Improved Decision-Making for Software Managers Using Bayesian Networks

Łukasz Radliński[†‡] Norman Fenton[‡] Martin Neil[‡] David Marquez[‡]

[†]Institute of Information Technology in Management
University of Szczecin
ul. Mickiewicza 64
71-101 Szczecin, Poland
+48 91 444 1911

[‡]Department of Computer Science
Queen Mary, University of London
Mile End Road
London E1 4NS, UK
+44 20 7882 7860 +44 20 7882 5221 +44 20 7882 8027

{lukrad, norman, martin, marquezd}@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.

Categories and Subject Descriptors

D.3.3 [**Software Engineering**]: Management – *productivity, software quality assurance (SQA), time estimation.*

General Terms

Your general terms must be any of the following 16 designated terms: Management, Measurement, Experimentation, Human Factors.

Keywords

software project modelling, project risk factors, Bayesian Nets.

1. INTRODUCTION

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

1. resources necessary to accomplish a software project.
2. 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 [9]. 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 [8]. 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* [9] that was developed as a part of EC Project MODIST [13], 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 (including use by organizations such as Siemens [23] who were not involved in the MODIST project), 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 the new model provides better predictions than the MOIST model and how software managers can use it 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, 12, 18]. Some of them were also Bayesian Nets [3, 10, 14, 17, 21, 22].

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 [9, 23], provides the greatest potential for decision support and is the easiest for adoption to our purposes. Figure 3 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.

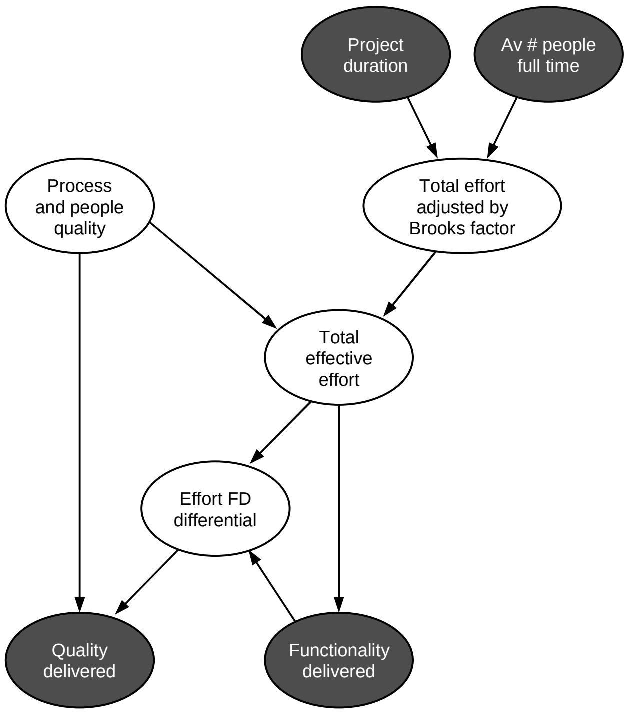


Figure 1. Project resource model (simplified), adapted from [7]

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 [9].

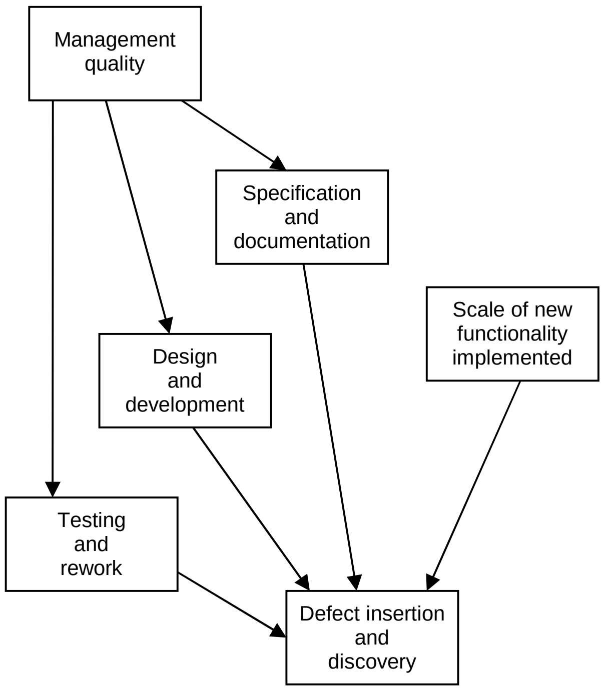

Management quality
Specification and documentation
Scale of new functionality implemented
Design and development
Testing and rework
Defect insertion and discovery

Figure 2. Schematic view of MODIST defect prediction model

In addition to the project level model a second MODIST model is concerned with prediction of number of defects [10]. A schematic view of this model is illustrated on Figure 2. The model predicts number of defects at several stages of software development: total, found in testing, fixed in rework, residual, etc. These numbers are affected by causal factors describing overall management quality level and process and people quality at different stages of software development: specification, design and development, testing and rework. These subnets also contain a reference to effort spent on each activity. However, effort is not expressed in absolute numerical value but on a ranked scale from "very low" to "very high". In this context it means: how appropriate is effort spent on a specific activity compared with the need.

Both MODIST models have been validated by various partners in the MODIST project and beyond and have also been incorporated into the AgenaRisk tool [1] that has several thousand users worldwide.

The main weaknesses of the MODIST models are:

1. These models are not integrated, i.e. end users are not able to perform full trade-off analysis between functionality, effort and quality using numerical values. The main assumption in estimating trade-offs between them is that knowing the values of two of them we can estimate the value of the third one. In the project-level model 'quality delivered' was primarily estimated on a 7-point ranked scale. The value for the numerical variable 'defects per KLOC' was then estimated depending purely on 'quality delivered'. Such a model structure could not give precise predictions for number of

defects. MODIST's second model was developed for precise defect prediction. But this one lacked the ability to express development effort explicitly on a numerical scale. In this model effort was expressed on a ranked scale.

2. The models use 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 (in project level model) 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.

3. Although the models contain several variables describing the (current) process and product of software development, they lack of ease of incorporating new empirical data by end users. Many of the prior distributions at the heart of the 'trade-off' part in the project-level 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 more easily to reflect such data. For example, among the easiest metrics for calculation from such databases are productivity and error rates for 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 models we have developed an improved BN model that provides support for:

1. Trade-off analysis between functionality, effort and quality where relationships between them are adjusted by additional factors missing in original MODIST models.
2. Different units of measurement for model variables, where (if they have some appropriate prior data) users can use any unit of measurement that they wish to. For example, the units for delivered software could be: lines of code, thousands of lines of code, function points, GUI screens, requirements, etc.
3. 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.

Figure 3 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.

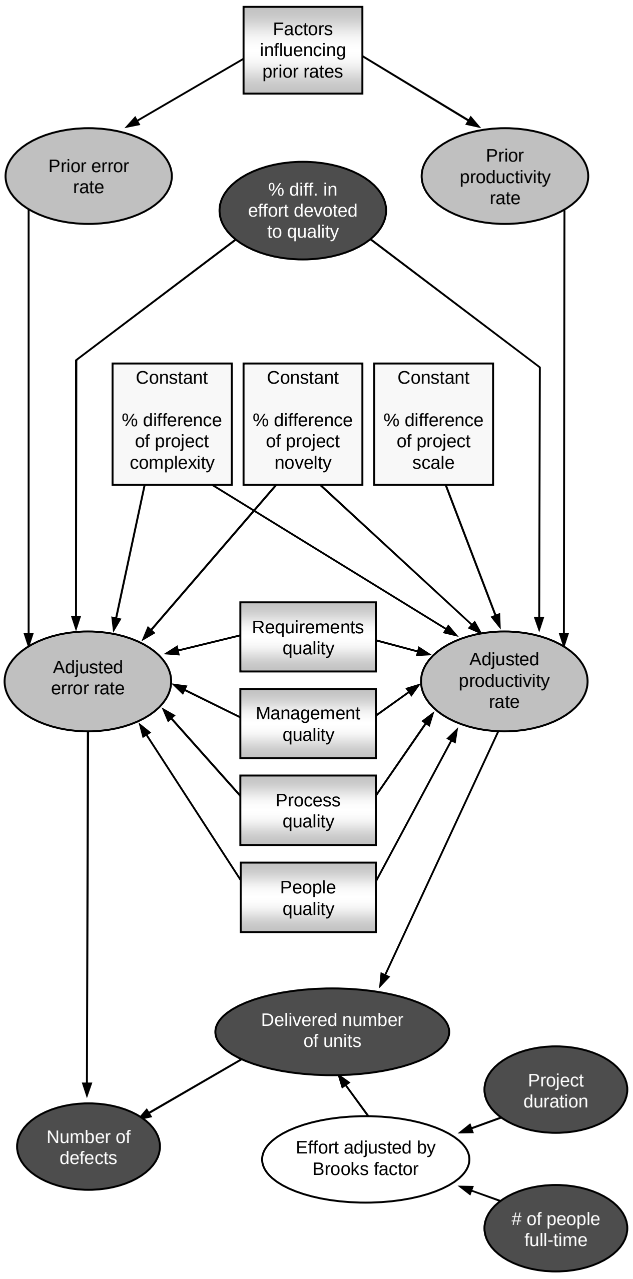

Factors influencing prior rates
Prior error rate
% diff. in effort devoted to quality
Prior productivity rate
Constant % difference of project complexity
Constant % difference of project novelty
Constant % difference of project scale
Requirements quality
Management quality
Process quality
People quality
Adjusted error rate
Adjusted productivity rate
Delivered number of units
Project duration
Number of defects
Effort adjusted by Brooks factor
# of people full-time

Figure 3. Schematic view of productivity model

The model consists of the following parts:

1. Factors influencing prior rates (Figure 3– 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 or distributions for the prior error and productivity rates. In such cases these rates are estimated by the model based on the values of descriptive environmental factors in this subnet, e.g. organization or application type, programming language type and other.

This part of the model incorporates results of analysis [20] which we performed using mainly the publicly available ISBSG database of software projects [11], and also compared these results with analyses available in the literature .

2. Prior error and productivity rates (Figure 3– 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 (Figure 3– 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 judgment. 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.).

4. Process and people factors that adjust error and productivity rates (Figure 3– 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 (“very low”, “low”, “normal”, “high”, “very high”) as opposed to the previously discussed constants, which are expressed on a continuous scale (real numbers). Their values are not absolute. Because of the difficulty users had in entering ‘absolute’ values for qualitative factors, in the productivity model we changed our approach. They reflect ***relative*** values compared to the typical past projects for which the prior productivity and error rate data is used. For example, setting the value of ‘staff quality’ to be “normal” means that in this project the staff quality 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 set to “high” it means that the staff quality is higher than on typical past projects. It does ***not*** mean that it is “high” in any absolute sense. Users of the model are always judging these qualitative factors relative to their own previous projects. If, on previous projects, the staff quality was always considered to be “very high” then setting it to “high” on a new project means that the new project is even higher than the “very high” previous average. In this model the values higher then “normal”, namely “high” and “very high”, increase productivity rate and decrease the error rate. The values below “normal”, namely “low” and “very low” decrease productivity rate and increase error rate.

5. Adjusted error and productivity rates (Figure 3– 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.

6. Trade-off component between the quality, functionality and effort (Figure 3– 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 written. Knowing the functionality and error rate the model calculates how many units of software 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 [9, 13]. 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” plays a key role in the analysis of trade-offs. The assumption here is a simple one: The greater the proportion of total project effort devoted to purely quality assurance activities, the smaller the proportion can be spent on writing new software . 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.

One other key improvement in the revised model compared to the MODIST model is that we have applied the dynamic discretisation algorithm [15, 16] 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 accurate predictions. We have previously tested this algorithm for the defect prediction model from the MODIST project [7]. 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 specific prior productivity and error rates, and total effort for the project but 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 what quality, can we produce?
- How good do process and people need to be if we actually need better quality software than the model predicts?
- How much more effort do we have to put to deliver better software?
- How much effort do we need to deliver software of specified functionality and quality?

Although the MODIST model can answer such questions, the correct predictions will only be given for "typical" project situations, where project constraints (such as complexity or novelty) and environmental factors (such as application type or programming language type used in project) are at the same level as in the past projects.

The improved model also gives answers to questions stated above. Furthermore, it strongly extents this list of questions it can answer. These additional questions are very important from the software managers' point of view for a better decision support, especially with changing project and process factors, and include the following:

- What are numerical relationships between functionality, effort and number of defects?
- How does the change in the process and people quality affect the functionality delivered and the quality of software.
- What impact on functionality and quality will have a proportional change of effort devoted to quality?
- How will our predictions for functionality, quality or effort look like if we want to use our own units of measurement for them?
- What software functionality and quality should we expect if actually our project is more complex than the previous ones?
- How does a change in inherent project factors effect predictions?
- What predictions should we expect if we feed the model with non-default values productivity and error rate?

The general task for software manager using the improved model is to assess how this project differs compared with typical past projects and enter these estimates into the model as observations. Not all of the factors need to be entered into the model because Bayesian Net will also be able to perform calculations even in the case of missing values. In such cases default probability distributions for the missing variables will be used.

4.1 Trade-offs Using Numerical Scale

As discussed in Section 2, the MODIST project–level model contains variables for predicting functionality, effort and quality. However, one of them, 'quality delivered' is expressed on a ranked scale only (Figure 4). That reduces potential of performing detailed trade-off analysis.

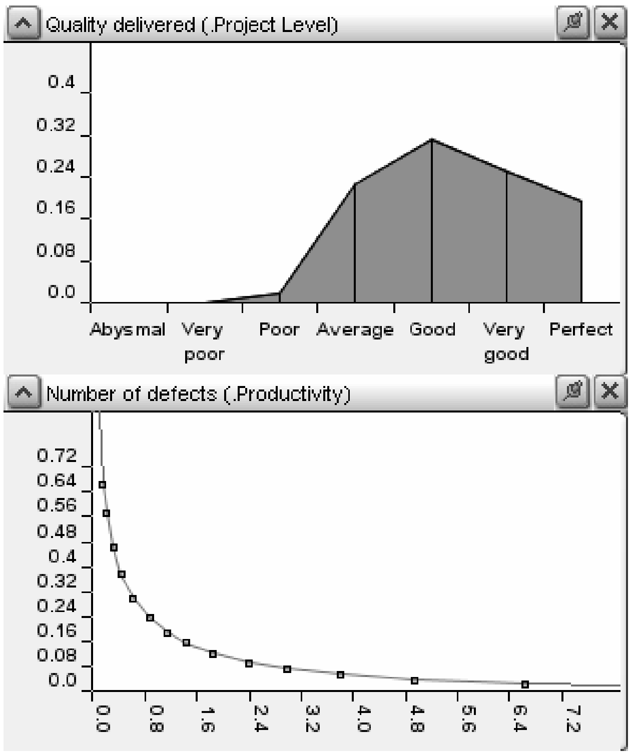


Figure 4. Comparison of prediction of quality in MODIST project-level and productivity model

The new productivity model fills the gap in MODIST model and contains variables describing project trade-offs between functionality, effort and quality on a numerical scale (Figure 4).

4.2 Entering Non-Default Productivity and Error Rates

MODIST project-level model incorporates trade-off relationships without the ability for users to specify their own prior productivity and error rates. Productivity model enables users to enter their own productivity and error rates which they can estimate outside the model.

For example, a software company delivers specific types of software which requires spending more development effort than for other software of similar size. There may also be a case where the nature of software and the development process may lead to situations where they achieve "non-standard" quality in terms if number of defects per unit of size. In such cases, which are not that rare, it may be inappropriate for them to use a model that does not enable them to use such information.

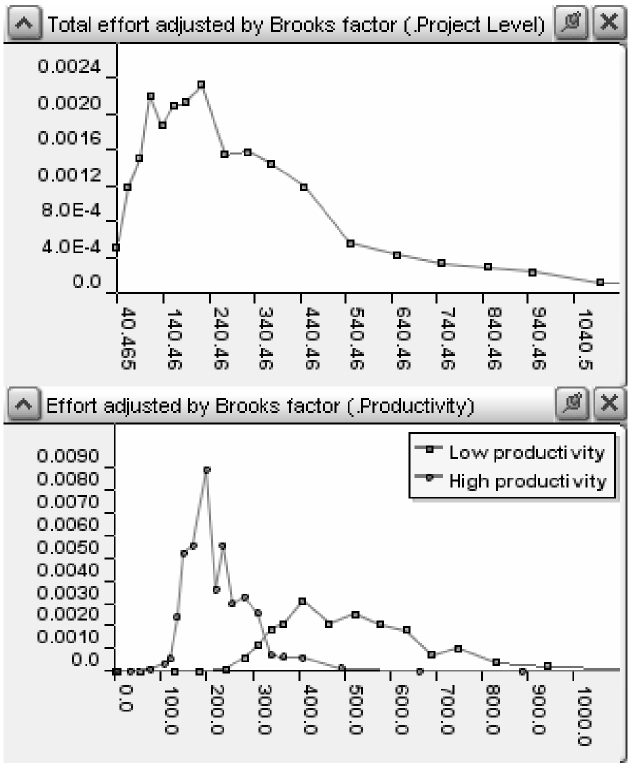


Figure 5. Effort predictions for different scenarios in MODIST project-level and productivity model

Figure 5 illustrates two examples of different predictions for development effort. In the first example we assume to have lower productivity. In the second one we assume to have higher productivity. As presented on the figure, if we want to develop software for a given size containing specific number of defects, in the second scenario we need less effort to reach these aims. MODIST model does not allow us to enter our own values for productivity or error rates and with the same constraints for functionality and size predicts a single probability distribution for quality (Figure 4).

4.3 Using Custom Units of Measurement

As described in Section 3, the productivity model does not have constraints in using custom units of measurement for entering observations and obtaining predictions. If users wish to use units of measurement of their choice they simply enter prior values as observations for variables expressed in their units of measurement.

For example, users want the model to predict the functionality which they could deliver when spending specific amount of effort and meet a constraint about the quality. Furthermore, suppose they want this functionality expressed in a non-standard unit of measurement: “number of requirements”. The MODIST model cannot be used in this case at all because functionality is expressed in number of function points there. In the productivity model users need to do the following:

- enter an observation for effort expressed in any unit of measurement, such as “person-month”, “person-week” or even “programming-group-week” if such non-standard unit of measurement is used in a company,
- enter an observation for their prior productivity rate (meaning productivity rate for a typical past project) in appropriate unit of measurement, such as “number of requirements per person-month” or “number of requirements per programming-group-week” or other – depending on unit of measurement used for effort,
- enter an observation for the “number of defects” node, which in this case would mean “number of defective requirements”.

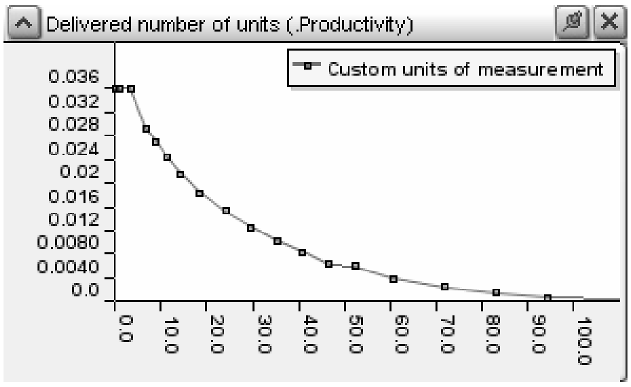


Figure 6. Using custom units of measurements

Figure 6 illustrates prediction for functionality in this example. The figure does not display units of measurement itself. Users have to infer them depending on units of measurement in other values they entered as observations. In our example we notice totally different scale of predicted values for functionality (X-axis) compared with other examples in this paper (Figure 7 and 10).

4.4 Change in Process and People Quality

As discussed in previous sections, both MODIST models and productivity model contain variables describing process and people quality. In both of them process and people quality influences on key variables, such as functionality, effort and quality delivered.

Figure 7 illustrates example in which our aim is to predict how much software we are able to develop for given effort in scenarios of having worse and better process and people quality. In MODIST we enter our estimates for process and people quality for the current project. In productivity model we define how process and people quality changed compared with the past.

As we could expect both models predict that having better development process and people increases the size of software which we could develop using limited effort. The key difference between the models here is that in productivity model we only need to define how process and people quality changed, while in MODIST we need to define the real process and people quality.

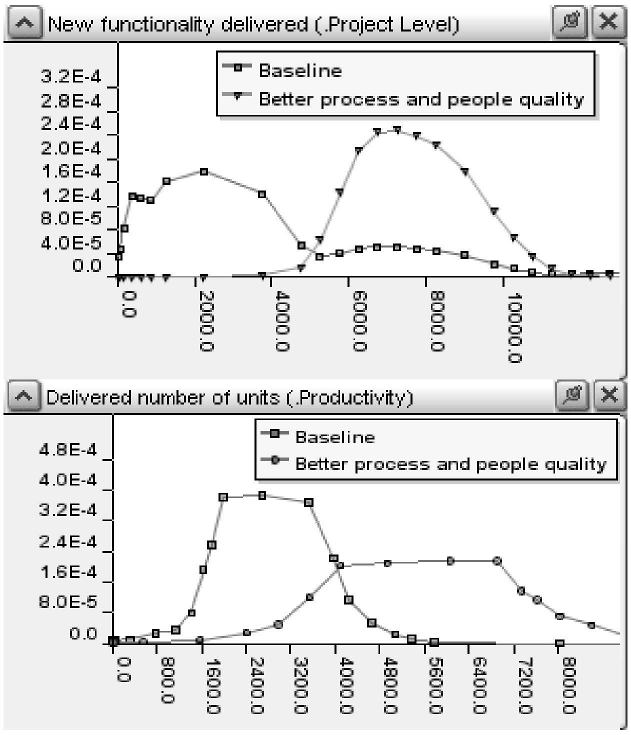


Figure 7. Predicted functionality for different process and people quality values in MODIST project-level and productivity model

4.5 Impact of Changed Project Constraints

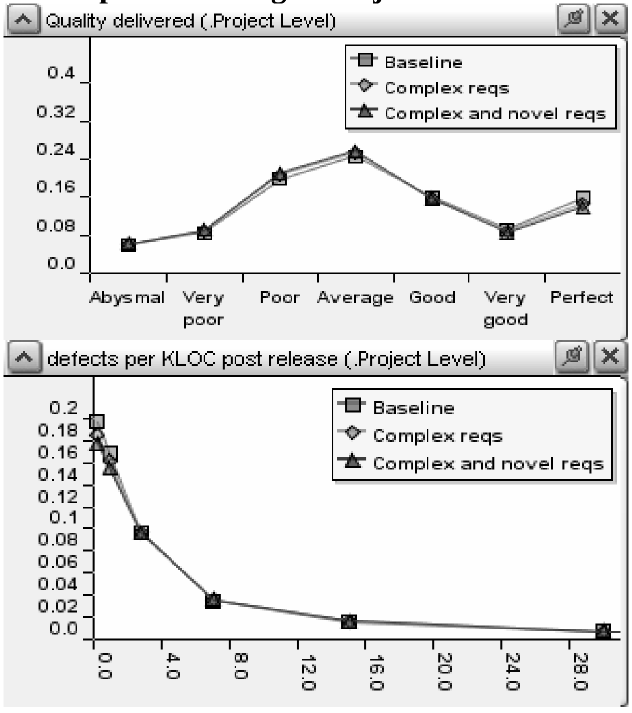
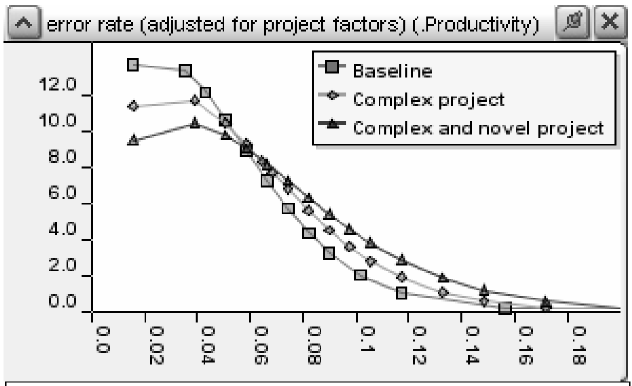


Figure 8. Predicted quality for complex and novel project in MODIST project-level model

The MODIST project-level model contains ranked node variables such as "requirements complexity", "requirements novelty", and "scale of distributed communications" that impact on overall process and people quality and thus on trade-off relationships. However, because of the structure of the model (high number of nodes with small number of states between project factors and quality nodes) the different values of project factors have almost no impact on predicted 'quality delivered' and 'error rate' (Figure 8).



Number of defects					
Baseline		**Complex project**		**Complex and novel project**	
Mean:	277.89	Mean:	295.56	Mean:	312.50
Median:	171.93	Median:	187.30	Median:	200.51
SD:	307.35	SD:	319.90	SD:	334.99

Figure 9. Predicted quality for complex and novel project in productivity model

In productivity model we can analyze the impact of project constraints on numeric quality variables. Figure 9 illustrates that, with this higher project complexity and the same resources, for a project of specific size, we should expect to deliver this software 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. Figure 9 illustrates that now we should expect even more defects compared with two previous scenarios.

We can also observe different number of states for 'error rate' node in MODIST project-level model (Figure 8) compared with productivity model (Figure 9). Because central tendency statistics are calculated using more shorter intervals in productivity model, we are automatically obtaining more precise predictions. The increased accuracy is even better because of applying dynamic-discretisation algorithm.

4.6 Impact of Environmental Factors

As we described it Section 3, prior productivity and error rates can be estimated by our model using descriptive factors if users cannot estimate them outside the model.

For example, a company has recently changed its profile and started to deliver different type of software than in the past. It means that they do not have enough volume of data about projects of similar type developed in the past. We enter observations for these descriptive nodes and as a result get different values (probability distributions) for prior error and productivity rates (Figure 10).

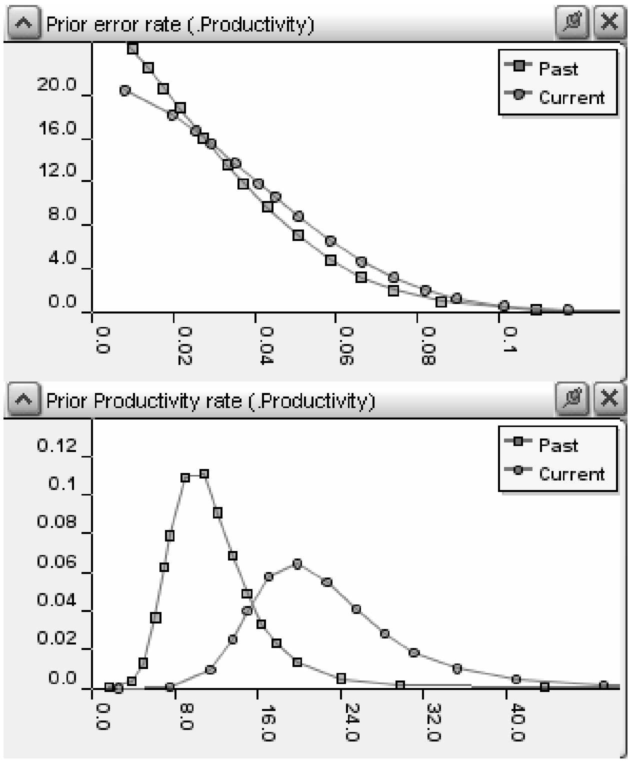


Figure 10. Predicting prior error and productivity rates for given environmental factors

MODIST model does not capture any descriptive factors about the company or project. From this point of view it may give incorrect predictions for functionality, effort, quality or other variables for companies and projects being outside the original MODIST scope. Productivity model incorporates several descriptive factors which adjusts relationship between functionality, effort and quality.

4.7 Change of Effort Devoted to Quality

In the MODIST project-level model we cannot enter directly information that we plan to spend more effort on testing than usually. However the model believes that if we used more effort than necessary for software of given size, then it must have been to balance worse process and people in achieving similar level of software quality. Actually, increased total effort may be due to increased project complexity or other factors describing project which are not under control of project managers but causing the need of additional effort on quality assurance. MODIST model predicts only very small (not significant) difference in 'requirements difficulty' (Figure 11).

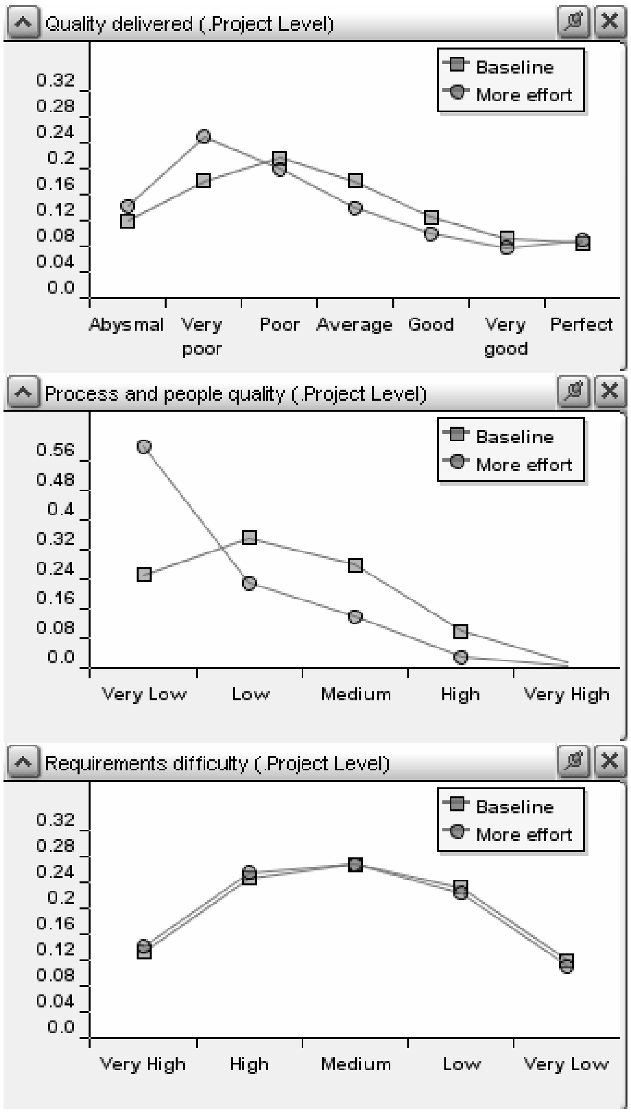


Figure 11. Predictions in MODIST project-level model when spending more effort than necessary

In the productivity model we can define the difference in effort spent on quality assurance in the current project compared with a "typical" past project.

For example, our aim is to predict the functionality and the quality of the developed software for a given effort (Figure 12). In the first scenario we are assuming that we spend the same proportion of effort on quality assurance as in the past. In the second scenario 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.

Because we are spending proportionally more effort on quality rather than on extending functionality the results show that in the second scenario we should expect to produce less software but with the better quality.

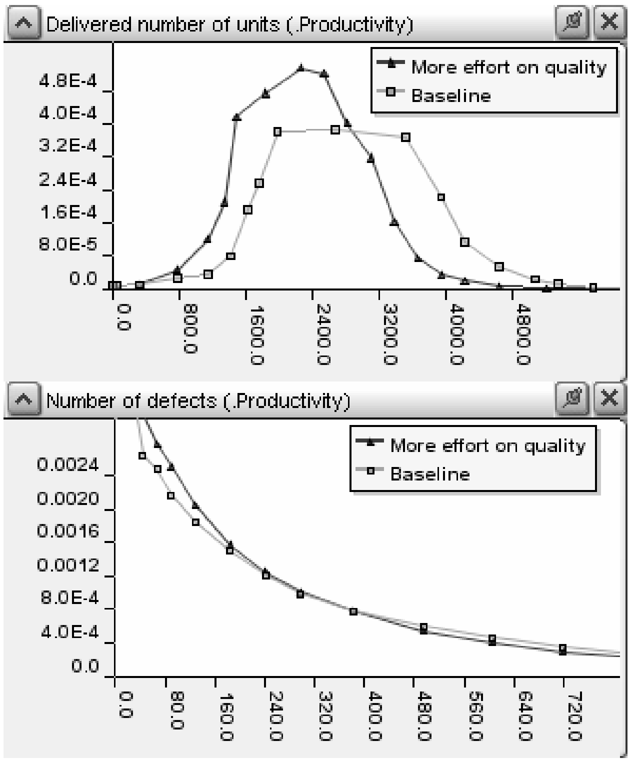


Figure 12. Predicted functionality and quality of delivered software when more effort is spent on quality

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 partially possible in a previous model (the MODIST project-level model) the model presented in this paper overcomes some significant weaknesses of the MODIST model:

- It fully captures trade-off relationships between functionality, effort and quality which are adjusted by several factors describing project and development process.
- 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.

6. REFERENCES

[1] AgenaRisk, www.agenarisk.com, 2007.

[2] Bajaj, N., Tyagi, A., and Agarwal, R. Software Estimation – A Fuzzy Approach, *ACM SIGSOFT Software Engineering Notes*, Vol. 31 No. 3 (2006)

[3] Bibi, S., and 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., and 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., and Steece, B. Bayesian Analysis of Empirical Software Engineering Cost Models, *IEEE Transactions on Software Engineering*, Vol. 25, No. 4. (1999).

[7] Fenton, N., Radliński, Ł., and 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.

[8] Fenton, N.E., and Neil, M. A Critique of Software Defect Prediction Models, *IEEE Transactions on Software Engineering*, Vol. 25, No. 3 (1999).

[9] Fenton, N.E., Marsh, W., Neil, M., Cates, P., Forey, S., and 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.

[10] Fenton, N.E., Neil, M., Marquez, D., Hearty, P., Marsh, W., Krause, P., and Mishra R. Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets, *Information & Software Technology*, Vol. 49, (2007) 32-43.

[11] ISBSG. *Estimating, Benchmarking & Research Suite Release 9*, International Software Benchmarking Standards Group, 2005.

[12] Mockus, A., Weiss, D.M., and Zhang, P. Understanding and Predicting Effort in Software Projects, Proc. 25th *International Conference on Software Engineering (ICSE)* (2003).

[13] MODIST. MODIST Bayesian Network models, www.modist.org.uk/docs/modist_bn_models.pdf, 2003.

[14] Moses, J., and Clifford, J. Improving Effort Estimation in Small Software Companies, *Proc. of EuroSPI 2000*, Copenhagen, Denmark, 2000.

[15] Neil, M., Tailor, M., and Marquez, D., Bayesian statistical inference using dynamic discretisation, *RADAR Technical Report*, Queen Mary College, University of London, 2005.

[16] Neil, M., Tailor, M., and Marquez, D., *Inference in Hybrid Bayesian Networks using dynamic discretisation*, RADAR Technical Report, Queen Mary College, University of London, 2005.

[17] Pendharkar, P.C., Subramanian, G.H., and Roger, J.A. A Probabilistic Model for Predicting Software Development Effort, *IEEE Transactions on Software Engineering*, Vol. 31, No. 7 (2005).

[18] Putnam, L.H. A general empirical solution to the macrosoftware sizing and estimating problem, *IEEE Trans. on Software Engineering*, Vol. 4, 4 (1978).

[19] Radliński Ł. Modelling Complex Nodes in Bayesian Nets for Software Project Risk Assessment, *Polish Journal of Environmental Studies*, Vol. 15, No. 4C, Hard, Olsztyn, Poland (2006).

[20] Radliński, Ł., Fenton, N., Neil, M., and Marquez, D. Modelling Prior Productivity and Defect Rates in a Causal Model For Software Project Risk Assessment, submitted to: *Congress of Young IT Scientists*, Świnoujście, Poland, Sept. 2007.

[21] Sentas, P., Angelis, L., Stamelos, I., and Bleris, G.L. Software Productivity and Effort Prediction with Ordinal Regression, *Journal of Information & Software Technology*, Elsevier, 47, 1 (2005), 17-29.

[22] Stamelos, I., Dimou, P., and Angelis, L. On the Use of Bayesian Belief Networks for the Prediction of Software Development Productivity, *Information & Software Technology*, Elsevier, 45, (2003) 51-60.

[23] Wang, H., Peng, F., Zhang, C., and Pietschker, A. Software Project Level Estimation Model Framework based on Bayesian Belief Networks, Sixth International Conference on Quality Software (QSIC'06) pp. 209-218, 2006