An Investigation of the Use of BBNs for Project Management

D. Rodriguez[1] R. Harrison[1] M. Satpathy[1] J. Dolado[2]

[1]School of Computer Science, Cybernetics and Electronic Engineering

University of Reading, Reading RG6 6AY, UK

{d.rodriguez-garcia, Rachel.Harrison, M.Satpathy}@reading.ac.uk

[2]P.M. Lardizabal, 1, 20.018 San Sebastián, Spain

dolado@si.ehu.es

Abstract

Dynamic models such as Bayesian Networks and System Dynamics are of late becoming increasingly popular among the Software Engineering research community. Such models take into account the causal relationship between the variables of a domain, and therefore may provide better solutions to many of the problems in this area. This paper concerns the use of Bayesian Networks (sometimes called Bayesian Belief Nets or BBNs) in project management. Creation of a BBN to reflect a software engineering problem is a challenging task. In this paper, we will concentrate on the creation of a BBN by using domain data. There are many semi-automatic approaches available in literature but there are also many open issues in these approaches. We will discuss the step by step construction of a BBN using the dataset from a Web-related project involving 37 Web sites. We will also perform an analysis of the generated network.

1 Introduction

A Bayesian Belief Network (BBN) [8] is a directed graph in which the nodes represent uncertain variables and the arcs represent the causal relationship between the variables. Each node has a probability table, which stores the conditional probabilities for each possible state of the node variable in relation to each combination of its parent state values. For a node without any parents, such a table stores the marginal probabilities for each possible state of that node. If the state of a certain node is known then its probability table is altered to reflect this knowledge. Such knowledge is then propagated to determine the changed probabilities of all possible values associated with other nodes. Note that the initial probabilities of the nodes in a BBN are obtained from expert judgment or past project data.

BBNs have been used in many application domains, such as medical diagnosis, price forecasting, pedigree analysis, automated vision, planning, debt detection, bottleneck detection, modelling of manufacturing processes etc. They provide the following advantages when compared with static models:

(i) BBN models can predict events based on partial or uncertain data, i.e., making good decisions with data that is scarce and incomplete. Uncertainty is inherent in Software Engineering. For example, Kitchenham and Linkman [9] state that estimates are a probabilistic assessment of a future condition and that is the main reason why managers do not obtain good estimates.

(ii) Software Engineering entities are typically influenced by many factors. BBNs allow us to create and manipulate complex models to understand chains of events (causal relationships) in a graphical way that might never be realised using conventional methods.

Recently, several papers have been published applying BBNs to Software Engineering. For example, Wooff *et al* [13] describe a Bayesian approach to the problem of software testing. The authors state that BBN can be used to drive test design, answer what-if questions, and provide decision support to managers and testers. As Fenton [7] states: *"The key to more powerful use of software metrics is not in more powerful metrics. Rather it is in a more intelligent approach to combining different metrics. […]. Once you are comfortable with such an approach you may need to look at more advanced statistical techniques for achieving a real prediction system. BBNs is such an approach that we feel provides the most promising way forward"*.

Neil and Fenton [7] estimate defect density taking into account high-level factors such as programmer/analyst skill, design methods and procedures in addition to traditional factor such as effort, complexity of the design etc. Therefore, using BBN, it is possible to include variables in a software reliability model that correspond to processes as well as product attributes. The authors have also developed, under the SERENE project in the area of safety and risk evaluation, a tool to create a larger network from smaller networks [11].

1.1 An Example

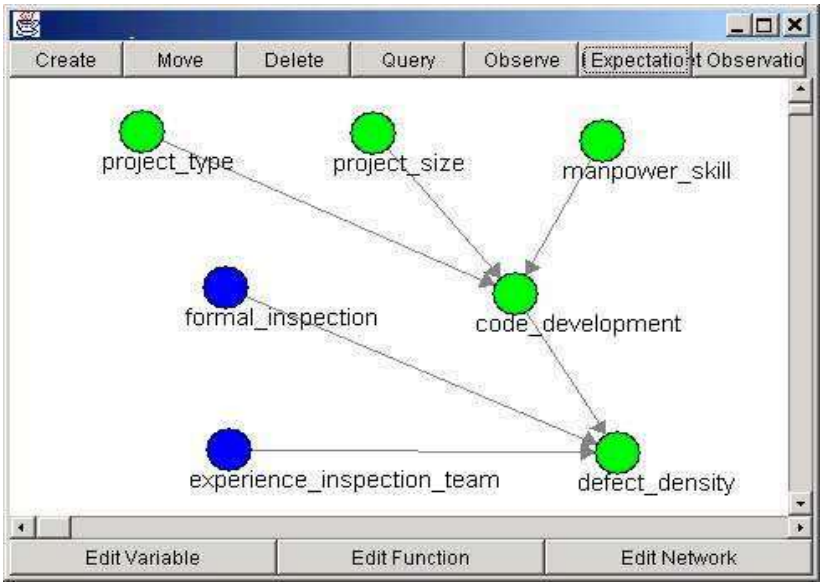

Figure 1 A BBN Example

Figure 1 shows the example of a BBN. The aim of this BBN is to predict the defect density of an implementation (before testing). It is assumed that formal inspection is performed over the code to find out the presence of errors. The BBN says that the estimation of defect density depends on the code development effort, formal inspection effort and the experience of the people doing the formal inspection. Similarly, code development effort depends on the type of the project, the size of the project and the skill of the development team. Conditional probabilities are assigned to each node through expert judgment. Consider the variable *project_size*. Assume, it can have three options: low (< 5000 LOC), medium (5000 to 20000 LOC) and high (<20000 LOC). By default, we assume that the probabilities of this value being high, medium and low are 20%, 60% and 20% respectively. Similarly, manpower skill can be low, high and medium. Project type can be organic, semi-detached or embedded. They can have their probabilities. One entry of the node *code_development* will be the probability of *code_development* effort being high provided that *manpower_skill* is low, *project_type* is embedded and *project_size* is low. What should be this probability value is usually obtained from expert judgment.

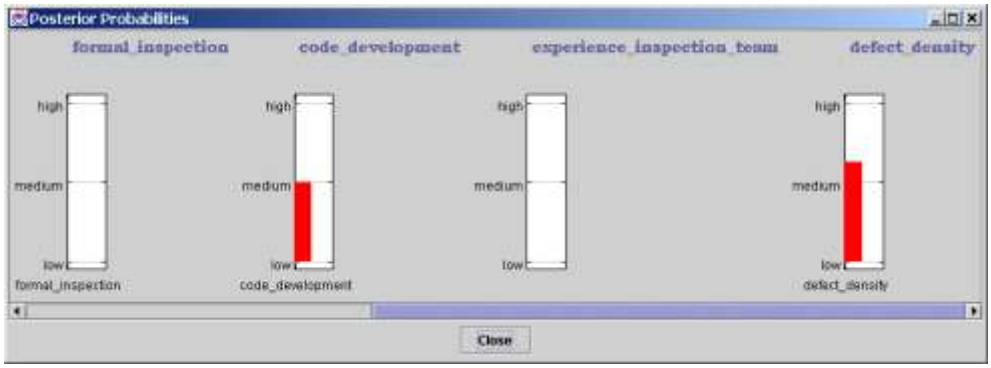

Figure 2 Analysis of BBN without Formal Inspection

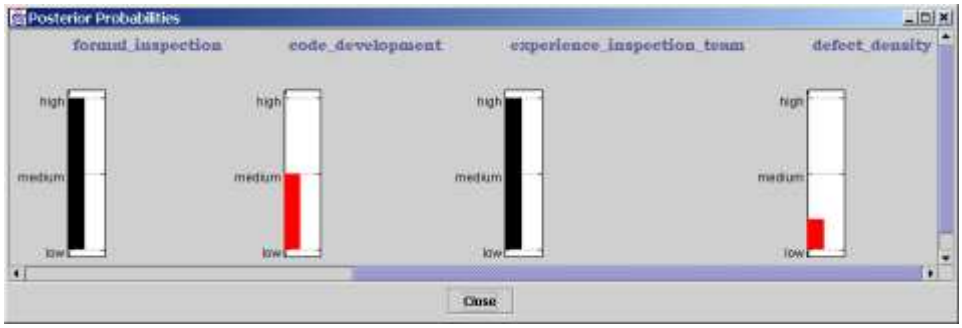

Figure 3 BBN analysis with an experienced formal inspection team

The usage of the tool incorporating this BBN is fairly intuitive. The tool has two modes of operation: *observation mode* which allows insertion of evidence, and a *query mode* where estimations are visualised. This means that in the beginning every node has its default probability values. Suppose we know that the experience of the formal inspection team is very high. This evidence is entered by physically altering the probability value of the corresponding node. This is done in observation mode. Now this evidence will be

propagated in the network which may alter all the values of the nodes in the network. The values are altered according to Bayesian laws of conditional probability.

For example, Figure 2 shows the default probability values of some variables. Next the evidences that *formal_inspection* effort is high and experience of the *inspection_team* team is high are entered. This alters the probabilities of the other nodes in the network. Such a scenario has been shown in Figure 3. This shows that the probability of the defect density being low is very high.

2 Mining Software Engineering Data using BBNs

Creation of a BBN to reflect the problem under consideration is an intellectual task. For instance, what if the probability entries in the above network are wrong? One way is to take past project data as well as intellectual advice to create the network and to assign the probability values.

In this section we will show how data mining techniques and BBNs can help us to acquire knowledge about the software engineering process of an organization. We will first define our terminology. *Domain knowledge* refers to information specific to a field. Users may have a limited subset of knowledge, called *background knowledge*. New or discovered knowledge can be obtained in a *deductive* or *inductive* way. Deduction infers information as a logical consequence of the data (e.g. average cost of a project). Induction infers information by generalisation. The computational techniques and tools designed to support the automation of inductive learning, i.e., extraction of useful knowledge from data, are called machine learning techniques. More recently, terms like *data mining* or *Knowledge Discovery in Databases* (KDD) have come to be used in place of machine learning. In general, KDD techniques try to extract in an automatic way the information useful for decision support or exploration of the data source [6].

Since data may not be organised in a way that facilitates the extraction of useful information, typical KDD processes are composed of the following steps:

(i) Data preparation. The data is formatted in a way that tools can manipulate it.
(ii) Data selection and cleaning. There may be missing, noisy and uncertain data in the raw dataset.
(iii) Data mining. It is in this step when the automated extraction of knowledge from the data is carried out.
(iv) Proper interpretation of the results, including the use of visualization techniques.
(v) Assimilation of the results.

Examples of such algorithms and some usual representations include: C4.5 for decision trees; COGITO for decision lists; Apriori for association rules; K2 for Bayesian Networks etc.

2.1 Learning BBNs from Data

When data is available, automated learning methods have been developed for learning, i.e. obtaining, BBNs from data. The process is composed of two main tasks: (i) induction of the best matching qualitative dependency model from data and prior knowledge; and (ii) estimation of the local probabilities.

There are two approaches to learning Bayesian Networks from data [1]: (i) search and scoring methods and (ii) dependency analysis methods.

In search and scoring methods, the problem is viewed as a search for a structure that fits the data best. Starting with a graph without edges, the algorithm uses a search method to add an edge to the graph. Then the scoring method checks whether the new structure is better than the old one. If so, the edge is kept and a new edge is tried. The algorithm stops when there is no better structure. This problem is known to be an NP-hard problem [1], therefore, many of the methods use heuristics to reduce the search space with node ordering as an input. Well-known search and score algorithms include: Chow and Liu Tree Construction Algorithm [3], K2 [4] etc.

Dependency analysis methods follow a different approach. These methods try to find dependencies from the data which in turn are used to infer the structure. Dependency relationships are measured using conditional independency tests.

2.2 The Process of Creating BBNs from Data

The typical steps of a Bayesian network based data mining process are presented in Figure 4. We will illustrate this process following an example in the context of estimation in Web Engineering using the dataset provided by Mendes *et al* [10].

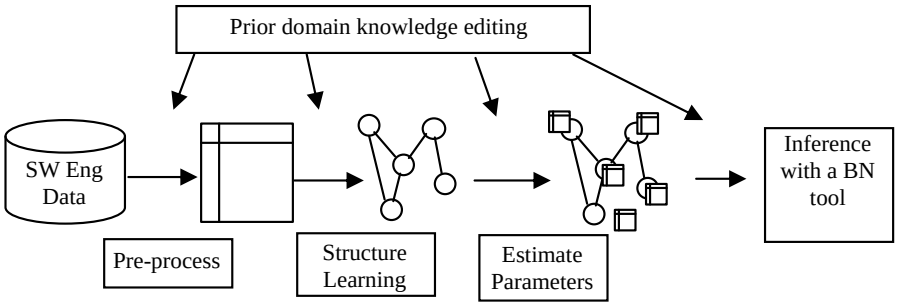

Figure 4 Process Steps to Obtain BBNs from Data

Data Pre-processing:

Our dataset is composed of 37 Web applications (3 were outliers) which were created by final year undergraduate students. This dataset has been shown in Appendix C; for clarity, we show a truncated version of it in Table 1. From this dataset, the authors identified 8 variables that characterise a Web hypermedia application and its development process.

ID	*Total Effort*	*Page Count*	*Media Count*	*…*	*Reused Program Count*	*Total Reused Media Allocation*	*Total Reused Code Length*
1	79.13	43	.00	…	0	804	0
2	133.1	53	53.00	…	0	1010	0
3	259.5	67	207.00	…	0	2704	0
…	…	…	…	…	…	…	…
37	141.4	52	48.00	…	0	589	0

Table 1 Raw Dataset

Metric	***Description***
Page Count (PaC)	Number of HTML or SHTML files used in the application
Media Count (MeC)	Number of Media files used in the application
Program Count (PRC)	Number of JavaScript files and Java applets used in the application
Reused Media Count (RMC)	Number of Reused/modified media files
Reused Program Count (RPC)	Number of Reused/modified programs
Connectivity Density (CoD)	Total number of internal links divided by *Page Count*
Total Page Complexity (TPC)	Average number of internal links divided by *Page Count*
Total Effort (TE)	Effort in person hours to design and author the application

Table 2 Size and Complexity Measures [10]

In order to create a BBN from data, it needs to be pre-processed. Typically, this process consists of discretizing data and dealing with missing values. In our case, we did not have any missing value but it was necessary to discretize most of the variables. We need to decide whether a variable is continuous, the number of intervals and the discretization method, usually by frequency or range. Naturally, we lose some of the available information in the process.

Table 3 partially shows the pre-processed dataset where new columns with discretized data have been added. Columns that were already discrete and new columns with discretized data will be used in the learning process.

ID	***TE***	***PaC***	***MeC***	***PRC***	***…***	***TE_d***	***PaC_d***	***MeC_d***
1	79.13	43	0	0	…	<92.175	<50.5	<0.5
2	133.1	53	53	1	…	>128.45<150.1	>50.5<53.5	>27.5<78
…	…	…	…	…	…	…	…	…
34	141.4	52	48	5	…	>128.45<150.1	>50.5<53.5	>27.5<78

Table 3 Pre-processed data

Learning process:

This process consists of learning the network structure and generating the node probability table.

In order to learn the structure, we need to apply domain knowledge both before and after the execution of the learning process. Before executing the learning process, it may be necessary to totally or partially define the structure of the network (node ordering for known causal relationships, forbidden links and root/leaf nodes) and some parameters such as thresholds for the accepted dependencies. After executing the algorithm, we may need to edit the network to add, remove or label arcs if it was impossible to do this from the data.

There are several projects under the GNU license that implement well-known *search and scoring* and *dependency analysis* algorithms in Java. Those algorithms can be integrated into the USEGESOFT tool easily. However, we found that using wizard-like GUI interfaces to define the structure facilitates the above task. Among these software packages we used BNPC tool [2] and Bayesware Discoverer [12]. Figure 6 shows a possible BBN generated after running the BBN Power Constructor tool.

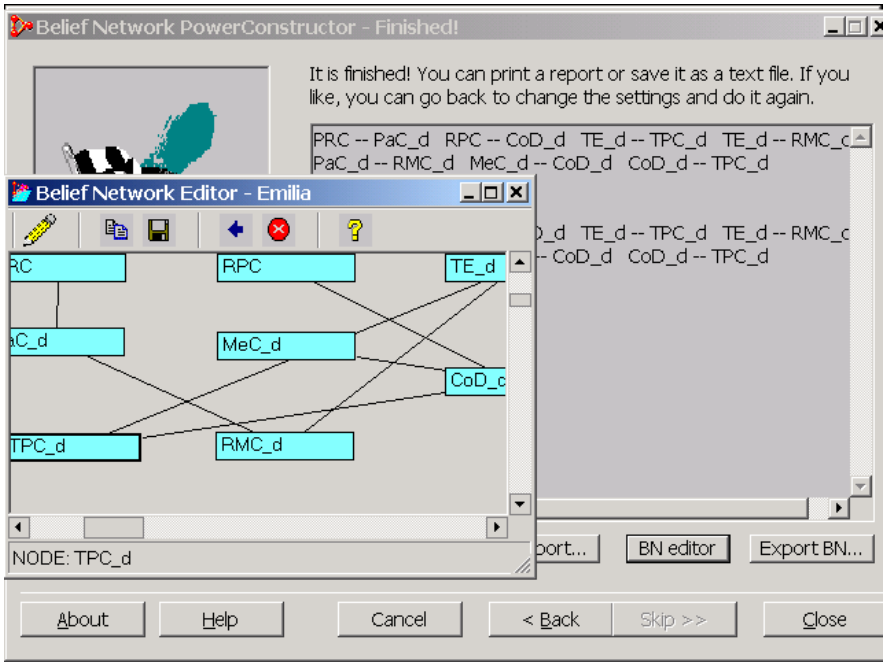

Figure 5 Learning Process using the BNPC Tool [2]

Figure 5 shows a screenshot of the BNPC tool and the generated network. In this case, the tool was not able to assign directions to the links between the nodes with the data available, hence the need to modify the network by hand. It is to be noted that expert domain knowledge may be required for such editing. Once the structure is defined, the probabilities of the links are estimated from the data by the BNPC tool.

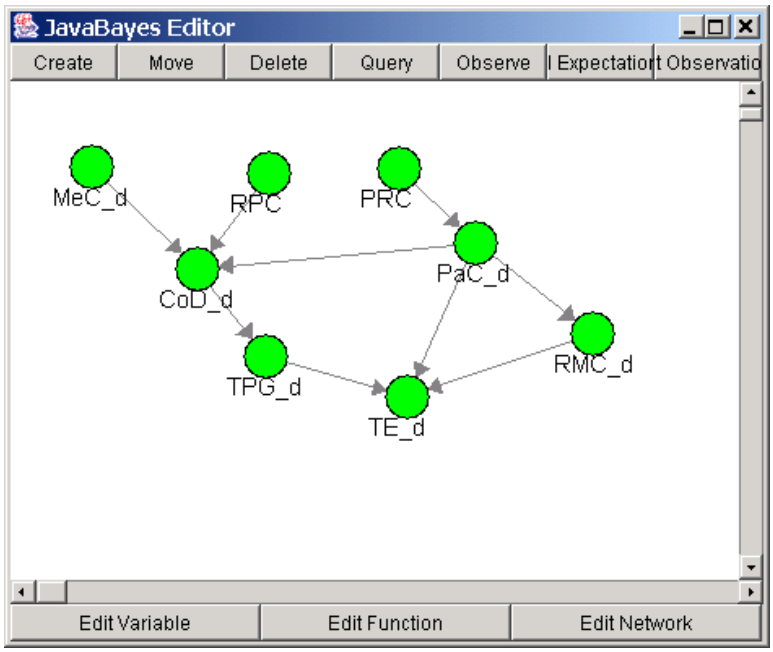

Figure 6 BBN generated from Mendes' dataset

At this stage the resulting network can be used by Bayesian network inference tools; in our case, we used USEGESOFT tool which includes a modified version of the JavaBayes

tool [5]. Figure 6 shows the final network from which we should be able to show how predictions might be made and historical results can be explained more clearly (Figure 7 and Figure 8). For example, it would be possible to estimate the *total effort* of the authoring of a Web application once we have approximate knowledge about other parameters of the Web application such as number of pages, amount of media used, number of JavaScripts etc. It is worth noting that in this case the output will be a distribution of probabilities instead of a single value. Usually, the value or the range with the highest probability is taken as the expected answer.

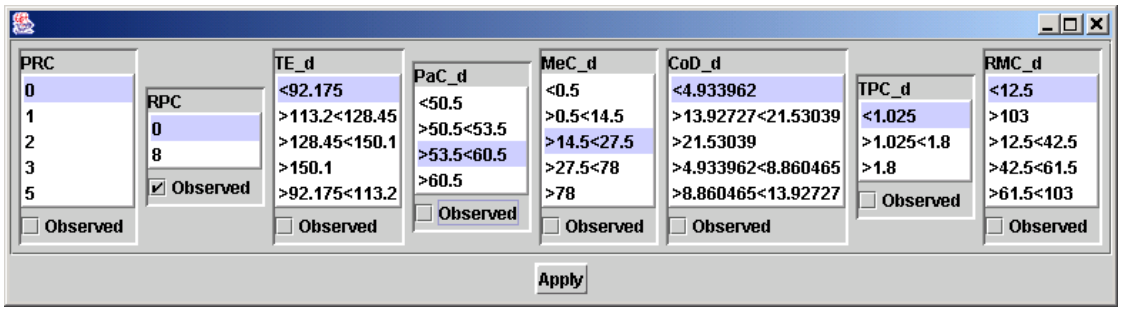

Figure 7 Evidences Window of a BBN for Web Authoring Estimates

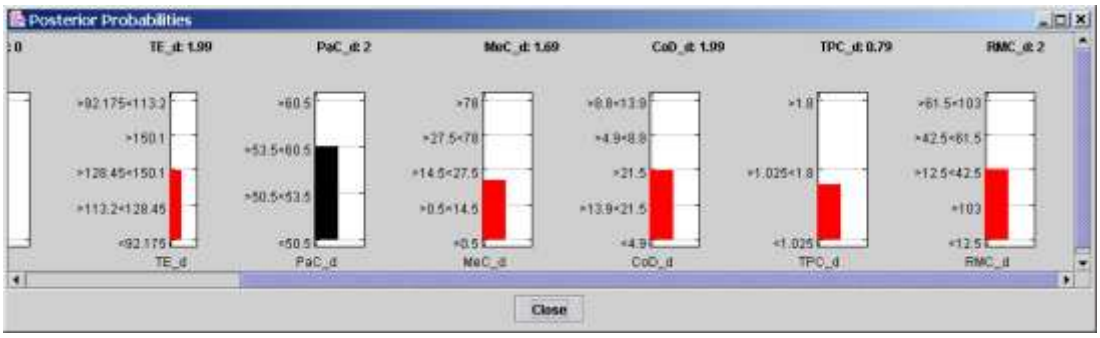

Figure 8 Probabilities Window of a BBN for Web Authoring Estimates

Validation of the BBN:

It is important that we need to validate a BBN once it is created from historical data. Presently, we will compare the results obtained from this network with the actual dataset from which we obtained the network. Note that our approach is not a proper validation technique; however, this will give us some confidence about the effectiveness of the network.

Table 4 shows the maximum, minimum and the mean values of the 8 variables in the original dataset. We use the network to obtain the value of TE (total effort) by feeding to the network the values of the remaining 7 variables. The results that we obtain from the BBN are shown in Figure 9. For the minimum values of the said 7 variables, Table 6.4 says that TE has a value of 58.36. The network shows (refer to Figure 6.9) that the probability that the value of TE remains less than 92 is 55%, and the other possible values of TE are very low. When the maximum values are considered, the table shows that value of TE is 153.78, whereas the Figure shows that the probability of TE being 150 is maximum (33%). So far as the mean values are considered, the table shows that TE has a value of 111.89, whereas the network shows that the probability of the TE value remaining between 92 and 128 is high.

What we can say from these results is that the values obtained by the network to a large extent approximates the actual values, although they are not very accurate. The reason is that the number of cases considered in the dataset is low (34). Furthermore, the network construction process uses expert knowledge at intermediate stages, which might have not been perfect in our case.

Metric	***Mean***	***Min***	***Max***
PaC	55.21	33	100
MeC	24.82	0	126
PRC	0.41	0	5
RMC	42.06	0	112
RPC	0.24	0	8
COD	10.44	1.69	23.3
TPC	1.16	0	2.51
TE	111.89	58.36	153.78

Table 4 Summary Statistics

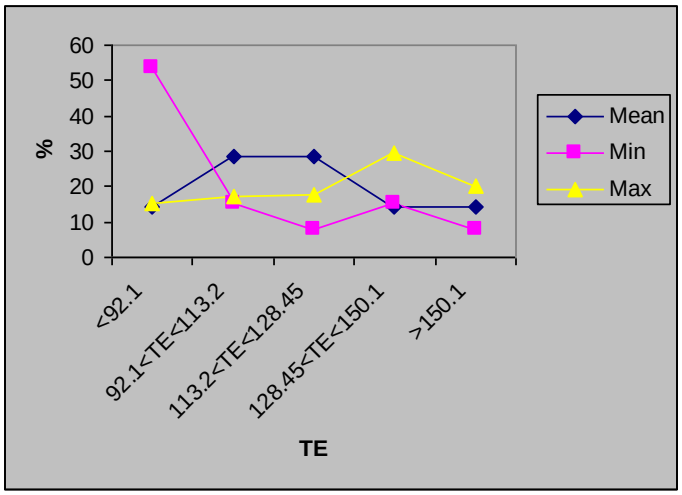

Figure 9 Output Probabilities for the Variable *Total Effort* (TE)

3 Conclusion

Computation with BBNs can be a very powerful technique in software engineering problems like estimation and risk analysis because: (i) they take into account the cause-effect relationships of the variables in the problem domain, and (ii) the backwards propagation of the probabilities in the BBN can help in taking managerial decisions that might not be possible using static models.

Construction of BBNs is an intellectual task. We have discussed a semi-automatic approach which uses many prototype tools for constructing BBNs. We have also discussed a practical example and constructed a BBN in relation to the given dataset. Research on validation of BBNs is an important part of our future work.

References

[1] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*: Springer-Verlag, 1997.

[2] J. Cheng, "Belief Network (BN) Power Constructor", 2.2 Beta ed, 2001.

[3] C. K. Chow and C. N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462-467, 1968.

[4] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, pp. 309-347, 1992.

[5] F. G. Cozman, "JavaBayes 0.346", 0.346 ed, 2001.

[6] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD Process for Extracting Useful Knowledge From Volumes of Data", in *Communications of the ACM*, vol. 39, 1996, pp. 27-34.

[7] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 25, pp. 675-689, 1999.

[8] F. V. Jensen, *An Introduction to Bayesian Networks*. London: UCL Press, 1996.

[9] B. A. Kitchenham and S. G. Linkman, "Estimates, Uncertainty, and Risk", in *IEEE Software*, vol. 14, 1997, pp. 69-74.

[10] E. Mendes, I. Watson, C. Triggs, N. Mosley, and S. Counsell, "A Comparison of Development Effort Estimation Techniques for Web Hypermedia Applications", 8th IEEE Metrics Symposium, Ottawa, Canada, 2002.

[11] M. Neil, N. E. Fenton, and L. Nielsen, "Building large-scale Bayesian Networks," *The Knowledge Engineering Review*, vol. 15, pp. 257-284, 2000.

[12] M. Ramoni and P. Sebastiani, "Bayesware Discoverer", 1.0 ed, 2002.

[13] D. A. Wooff, M. Goldstein, and F. P. A. Coolen, "Bayesian Graphical Models for Software Testing," *IEEE Transactions on Software Engineering*, vol. 28, pp. 510-525, 2002.