Empir Software Eng (2008) 13:499–537
DOI 10.1007/s10664-008-9072-x

On the effectiveness of early life cycle defect prediction with Bayesian Nets

Norman Fenton · Martin Neil · William Marsh · Peter Hearty · Łukasz Radliński · Paul Krause

Published online: 27 June 2008

Editor: Tim Menzies

Abstract Standard practice in building models in software engineering normally involves three steps: collecting domain knowledge (previous results, expert knowledge); building a skeleton of the model based on step 1 including as yet unknown parameters; estimating the model parameters using historical data. Our experience shows that it is extremely difficult to obtain reliable data of the required granularity, or of the required volume with which we could later generalize our conclusions. Therefore, in searching for a method for building a model we cannot consider methods requiring large volumes of data. This paper discusses an experiment to develop a causal model (Bayesian net) for predicting the number of residual defects that are likely to be found during independent testing or operational usage. The approach supports (1) and (2), does not require (3), yet still makes accurate defect predictions (an R^2 of 0.93 between predicted and actual defects). Since our method does not

N. Fenton · M. Neil · W. Marsh · P. Hearty · Ł. Radliński
Department of Computer Science, Queen Mary, University of London, Mile End Road, London, UK

N. Fenton
e-mail: norman@dcs.qmul.ac.uk

M. Neil
e-mail: martin@dcs.qmul.ac.uk

W. Marsh
e-mail: william@dcs.qmul.ac.uk

P. Hearty
e-mail: hearty@dcs.qmul.ac.uk

Ł. Radliński (✉)
Institute of Information Technology in Management, University of Szczecin, Szczecin, Poland
e-mail: lukrad@dcs.qmul.ac.uk

P. Krause
Department of Computing, University of Surrey, Guildford, Surrey, UK
e-mail: p.krause@surrey.ac.uk

require detailed domain knowledge it can be applied very early in the process life cycle. The model incorporates a set of quantitative and qualitative factors describing a project and its development process, which are inputs to the model. The model variables, as well as the relationships between them, were identified as part of a major collaborative project. A dataset, elicited from 31 completed software projects in the consumer electronics industry, was gathered using a questionnaire distributed to managers of recent projects. We used this dataset to validate the model by analyzing several popular evaluation measures (R^2, measures based on the relative error and Pred). The validation results also confirm the need for using the qualitative factors in the model. The dataset may be of interest to other researchers evaluating models with similar aims. Based on some typical scenarios we demonstrate how the model can be used for better decision support in operational environments. We also performed sensitivity analysis in which we identified the most influential variables on the number of residual defects. This showed that the project size, scale of distributed communication and the project complexity cause the most of variation in number of defects in our model. We make both the dataset and causal model available for research use.

Keywords Software defect prediction · Qualitative factors · Quantitative data · Bayesian network · Decision support · Sensitivity analysis

1 Introduction

The ultimate goal of research in software metrics (Fenton and Pfleeger 1998; Jensen 1996; Jones 1999) is to help project managers make decisions under uncertainty. In particular, central aids to decision making are the abilities to estimate the cost of developing software, and to predict the quality likely to be achieved from a given development effort. The MODIST ('Models of Uncertainty and Risk for Distributed Software Development') Project (MODIST 2003), which was part-funded by the European Commission, was concerned with these problems in large distributed software projects. The project partners were Agena, Israel Aircraft Industries, QinetiQ and Philips Electronics. As part of this project a group of experienced project managers identified a set of factors influencing cost and quality outcomes, which were formed into a number of causal models. The primary objective of one such model, the focus of this paper, is to predict the number of residual defects in major software systems. We believe this model is relevant for many large commercial software systems, where it is accepted that residual ("non-blocking") defects have to be lived with. As it stands, the model will be less relevant for safety critical software or core algorithmic software where very few post-release defects can be tolerated, and so where few defects will be found later in testing phases or in operational usage.

The use of a generic version of the MODIST defect prediction model has been described in (Fenton et al. 2007b) including how the model can be used in multiple life-cycle iterations. Some brief details of both the validation of the core model and the qualitative factors used were presented in (Fenton at al. 2007c). The objective of this paper is to extend the work of (Fenton at al. 2007c) and to describe in more detail both the rationale for the qualitative factors in the causal model and its validation. We also make the data and model available to other researchers.

In Section 2 we describe the rationale for our interest in developing causal models (Bayesian nets) as opposed to classical (notably regression-based) methods. In Section 3 we provide the details about the structure of our model.

The data needed for the model was not available in any publicly accessible form, even though similar factors are used in popular models supporting software managers, most notably COCOMO-II (Boehm et al. 1995) for software cost estimation. For example, the ISBSG dataset (ISBSG 2007), containing data on about 4,106 projects, helps us to quantify some of the relationships in the model, but it does not help in validation because of the absence of the qualitative, causal factors. To get the necessary data, senior project managers in one organization provided the information for 31 projects. We had to provide refined and more detailed descriptions and measurement schemes for most of the factors in the model. This process is described in Section 4. The resulting quantitative data is presented in Section 5, with the qualitative data in Section 6. Section 7 describes some issues arising from data collection, while in Section 8 we summarise the model validation results. The results show that the causal model, independently built using a combination of expert judgement and historical data, was able to make reasonably accurate predictions for the new projects. In Section 9 we present examples of the model's usage to help managers in decision-making. Section 10 contains the details of the sensitivity analysis that we performed on the model.

2 The Need for Causal Models

There have been many non-causal models for software defect prediction and some of these have achieved very good accuracy with few input variables and no qualitative factors. For example, in a model for predicting the probability of detecting a defect (pd) and the probability of false alarm in detecting a defect (pf) (Menzies et al. 2007) the authors achieve a very respectable predictive accuracy of: pd=71% and pf=25%. However, their model uses static code attributes as input variables. This means that their model cannot be used before completing and delivering some source code. Ostrand et al. (2005) also achieve respectable accuracy in their models (where the focus is on identifying especially fault-prone components)—20% of the files with the highest predicted number of faults contained an average of 83% of the faults that were actually detected. However, this type of prediction depends on the availability of relevant data from previous releases of the same system.

The problem we were addressing goes far beyond the constraints of these assumptions. In our target domains, we certainly could not assume that the system under consideration was merely an incremental release of some existing system. Nor could we assume that we had any detailed information about the code base of the system. Our problem was to make predictions as early as possible in the project life-cycle including even before any code had been produced or any defects found in testing. But our model had to be suitably robust and complete to enable updated predictions once code was developed and defects were being found. Inevitably this meant our model had to use process factors more than code attributes, but has the benefit of being usable for development planning where various trade-off scenarios can be assessed.

In many respects what we are doing is confirming the ideas described in previous studies such as (Fenton and Neil 1999) and (Chulani and Boehm 1999). Although the Chulani and Boehm work was focused on effort prediction rather than fault prediction, they identify the problems associated with parametric models being empirically calibrated to actual data from completed software projects. In particular, they explain why the most commonly used technique (multiple regression) imposes assumptions frequently violated by software engineering datasets. By using Bayesian analysis incorporating qualitative factors their model for estimating project effort was shown to be significantly more accurate than regression models.

To emphasize the radically different nature of our modelling approach we note that the standard practise in building models in software engineering normally involves three steps:

1. Collecting domain knowledge (previous results, expert knowledge);
2. Building a skeleton of the model based on step 1 including as yet unknown parameters;
3. Estimating the model parameters using historical data to either point values (as, for example, in the regression models) or probability distributions (as, for example, in Bayesian analysis).

The novelty in our approach in developing a model is to follow only steps 1 and 2 without requiring step 3. Our experience shows that it is extremely difficult to obtain reliable data of the required granularity, or of the required volume with which we could later generalize our conclusions. Therefore, in searching for a method for building a model we cannot consider methods requiring large volumes of data.

The pitfalls of building a defect-prediction model based purely on empirical data were first laid out in (Fenton and Neil 1999). The biggest danger is in missing explanatory variables. For example, a number of empirical studies have attempted to use information on component defects found pre-release to predict component defects post-release, based on the assumption of some positive correlation. Yet the study (Fenton et al. 2002b) found in many cases that the components with higher numbers of defects found pre-release experienced fewer defects in operational usage. Once the systems exhibiting these apparently counter-intuitive results were further analysed the explanation was (of course) that, in most cases, the modules for which higher numbers of defects were found during testing were simply those that were tested more effectively. This particular example was a key motivation for the MODIST model and it reminds us of the following intriguing question:

> If you discover very few defects in your software is that a 'good thing' or a 'bad thing'?

The general assumption is that, especially if the question concerns defects found by customers in operation, then the answer must be 'a good thing'. But, at an international software metrics conference some years ago a leading metrics expert recounted an interesting story about a company-wide metrics programme that he had been instrumental in setting up. He said that one of the main objectives of the programme was to achieve process improvement by learning from metrics what process activities worked and what ones did not. To do this the company looked at those projects that, in metrics terms, were considered most successful. These were the projects with especially low rates of customer-reported defects, measured by defects per thousand lines of code (KLOC). The idea was to learn what processes characterised such successful projects. A number of such 'star' projects were identified, including some that apparently achieved the magical perfect reliability target of zero defects per KLOC in the first 6 months post-release. But, it turned out that what they learned from this was very different to what they had expected. Few of the star projects were, in fact, at all successful from any commercial or subjective perspective. In fact, most were disasters. The reason for the very low number of defects reported by customers was that they were generally so poor that they were little used, or not used at all. The programme had completely missed "usage" as a causal impactor on the number of observed defects.

Causal models, also known as Bayesian Nets (BNs; Jensen 1996; Neapolitan 2004; Winkler 2003), enable us to incorporate qualitative explanatory factors such as testing

quality or usage to avoid the above problems, even if we have no relevant empirical data. A BN consists of two parts:

- A directed acyclic graph—Each node is a model variable, and links between the nodes reflect (causal) influences between the variables.
- Probability distributions—Unconditional probabilities for the nodes without parents and conditional probabilities for nodes with parents (depending on the parents' states).

Particular advantages of BNs for our research purposes are:

- Handling the missing data—In a BN each variable is assigned a prior probability. If users do not provide an observation for such a variable the default prior probability will be used in calculations.
- No fixed list of input and output variables—If a user provides an observation for a variable such a variable becomes an input variable; if a variable is left without an observation, it becomes an output variable.
- Explicit capturing of uncertainty about each unknown variable—all predictions are in the form of probability distributions rather than point values.
- Easier understanding of the relationships between variables—the model explicitly captures causal/influential relationships between variables shown on the graph.

3 Defect Prediction Causal Model

3.1 Overview of the Model

This section is an overview of the BN model whose factors were elicited from experienced project managers in the MODIST project (MODIST 2003). The model is presented in schematic form in Fig. 1. Each rectangle represents a subnetwork illustrated in detail in Figs. 2, 3, 4, 5, 6, 7, 8. A detailed description of the previous versions of the model can be found in (Fenton et al. 2002a; Fenton et al. 2002b; Fenton et al. 2004; Fenton et al. 2007b; Neil et al. 2003). The model itself can be downloaded from (MODIST BN 2007 and

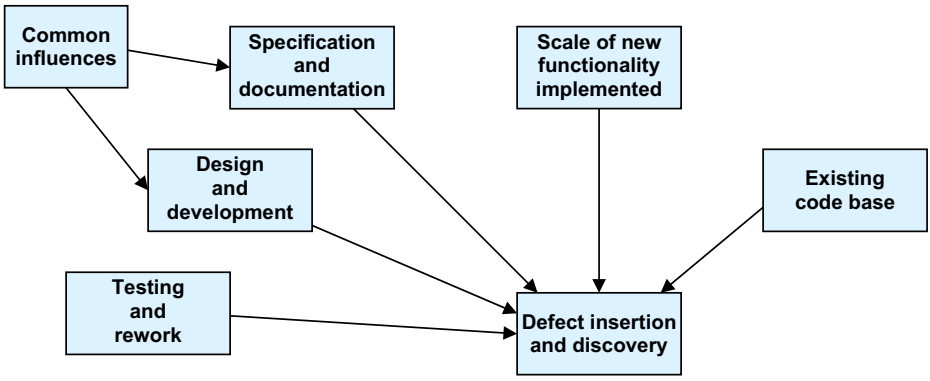

Fig. 1 Schematic view of defect prediction model

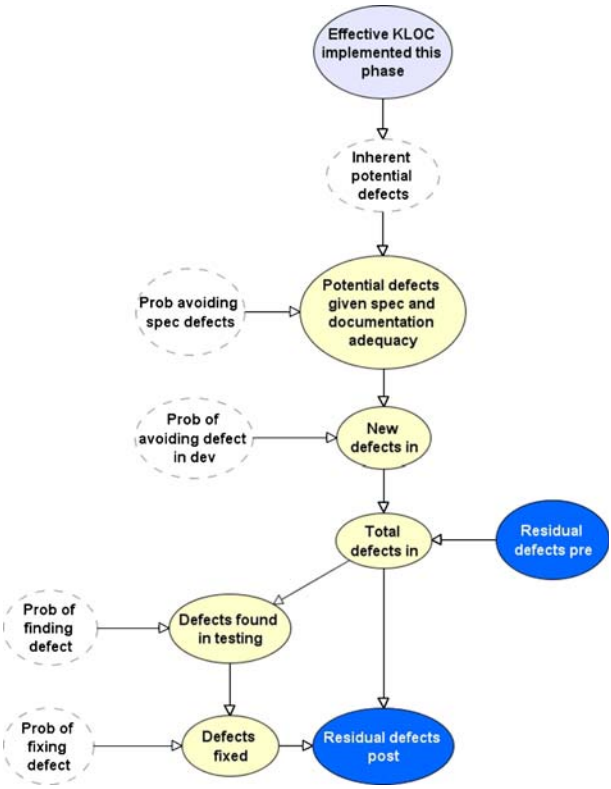

Fig. 2 Defect insertion and discovery subnet

Boetticher et al. 2008) and viewed and executed using Bayesian net software which can be downloaded for free from (Agenarisk 2007).

Each subnetwork is a part of the Bayesian network, with nodes representing probabilistic variables and arcs representing causal relationships between variables. It is important to note that the model not only reflects relationships between variables, which could be reflected in regression-type models, but also direct cause–effect relationships. For example, a more rigorous testing process leads to an increased probability of finding and fixing a

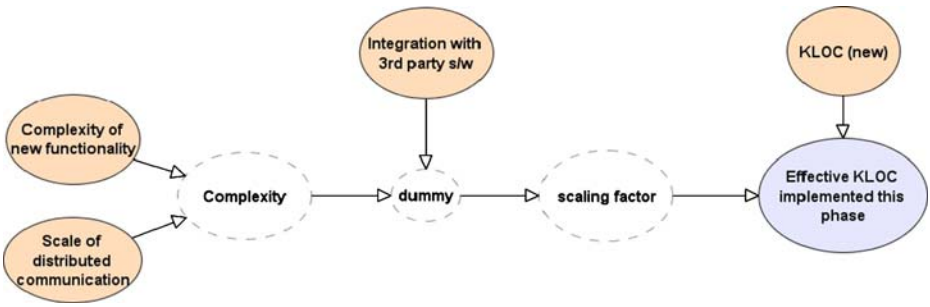

Fig. 3 Scale of new functionality implemented

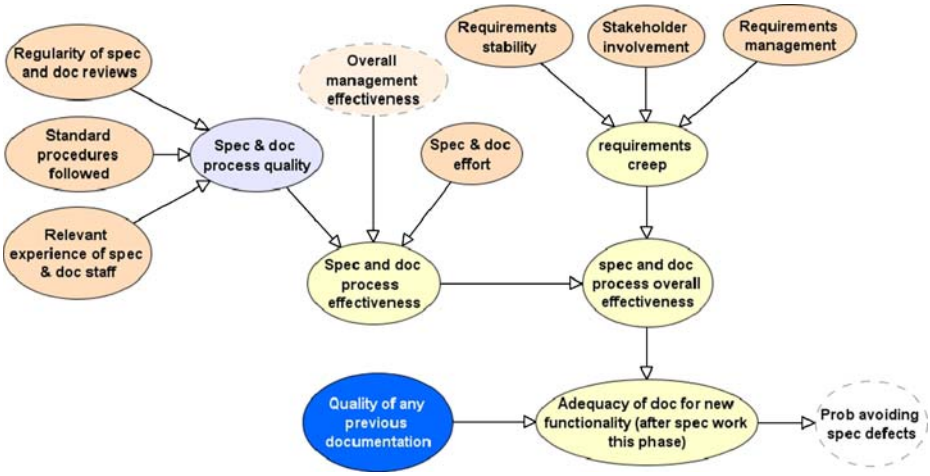

Fig. 4 Specification and documentation subnet

defect and thus to a reduced number of defects left in the software after testing. As an extreme case, the model includes the knowledge that no defects will be found if no testing is done. Such causal knowledge is almost impossible to 'learn' from limited data alone, yet can be readily and consistently elicited from experts in the subject domain. This is one of the reasons that Bayesian networks appear to be so useful.

Descriptions of the qualitative factors used in this model are presented in Section 4.

We developed this causal model based on a combination of the following sources:

- empirical data from the literature;
- empirical data from the project partners;
- subjective judgment of project managers and other experts in the collaborative project, where no relevant data was available.

This causal model was not developed from the data reported in this paper. The data presented here were available for us after the model was developed. Thus, we used them *only* to validate the model.

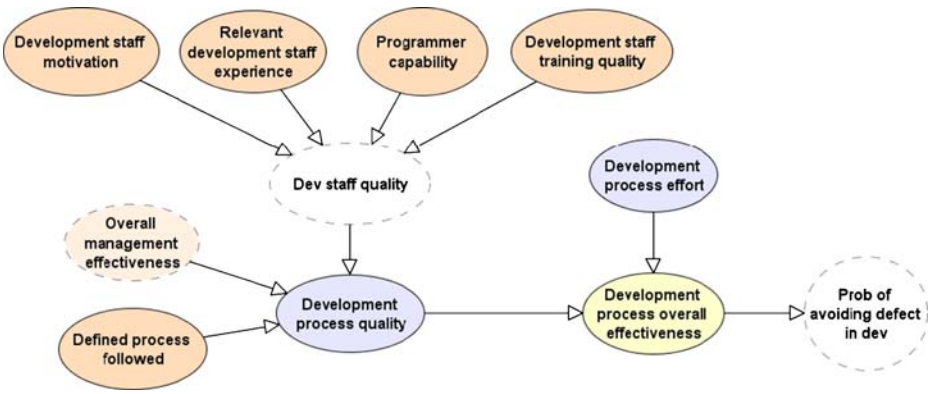

Fig. 5 Design and development subnet

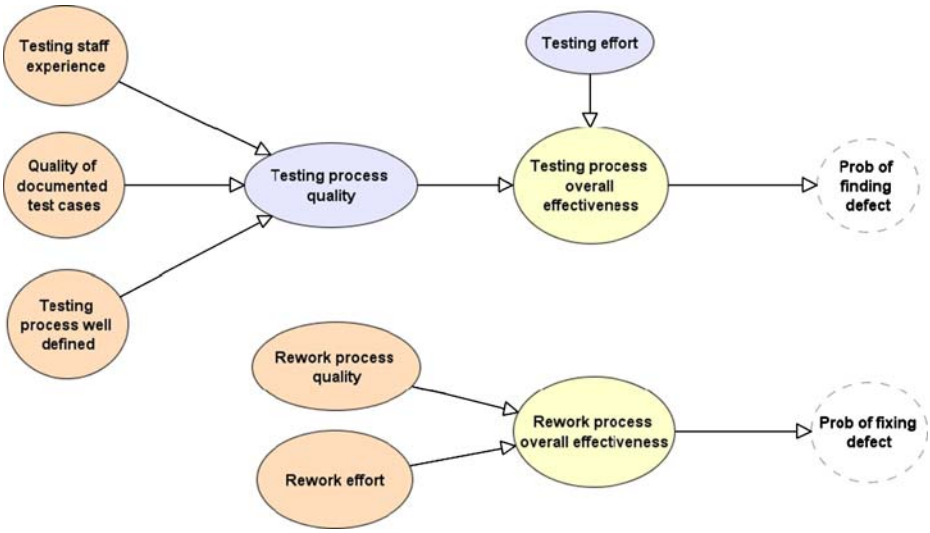

Fig. 6 Testing and rework subnet

3.2 Defect Insertion and Discovery Subnet

Figure 2 shows the core of this Bayesian network: the defect insertion and discovery subnet. This subnet contains various categories of defects which are sequentially used to calculate each other. The nodes with dashed edges are unobservable quantities whose values are predicted from:

- variables describing new functionality (Fig. 3),
- qualitative process factors (Figs. 4, 5, 6).

The variables with yellow and blue backgrounds represent different categories of defects. They are explained in Table 1. The variables with blue background have a special meaning. They are input and output nodes that can be used to link the model instances to reflect different development lifecycles. 'Residual defects post' reflects the number of

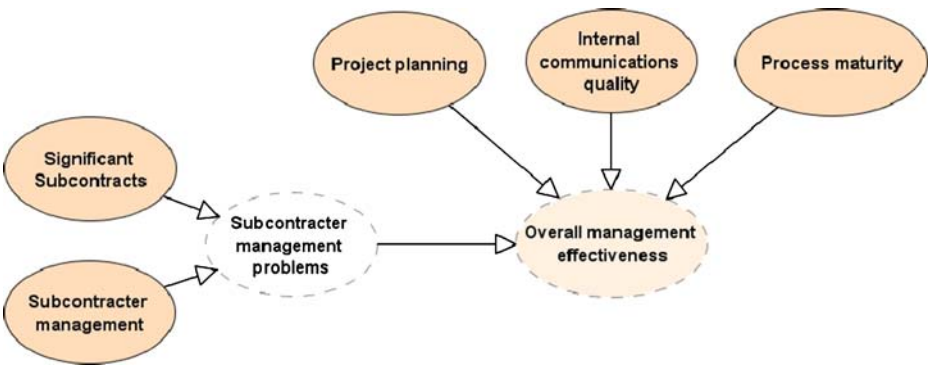

Fig. 7 Common influences subnet

Fig. 8 Existing code base subnet

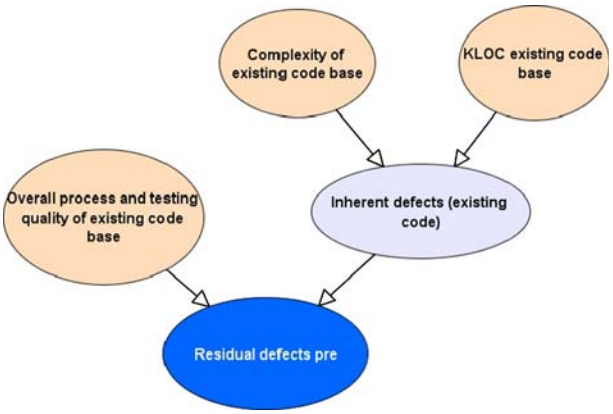

Table 1 Nodes in the defect insertion and discovery subnet

Symbol	Name	Description	Expression[a]
IPD	Inherent potential defects	The number of defects that are likely to occur in software of specific size	TNormal(min(10000, 30 * EKLOC$^{1.1}$), 100 * EKLOC, 0, 10000) *EKLOC: effective KLOC implemented this phase*
PDSDA	Potential defects given specification and documentation adequacy	The number of potential defects in software adjusted by the 'probability of avoiding specification defects' (estimated by the subnet described in Section 3.4)	Binomial (IPD, 1—P(ADS)) *P(ADS): probability of avoiding specification defects*
ND	New defects in	The number of defects inserted during design and development (coding) activities (Section 3.4)	Binomial (PDSDA, 1—P(ADD)) *P(ADD): probability of avoiding defect in development*
TD	Total defects in	The total number of defects existing in the whole code base including residual defects from the existing code base if such is used (Section 3.6)	ND+RDPre *RDPre: residual defects pre*
DT	Defects found in testing	The number of all project defects found during testing activity (Section 3.4)	Binomial (TD, P(DT)) *P(DT): probability of finding defect*
DF	Defects fixed	The number of all defects found during testing are fixed during rework (Section 3.4)	Binomial (DT, P(DF)) *P(DF): probability of fixing defect*
RDPost	Residual defects post	The number of defects left in software after its release	TD–DF

[a] The syntax for the distributions used here is: TNormal (mean, variance, lower bound, upper bound), Binomial(number of trials, probability of success)

defects remaining after a phase of development. It can be connected to the 'residual defects pre' of another instance of the model to reflect the number of defects existing before the next phase of development. Several examples on using the model in this way are presented in (Fenton et al. 2007b).

The expression for estimating the number of 'inherent potential defects' depending on the project size has been built based in part on C. Jones' data (Jones 1986; Jones 1999). The variability coded into this expression captures not only our uncertainty about the Jones' data but also the spread from the other reported data.

3.3 Scale of New Functionality Implemented Subnet

As in the other defect prediction models (Chulani and Boehm 1999; Compton and Withrow 1990; Gaffney 1984; Henry and Kafura 1884; Lipow 1982) the main factor influencing the number of defects in software is the size of the project. Since the "size" in KLOC of the project may not describe the project well enough our model adjusts the value of the project size by two ranked variables: 'complexity of new functionality', 'scale of distributed communication', and the Boolean variable: 'integration with third party software' (Fig. 3). As a result this subnet estimates the 'effective KLOC implemented this phase'.

3.4 Subnets for Development Activities

The model distinguishes three development activities:

- specification and documentation (Fig. 4) with the outcome 'probability of avoiding specification defects';
- design and development (Fig. 5) with the outcome 'probability of avoiding defect in development';
- testing and rework (Fig. 6) with the outcomes 'probability of finding defect' and 'probability of fixing defect'.

They all have a similar structure in the sense that qualitative factors are used in the same way:

- Process and people quality for each activity is established;
- The appropriate effort allocated to each activity is included.

Apart from the factors mentioned above, the specification and documentation subnet includes:

- 'overall management effectiveness' (the outcome from the 'Common influences' subnet);
- 'requirements creep'—reflecting the quality of requirements passed from the customer;
- 'quality of any previous documentation'—important for projects which are enhancements or re-developments.

The values for child nodes in these subnets are calculated using weighted expressions (Fenton et al. 2007a):

- weighted max—for 'requirements creep';
- weighted min—for 'spec and doc process effectiveness';
- weighted means—for all other child nodes.

The weights in these expressions have been estimated by experienced software managers. However, users can also add their own factors describing process and people quality as well as modify their importance by changing the weights.

Each of the development activity subnets ends with a numeric 'probability' node used in the defect insertion and discovery subnet (Fig. 2). The expressions for these 'probability' nodes are partitioned expressions where, for each state of the parent node (overall effectiveness of the specific activity), the Truncated Normal distribution is entered. For example, the expression for the probability of finding a defect for the testing when the testing process overall effectiveness is at its lowest value is defined as: TNormal(0.01, 0.001, 0, 1); since the mean here is 0.01 this says that typically we would expect to find only 1% of the defects when the testing process overall effectiveness is at its worst level. When the testing process overall effectiveness is at its highest level the distribution for defects found is defined as: TNormal(0.9, 0.001, 0, 1)—so this says that typically in the best case we would expect to find 90% of the defects. The values for the means in these distributions depending on the value of the parent node are presented in Table 2. The variance for these distributions was 0.001 in all 'probability' nodes except in 'probability of avoiding defect in development' where it was 0.005. Because of the range of possible probability values we defined truncation points of 0 and 1.

3.5 Common Influences Subnet

This subnet describes the quality of management (Fig. 7). The outcome from this subnet is the ranked variable: 'overall management effectiveness'. It does not influence any 'defects' variable directly. Rather, it influences the quality of development activities.

3.6 Existing Code Base Subnet

During the initial model validation we realized that predictions for some projects were heavily biased by the fact that an existing code base was used in these projects. We added a subnet that predicts the number of residual defects in this existing portion of code. Figure 8 illustrates the structure of this subnet.

The main factor in this subnet is the size of the existing code ('KLOC existing code base'). Adjusting this by the 'complexity of existing code base', the model estimates the

Table 2 The mean values in expressions of the 'probability' nodes

Ordinal value of activity effectiveness	Probability of avoiding specification defects	Probability of avoiding defect in development	Probability of finding defect	Probability of fixing defect
1 (worst)	0.0001	0.01	0.01	0.001
2	0.01	0.1	0.1	0.1
3	0.03	0.2	0.2	0.2
4	0.1	0.3	0.3	0.3
5	0.2	0.4	0.4	0.4
6	0.27	0.5	0.5	0.5
7	0.35	0.6	0.6	0.6
8	0.5	0.7	0.7	0.7
9	0.6	0.8	0.8	0.8
10 (best)	0.75	0.9	0.9	0.9

number of inherent defects in the existing code. This value is then adjusted by the 'overall process and testing quality of existing code base'. The latter is a ranked node with neither indicators nor parents, which could help users to estimate its value. However the users can add indicators to explain the overall process and testing quality.

4 Qualitative Factors

The first stage of developing the causal model outlined in Section 3, required partners in the MODIST Project (MODIST 2003) to identify qualitative factors that they believed had a significant influence on the outcome of a software project. Once the model had been built the second stage was to gather a dataset to validate the model; to do this effectively a more detailed description of each factor was needed. In Section 4.1 we describe the set of factors, together with the first level of detailed description. Section 4.2 gives an extract of the subsequent questionnaire given to project managers to gather data from completed projects. Section 4.3 discusses some issues arising from this method of measuring the qualitative project factors.

Although it was intended that the validation dataset would cover all the data used in the Bayesian network, this was not achieved; some project managers did not answer all the questions. A small number of variables were omitted altogether. This arose when the data recording practices of the project did not match the assumptions of the questionnaire: in particular rework was not distinguished from testing, and the concept of the 'effort' spent on a project phase, relative to what would be expected on average, was not used by project managers. Also the data about the code base existing prior to the main development cycle was not included in the dataset by our industrial partners. Fortunately, a Bayesian network handles missing data (see Section 7.3).

4.1 Factor Descriptions

For ease of understanding and presentation the factors are grouped under five topics: specification and documentation process (Table 3), new functionality (Table 4), design and development process (Table 5), testing and rework (Table 6) and finally project management (Table 7). This grouping also reflects the Bayesian Network structure. However, despite the grouping the factors are all be considered to be project attributes.

Each factor is named and described by a question to be answered. The descriptive questions were specifically tailored for the organisation providing the project data. These tables also show on which figures these factors are present or indicate that some of these factors are missing in this version of the model (but were present in the original MODIST model).

4.2 Questionnaire Design

Qualitative data are expressed on a five-point ordinal scale. The ordinal values used are: Very High, High, Medium, Low, Very Low. The data values were gathered using a questionnaire, which was completed by the project manager, project quality manger or other senior project staff. Each questionnaire item consists of:

- More detailed questions,
- An interpretation of the ordinal scale.

Table 3 Specification and documentation process

	Factor name	Descriptive question	Figure reference
S1	Relevant experience of spec and doc staff	How would you rate the experience and skill set of your team members for executing this project during the requirements and specifications phase?	Fig. 4
S2	Quality of documentation inspected	How would you rate the quality of the requirements given by the client or other groups?	Fig. 4 as 'Quality of any previous documentation'
S3	Regularity of spec and doc reviews	Have all the Requirements, Design Documents and Test Specifications been reviewed in the project?	Fig. 4
S4	Standard procedures followed	In your opinion, how effective was the review procedure?	Fig. 4
S5	Review process effectiveness	What was the review effectiveness in the project for the requirements phase?	–
S6	Spec defects discovered in review	In your opinion, is the defect density of spec reviews on the high side?	–
S7	Requirements stability	How stable were the requirements in your project?	Fig. 4

For example, for factor S1 'Relevant Experience of Spec and Doc Staff', the additional questions are:

1. Did the Requirements team have adequate experience in analysing and generating requirements?
2. Did the Requirements team have adequate domain expertise?

and the ordinal scale points are:

Very High: Software engineers with greater than 3 year's experience in requirements management, and with extensive domain knowledge.
High: Software engineers with greater than 3 year's experience in requirements management, but with limited domain knowledge.
Medium: Software engineers having between 1 and 3 year's experience in requirements management.

Table 4 New functionality

	Factor name	Descriptive question	Figure reference
F1	Complexity of new functionality	What was the complexity of the new development or new features that happened in your project?	Fig. 3
F2	Scale of new functionality implemented	How large was the extent of working on new functionality rather than just enhancing the older functionalities in your project?	–
F3	Total no. of inputs and outputs	For your product domain, would you rate the total no of outputs/inputs (newly developed/ enhanced) as high?	–

Table 5 Design and development process

	Factor name	Descriptive question	Figure reference
D1	Relevant development staff experience	How would you rate the experience and skill set of your team members for executing this project during the design and development phase?	Fig. 5
D2	Programmer capability	On an average, how would you assess the Quality of code produced by the team members?	Fig. 5
D3	Defined processes followed	What was the review effectiveness in the project for the Design and Development phase?	Fig. 5
D4	Development staff motivation	What is your opinion about the motivation levels of your team members?	Fig. 5

Low: Software engineers having between 1 and 3 year's experience, but with no experience in requirements management.
Very Low: Software engineers with less than 1 year's experience, and with no previous domain experience.

In some cases the questionnaire used a set of criteria and a score. An example is the factor S4 'Standard (Review) Procedures Followed'. The detailed questions, giving the criteria, are:

1. In case of changes after baselining, have the major changes been re-reviewed?
2. Are there any re-review triggers/criteria defined?
3. Have some domain specific standards been adhered to (like design rules, re-engineering guidelines, architectural guidelines, etc)?
4. Was the requirements document checked for review worthiness or pre-review checklist filled before the review?
5. Have the reviews been planned upfront?
6. Have the reviewers been assigned upfront?
7. Were the reviews role-based?
8. Were the reviewers identified appropriate and experienced enough for reviewing?
9. Was there adequate preparation time available for the reviewers?
10. Were there overview sessions for all complex work products?

Table 6 Testing and rework

	Factor name	Descriptive question	Figure reference
T1	Testing process well defined	How effective was the testing process adopted by your project?	Fig. 6
T2	Staff experience—unit test	What was the level of software test competence of those performing the unit test?	Fig. 6 as'Testing staff experience'
T3	Staff experience—independent test	How would you rate the experience and skill set of the independent test engineers (Integration, functional or sub-system testing, Alpha, Beta)?	Fig. 6 as'Testing staff experience'
T4	Quality of documented test cases	What was the extent of the defects that were found using formal testing against the intuitive/random testing?	Fig. 6

Table 7 Project management

	Factor name	Descriptive question	Figure reference
P1	Dev. staff training quality	What is the coverage of the identified project/ process related trainings as well as trainings identified as per the roles, by the team members?	Fig. 5
P2	Configuration management	How effective is the project's document management and configuration management?	–
P3	Project planning	Has the project planning been done adequately?	Fig. 7
P4	Scale of distributed communication	How many sites/groups were involved in the project.	Fig. 3
P5	Stakeholder involvement	To what extent were the key project stakeholders involved?	Fig. 4
P6	Customer involvement	How good was customer interaction in the project?	–
P7	Vendor management	How would you rate the Vendor/Sub-contractor Management (if applicable)?	Fig. 7 as 'Subcontractor management' and 'Significant subcontracts'
P8	Internal communication/ interaction	How would you the rate the quality of internal interactions/communication within the team?	Fig. 7 as 'Internal Communications Quality'
P9	Process maturity	What's your opinion about process maturity in the project?	Fig. 7

The scale point is then derived as follows:

Very High: All the ten subquestions answered 'yes',
High: 7–9 of the subquestions answered 'yes',
Medium: 5–6 of the subquestions answered 'yes',
Low: 4 of the subquestions answered 'yes',
Very Low: less than 4 of the subquestions answered 'yes'.

4.3 Measurement Issues

The factors used in the model were originally identified by a group of project managers from different partners in the MODIST project. Although from different organisations, it was possible for the project managers to agree on the importance of factors such as 'Requirements Stability'. A further issue is whether it is possible to measure such values consistently between organisations.

As shown by the example in Section 4.2, we designed the questionnaire to use objective criteria, such as the number of years of experience, whenever possible. However, we do not claim external validity of these measurements, since this is not needed for our approach, as we explain below.

One way experts were used in building the model was to estimate the 'strength' of the effect of each qualitative factor in the causal model. This information is represented in the conditional probability table for each node in the Bayesian network. As a result of this process, the model is applicable within the organisation where the experts have gained their

experience. Since the model itself is not universal, there is no need for the measurements to be so. It does not follow that the scope of the validation (see Section 8) becomes trivial, even tautological, as a result. Instead, the validation shows that a model constructed using expert judgement and historical data, within one organisation, can be used within the same organisation to predict accurately the outcome of new projects. On the other hand, the validation described here does not consider issues such as the external validity of the causal structure of our model (see Section 7.4).

5 Quantitative Data

The projects developed software embedded in consumer electronics products. Each project developed or enhanced some functionality provided by a product. The developed software was not stand-alone but was integrated with other software subsystems in the product.

A waterfall lifecycle was followed. The software engineering part of the lifecycle covered a specification review, design, a design review and development up to unit testing. The software was then passed to independent test in several phases, from software integration testing to overall system (i.e. product) testing.

Most of the software development was at one site, but the overall development was distributed over different locations in a global organisation. Both the software specification and the independent testing were typically at a different location to the software development.

The data values are shown in Table 8:

- Software size: the size, in KLoC of the developed code and the development language (Fig. 9 shows the distribution of code size in the dataset). Note that for two projects, this data was not available: the Bayesian network can still be used and it will assume the projects to be 'medium' but of uncertain size.
- Effort: development effort measured in person hours for the software development, from specification review to unit test.
- Defects: functional defects discovered during all the independent testing phases, following the software development.

In some projects existing software was reused as part of the development. The impact of this on the dataset is considered in Section 7.

This new dataset could, of course, be used to build traditional statistical/regression based models, as indicated, for example, by Fig. 10. This could be the basis for a simple regression model relating KLoC to defects; indeed the correlation coefficient here is quite high (0.78). However, this does not correspond to the way that we used this data, which was to validate a model created before the data was gathered. Therefore, we do not pursue this comparison.

6 Qualitative Data

The data values are shown in Tables 9, 10, 11, 12. Missing data values are marked with '–' (see Section 7.3). The letters VL, L, M, H, VH correspond to the ordinal scale described in Section 4.2 ('very low' to 'very high').

Table 8 Size, effort and defects

Project	Hours	KLoC	Language	Defects
1	7,109	6.0	C	148
2	1,308	0.9	C	31
3	18,170	53.9	C	209
4	7,006	–	C	228
5	9,434	14.0	C	373
6	9,441	14.0	C	167
7	13,888	21.0	C	204
8	8,822	5.8	C	53
9	2,192	2.5	VC++,MFC	17
10	4,410	4.8	C	29
11	14,196	4.4	C	71
12	13,388	19.0	C	90
13	25,450	49.1	C	129
14	33,472	58.3	C	672
15	34,893	154.0	C	1,768
16	7,121	26.7	C	109
17	13,680	33.0	C	688
18	32,366	155.2	C	1,906
19	12,388	87.0	C	476
20	52,660	50.0	C	928
21	18,748	22.0	C	196
22	28,206	44.0	C	184
23	53,995	61.0	C	680
24	24,895	99.0	C	1,597
25	6,906	23.0	C	546
26	1,642	–	C	261
27	14,602	52.0	C	412
28	8,581	36.0	C	881
29	3,764	11.0	C	91
30	1,976	1.0	C	5
31	15,691	33.0	C	653

7 Issues Arising from the Data Collection

The complexity of software projects makes gathering data a challenge. The most important challenges we faced and lessons we learned during this work are described below. Some of

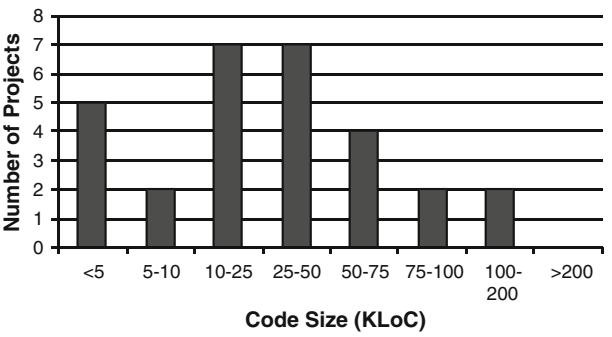

Fig. 9 Code size distribution

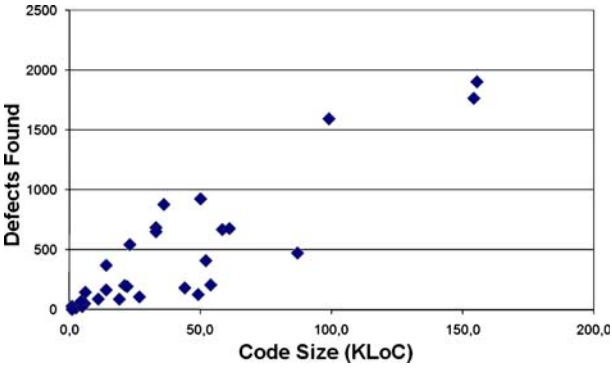

Fig. 10 Code size versus defects

Table 9 Specification and documentation process data

Project	S1	S2	S3	S4	S5	S6	S7
1	H	M	VH	H	M	H	L
2	H	H	VH	H	M	H	H
3	H	H	VH	H	H	VH	H
4	L	L	M	L	L	L	L
5	H	M	H	M	H	–	M
6	VH	M	VH	M	H	–	H
7	L	M	VH	H	H	L	M
8	M	M	H	M	H	L	H
9	H	VH	VH	H	VH	M	VH
10	H	H	H	M	H	M	H
11	H	M	H	M	H	H	H
12	H	M	H	M	M	M	L
13	VH	M	M	L	M	H	L
14	H	H	H	H	H	H	H
15	H	H	H	H	H	VH	VL
16	H	H	H	H	H	H	M
17	VH	H	M	L	H	H	M
18	M	H	H	H	H	VH	VL
19	H	M	H	H	H	H	M
20	L	L	M	VL	L	M	VL
21	H	H	H	M	L	M	M
22	L	L	M	M	M	M	L
23	M	H	VH	H	L	M	M
24	M	M	M	H	M	H	L
25	M	H	–	H	M	M	M
26	M	M	H	M	H	H	H
27	H	M	VH	M	M	VH	M
28	H	L	VH	M	M	M	L
29	H	M	VH	H	M	M	VH
30	H	H	VH	H	H	M	VH
31	–	H	H	M	M	H	M

Table 10 Data for new functionality, design and development process

Project	F1	F2	F3	D1	D2	D3	D4
1	M	L	M	L	H	H	H
2	L	VL	M	L	H	H	H
3	H	H	VH	H	VH	H	VH
4	M	L	M	L	M	L	M
5	H	H	VH	L	M	H	H
6	M	M	VH	M	H	M	M
7	L	VL	M	M	VH	H	H
8	M	L	M	H	H	M	M
9	L	L	M	H	VH	VH	H
10	M	L	M	H	H	H	H
11	H	H	H	H	H	H	H
12	H	H	H	VH	M	M	H
13	H	H	H	H	H	H	H
14	VH	H	H	H	H	H	H
15	H	H	M	H	H	H	H
16	L	VL	M	H	H	H	H
17	L	VL	M	M	M	H	H
18	VH	VH	H	M	H	H	H
19	H	H	H	H	H	H	H
20	VH	H	VH	VL	VL	L	H
21	L	M	VH	H	H	H	H
22	M	M	VH	H	M	L	H
23	H	VH	VH	L	H	H	H
24	M	M	H	M	H	H	M
25	H	VL	H	M	H	M	H
26	M	H	M	L	M	M	M
27	H	VH	VH	M	L	M	H
28	VH	VH	VH	M	L	H	H
29	M	M	H	VH	VH	H	H
30	L	L	M	H	H	H	H
31	M	M	H	H	H	H	H

these challenges are not fully resolved by the data included in this dataset; how these issues were addressed in our models is described elsewhere (Fenton et al. 2007b).

7.1 Software Size: Intrinsic Complexity

Because of the need to have a size based measure based only on the amount of functionality to be implemented, we had hoped to use function points as the key metric for this purpose, as recommended from the MODIST work. The model was developed before gathering the data but then it was modified when we realized that function points were not being used by the software development organisation providing this data. It is well known (Fenton and Pfleeger 1998) that KLoC measures program length but the length of the program is only one aspect of the size of the development task. There is an additional measure which we term the ‘intrinsic complexity’. The factors F1–F3 were included in the data gathered to give a better estimate of the intrinsic complexity than code size alone. Unfortunately, intrinsic complexity is not an observable quantity, so finding sufficient factors to estimate the size of the development task remains a challenge.

Table 11 Testing and rework data

Project	T1	T2	T3	T4
1	M	H	L	H
2	H	H	L	H
3	H	H	H	H
4	VL	VL	VL	L
5	M	M	L	M
6	H	–	M	M
7	H	M	M	H
8	H	M	M	M
9	H	VH	VH	H
10	H	M	M	M
11	H	H	M	M
12	H	H	M	M
13	M	M	L	M
14	H	H	H	H
15	M	H	M	M
16	M	H	M	M
17	M	L	L	H
18	H	H	M	M
19	H	M	M	H
20	VL	VL	VH	H
21	H	H	H	H
22	H	M	M	H
23	H	H	H	H
24	H	M	M	M
25	VL	M	H	L
26	M	L	H	M
27	M	M	M	M
28	M	M	M	M
29	H	VH	VH	H
30	H	H	H	H
31	M	H	M	M

7.2 Code Reuse

It is very common for software development to be carried out as part of a product line development, naturally giving rise to software reuse. This complicates the measurement of software size—the lines of developed software differs from the length of the developed program—and also impacts the prediction of defects, since the quality of the reused software is variable. The original MODIST model did not take into account the code reuse. After one of the validation rounds we added the 'existing code base' subnet to the model for the final validation round (Section 8).

7.3 Missing Data Values

Given the complexity of a dataset that attempts to cover relevant software cost and quality drivers, it is inevitable that some data values will be missing. It is essential that software prediction methods are able to cope with missing data values.

The Bayesian net model used in this study is one such method that handles missing data, since the model includes prior probability distributions for all the project data.

Table 12 Project management data

Project	P1	P2	P3	P4	P5	P6	P7	P8	P9
1	VH	H	H	L	H	M	–	VH	H
2	VH	H	H	L	H	M	–	VH	H
3	H	VH	H	–	VH	VH	–	VH	VH
4	L	M	VL	L	M	M	M	H	M
5	H	H	H	M	M	H	L	VH	M
6	H	H	H	M	M	VH	L	VH	H
7	H	H	VH	VL	VH	VH	–	H	VH
8	M	H	H	VL	H	H	–	H	H
9	VH	VH	VH	L	VH	VH	–	VH	VH
10	H	H	H	VL	H	H	–	M	H
11	H	H	H	VL	H	H	–	M	M
12	H	H	H	L	H	H	–	M	H
13	M	H	H	VL	H	M	H	M	M
14	H	H	H	–	H	H	–	H	H
15	VH	M	H	M	VH	VH	–	VH	H
16	VH	M	H	M	VH	VH	–	VH	H
17	M	M	M	M	M	H	–	H	M
18	VH	M	H	H	VH	VH	–	VH	H
19	M	H	H	L	H	H	–	H	H
20	H	M	L	H	H	M	–	H	H
21	H	H	H	H	H	H	–	H	H
22	H	H	M	H	H	H	–	H	H
23	H	H	H	H	H	M	–	H	H
24	H	H	M	L	M	H	–	VH	H
25	M	M	M	M	M	M	M	H	H
26	L	M	M	L	H	H	L	H	M
27	H	M	L	L	M	H	H	H	M
28	H	M	L	L	M	M	–	H	M
29	M	H	H	L	VH	VH	–	H	H
30	M	H	H	L	VH	VH	–	H	H
31	H	H	H	H	VH	VH	–	VH	VH

7.4 Generality of the Data

An objective of the partners in the MODIST project was to identify only factors (and the means of measuring them) that were generally relevant to complex software projects. Achieving this objective would enable different organizations to make use of the causal model. We recognize that the more detailed descriptions and questionnaires refer to process-specific information. The objective of generality would still be partly achieved if other organizations (using different processes) used the same factors, but adapted the questionnaire as a means of measuring them.

8 Model Validation

8.1 Evaluation Measures

To validate the model which we developed using the dataset provided in Sections 5 and 6 we used the following evaluation measures which are commonly used and suggested to

validate such types of models (Chulani et al. 1999; Fenton et al. 2007b; Kitchenham et al. 2001; Stensrud et al. 2002):

1. Coefficient of determination (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \bar{y}_i\right)^2}$$

2. Mean Magnitude of Relative Error (MMRE)

$$\text{MMRE} = \frac{1}{n} \sum_{i=1}^{n} \text{MRE}_i$$

$$\text{MRE}_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

3. Median Magnitude of Relative Error (MdMRE)

$$\text{MdMRE} = \text{Median}(\text{MRE}_i)$$

4. Balanced Mean Magnitude of Relative Error (BMMRE)

$$\text{BMMRE}_i = \frac{1}{n} \sum_{i=1}^{n} \frac{\left|y_i - \hat{y}_i\right|}{\text{Min}\left(y_i, \hat{y}_i\right)}$$

5. Mean Magnitude of Relative Error relative to the Estimate (MEMRE)

$$\text{MEMRE} = \frac{1}{n} \sum_{i=1}^{n} \text{EMRE}_i$$

$$\text{EMRE}_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

6. Median Magnitude of Relative Error relative to the estimate (MdEMRE)

$$\text{MdEMRE} = \text{Median}(\text{EMRE}_i)$$

7. Prediction at level l—measures the fraction of observations for which predictions are within $\pm l$ percent of actuals.

$$\text{Pred}(l) = \frac{1}{n} \sum_{i=1}^{n} a_i$$

$$a_i = \begin{cases} 1 & \text{if} \quad \text{MRE}_i \leq l \\ 0 & \text{if} \quad \text{MRE}_i > l \end{cases}$$

8.2 Validation Results

We validated the causal model using the presented project dataset. We did this by entering, for each project, data excluding the defect data and ran the Bayesian net model. This produces a (predicted) probability distribution for number of defects found in independent testing. Using the median values of these distributions enables us to calculate the accuracy of the predictions. Table 13 illustrates the actual and predicted number of defects.

Table 13 Actual and predicted number of defects

Project	Actual defects	Defects predicted
1	148	75
2	31	52
3	209	254
4	228	355
5	373	349
6	167	123
7	204	262
8	53	48
9	17	57
10	29	203
11	71	51
12	90	347
13	129	516
14	672	674
15	1,768	1,526
16	109	145
17	688	444
18	1,906	1,886
19	476	581
20	928	986
21	196	259
22	184	501
23	680	722
24	1,597	1,514
25	546	641
26	261	407
27	412	430
28	881	721
29	91	116
30	5	46
31	653	505

As presented in Fig. 11, we achieved a very high accuracy measured with an R^2=0.9311. We can observe the greater variation between the actual and predicted defects for generally smaller projects which overall have smaller numbers of defects.

The R^2 value is vulnerable to outliers—three projects with number of defects >1,500 clearly cause the increase in the value of R^2. That is why we decided to calculate other measures reflecting the prediction accuracy. Apart from calculating these measures for the whole dataset we also analyzed the prediction accuracy of the model depending on the size of the project: small, medium and large projects. The values for validation measures are listed in Table 14.

We can observe that the predictive accuracy of the model, expressed by the different measures presented in Table 14, increases with the size of the project and achieves highly desirable values in such types of models:

- the measures based on relative error (MMRE, MdMRE, BMMRE, MMER, MdMER) decrease significantly, as project size increases;
- Pred with different *l* levels increases (with one exception for Pred10 for medium-sized projects).

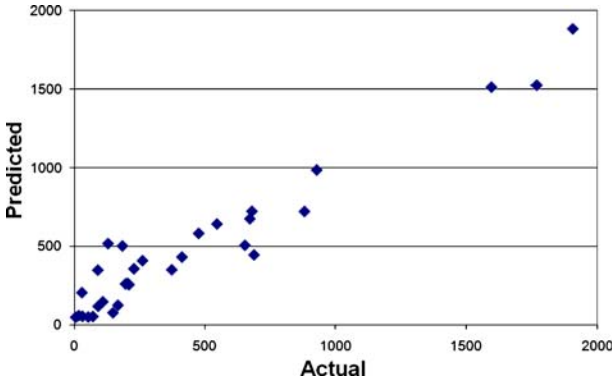

Fig. 11 Predicted and actual values

The lower prediction accuracy for smaller projects can be explained by the fact that the prior knowledge which we incorporated into the model based on expert opinions from the partner companies did not involve smaller projects. The validation using the data provided in previous sections shows that using the model with the data outside the original model scope does not ensure accurate predictions.

We can observe a high value for MMRE compared to other measures based on the magnitude of relative error both for the dataset as a whole and the subsets depending on the project size. This was due to the fact that in six projects the magnitude of relative error was >1 (the absolute prediction error was higher than the actual value)—even several times higher than typical relative errors in other projects. These six projects caused the shift in MMRE but not so much in other measures based on relative error, which were all of a similar level.

These satisfactory results from the model validation gives confidence in the value of the causal model, but of course further validation using additional datasets would provide even greater confidence in the integrity and robustness of the model. Moreover, the validation we have described does not do full justice to the benefits of a Bayesian net model. For example, in the Bayesian net model it is possible to enter data at any of the variables and obtain the probability distributions at any of the unobserved variables. We will expand on this flexibility of use in the next section.

Table 14 Values of model evaluation measures

Evaluation measures	Dataset for validation			
	Projects <10 KLoC (n=7)	Projects ≥10 and < 50 KLoC (n=13)	Projects ≥50 KLoC (n=9)	All projects (n=31)
R^2	0.003	0.523	0.984	0.931
MMRE	2.55	0.77	0.09	0.96
MdMRE	0.49	0.28	0.06	0.27
BMMRE	0.53	0.33	0.08	0.30
MMER	0.62	0.36	0.08	0.34
MdMER	0.70	0.25	0.06	0.24
Pred30	0.29	0.54	1	0.58
Pred10	0.14	0.08	0.67	0.26

9 Decision Support for Software Manager

A clear benefit of a Bayesian net model, compared for example to a more traditional statistical model derived purely from regression analysis, is its provision of a range of decision-support and risk assessment capabilities. These are potentially extremely valuable to project managers. We illustrate this with two example scenarios in this section.

9.1 Example 1

In the first scenario we assume that the software company is about to deliver software with predicted size of 100 KLoC of 'medium' complexity. Without entering any other data as inputs to the model we get a prediction of 1438 (median) residual defects. Let us further assume that there will be no special allowances for testing and reworking on this project and so we set the value for 'Testing process overall effectiveness' and the 'Rework process overall effectiveness' as 'medium'. The revised prediction for number of residual defects is 1709 (median). The software manager might decide that this is far too high and that the maximum number of residual defects that can be tolerated is 200. Then in the BN model we can enter the value 200 in the residual defects node. Since the BN is able to propagate evidence backwards as well as forwards we can compare the predictions for unknown variables such as development process effort and development process quality. Figure 12 illustrates the resulting probability distributions for the two scenarios (one where we do not incorporate the constraint on defects and the other where we do). What the model is saying is that we are unlikely to achieve the low defect target unless both the development process effort and quality are significantly higher than normal.

The model allows managers to perform various types of what-if-analyses and trade-offs. For example, suppose resource constraints make it impossible to increase development effort. If we enter 'medium' effort then the distribution for development process quality shifts even further towards very high (Fig. 13) making it clear that, in the absence of other information it is extremely unlikely we will meet the target unless there is some drastic overall improvement to the development process compared to previous projects.

9.2 Example 2

In this example we again have to develop 100 KLoC of medium complexity. Let us assume that this is a completely new project (no existing code base and thus no residual defects in the existing code base). The prediction for number of residual defects given only this

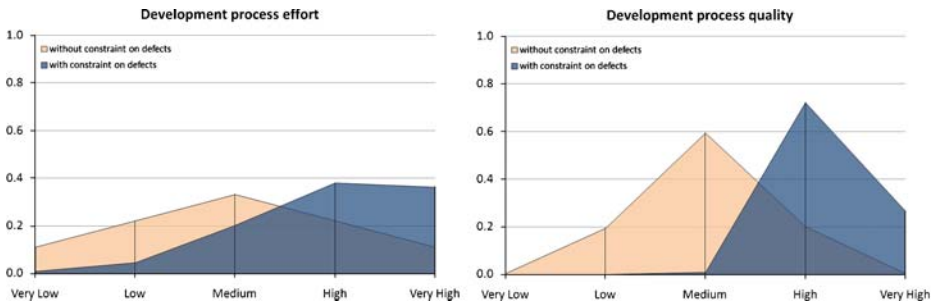

Fig. 12 Revised prediction for development effort and process quality with a product quality constraint

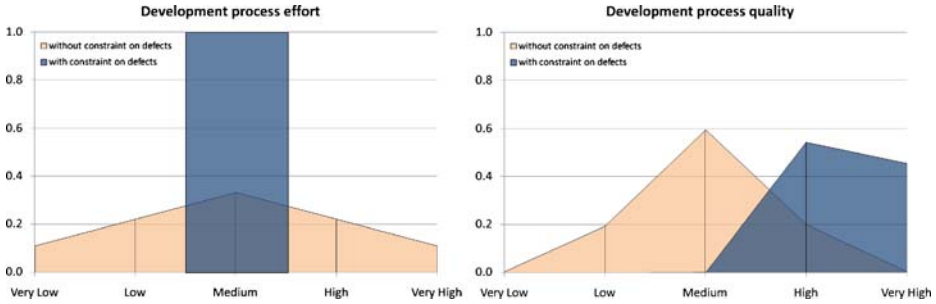

Fig. 13 Revised prediction for development process quality with an effort constraint

information is 1497 (median). But this time we want to analyze the impact of differences in various aspects of process quality among development activities. Assuming that every time we allocate appropriate effort for a specific activity (further assuming we know from past projects what is the appropriate effort allocation) and 'requirements creep' is fixed ('Medium') we analyze the impact of process quality in development activities in various combinations. The results are presented in Table 15.

Based on the predicted results we can observe that:

- As expected, increasing the process quality of any activity increases the quality of the software (number of residual defects decreases)—e.g. by comparing scenario 1 and 2, 3 and 4 etc;
- When the process quality is 'low' only in one of the activities and 'high' in the others (scenarios 4, 6, 7), the best is the scenario with low "specification and documentation process quality' (no. 4) and the worst is the scenario with low "development process quality" (no. 6);
- When the process quality is 'high' only in one of the activities and 'low' in the others (scenarios 2, 3, 5), the best is the scenario with high "development process quality" (no. 3) and the worst is the scenario with high "specification and documentation process quality" (no. 5).

These observations provide useful insights into where to prioritise effort if resources are constrained.

Table 15 Predictions for number of defects for various combinations of process quality

Scenario No.	Entered observations			Predicted numberof residual defects	
	Specification and documentation process quality	Development process quality	Testing and rework process quality	Mean	Median
1.	Low	Low	Low	2,185	2,210
2.	Low	Low	High	1,668	1,663
3.	Low	High	Low	1,326	1,284
4.	Low	High	High	1,020	980
5.	High	Low	Low	2,101	2,101
6.	High	Low	High	1,605	1,593
7.	High	High	Low	1,248	1,188
8.	High	High	High	961	914

10 Sensitivity Analysis

In this section we perform two levels of sensitivity analysis. The first (Section 10.1) is a basic analysis that can be viewed as an extension of the approach in the example of Section 9.2. The second (Section 10.2) is a deeper 'global sensitivity analysis' in which the sensitivity estimates of specific factors are evaluated incorporating changes in all other factors and which also incorporate the probability distributions associated with the variables. In Section 10.3 we discuss the overall impact of the sensitivity analysis results.

10.1 Basic Sensitivity Analysis

In the basic analysis we analyze the impact on the key model "output" variable (namely the number of residual defects) of one or two model input variables at a time in different combinations. The aim of this analysis is to visualize the impact of the model inputs on the model output in the selected scenarios.

10.1.1 Case 1

In this case (Fig. 14) we analyze the impact of different combinations of two factors on number of residual defects:

- (Left hand side) *project size* (KLOC) with *overall process effectiveness* (ranked scale from 'very low' (VL) to 'very high' (VH)),

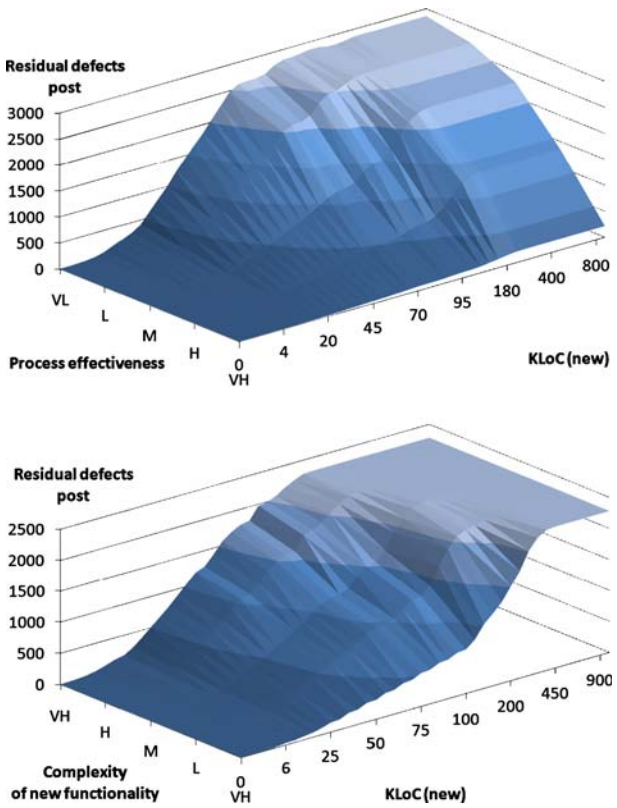

Fig. 14 Impact of project size, process effectiveness and project complexity

- (Right hand side) *project size* (KLOC) with *complexity of new functionality* (ranked scale from 'very low' (VL) to 'very high' (VH)).

To simplify the analysis we assume that no code will be reused and all other factors are kept constant. Specifically:

- Factors measured on a ranked scale are set to 'Medium',
- Boolean factors are set to 'No'.

Figure 14 illustrates the results for this example. Note that, since the model predicts the number of residual defects as a probability distribution, we use the median value of the predicted distribution. We can observe that both process effectiveness and project complexity cause increasing variation in predicted number of residual defects as the size of the project increases.

We can also observe that when process effectiveness is 'very high' the number of defects increases much more slowly with project size than when process effectiveness is 'very low'. The similar relationship cannot be observed between project complexity and project size. This may suggest that project complexity tends to cause lower variation in the number of defects than process effectiveness.

We found that the model becomes less sensitive to changes in project size and other factors influencing the 'effective KLoC implemented this phase' (Fig. 3) for larger projects. The limit in project size is around 150 KLoC for the most complex projects and around 400 KLoC for the least complex. This can be observed on Fig. 14 as a flatter area in which the predicted number of residual defects does not increase with an increase in project size. These limits arise from the original expert-elicitation sessions; specifically projects beyond that range had never been considered (primarily because the experts were focused on knowledge of components or subsystems). We observed similar insensitivity to project size increases in subsequent scenarios.

10.1.2 Case 2

Here we analyze the impact of project complexity, overall process effectiveness and factors describing requirements quality. We consider four scenarios depending on the project size: 10, 30, 70 and 150 KLoC. From Fig. 15, we can observe that the variation in process effectiveness indeed causes higher variation in the number of residual defects than the variation of project complexity or requirements quality. The variation of project complexity seems to cause increasing variation in the number of defects together with decreasing process effectiveness. The requirements quality does not follow such behaviour. Although Fig. 15 illustrates predictions for constant project size (KLoC new=70) we also observed similar relationships in scenarios with different project sizes.

10.1.3 Case 3

Here we analyze the impact of each activity's effectiveness on the number of residual defects. The results in Fig. 16 suggest that we can conclude that the development effectiveness (coding) has the most important impact. Testing and rework effectiveness is only slightly less important. Much less important appears to be specification and documentation effectiveness. We observed similar relations in scenarios with other values of project size.

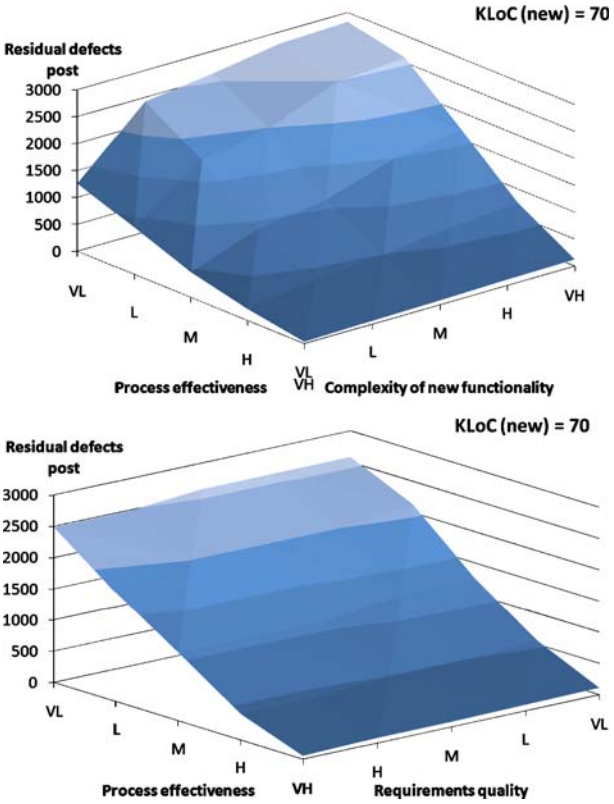

Fig. 15 Impact of overall process effectiveness, project complexity and requirements quality

10.1.4 Case 4

Here we analysed the impact in predicted number of residual defects of all qualitative factors independently. Each single factor is changed from its lowest to highest value while all others are kept constant (set as their most probable value). We assume that there is a project to be developed which will reuse 3 KLoC from previous projects. We performed this analysis in four scenarios of varying project size: 10, 30, 70 and 150 KLoC of new functionality. The results for the case KLoC=10 are illustrated in the Tornado graph shown in Fig. 17 (the overall relative sensitivity of the factors observed for the other project size scenarios were similar). The graph plots the range of predicted residual defects resulting from changes in each factor. For example, for the factor 'complexity of new functionality' the residual defects ranges from 75 (when complexity of new functionality is at its lowest) to 195 (when complexity of new functionality is at its highest)

Clearly there are three factors significantly more important than the rest:

- complexity of new functionality,
- scale of distributed communication,
- integration with third party software—the actual variation in number of defects caused by change in this variable ranged from 132 to 529 (the high boundary is truncated on the graph to improve its overall clarity).

It is also worth noting the influence of the last two factors in Fig. 17, namely 'complexity of existing code base' and 'overall process and testing quality of existing code

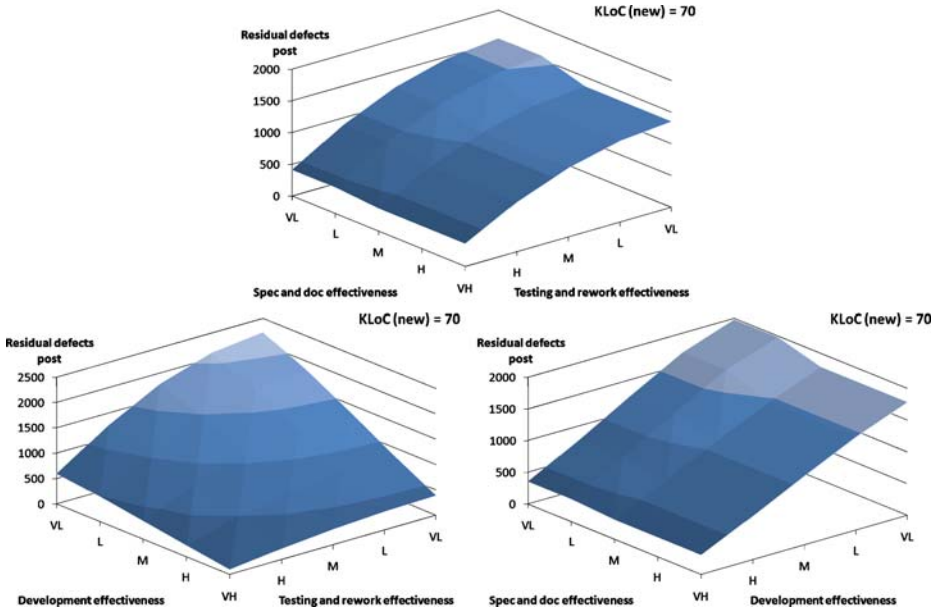

Fig. 16 Comparison of impact of different development activities effectiveness

base'. It turns out that the influence of these factors increased with the increase in the proportion of size of reused code to the size of the new code. So, not surprisingly, the model suggests that the more code (proportionally) we reuse the more important it becomes as to how this existing code was developed.

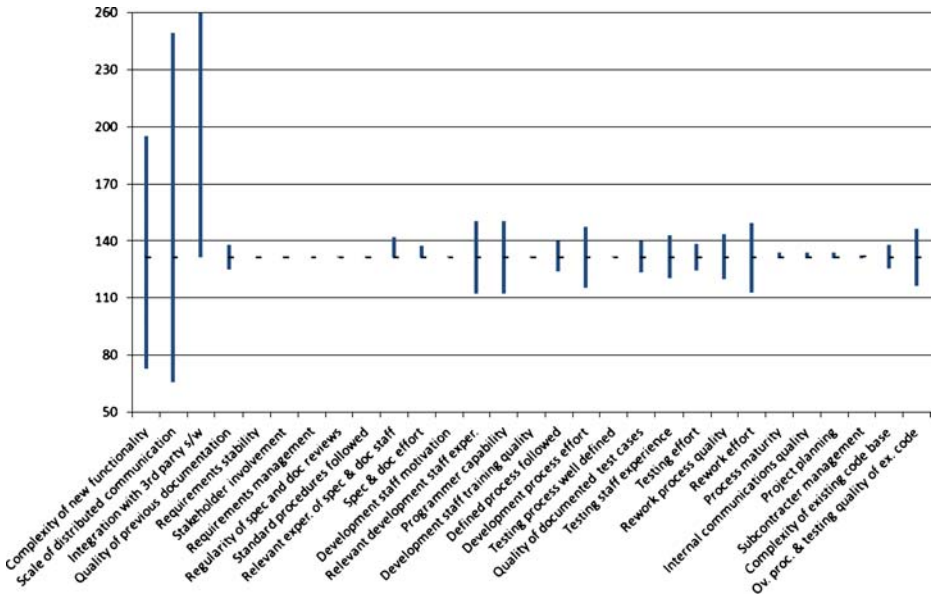

Fig. 17 Impact of each qualitative factor

10.2 Global Sensitivity Analysis

In the previous subsection we analyzed the change of one or two input variables at a time and we did not make any use of the probability distributions associated with the variables. Now we perform a deeper 'global sensitivity analysis' in which the sensitivity estimates of specific factors are evaluated incorporating changes in all other factors and which also incorporate the probability distributions associated with the variables (Saltelli 2000; Wagner 2007b). Such global sensitivity analysis has been previously used in analyzing other software engineering models (Cangussu et al. 2003; Musilek et al. 2002; Wagner 2007a; Wagner 2007b).

To perform this analysis we first generated 3000 random sample data based on the prior probability distributions of the nodes without parents in the model (such nodes may be thought of as the 'true' inputs in our model). The number 3000 provides a reasonable balance between coverage and calculation time efficiency.

Next, we analyzed visually the relationship between the input variables and the output variable. For example, Fig. 18 illustrates the relationship between project size (KLoC new) and the number of residual defects. This relationship closely matches a number of empirically reported studies (Fenton and Neil 1999) showing that, despite the positive correlation, other factors can minimise its predictive impact

Finally, we used the SimLab tool (SimLab 2004) to calculate and analyze three numerical measures of the global sensitivity:

- Spearman's rank correlation coefficient—measuring the degree of correlation between each input variable and the output variable. We did not use the more popular Pearson product-moment correlation coefficient because most of our input variables were ranked scale variables rather than numeric. For the same reason the other measures were calculated on ranks rather than on exact values.
- Standardised rank correlation coefficient—measuring the effect of varying each input variable away from its mean (rank) by a fixed fraction of its variance, while maintaining all other variables at their expected values (rank).
- Partial rank correlation coefficient—measuring the strength of correlation between an input factor and an output factor with any effect of possible correlation of this particular input factor with other input factors removed.

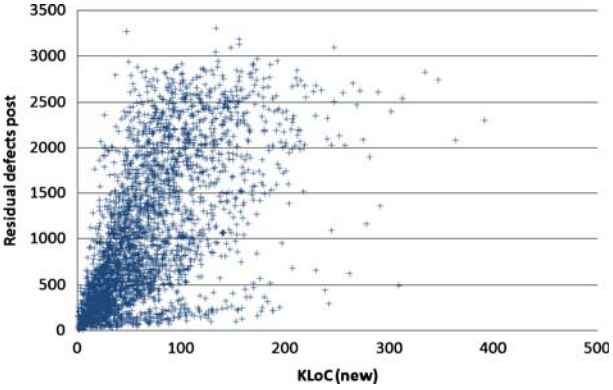

Fig. 18 Relationship between estimated project size and number of residual defects

The results of this analysis are presented in Table 16, with the six most influential factors on the number of residual defects being:

- KLoC (new);
- scale of distributed communication;
- complexity of new functionality;
- KLOC existing code base;
- testing staff experience;
- rework effort.

Table 16 Results of global sensitivity analysis

Input variable	Sensitivity measure					
	Spearman's rank correlation coefficient		Standardised rank correlation coefficient		Partial rank correlation coefficient	
KLOC (new)	0.670[a]	*1*	0.663[a]	*1*	0.853	*1*
Complexity of new functionality	0.353[a]	*3*	0.339[a]	*3*	0.640	*3*
Scale of distributed communication	0.454[a]	*2*	0.455[a]	*2*	0.746	*2*
Integration with third party s/w	0.041[a]	*18*	0.056[a]	*12*	0.136	*12*
Quality of any previous documentation	−0.038	*22*	−0.043	*17*	−0.105	*17*
Requirements stability	−0.022	*25*	−0.010	*31*	−0.024	*31*
Stakeholder involvement	0.038[a]	*21*	−0.001	*32*	−0.002	*32*
Requirements management	−0.019	*27*	−0.018	*26*	−0.043	*26*
Regularity of spec and doc reviews	−0.010	*29*	−0.017	*28*	−0.041	*28*
Standard procedures followed	−0.053	*13*	−0.014	*30*	−0.034	*30*
Relevant experience of spec and doc staff	−0.050	*14*	−0.055	*13*	−0.133	*13*
Spec and doc effort	−0.007	*31*	−0.016	*29*	−0.039	*29*
Development staff motivation	−0.029	*23*	−0.018	*25*	−0.043	*25*
Relevant development staff experience	−0.040	*20*	−0.065	*9*	−0.158	*9*
Programmer capability	−0.048	*16*	−0.067	*8*	−0.163	*8*
Development staff training quality	−0.023	*24*	−0.031	*22*	−0.075	*22*
Defined process followed	−0.085	*7*	−0.056	*11*	−0.137	*11*
Development process effort	−0.061	*10*	−0.071	*7*	−0.171	*7*
Testing process well defined	−0.011	*28*	−0.018	*24*	−0.045	*24*
Quality of documented test cases	−0.019	*26*	−0.032	*20*	−0.077	*20*
Testing staff experience	−0.088	*6*	−0.089	*5*	−0.213	*5*
Testing effort	−0.040	*19*	−0.033	*19*	−0.080	*19*
Rework process quality	−0.068	*8*	−0.053	*14*	−0.129	*14*
Rework effort	−0.092	*5*	−0.083	*6*	−0.199	*6*
Process maturity	−0.050	*15*	−0.043	*16*	−0.105	*16*
Internal communications quality	0.001	*32*	−0.020	*23*	−0.050	*23*
Project planning	−0.053	*12*	−0.031	*21*	−0.077	*21*
Subcontractor management	−0.010	*30*	−0.017	*27*	−0.042	*27*
Significant subcontracts	0.059[a]	*11*	0.043	*15*	0.106	*15*
KLOC existing code base	0.105[a]	*4*	0.089[a]	*4*	0.214	*4*
Complexity of existing code base	0.043[a]	*17*	0.038	*18*	0.094	*18*
Overall process and testing quality of existing code base	−0.062	*9*	−0.065	*10*	−0.157	*10*

Values in italics indicate the rank of the specific factor for each measure

[a] For SPEA and SRCC—values significant at $p=0.95$

Box-plots presented on Fig. 19 illustrate the impact of the two most influential qualitative model inputs on the number of residual defects. The plots show, for example, that while the predicted number of defects increases significantly as the complexity of new functionality increases, the variance also increases significantly (as seen by the increase in size of the 25–75% range).

The major difference between the results of the global analysis and that of the simple analysis in Section 10.1 concerns the factor 'integration with third party software'. In fact, its apparent lesser importance based on the global analysis results has a simple explanation. The value 'Yes' for this factor has a small prior probability assigned in the model (0.05) compared to the value 'No' (0.95; this simply reflects the empirical observation that only 5% of the projects involved integration with third party software). Consequently the 3000 sampled data (whose generation was based on the prior distributions) contained very few in which 'integration with third party software' was set as 'Yes'. The previous analysis considered Yes and No values equally.

10.3 Discussion

The results of the sensitivity analysis are useful for two main reasons:

1. They provides a basis for 'internal' validation of the model
2. They provide a practical method for using the model for 'fast' prediction

In the case of 1 the results can be fed back to the experts involved in the model development. The sensitivity analysis results provide a 'holistic' view of the model that was never part of the original expert elicitation. When the model was built experts were not asked to rank attributes in order of their expected impact on residual defects; nor were they asked to consider the impact on residual defects of any given factor across the range of possible values of that factor. Yet the sensitivity analysis results provide exactly this information in a way that summarises the overall cumulative effects of many individual assumptions built into the model. If either

a. the experts disagree with the results of the sensitivity analysis (for example, if they disagree about the relative impact of a certain factor) or

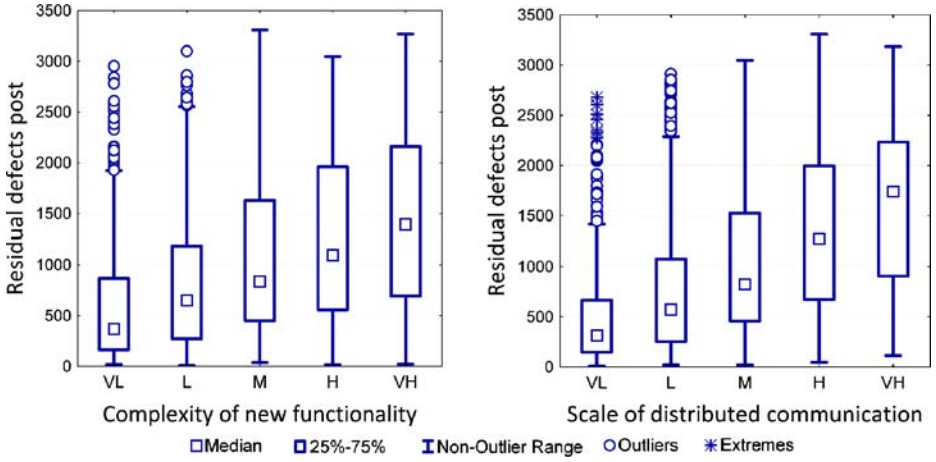

Fig. 19 Impact of the two most influential qualitative factors

b. empirical results from real projects clearly contradict the results of the sensitivity analysis

then this would suggest that some aspect of the model needs to be fixed. As yet we make no claim about *a.* and *b.* other than that the sensitivity analysis results did not throw up any obvious contradictions to either the experts' judgements or to what has been observed empirically in real projects. Having said that, we offer one caveat. The analysis indicated that the variables covering the requirements process had little impact on the number of defects. Yet, poor requirements processes are often the first focus of any improvement activities. We believe this point does need further investigation, although given the relatively informal requirements processes of the projects being studied, "scale of distributed communication" acted to a certain extent as a proxy for the requirements process.

In the case of 2 the idea is that we can use the results of the sensitivity to identify a very small set of factors for early predictions. One of the benefits of a Bayesian Net model is that it is not necessary to enter values for all of the project factors in order to get a prediction. It turns out that, for this model, reasonably accurate predictions of the residual defects can be achieved by entering just a small number of the most influential factors. Typically we have found that entering only values for size, scale of distributed communication, complexity of new functionality, and third party integration, results in reasonably accurate predictions. Hence, this kind of model can be used for effective decision-support and trade-off analysis during early development phases (in the way demonstrated in the examples of Section 9).

11 Conclusions

We have presented a causal model for defect prediction that is a revised version of the MODIST model (MODIST 2003). The main feature that distinguishes it from other defect prediction models is the fact that it explicitly combines both quantitative and qualitative factors.

We have also presented a dataset for 31 software development projects. This dataset incorporates the set of quantitative and qualitative factors that were previously built into a causal model of the software process. The factors (which had been identified by a consortium of software project experts) include values for code size, effort and defects, together with qualitative data values judged by project managers (or other project staff) using a questionnaire. We have used this data to evaluate the causal model and the results are promising. Specifically, the model predicts, with satisfactory accuracy, the number of software defects that will be found in independent testing. This accuracy increases with increasing project size.

To determine which model variables have the greatest impact on the number of residual defects predicted, we have performed various sensitivity analyses. We found that the most influential qualitative factors are *project complexity* and *scale of distributed communication*. Although none of the individual process factors appear to be highly influential on their own, the aggregation of such factors as 'process effectiveness' are highly influential. One of the benefits of a Bayesian Net model is that it is not necessary to enter values for all of the project factors in order to get a prediction. In fact, for this model, reasonably accurate predictions of the defects can be achieved typically by entering values for just the three or four most influential factors identified in the sensitivity analysis. Hence, this kind of model

can be used for effective decision-support and trade-off analysis during early development phases.

The model presented in this paper was developed for a specific context (distributed projects in which the focus was typically a 'subsystem' type component of between 10 and 150 KLOC) and the validation was also largely within this context. Crucially the notion of what was a 'defect' was well-defined within this context. However, the experts involved in building the model were always conscious of the need to make it as general as possible. As a result we feel that the model could be generally used by companies developing commercial software, providing some minimal calibration is considered. There is ongoing research (Radliński et al. 2007) on how to calibrate such models to take account of company specific defect counting methods and defect rates. There may also be a need to add other process factors that are identified as important in a particular company. The information about such factors can be obtained through a modified questionnaire. After this, adding new process variables to the model only requires changing expressions in the child nodes. The process of finding the most appropriate expressions may be tricky if there are significant multi-dimensional correlations between the input variables that we may need to incorporate in the model.

By presenting the raw data in this paper, we hope to enable other researchers to evaluate similar models and decision-support techniques for software managers (the dataset can of course also be used for evaluating more traditional types of software prediction models). We also hope that similar datasets will become more widely available in future.

To ensure full visibility and repeatability, we also provide an electronic version of the causal model for researchers (MODIST BN 2007). The model can be viewed and executed by downloading the free trial version of the Bayesian network software (AgenaRisk 2007).

Acknowledgments This paper is based in part on work undertaken on the following funded research projects: MODIST (EC Framework 5 Project IST-2000-28749), SCULLY (EPSRC Project GR/N00258), SIMP (EPSRC Systems Integration Initiative Programme Project GR/N39234), and eXdecide (2005). We acknowledge the insightful suggestions of two anonymous referees that led to significant improvements in the paper. We also acknowledge the contributions of individuals from Agena, Philips Electronics, Israel Aircraft Industries, QinetiQ and BAE Systems. We dedicate this paper to the late Colin Tully, who acted as an enthusiastic and supportive reviewer of this work during the MODIST project.

References

AgenaRisk (2007) Bayesian Network Software Tool. www.agenarisk.com

Boehm B, Clark B, Horowitz E, Westland C, Madachy R, Selby R (1995) Cost models for future software life cycle process: COCOMO 2.0. Ann Softw Eng 1(1):57–94

Boetticher G, Menzies T, Ostrand T (2008) PROMISE Repository of Empirical Software Engineering Data http://promisedata.org/ repository, West Virginia University, Department of Computer Science

Cangussu JW, DeCarlo RA, Mathur AP (2003) Using sensitivity analysis to validate a state variable model of the software test process. IEEE Trans Softw Eng 29(5):430–443

Chulani S, Boehm B (1999) Modelling Software Defect Introduction and Removal: COQUALMO (COnstructive QUAlity MOdel). Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering

Chulani S, Boehm B, Steece B (1999) Bayesian analysis of empirical software engineering cost models. IEEE Trans Softw Eng 25(4):573–583

Compton T, Withrow C (1990) Prediction and Control of Ada Software Defects. J Syst Softw 12:199–207

eXdecide (2005) Quantified Risk Assessment and Decision Support for Agile Software Projects. EPSRC project EP/C005406/1. www.dcs.qmul.ac.uk/~norman/radarweb/core_pages/projects.html

Fenton NE, Pfleeger SL (1998) Software Metrics: A Rigorous and Practical Approach (2nd Edition). PWS Publishing, Boston
Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Software Eng 25 (5):675–689
Fenton NE, Krause P, Neil M (2002a) Probabilistic modelling for software quality control. J Appl Non-Class Log 12(2):173–188
Fenton NE, Krause P, Neil M (2002b) Software measurement: uncertainty and causal modelling. IEEE Software. 10(4):116–122
Fenton NE, Marsh W, Neil M, Cates P, Forey S, Tailor M (2004) Making Resource Decisions for Software Projects. Proceedings of 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, United Kingdom, 397–406
Fenton NE, Neil M, Caballero JG (2007a) Using ranked nodes to model qualitative judgments in Bayesian networks. IEEE Trans Knowl Data Eng 19(10):1420–1432
Fenton NE, Neil M, Marsh W, Hearty P, Marquez D, Krause P, Mishra R (2007b) Predicting software defects in varying development lifecycles using Bayesian nets. Inf Softw Technol 49(1):32–43
Fenton N, Neil M, Marsh W, Hearty P, Radliński Ł, Krause P (2007c) Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction. Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering. International Conference on Software Engineering. IEEE Computer Society, Washington, DC: 2
Gaffney JR (1984) Estimating the Number of Faults in Code. IEEE Trans Softw Eng 10(4):141–152
Henry S, Kafura D (1984) The evaluation of software system's structure using quantitative software metrics. Softw Pract Exp 14(6):561–573
ISBSG (2007) Repository Data Release 10. International Software Benchmarking Standards Group. www.isbsg.org
Jensen FV (1996) An introduction to Bayesian networks. UCL Press, London
Jones C (1986) Programmer productivity. McGraw Hill, New York
Jones C (1999) Software sizing. IEE Review 45(4):165–167
Kitchenham BA, Pickard LM, MacDonell SG, Shepperd MJ (2001) What accuracy statistics really measure. IEE Proc Softw 148(3):81–85
Lipow M (1982) Number of Faults per Line of Code. IEEE Trans Softw Eng 8(4):437–439
Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng 32(11):1–12
MODIST (2003) Models of Uncertainty and Risk for Distributed Software Development. EC Information Society Technologies Project IST-2000-28749. www.modist.org
MODIST BN Model (2007) http://promisedata.org/repository/data/qqdefects/
Musilek P, Pedrycz W, Nan Sun, Succi G (2002) On the Sensitivity of COCOMO II Software Cost Model. Proc of the 8th IEEE Symposium on Software Metrics: 13–20
Neapolitan RE (2004) Learning Bayesian networks. Pearson Prentice Hall, Upper Saddle River
Neil M, Krause P, Fenton NE (2003) Software Quality Prediction Using Bayesian Networks Software Engineering with Computational Intelligence (Ed T.M. Khoshgoftaar). Kluwer, Chapter 6
Ostrand TJ, Weyuker EJ, Bell RM (2005) Predicting the location and number of faults in large software systems. IEEE Trans Softw Eng 31(4):340–355
Radliński Ł, Fenton N, Neil M, Marquez D (2007) Improved Decision-Making for Software Managers Using Bayesian Networks, Proc. of 11th IASTED Int. Conf. Software Engineering and Applications (SEA), Cambridge, MA: 13–19
Saltelli A (2000) What is Sensitivity Analysis. In: Saltelli A, Chan K, Scott EM (eds) Sensitivity Analysis. John Wiley & Sons, pp. 4–13
SIMLAB (2004) Simulation Environment for Uncertainty and Sensitivity Analysis Version 2.2. Joint Research Centre of the European Commission. http://simlab.jrc.cec.eu.int/
Stensrud E, Foss T, Kitchenham B, Myrtveit I (2002) An Empirical Validation of the Relationship Between the Magnitude of Relative Error and Project Size. Proc. of 8th IEEE Symposium on Software Metrics: 3–12
Wagner S (2007a) An Approach to Global Sensitivity Analysis: FAST on COCOMO. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement: 440–442
Wagner S (2007b) Global Sensitivity Analysis of Predictor Models in Software Engineering. Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering. International Conference on Software Engineering. IEEE Computer Society, Washington, DC: 3
Winkler RL (2003) An introduction to Bayesian inference and decision,, 2nd edn. Probabilistic Publishing, Gainesville

Norman Fenton is Professor of Computer Science at Queen Mary (London University) and is also Chief Executive Officer of Agena, a company that specialises in risk management for critical systems. At Queen Mary he is the Director of the Risk Assessment and Decision Analysis Research Group (RADAR). He is an Affiliated Professor to the University of Haifa, Israel. He has held previous academic posts at City University (Professor in Centre for Software Reliability), South Bank (Director of Centre for Systems and Software Engineering), Oxford University and University College Dublin (both as Research Fellow) and was a visiting researcher at GMD in Germany. He has published extensively in various areas of mathematics, software engineering, and risk analysis. His recent work has focused on causal models (Bayesian Nets) for risk assessment in a wide range of application domains.

Martin Neil is Professor in Computer Science and Statistics at the Department of Computer Science, Queen Mary, University of London, where he teaches decision and risk analysis and software engineering. He is also a joint founder and Chief Technology Officer of Agena Ltd, who develop and distribute AgenaRisk, a software product for modeling risk and uncertainty and a Visiting Professor in the Faculty of Engineering and Physical Sciences, University of Surrey. He has over twenty years experience in academic research, teaching, consulting, systems development and project management and has published or presented over 40 papers in refereed journals and at major conferences. His interests cover Bayesian modeling and/or risk quantification in diverse areas: operational risk in finance, systems and design reliability, software project risk, decision support, simulation cost benefit analysis, AI and personalization, and statistical learning. He has consulted to Motorola, Philips, NATS, QinetiQ, Advantica, DSTL (UK MOD), ABSA, Ericsson, Royal Bank of Canada, TNO and others, either providing advanced risk modeling expertise or systems deployment and integration

using AgenaRisk. Before setting up Agena and joining academia he previously held senior positions with JP Morgan and Lloyds Register in the areas of software project governance and safety critical systems evaluation respectively. He earned a B.Sc. in Mathematics, a Ph.D. in Statistics and Software Metrics and is a Chartered Engineer.

William Marsh is a lecturer in the Department of Computer Science, Queen Mary, University of London, where he teaches Systems Analysis, Operating Systems and Networks. He has seven years' experience in academic research; his interest cover software project risk, system safety risk and medical decision support. Before joining academia, William had over sixteen years' experience covering the development and safety assessment of real time and safety critical software. He earned a B.A. in Engineering, M.Sc. in Computation and a Ph.D. in Computer Science. He is a member of the BCS.

Peter Hearty is a research assistant and Ph.D. student at Queen Mary, University of London. He graduated in Mathematics and Physics from Stirling University in 1982 before working as a mathematical modeller and later as an IT consultant for GEC-Marconi, Reuters, Bankers Trust and NatWest. From 1997 to 2005 he ran his own database and software licensing company. His interests include Dynamic Bayesian Nets and Agile software development methods.

Łukasz Radliński obtained M.A. degree in Computer Science and Econometrics from University of Szczecin, Poland in 2000. Since then he has been working as an assistant at the same university in the Institute of Information Technology in Management conducting research and teaching various software engineering courses. He has also worked as a software developer for some local companies. Since 2005 he has been a Ph.D. student at the Department of Computer Science, Queen Mary, University of London, UK. His research interests include software project risk management, quality assurance and applications of Bayesian nets.

Paul Krause has over 20 years experience in research and application of advanced software engineering techniques. He has been Professor of Software Engineering at the University of Surrey since January 2001. Primary research interests are in the specification and testing of high integrity distributed software applications. Specific research contributions in formal modelling of interacting software components, automated generation and execution of software test cases for highly concurrent software applications, and software quality prediction using Bayesian Networks. Prior to 2001, Prof. Krause was Senior Principal Scientist at Philips Research Laboratories, where he successfully migrated results from a range of his research projects on the specification and testing of software into development organisations within Philips Electronics. Since moving to Surrey, his research interests have developed in the field of "Digital Ecosystems" as a partner in the EU funded DBE and OPAALS projects. He is Computer Science Coordinator in the latter.