ISSN No. 0976-5697

Volume 1, No. 4, Nov-Dec 2010

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Web Effort and Defect Prediction using Bayseian Model

Nandakumar.P*
School of Information Technology
VIT University, Vellore, India
Nandakumarp2005@vit.ac.in

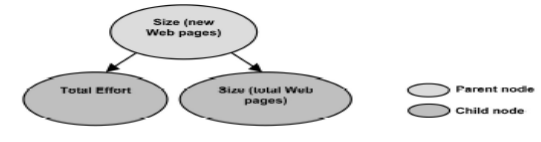
Magesh.G
School of Information Technology
VIT University, Vellore, India
Magesh.g@vit.ac.in

Anitha Kumari.A
School of Information Technology
VIT University, Vellore, India
Anithakumari.a@vit.ac.in

Abstract: objective. The objective of the paper is the web effort prediction how effectively the web application project will be developed and what are all the defects, quality control also developed in the web application projects by using the COCOMO and software effort techniques are also been involved. In the project how effectively those projects will be developed. The approach allows us to incorporate causal process factors as well as combine qualitative and quantitative measures, hence overcoming some of the well-known limitations of traditional software metrics methods. What are all the effort prediction is applied and what are all the defects present in that by using the Bayesian network (BN) model and defect prediction using the "McCabe's versus Halstead versus lines of code counts" for generating defect predictors. By suing this method can easily find the defect prediction in which modules the most errors are occurring. We show here that such debates are irrelevant since how the attributes are used to build predictors is much more important than which particular attributes are used. Also, contrary to prior pessimism, we show that such defect predictors are demonstrably useful.

I. INTRODUCTION

In the [39] Web effort project is the most important one where to check the effort Estimation, the process by which effort is most predictable one and used to determine Costs and resource should be allocated, enabling projects to be delivered on time and within budget. Effort estimation is a very tough to determine in web application project it should be more comparatively work with each domain. Within the context of Web effort estimation, numerous studies investigated the use of effort prediction techniques. However, to date, only Mendes [2], [3], [4] has investigated the inclusion of uncertainty, inherent to effort estimation, into a model for Web effort estimation. A BN is a model Which supports reasoning with uncertainty due to the way in which it incorporates existing complex domain knowledge [1], [7]? Herein, knowledge is represented using two parts. The first, which is the qualitative part, represents the structure of aBNas depicted by a directed acyclic graph (digraph; see Fig. 1).



NPT for node Size (new Web pages)	
Low	0.2
Medium	0.3
High	0.5

NPT for node Total Effort (TE)			
Size (new Web pages)	*Low*	*Medium*	*High*
Low	0.8	0.2	0.1
Medium	0.1	0.6	0.2
High	0.1	0.2	0.7

Fig. 1. A small BN model and two NPTs [3].

The digraph's nodes represent the relevant variables (factors) in the domain being modeled, which can be of different types (e.g., observable or latent, categorical). The digraph's arcs represent the causal relationships between variables, where relationships are quantified probabilistic cally [1], [6], [8]. The second, which is the quantitative part, associates a node probability table (NPT) to each node, its probability distribution. A parent node's NPT describes the relative probability of each state (value); a child node's NPT describes the relative probability of each state conditional on every combination of states of its parents (e.g., the relative probability of total effort (TE) being "Low" conditional on Size (new Web pages; SNWP) being "Low" is 0.8). Each column in an NPT represents a conditional probability distribution and, therefore, its values sum up to 1 [1]. Once the BN is valued in each module automatically in all other modules the values will be generated automatically. In this paper where the BN method has been used for web effort estimation and we having the opportunity to gather data on the industrial web application projects in that newly created dp (web effort and defect prediction) database in that data to create the BNs presented herein. The project data characterize Web projects using size measures and cost drivers targeted at early effort estimation. Since we had a data set of real industrial Web projects, we were also able to compare the accuracy of the Web effort BNs to that using Manual Stepwise Regression (MSWR) [2].

A. P1used data in the WDP database.
B. P2 used data in another Web projects as Data.

P1 must be used in a single BN tool, Hug in, for structure and parameter learning, p2 used two tools Hug in and power soft. S1 used the entire WDP database to elicit

© 2010, IJARCS All Rights Reserved

the initial BN causal graph S1 in effect used a hybrid BN model, where the causal graph was expert driven and its probabilities data driven. [a1]Their BN model was validated. Then the next will be the checking what are all the defects must be presenting in the web application projects. In this defect prediction mostly the cost is very cheap and effectively can use this method by using the Bayesian method is that minor changes in the data(such as a slightly different sample used to learn a predictor) can make different attributes appear most useful for defect prediction. When we using the McCabe's complexity attribute, just because of small variations to the data. [9] Where the Halstead method used for the determining the defects in the web application projects using these two methods McCabe's and Halstead. Whether the Web application project will give better solution or not, what are all defects must be presented in that, what are all should be remove from that one, which area should be modified and data's occurring there should be valuable one or not . Those things should be evaluated in this paper.

II. RELATED WORK

There have been numerous attempts to model effort estimation for Web projects, but, except for S1, none have used a probabilistic model beyond the use of a single probability distribution. TSE.2008.64, presents a summary of previous studies. Whenever two or more studies compared different effort estimation techniques using the same data set, we only included the study that used the greatest number of effort estimation techniques. For the defect predictions McCabe [10] and Halstead [9]. McCabe and Halstead are "module"-based metrics, where a module is the smallest unit of functionality.2 we study defect predictors learned from static code attributes since they are useful, easy to use, and widely used. Useful. This paper finds defect predictors with a probability of detection of 71 percent. Easy to use. Static code attributes like lines of code and the McCabe/Halstead attributes can be automatically and cheaply collected, even for very large systems [11]. By contrast, other methods, such as manual code reviews, are labor-intensive. Depending on the review methods, 8 to 20 LOC/minute can be inspected and this effort repeats for all members of the review team, which can be as large as four or six [12]. [14] In another papers where the Selecting a defect prediction model for maintenance resource planning and software insurance in this paper where the defect prediction models could lead to better maintenance resource and potentially a software system. So mostly where the data are been defect checking separately in any other paper web effort and defect can be checked properly. [13]This paper reviews the use of Bayesian Networks (BNs) in redacting software defects and software reliability. For the best quality of the projects the COCOMO is the one method to approach how the project has been most cost effective one for the organization and the customers then using software effort techniques with the most techniques can easily judge the effort of the project. The approach allows us to incorporate causal process factors as well as combine qualitative and quantitative measures, hence overcoming some of the well-known limitations of traditional software metrics methods. Finally in all other survey checking separately in each field. So effort and defect can be checking in a single paper.

III. BUILDING THE WEB EFFORT

The BNs were built and validated using an adapted Knowledge Engineering of BN (KEBN) process [15], [16], [17] The three main steps that are part of the KEBN process are the Structural Development, Parameter Estimation, and Model validation. The KEBN process iterates over these steps until a complete BN is built and validated. Each of these steps is briefly described. Structural Development entails the creation of the BN's graphical structure (causal graph) containing nodes (variables) and causal relationships. These can be identified by DEs, directly from data, or using a combination of both. Within the context of this work, the BNs' graphs were obtained using data from the wdp database and current knowledge from a DE, The identification of values and relationships was initially obtained automatically using two BN tools, Hug in and Power Soft, and two training sets each containing 130 projects randomly chosen, leading to four of the BN models used in this study. Later, another four BN models were created, all using a single model structure elicited by the DE and probabilities obtained by automatically fitting this structure to the same two training sets and tools previously used to be used with Hug in Expert and Power Soft. There are no strict rules as to how many discrete approximations should be used. Some studies have employed three [18], others five [14], seven [4], and eight [19]. We chose five because the DE participating in this study was happy with this choice and also because An edictal evidence from eliciting BNs with local Web companies has shown that companies find three to five categories sufficient. Both Hug in and Power Soft offer several discretization algorithms. We used the equal frequency intervals algorithm, as suggested in [20] and used in [21], [22], [23], and five intervals, as also done in [21], [22], [23]. Therefore, each interval contained approximately 130/5 data points. Sometimes, a variable presented repeated values, making it impossible to have exactly the same number of data points per interval. This was the case for variables Fots, HFotsA, Hnew, totHigh, FotsA, and New. None of the eight BN structures was optimized [17], [12],[24] (a technique used to reduce the number of probabilities that need to be assessed for the network) to guarantee that every BN node would have its NPT generated solely using the WDP data. The five effort categories used with both Hug in and Power Soft were given as follows: [1, 1,000.88), [1,000.88, 2,000.66), [2,000.66, 3,000.44), [3,000.44, 4,000.22), [4,000.22,5,000.11).Parameter Estimation represents the quantitative component of a BN, which results in conditional probabilities that quantify the relationships between variables [17]. Probabilities can be obtained via Expert Elicitation, automatically, or using a combination of both. For all eight BN causal graphs in this paper, parameters were obtained by automatically fitting a BN graph to two training sets each of 130 Web projects (automated learning). Hug in used the EM-Learning algorithm [22] and Power Soft used a proprietary algorithm [7]. Two validation sets, each containing 65 projects, were then employed for the Model Validation step to assess the effort prediction accuracy of each BN model. Since there is no de facto standard of how many projects a validation set should contain, we chose to use a 66:33 split, as in [5], [25]. Model Validation. This step validates the BN constructed

© 2010, IJARCS All Rights Reserved

from the two previous steps and determines the necessity to revisit any of those steps. Two different validation methods are generally used—Model Walkthrough and Predictive Accuracy [26]. Both verify if predictions provided by a BN are, on average, better than those currently obtained by a DE. Predictive Accuracy is normally carried out using quantitative data and was the validation approach employed by this paper. Estimated effort for each of the projects in a validation set was obtained using a point forecast, computed using the method described in [27]. Carried out a Predictive accuracy procedure using two validation sets of real data volunteered by numerous Web companies worldwide.

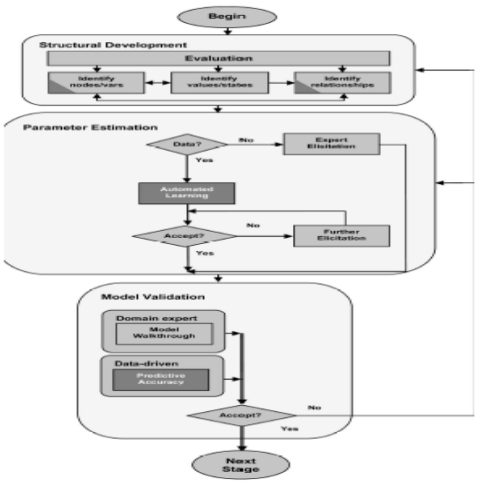


Fig. 2 KEBN adapted from [26]

IV. DEFECT PREDICTION

We learn defect predictors from static code attributes defined by McCabe [29] and Halstead [28]. McCabe and Halstead are module"-based metrics, where a module is the smallest unit of functionality.2 We study defect predictors learned from static code attributes since they are useful, easy to use, and widely used. Useful. This paper finds defect predictors with a probability of detection of 71 percent. This is markedly higher than other currently used industrial methods such as manual code reviews: A panel at IEEE Metrics 2002 [26] concluded that manual software reviews can find _ 60 percent of defects.3. Raffo found that the defect detection capability of industrial review methods can vary from pd = TR (35; 50; 65)% for full agan inspections4 [29] to pd = TR(13; 21; 30)% for less-structured inspections. Easy to use. Static code attributes like lines of code and the McCabe/Halstead attributes can be automatically and cheaply collected, even for very large systems [30]. By contrast, other methods, such as manual code reviews, are labor-intensive. Depending on the review methods, 8 to 20 LOC/minute can be inspected and this effort repeats for all members of the review team, which can be as large as four or six [34]. Our experimental method seeks the "best"subsets of the available attributes that are most useful for predicting defects. We will show that the best size for the "best" set is larger than 1; i.e., predictors based on single (as rgued by Shepherd and Ince and Fenton and fleeger), then we would expect lower probabilities of detection and Uch higher false alarm rates. 2. These new (pd; pf) figures are much larger than any of our prior results of mean (pd; pf) (36%; 17%) [4] (See Fig. 2). Despite much experimentation [36], [35], the only way we could achieve a pd > 70% was to accept a 50 percent false alarm rate. 3. These new results of mean(pd) = 71% are better than currently used industrial ethods, such as the pd _ 60% reported at the 2002 IEEE Metrics panel or the edian(pd) = 21::50 reported by Raffo.4. There is still considerable room for improvement, such as lower pfs and higher pds. We are actively researching better tode metrics which, potentially,will yield "better" predictors.

Table I data sets

system	language	sub-system data set	total LOC	# modules	% defective
spacecraft instrument	C	cm1-05	17K	506	9
storage management for ground data	Java	kc3	8K	459	9
		kc4	25K	126	49
Db	C	mw1	8K	404	7
Flight software for earth orbiting satellite	C	pc1-05	26K	1,108	6
		pc2	25K	5,590	0.4
		pc3	36K	1,564	10
		pc4	30K	1,458	12

V. VALIDATION OF DATA

An experiment needs three things:

A. data to be processed,
B. a processing method, and
C. a reporting method.

This section discusses the data used in this study. Processing via data miners and our reporting methods are discussed later. All our data comes from the MDP. At the time of this writing, 10 data sets are available in that repository. Two of those data sets have a different format from the rest and were not used in this study. This left eight, shown in Fig. 3. Each module of each data sets describes the attributes of Table: 1 Data sets used in this study. The data sets cm1-05 and pc1-05 update data sets cm1 and pc1 processed previously by the authors [that module, plus the number of defects known for that module. This data comes from eight subsystems taken from four systems. These systems were developed in different geographical locations across North America. Within a system, the subsystems shared some a common code base but did not pass personnel or code between subsystems. Fig. 4 shows the module sizes of our data; for example,there are 126 modules in the kc4 data set; most of them are under 100 lines of code, but a few of them are more than 1,000 lines of code long. Each data set was preprocessed by removing the module identifier attribute (which is different for each row). Also, the error count column was converted into a Boolean attribute called defective? as follows:defective? =(error count >= 1) Finally, the error density column was removed (since it can be derived from line counts and error count). The preprocessed data sets had 38 attributes plus one target attribute (defective?), shown in Fig. 5, and included Halstead, McCabe, lines of code, and other miscellaneous attributes. The Halstead attributes were derived by Maurice Halstead in 1977. He argued that modules that are hard to read are more likely to be fault prone [1]. Halstead estimates reading complexity by counting the number of operators and operands in a module: See the h attributes of Fig. 5. These three raw h Halstead attributes were then used to compute the H: the eight derived Halstead attributes using the

© 2010, IJARCS All Rights Reserved

equations shown in Fig. 5. In between the raw and derived Halstead attributes are certain intermediaries (which do not appear in the MDP data sets): .->m=m1+m2 . Minimum perator count: m1* = 2, and m2* is the minimum operand count and equals the number of module parameters. An alternative to the Halstead attributes are the complexity attributes proposed by Thomas McCabe in 1976. Unlike Halstead, McCabe argued that the complexity of pathways between module symbols is more insightful than just a count of the symbols [29]. The first three lines of Fig. 5 show McCabe's three main attributes for this pathway complexity. These are defined as follows: A module is said to have a flow graph; i.e., a directed graph where each node corresponds to a program statement and each arc indicates

Table 2 Halstead and Mccabe methods

m = Mccabe		$v(g)$	cyclomatic_complexity
		$iv(G)$	design_complexity
		$ev(G)$	essential_complexity
locs	loc		loc_total (one line = one count
	loc(other)		loc_blank loc_code_and_comment loc_comments loc_executable number_of_lines (opening to closing brackets)
Halstead	h	N_1	num_operators
		N_2	num_operands
		μ_1	num_unique_operators
		μ_2	num_unique_operands
	H	N	length: $N = N_1 + N_2$
		V	volume: $V = N * log_2\mu$
		L	level: $L = V^*/V$ where $V^* = (2 + \mu_2{}^*)log_2(2 + \mu_2{}^*)$
		D	difficulty: $D = 1/L$
		I	content: $I = \hat{L} * V$ where $\hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$
		E	effort: $E = V/\hat{L}$
		B	error_est
		T	prog_time: $T = E/18$ seconds

misc = Miscellaneous	branch_count
	call_pairs
	condition_count
	decision_count
	decision_density
	design_density
	edge_count
	global_data_complexity
	global_data_density
	maintenance_severity
	modified_condition_count
	multiple_condition_count
	node_count
	normalized_cyclomatic_complexity
	parameter_count
	pathological_complexity
	percent_comments

The flow of control from one statement to another. The Table: 2 [38] cyclomatic complexity of a module is v (G) =e -n +2, where G is a program's flow graph, e is the number of arcs in the flow graph, and n is the number of nodes in the flow graph [37]. The essential complexity (ev(G)) of a module is the extent to which a flow graph can be "reduced" by decomposing all the subflowgraphs of G that are D-structured primes (also sometimes referred to as "proper one-entry one-exit subflowgraphs"[37]).

Ev(G)=v(G)_ m, where m is the number of subflowgraphs of G that are D-structured primes [37]. Finally, the design complexity (iv(G)) of a module is the cyclomatic complexity of a module's reduced flow graph. At the end of Fig. 5[38] are a set of misc attributes that are less well-defined than lines of code attributes or the Halstead and McCabe attributes. The meaning of these attributes is poorly documented in the MDP database. Indeed, they seem to be values generated from some unknown tool set that was available at the time of uploading the data into the MDP. Since there are difficulties in reproducing these attributes at other sites, an argument could be made for removing them from this study. A counterargument is that if static code attributes are as weak as suggested by Shepherd and Ince and Fenton and Pfleeger, then we should use all possible attributes in order to make maximum use of the available information. This study took a middle ground: All these attributes were passed to the learners and they determined which ones had the most information. An interesting repeated pattern in our data sets are exponential distributions in the numeric attributes. For example, Fig. 6a shows the sorted McCabe v(g) attributes from cm1. These values form an exponential distribution with many small values and a few much larger values. Elsewhere, we have conducted limited experiments suggesting that a logarithmic filter on all numeric values might improve predictor performance [36]. Such a filter replaces all numerics n with their logarithms. ln(n). The effects of such a filter are shown in Fig. 6b: The log-filtered values are now more evenly spread across the y-range, making it easier to reason about them. To test the value of logfiltering, all the data was passed through one of two filters: 1. none; i.e., no change, or 2. logNums; i.e., logarithmic filtering. To avoid numerical errors with ln(0), all numbers under 0.000001 are replaced with ln(0:000001)

VI. SOFTWARE EFFORT ESTIMATING TECHNIQUES

Barry Boehm, in his classic work on software effort models, identified the main ways of deriving estimates of software development effort as:Expert judgment, where the advice of knowledgeable staff is solicited ;Analogy, where a similar, completed, project is identified and its actual effort is used as the basis of the estimate for the new project; Parkison, which identifies the staff effort available to do a project and uses that as the estimate?

A. Top-down, where an overall estimate is formulated for the whole project which is then broken down into the effort required for component tasks;
B. Bottom –up, where component tasks are identified and sized and these individual estimates are aggregated.
C. Bottom up estimating Estimating methods can be generally divided into bottom-up and top-down approaches. With the bottom-up approach the estimator breaks the project into its component tasks and then estimates how much effort will be required to carry out each task. With a large project, the process of breaking down into tasks would be a repetitive one. Each task would be analyzed in to its component subtasks and theses would be further analyzed. It is suggested that this is repeated until you get to components that can be executed by a single person in about a week or two. The reader may wonder why this is not called a topdown

© 2010, IJARCS All Rights Reserved

approach: after all, are starting from the top and working down. Although this top-down analysis is an essential precursor to bottom-up estimating, it is really a separate one- that of producing a work breakdown schedule (WBS). The bottom-up part comes in adding up the calculated effort for each activity to get an overall estimate.

The bottom-up approach is most appropriate at the later, more detailed, stages of project planning. If this method os used early on in the project cycle then the estimator will have to make some assumptions about the characteristics of the final system, for Example the number and size of software components. These will be working assumptions that imply no commitment when it comes to the actual design of the application. Where a project is completely novel or there is no historical data available, the estimator would be forced to use the bottom-up approach.

VII. COCOMO: A Parametric Model

Boehm's COCOMO (Constructive Cost Model) is often referred to in the literature on software project management, particularly in connection with software estimating. The term COCOMO really refers to a group of models. The basic model was built around the equation (effort)=c(size)k Where effort was measured in pm or the number of 'persons-months 'consisting , size was measured in kdsi, thousands of delivered source code instructions, and c and k were constants.

A. Organic mode
B. Embedded mode
C. Semi mode

Table: 3 COCOMO model values

System type	c	k
organic	2.4	1.05
Semi-detached	3.0	1.12
embedded	3.6	1.20

(a) Basic cocomo

Effort applied=ab(kloc)^b.b[months]
Development time=cb(effort applied)^b.d[months]
People required =effort applied /evp time[count]
Intermideate cocomo:
E=ai(kloc)^b.iEAF(Effort adjucemt Factor)

VIII. DEFECT DENSITY

Defect Density (at System Testing stage)
[Total number of Defects identified during system Testimg]/Actual Size of the product

A. Defect rate

Is th expected number of defects over a certain time period specified is important for cost and resource estimates of maintainence phase of the software life cycle. It should be noted that "defect rate" and defect injection rate [(Number of in-process-defectss)+(Number of Customer-reported Defects)]/Actual size of product to define defect removal effectiveness, we must first understand the activities in the development process that are related to defect injections and defect removals. Instead of finding the defect from the overall system we shall find the defects from individual modules or subsystems for this process the following expression can be used Number of Defects removed(at the step entry)/(Number of Defects existing at step entry)+(Number of Defects injected during development)*100

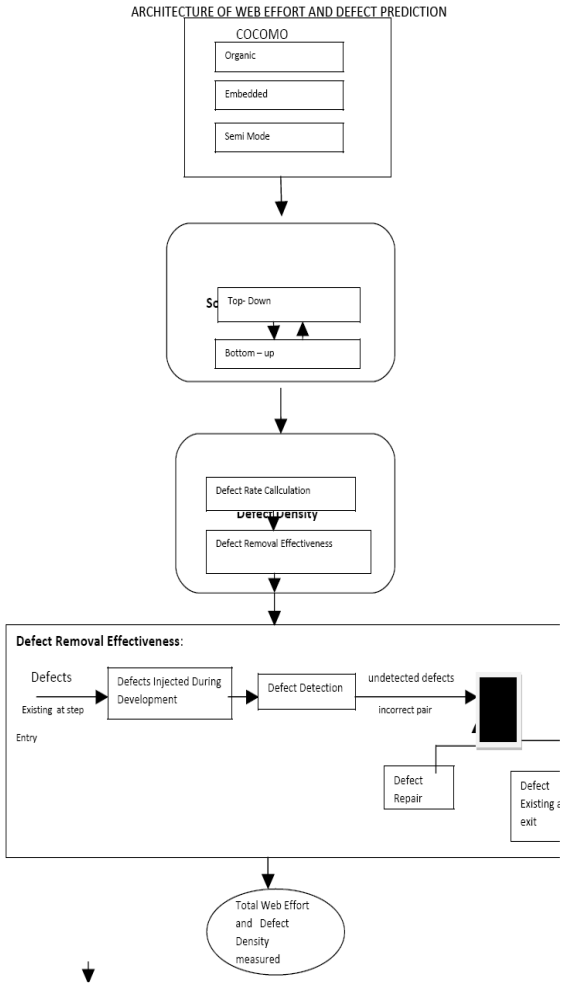


Figure 3

IX. CONCLUTION

In this paper has been fully explained with the Bayseian networks how to implement the bayseian model. With that process to check the web application effectiveness in that process with the help of the WDP database. From this process mainly defectiveness and quality checking in this process are also have been involved with COCOMO model and MCcabes and Halsted model. Then effectiveness of the process and quality control are also been checked with some equations. Finally with this paper web application project can be fully controlled and quality product can get from this method.

© 2010, IJARCS All Rights Reserved

X. ACKNOWLEDGEMENT

This work is done with the help of **Prof. Raju** is a graduate of the Indian and US universities with degrees in Physics, Electronics, Industrial Engineering and Management.

XI. REFERENCES

[1] F.V. Jensen, An Introduction to Bayesian Networks. UCL Press, 1996.
[2] B.A. Kitchenham, "A Procedure for Analysing Unbalanced Data Sets," IEEE Trans. Software Eng., vol. 24, no. 4, pp. 48-301, Apr. 1998.
[3] E. Mendes, "PredictingWeb Development Effort Using a Bayesian Network," Proc. Int'l Conf. Evaluation and Assessment in Software Eng., pp. 83- 93, 2007.
[4] E. Mendes, "The Use of a Bayesian Network for Web Effort Estimation," Proc. Seventh Int'l Conf. Web Eng., pp. 90-104, 2007.
[5] E. Mendes, "A Comparison of Techniques for Web Effort Estimation," Proc. ACM/IEEE Int'l Symp. Empirical Software Eng.,pp. 334-343, 2007.
[6] M. Neil, N. Fenton, and L. Nielsen, "Building Large-Scale Bayesian Networks," The Knowledge Eng. Rev. (KER), vol. 15, no. 3, pp. 257-54, 2000.
[7] J. Pearl, Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, 1988.
[8] O. Woodberry, A. Nicholson, K. Korb, and C. Pollino, "Parameterising Bayesian Networks," Proc. Australian Conf. Artificial Intelligence, pp. 1101- 1107, 2004
[9] M. Halstead, Elements of Software Science. Elsevier, 1977.
[10] T. McCabe, "A Complexity Measure," IEEE Trans. Software Eng., vol. 2, no. 4, pp. 308-320, Dec. 1976.
[11] N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," Proc. Int'l Conf. Software Eng., 2005
[12] T. Menzies, D. Raffo, S. Setamanit, Y. Hu, and S. Tootoonian, "Model-Based Tests of Truisms," Proc. IEEE Automated Software Eng. Conf., 2002
[13] Using Bayesian Networks to Predict Software Defects and Reliability Norman Fenton, Martin Neil, David Marquez**,** Dep. of Computer Science Queen Mary, University of London.
[14] Selecting a defect prediction model for maintenanceresource planning and software insurance Paul Luo Li, Mary Shaw, Jim Herbsleb, Carnegie Mellon University 5000 Forbes Ave Pittsburgh PA 15213 1-72-38-3043
[15] M.J. Druzdzel, A. Onisko, D. Schwartz, J.N. Dowling, and H. Wasyluk, "Knowledge Engineering for Very Large Decision- Analytic Medical Models," Proc. Ann. Meeting Am. Medical Informatics Assoc., pp. 1049-1054, 1999.
[16] S.M. Mahoney and K.B. Laskey, "Network Engineering for Complex Belief Networks," Proc. 12th Ann. Conf. Uncertainty in Artificial Intelligence, pp. 389-396, 1996.
[17] O. Woodberry, A. Nicholson, K. Korb, and C. Pollino, "Parameterising Bayesian Networks," Proc. Australian Conf. Artificial Intelligence, pp. 1101- 1107, 2004.
[18] P.C. Pendharkar, G.H. Subramanian, and J.A. Rodger, "A Probabilistic Model for Predicting Software Development Effort," IEEE Trans. Software Eng., vol. 31, no. 7, pp. 615-624, July 2005.
[19] I. Stamelos, L. Angelis, P. Dimou, and E. Sakellaris, "On the Use of Bayesian Belief Networks for the Prediction of Software Productivity," Information and Software Technology, vol. 45, no. 1, pp. 51-60(10), Jan. 2003.
[20] A.J. Knobbe and E.K.Y. Ho, "Numbers in Multi-Relational Data Mining," Proc. Ninth European Conf. Principles and Practice of Knowledge Discovery in Databases), 2005.
[21] E. Mendes, "PredictingWeb Development Effort Using a Bayesian Network," Proc. Int'l Conf. Evaluation and Assessment in Software Eng., pp. 83- 93, 2007.
[22] E. Mendes, "The Use of a Bayesian Network for Web Effort Estimation," Proc. Seventh Int'l Conf. Web Eng., pp. 90-104, 2007.
[23] E. Mendes, "A Comparison of Techniques for Web Effort Estimation," Proc. ACM/IEEE Int'l Symp. Empirical Software Eng., pp. 334-343, 2007.
[25] J. Pearl, Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, 1988.
[26] O. Woodberry, A. Nicholson, K. Korb, and C. Pollino, "Parameterising Bayesian Networks," Proc. Australian Conf. Artificial Intelligence, pp. 1101- 1107, 2004.
[27] P.C. Pendharkar, G.H. Subramanian, and J.A. Rodger, "A Probabilistic Model for Predicting Software Development Effort," IEEE Trans. Software Eng., vol. 31, no. 7, pp. 615-624, July 2005.
[28] M. Halstead, Elements of Software Science. Elsevier, 1977.
[29] T. McCabe, "A Complexity Measure," IEEE Trans. Software Eng., vol. 2, no. 4, pp. 308-320, Dec. 1976.
[30] N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," Proc. Int'l Conf. Software Eng., 2005.
[31] F. Shull, V.B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What We Have Learned About Fighting Defects," Proc. Eighth Int'l Software Metrics Symp., pp. 249-258, 2002.
[32] M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems J., vol. 15, no. 3, 1976
[33] T. Menzies, D. Raffo, S. Setamanit, Y. Hu, and S. Tootoonian, "Model-Based Tests of Truisms," Proc. IEEE Automated Software Eng. Conf., 2002.
[34] C. Blake and C. Merz, "UCI Repository of Machine Learning Databases," http://www.ics.uci.edu/~mlearn/MLRepository html, 1998.
[35] T. Menzies, J.D. Stefano, K. Ammar, K. McGill, P. Callis, R. Chapman, and J. Davis, "When Can We Test Less?" Proc. IEEE Software Metrics Symp., 2003.
[36] T. Menzies, J.S. DiStefeno, M. Chapman, and K. Mcgill, "Metrics that Matter," Proc. 27th NASA SEL Workshop Software Eng., 2002.. [37]N.E. Fenton and S. Pfleeger, Software Metrics: A Rigorous and Practical Approach, second ed. Int'l Thompson Press, 1995.
[38] Data Mining Static Code Attributes to Learn Defect Predictors

© 2010, IJARCS All Rights Reserved

[39] Bayesian Network Models for Web Effort Prediction: A Comparative Study Emilia Mendes and Nile Mosley

Book: Software quality assurance, principles and practice by: Nina S Godbole

© 2010, IJARCS All Rights Reserved