	3244260	[2539584, 3830543]	0.40
9	61	2179800	2547900	[1736400, 2710664]	0.45
10	64	2896200	3187860	[2348647, 3550162]	0.41
11	77	2563200	2466060	[2209977, 3155391]	0.37
12	89	3081600	3088860	[2688827, 3818528]	0.37
13	45	2541600	3029400	[2005434, 3086205]	0.43

Company B – Size categorized data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	113	5180400	6039060	[4172255, 5570495]	0.27
2	111	4434300	4708260	[3690582, 5196838]	0.34
3	75	3981600	3560700	[3283956, 4690502]	0.35
4	37	2620800	2529600	[1867851, 3356077]	0.57
5	36	2689200	2203680	[1855599, 3594827]	0.65
6	64	3708000	3505740	[2888790, 4311748]	0.38
7	30	2232000	2280600	[1584825, 2734145]	0.51
8	42	2728800	2038380	[2030878, 3367404]	0.49
9	37	2059200	1898400	[1522755, 2463720]	0.46
10	29	1753200	1677600	[1251076, 2223098]	0.55
11	38	2624400	2154480	[1927280, 3182145]	0.48
12	58	1627200	1573440	[1280276, 2008747]	0.45
13	51	1854000	2058720	[1425970, 2348760]	0.50
14	28	892800	822840	[676670, 1179900]	0.56
15	88	3769200	3099900	[3071526, 4399710]	0.35
16	103	4766400	3950520	[3869770, 5414419]	0.32
17	82	3600000	3183540	[2840249, 4101067]	0.35
18	113	5882400	5190480	[4763587, 6837301]	0.35
19	144	7522200	5574240	[6118624, 8039233]	0.26
20	103	5662800	4198440	[4447555, 6063049]	0.29
21	93	4896000	4386120	[3727434, 5406256]	0.34

Company C – Size-categorized data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	104	2527200	2141280	[2024332, 2954375]	0.37
2	143	3313800	2545560	[2778266, 3716556]	0.28
3	109	2858400	2475120	[2190336, 3081900]	0.31
4	74	2318400	1489020	[1560150, 2542120]	0.42
5	11	273600	207900	[146820, 405309]	0.58
6	106	3535200	2964780	[2485580, 3769030]	0.36
7	75	1980000	1408500	[1421363, 2206208]	0.40
8	90	3088800	2325120	[2199328, 3170070]	0.31
9	57	1521480	1486800	[999755, 1621443]	0.41
10	107	2440800	2172600	[1816531, 2626104]	0.33
11	135	3790800	3060060	[2857343, 3876171]	0.27
12	119	3664800	3042540	[2689637, 3678711]	0.27
13	53	1454400	1162320	[998964, 1574742]	0.40
14	60	1890000	1542840	[1210700, 2001001]	0.42

Result of combining the two previous approaches

Company A – Size categorized and smoothed data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	43	2156400	2073960	[1549620, 2444508]	0,41
2	53	2180700	2309400	[1505340, 2441313]	0,43
3	39	1126800	1011600	[786348, 1319472]	0,47
4	42	1513800	1994400	[1069632, 1689588]	0,41
5	45	1854900	2049360	[1343808, 2053791]	0,38
6	43	1472400	1308900	[1084896, 1678356]	0,40
7	72	2645100	2703900	[1955133, 2907009]	0,36
8	68	3193200	3244260	[2358720, 3478680]	0,35
9	61	2179800	2547900	[1577592, 2475288]	0,41
10	64	2896200	3187860	[2146968, 3187584]	0,36
11	77	2563200	2466060	[1951200, 2778552]	0,32
12	89	3081600	3088860	[2363994, 3312936]	0,31
13	45	2541600	3029400	[1808424, 2795472]	0,39

Company B – Size categorized and smoothed data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	113	5180400	6039060	[4249656, 5557752]	0,25
2	111	4434300	4708260	[3658887, 4995198]	0,30
3	75	3981600	3560700	[3099888, 4385160]	0,32
4	37	2620800	2529600	[1770480, 3091788]	0,50
5	36	2689200	2203680	[1808604, 3510072]	0,63
6	64	3708000	3505740	[2842092, 4132800]	0,35
7	30	2232000	2280600	[1534104, 2666520]	0,51
8	42	2728800	2038380	[1909512, 3232512]	0,48
9	37	2059200	1898400	[1457496, 2413224]	0,46
10	29	1753200	1677600	[1229004, 2107584]	0,50
11	38	2624400	2154480	[1851912, 3122316]	0,48
12	58	1627200	1573440	[1275174, 1992546]	0,44
13	51	1854000	2058720	[1463796, 2292588]	0,45
14	28	892800	822840	[652536, 1153800]	0,56
15	88	3769200	3099900	[3042684, 4253616]	0,32
16	103	4766400	3950520	[3753396, 5225652]	0,31
17	82	3600000	3183540	[2845962, 4059090]	0,34
18	113	5882400	5190480	[4605192, 6704154]	0,36
19	144	7522200	5574240	[6127902, 7963901]	0,24
20	103	5662800	4198440	[4526694, 6021756]	0,26
21	93	4896000	4386120	[3805614, 5449716]	0,34

Company C – Size-categorized and smoothed data					
Sprint	**Tasks**	**Estimate**	**Actual**	**Prediction interval**	**PI width**
1	104	2527200	2141280	[2092860, 2824020]	0,29
2	143	3313800	2545560	[2851200, 3661578]	0,24
3	109	2858400	2475120	[2264328, 3025260]	0,27
4	74	2318400	1489020	[1657620, 2518992]	0,37
5	11	273600	207900	[166284, 398916]	0,85
6	106	3535200	2964780	[2635920, 3865139]	0,35
7	75	1980000	1408500	[1550700, 2212884]	0,33
8	90	3088800	2325120	[2363832, 3239352]	0,28
9	57	1521480	1486800	[1111024, 1677363]	0,37
10	107	2440800	2172600	[1957068, 2705508]	0,31
11	135	3790800	3060060	[3040866, 3924071]	0,23
12	119	3664800	3042540	[2880036, 3830508]	0,26
13	53	1454400	1162320	[1094976, 1602180]	0,35
14	60	1890000	1542840	[1348128, 2025612]	0,36