Improved Decision-Making for Software Managers Using Bayesian Networks

Łukasz Radliński[1, 2], Norman Fenton[1], Martin Neil[1]

[1]Department of Computer Science, Queen Mary, University of London,
Mile End Road, London, England E1 4NS
[2]Institute of Information Technology in Management, University of Szczecin,
ul. Mickiewicza 64, 71-101 Szczecin, Poland
{lukrad, norman, martin}@dcs.qmul.ac.uk

Abstract. Although there have been many models for predicting resources in software development they provide little in the way of decision-support for software managers. It has been argued that models based on Bayesian Nets give more benefits, in terms of decision-support, than traditional models. The model described here is an improvement on one such widely used model that evolved from the EC project MODIST. Unlike the MODIST model the new model gives users the ability to adjust the model either by their subjective beliefs or by feeding the model with empirical data from past projects. Also, the new model gives freedom of choice of units of measurement for expressing model variables. Consequently, the new model is significantly more flexible.

Keywords. Software development, predicting resources, trade-off analysis, Bayesian Nets

1 Introduction

In the software engineering domain much effort has been spent on building models for two areas:

1. Predicting resources necessary to accomplish a software project.
2. Predicting quality of a developed software product.

Indeed, it has been argued that almost all research under the classification of 'software metrics' is traceable to these two objectives [7]. Yet, few models have addressed the ultimate objective of software metrics, which is to provide software managers support for improved decision-making and risk assessment based on quantification. Such an objective requires a combination of both the resource and quality perspective of a project. One approach that has shown considerable promise in addressing this requirement is Bayesian Nets [10]. A Bayesian Net (BN) is an acyclic graph in which the nodes indicate variables expressed as probability distributions. Nodes are connected according to the causal/relevance relationships between them. Thus, they enable us to analyze the impact of one variable on others in many useful combinations.

A widely used BN model, called the project-level model [7] that was developed as a part of EC Project MODIST [12], attempted to address the requirements for decision making and risk assessment in software projects, while taking account of the best empirical results that had informed earlier resource prediction and defect prediction models. In particular, the model attempted to reflect the trade-offs that we can normally observe in software projects between:

- the size of delivered software,
- the quality of delivered software,
- the effort required for developing the software (in terms of both project duration and number of people).

While the model has been widely used and quite successful, it is limited in the sense that the prior probability distributions in the model are heavily dependent on previous empirical data that may not always be relevant. Hence this paper focuses on a new model that adopts the basic philosophy of the MODIST model, but which can be much more easily adjusted for company-specific needs.

In Section 2 we briefly present the original MODIST project-level model and we point out its limitations. We present our revised model in Section 3 that addresses the key weaknesses of the MODIST model. In Section 4 we demonstrate how software managers can use the revised model for better decision support and risk assessment.

2 Existing Bayesian Nets for Software Managers

There have been many different software engineering models incorporating resource prediction [2, 4, 6, 11, 17]. Some of them were also Bayesian Nets [3, 8, 13, 16, 19, 20].

We decided to base our improved model on the MODIST project-level model because it explicitly contains the trade-off component, has been validated in several trials [7], provides the greatest potential for decision support and is the easiest for adoption to our purposes. Fig. 1 illustrates the structure of the main part of this model. Based on project duration (expressed in person-months) and average number of people full time, the model calculates effort, which is adjusted by the Brooks factor [5]. Then effort is adjusted by process and people quality. Functionality delivered (in function points) is calculated based on the adjusted effort. Knowing the functionality and the real effort for the project the model calculates the software quality, which is also adjusted by process and people quality. Because propagation in BNs enables both forward and backward inference, it is possible to enter ‘observations’ into any node of the model and let the model produce revised probability distributions for all the (as yet) unknown nodes. For example, if there is a known quality requirement then the model will produce predicted distributions for resources and functionality. If, in addition, there are certain fixed resources then the model will again produce a revised distribution for functionality.

The whole model takes into account other factors such as: process, people and requirements specification quality as well as distributed communications and

management factors. It is too complex to show them in detail on a single diagram. More on its structure and usage can be found in [7].

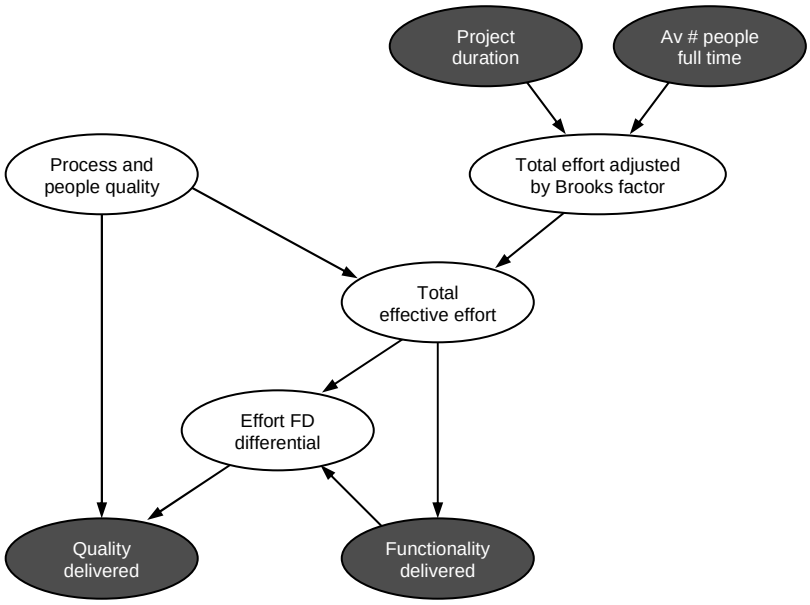

Fig. 1. Project resource model (simplified), adapted from [7]

This project level model has been validated by various partners in the MODIST project and has also been incorporated into the AgenaRisk tool [1] that has several thousand users worldwide.

There are two main weaknesses of the MODIST project-level model:

1. The model uses fixed units of measurement for some factors. Functionality is expressed in function points and, partially, KLOC (thousands lines of code). If users decide to use KLOC, they need to provide the programming language name and the model still calculates the value expressed in function points for the further calculations. Effort is measured in person-months. Companies may wish to use other units of measurement (in particular many organizations involved in the MODIST trials were uncomfortable using function points). In such cases they have to calculate their values/estimations outside the model to be expressed in the units acceptable by the model.
2. Although the model contains several variables describing the (current) process and product of software development, it lacks of ease of incorporating new empirical data by the end users. Many of the prior distributions at the heart of the 'trade-off' part of the model are based on empirical data that may not be relevant. As is typical in any Bayesian model, while such priors are extremely useful for organizations that have no previous relevant data of their own, they can significantly bias the predictions even once project-specific variables are observed. Since software companies increasingly gather their own data about past projects, it is important to allow the model to be adjusted to easily to reflect such data. For

example, among the easiest metrics for calculation from such databases are productivity and error rates for the past projects. Unfortunately, it is not possible to "feed" the model with such data.

3 Improved Bayesian Net for Software Managers

By considering the weaknesses of the existing model we have developed an improved BN model that provides support for:

1. Different units of measurement for model variables, possibly even to the extent, where users can use any unit of measurement that they wish to.
2. Easy incorporation of new (more relevant) empirical data into the model.

We have retained the crucial trade-off component between various software development factors, but have simplified it by including only the most important variables which are:

- easy to understand and interpret by users,
- easy to estimate based on the past data.

Fig. 2 illustrates the schematic view of the improved Bayesian Net for predicting resources in software development. Because it explicitly captures productivity we called this new model the "productivity model". All ellipses on this figure reflect nodes in the net, rectangles with light-grey background reflect model constants and rectangles with gradient background reflect subnets containing more detailed nodes.

The model consists of the following parts:

1. **Factors influencing prior rates** (Fig. 2 – gradient-filled rectangle).
   This subnet contains nodes which are general factors influencing prior error and productivity rates. This subnet is used only if the end user does not enter observations for the prior error and productivity rates. In such cases these rates are estimated by the model based on the values of the nodes in this subnet, e.g. organization or application type.
2. **Prior error and productivity rates** (Fig. 2 – grey ellipses).
   These rates are the values for the past projects. The user enters the values as calculated mean values from the past data. If they are unable to calculate them the model will estimate them based on the factors influencing them in the subnet described above.
3. **Constants describing process and project attributes which adjust prior error and productivity rates** (Fig. 2 – light-grey-filled rectangles).
   In each case the idea is to capture any key differences between the current project and the typical past projects for which we entered the prior error and productivity rates. This difference (which is expressed simply as a percentage) can be estimated using complexity metrics or expert judgement. The constants are:

- Percentage difference of software complexity.

- Percentage difference of software project scale (by which we mean scale factors affecting infrastructure rather than pure development).
- Percentage difference of software novelty (by which we mean what part of the project will be built from scratch as opposed to reuse of existing documentation, design, code, etc.).

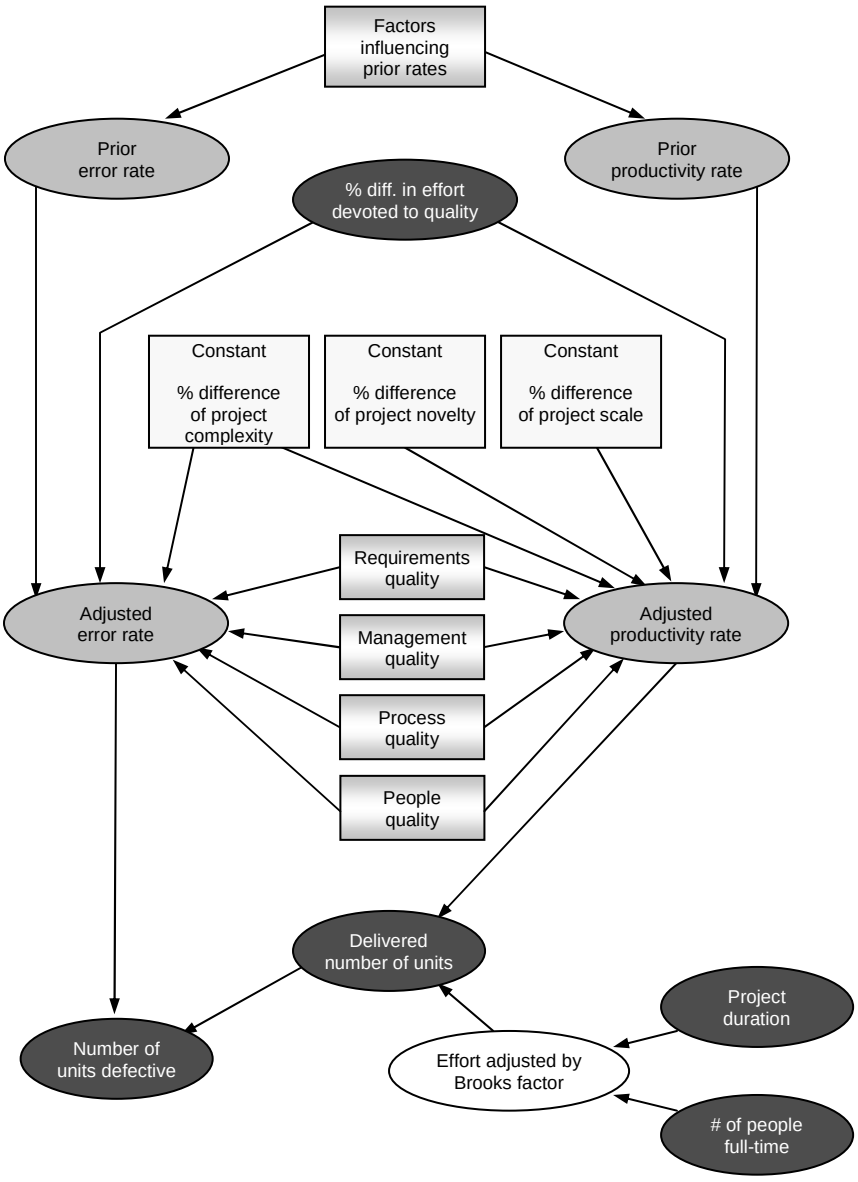

Fig. 2. Schematic view of productivity model

4. **Process and people factors which adjust error and productivity rates** (Fig. 2 – gradient-filled rectangles).

These factors are incorporated in the model as the following subnets: requirements quality, management quality, process quality and people quality. Formulating them as subnets containing more detailed nodes enables end users to either use those detailed nodes (e.g. staff motivation and/or staff experience) for expressing the quality of people or to directly use an aggregated people quality node.

These factors are expressed on a ranked scale (from "very low" to "very high") as opposed to the previously discussed constants, which are expressed on a continuous scale (real numbers). Their values are not absolute. They reflect relative values compared to the mean values for the past projects. If the value is "normal" then it means it is the same as it was on average for the past projects. In this case it would not affect either error or productivity rate. If it is "high" it means that this quality is higher than it used to be in the past. It does not mean that it is absolutely "high". Assuming that in the past the quality was always "very high" (the highest value possible in this model) and now it is set to "high", it means that currently it is even higher. Although the common meaning would suggest that "high" is lower than "very high". In this model the values higher then "normal": "high" and "very high" increase productivity rate and decrease the error rate. The values below "normal": "low" and "very low" decrease productivity rate and increase error rate.

5. **Adjusted error and productivity rates** (Fig. 2 – grey ellipses).

These two nodes reflect error and productivity rates which have been adjusted by all constants and factors. Therefore they are the estimated rates for the current project They influence the most important part of the model: the trade-off component.

In the model these two nodes are not calculated depending on their parent nodes directly. This would lead to very long calculation times because each of them has several parent nodes [18]. In fact there are hidden nodes that represent these rates adjusted sequentially by constants and factors. We do not show them on the figure illustrating the model to keep it simple and clear.

6. **Trade-off component between the quality, functionality and effort** (Fig. 2 – dark grey ellipses).

This is the main part of the model. Knowing the productivity rate and effort the model calculates the functionality – how much software can be done. Knowing the functionality and error rate the model calculates how many software units we should expect to be defective (the quality).

Effort in this model is expressed as a combination of project duration and number of people working full-time at the project. This effort is adjusted by a Brooks factor [5], like it was in the project-level MODIST model [7, 12]. This adjustment means that, for example, the total productive effort of 2 people working for 10 months is not the same as 20 people working for one month, even though the total effort in both cases is 20 person-months. We introduced this adjustment.

The node "percentage difference in effort devoted to quality" also takes part in the analysis of trade-offs. Normally part of total development effort is devoted to improve the quality of the software rather than extend its functionality. This node describes how much the effort spent on improving software quality differs in the current project compared to this effort in the past projects (for which the prior error and productivity

rates have been estimated). The higher positive difference we have, the lower error rate (better quality) and the lower productivity rate (less functionality) we should expect. This is because, given the value of effort, the more of it we spend on increasing quality the less we can spend on extending the functionality of software.

One other key improvement in the revised model compared to the MODIST model is that we have applied the dynamic discretisation algorithm [14, 15] by marking making all numeric nodes 'simulation' nodes. This means that we did not need to define the fixed node states when the model was created. This resulted in more precise predictions. We have previously tested this algorithm for the defect prediction model from the MODIST project [9]. The model has been implemented using the AgenaRisk software [1].

4 Decision Support in Improved Bayesian Net

Using the productivity model we can perform estimations for the size and quality of delivered software and the effort required for developing the software. But it is not only that. The key feature of the model is the ability to perform a trade-off analysis between these variables: how the change in one of them affects the remaining ones.

Because our model is a Bayesian Net when users wish to estimate the predictive variable they do not need to provide observations for all of the predictor variables. That is because predictor variables always have the probability distributions assigned (priors) even if they are not passed directly by users. This is a useful feature because usually it is not possible or is too costly to estimate the values for all predictor variables during a software project.

From such a model we expect to get answers to the following types of questions:

- Given the certain prior productivity and error rates, total effort for the project and leaving default values for the remaining variables (which means that both project and process factors and constants are the same as for the past projects) how much functionality and of which quality we can produce?
- How good do process and people need to be if actually we need better quality software than the model originally suggests?
- How much more effort do we have to put to deliver better software?
- How much effort do we need to deliver software of certain functionality and quality?
- How does the change in the process and people quality affect the functionality delivered and the quality of software.
- What software functionality and quality should we expect if actually our project is more complex than the previous ones?
- What impact on functionality and quality will have a proportional change of effort devoted to quality?

4.1 Predicting Functionality and Quality From Resources

In this first example we provide information about the prior productivity rate (20 function points per man-month) and prior error rate (0.01). Expressing productivity rate in function points per person-month means that the "functionality delivered" will be expressed in function points, "project duration" in months and effort in person-months. We will keep such assumptions about both the values of these rates and units of measurement for all of the examples. But it is important to note that the choice of units is completely flexible.

We declare that the effort we can spend on this project is: project duration (12 months), and number of people (20). We also assume that, compared to the previous projects:

- this project is of similar complexity, novelty and scale, as are the staff and processes
- we devote the same proportion of effort on quality.

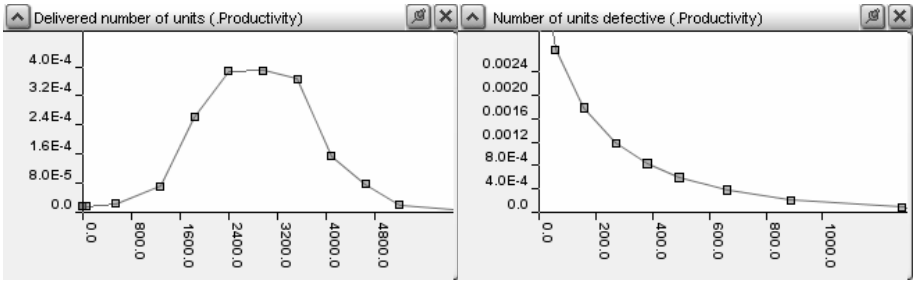

Fig. 3. Predicted functionality and quality of delivered software

Given these constraints the model predicts that we should expect to develop software of around 2900 function points and containing around 215 defective units (function points). Fig. 3 illustrates distributions for these predictive variables.

4.2 Delivering Better Software

Let us assume that we are not satisfied with the predicted quality. We enter a lower value (100) for the node "number of units defective" than the model originally predicted. We can reach the target by either:

- Improving the process and people quality while keeping the initial effort unchanged – Fig. 4 a),
- Spending more effort assuming that we will not process and people quality – Fig. 4 b).

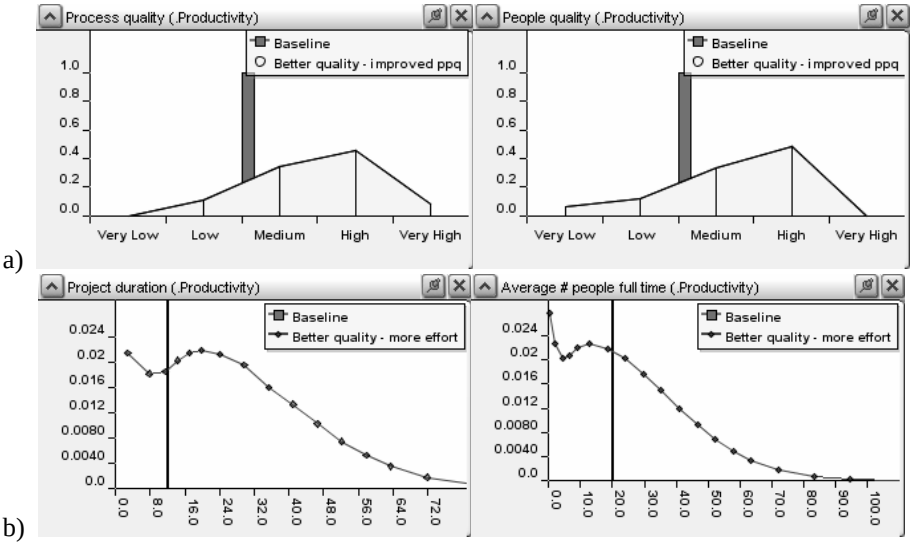

Fig. 4. Delivering better software

4.3 Predicting Effort

In this scenario our aim is to predict effort required to develop software of given functionality (3500 function points) and quality (100 defective units). Like in the first example, we are assuming that the same level of project complexity, novelty, scale proportion of effort devoted to quality, process and people quality.

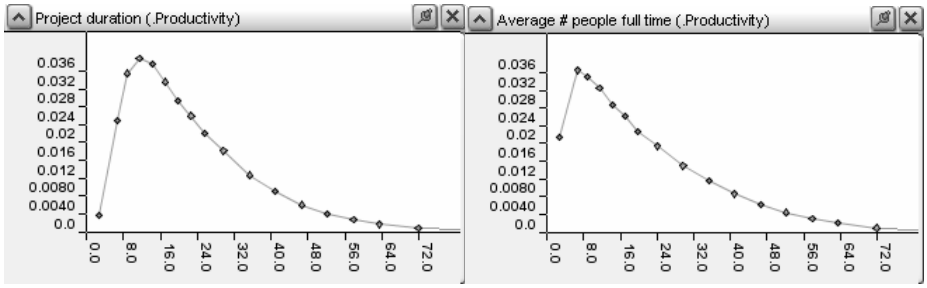

Fig. 5. Predicted effort for developing a system

From such constraints the model predicts the total effort as well as separately project duration and number of people. Fig. 5 illustrates that we need around 18 people for 20 months.

4.4 Predicting Functionality and Quality Change Influenced by Change in Process and People Quality

Now we are assuming the same project constraints as in the first scenario. However, now we are sure to reach much higher process quality and we believe that we have much better people than in the past projects. Our aim is again to predict how much functionality we can develop and what will be its quality.

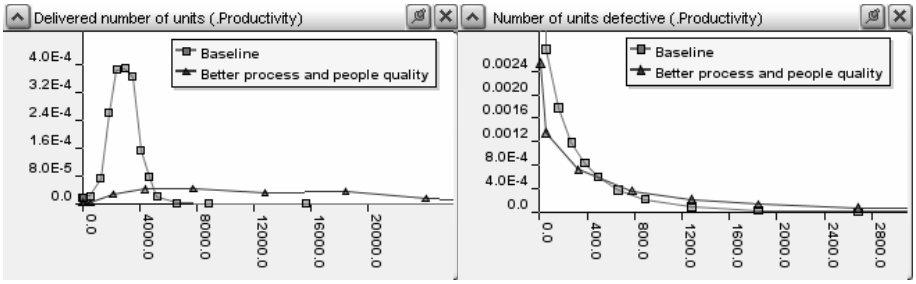

Fig. 6. Predicted functionality and quality of delivered software for better process and people quality

The model predicts that we should expect to be able to develop around four times more software then initially and that we should expect that this software will contain around two times more defects then initially (Fig. 6). However, that increase in number of defective units does not mean the decrease in quality. In fact we should expect error rate to drop by a half. It means that the larger number of defective units is purely because the functionality of delivered software will increase even more.

4.5 Predicting Functionality and Quality for a More Complex Project

In this scenario we wish to predict the functionality and quality of delivered software. Our assumptions are the same as in the first scenario. The only difference is that now we estimate that the current project is 70% more complex than past projects.

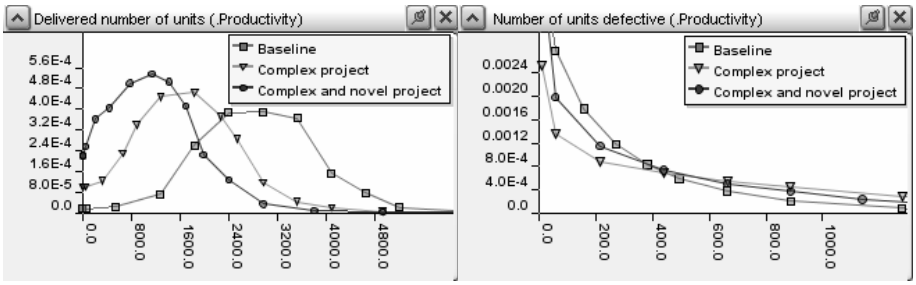

Fig. 7. Predicted functionality and quality for complex and novel project

Fig. 7 illustrates that, with this higher project complexity and the same resources, we can produce less and with lower quality compared to the scenario which assumes no change in project complexity.

In addition to the higher project complexity, suppose we also estimate higher novelty in a sense that we will be able to reuse a smaller part of software compared to past projects. Fig. 7 illustrates even more decrease in the functionality delivered. We observe that this change causes us to expect less number of defective software units. But this is entirely due to the fact that it will be smaller. Project novelty has no impact on the error rate.

4.6 Predicting Functionality and Quality When More Effort is Devoted to Quality

Our aim is again to predict the functionality and the quality of the developed software. We are again assuming the same values for variables mentioned in the first scenario. The only difference is that we would like to analyze what will happen if we spend 25% more effort on improving software quality compared to the "typical" project in the past. We are not assuming the change in the total effort for the project.

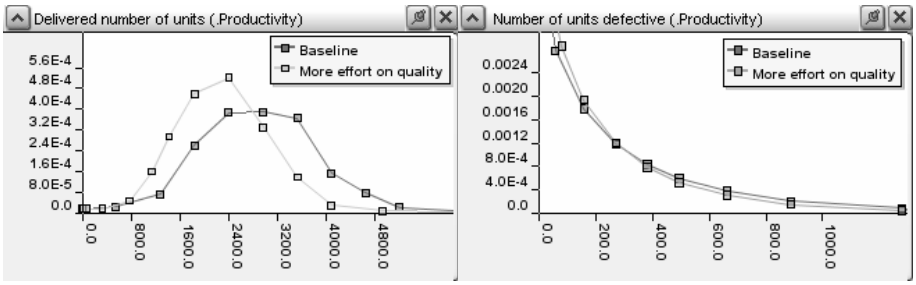

Fig. 8. Predicted functionality and quality of delivered software when more effort is spent on quality

Because we are spending proportionally more effort on quality rather than on extending functionality the results show that now we should expect to produce less software but with the better quality. Fig. 8 illustrates this.

5 Summary and Future Work

We have described a new model that can produce resource and quality predictions for software projects, but which more importantly can perform powerful what-if analysis and trade-off analysis to support project managers confronted with changing project realities.. Although this type of analysis was possible in a previous model (the MODIST project-level model) the model presented in this paper overcomes the two major weakness of the MODIST model:

- It is independent on the units of measurement for effort and functionality,

- It is much easier to use basic metrics, such as error and productivity rates, extracted from past project databases to adjust the model for the specific software company's needs.

In addition the model is also more accurate by virtue of the use of dynamic discretisation of numeric nodes.

This new 'productivity' model can be of immediate practical use since it directly addresses the improvements requested by the many users of the MODIST project level model. However, there are opportunities for still further improvements and refinements. For example, many of the variables, such as effort, process and people quality, are aggregations. This means that they describe project and process by a single value (distribution). It is useful to have an opportunity to split such variables into smaller parts, for example according to the development activities: requirements and specification, design, implementation, testing and rework. We would than be able, for example, to differentiate process quality by these phases or estimate/assign effort for these specific activities instead of only for the whole project. We are now developing such an extended productivity model that will provide even better decision support for project managers.

References

1. AgenaRisk, www.agenarisk.com (2006)
2. Bajaj N., Tyagi A., Agarwal R., Software Estimation – A Fuzzy Approach, ACM SIGSOFT Software Engineering Notes, Vol. 31 No. 3 (2006)
3. Bibi S., Stamelos I.: Software Process Modeling with Bayesian Belief Networks, Proc. of 10th International Software Metrics Symposium (Metrics 2004), Chicago (2004)
4. Boehm B., Clark B., Horowitz E., Westland C., Madachy R., Selby R.: Cost models for future software life cycle process: COCOMO 2.0, Annals of Software Engineering (1995)
5. Brooks F.P.: The Mythical Man-Month: essays on software engineering, 2nd edition, Addison Wesley, (1995)
6. Chulani S., Boehm B., Steece B.: Bayesian Analysis of Empirical Software Engineering Cost Models, IEEE Transactions on Software Engineering, Vol. 25, No. 4. (1999)
7. Fenton N.E., Marsh W., Neil M., Cates P., Forey S., Tailor M.: Making Resource Decisions for Software Projects, 26th International Conference on Software Engineering (ICSE 2004), May 2004, Edinburgh, United Kingdom. IEEE Computer Society (2004) 397-406
8. Fenton N.E., Neil M., Hearty P., Marsh W., Krause P., Mishra R.: Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets, Information & Software Technology, Vol. 49, (2007) 32-43
9. Fenton N., Radliński Ł., Neil M.: Improved Bayesian Networks for Software Project Risk Assessment Using Dynamic Discretisation, Software Engineering Techniques: Design for Quality (Ed. K. Sacha), IFIP International Federation for Information Processing, Vol. 227, Springer, Boston (2006)
10. Fenton N.E., Neil M.: A Critique of Software Defect Prediction Models, IEEE Transactions on Software Engineering, Vol. 25, No. 3 (1999)

11. Mockus A., Weiss D.M., Zhang P.: Understanding and Predicting Effort in Software Projects, Proc. 25th International Conference on Software Engineering (ICSE) (2003)
12. MODIST: MODIST Bayesian Network models, www.modist.org.uk/docs/modist_bn_models.pdf, (2003)
13. Moses J., Clifford J.: Improving Effort Estimation in Small Software Companies, EuroSPI 2000, Copenhagen, Denmark (2000)
14. Neil M., Tailor M., Marquez D., Bayesian statistical inference using dynamic discretisation, RADAR Technical Report, Queen Mary College, University of London (2005)
15. Neil M., Tailor M., Marquez D., Inference in Hybrid Bayesian Networks using dynamic discretisation, RADAR Technical Report, Queen Mary College, University of London (2005)
16. Pendharkar P.C., Subramanian G.H., Roger J.A.: A Probabilistic Model for Predicting Software Development Effort, IEEE Transactions on Software Engineering, Vol. 31, No. 7 (2005)
17. Putnam L.H.: A general empirical solution to the macrosoftware sizing and estimating problem, IEEE Trans. on Software Engineering, Vol. 4 (4) (1978)
18. Radliński Ł.: Modelling Complex Nodes in Bayesian Nets for Software Project Risk Assessment, Polish Journal of Environmental Studies, Vol. 15, No. 4C, Hard, Olsztyn (2006)
19. Sentas P., Angelis L., Stamelos I., Bleris G.L.: Software Productivity and Effort Prediction with Ordinal Regression, Journal of Information & Software Technology, Elsevier, 47 (1) pp. 17-29 (2005)
20. Stamelos I., Dimou P., Angelis L.: On the Use of Bayesian Belief Networks for the Prediction of Software Development Productivity, Information & Software Technology, Elsevier, 45, pp. 51-60 (2003)