A Bayesian Belief Network for Modeling Open Source Software Maintenance Productivity

Stamatia Bibi, Apostolos Ampatzoglou, Ioannis Stamelos

A Bayesian Belief Network for Modeling Open Source Software Maintenance Productivity

Stamatia Bibi[1], Apostolos Ampatzoglou[2], and Ioannis Stamelos[3]

[1] Department of Informatics & Telecommunications, University of Western Macedonia, Greece
[2] Department of Computer Science, University of Groningen, Netherlands
[3] Department of Computer Science, Aristotle University of Thessaloniki, Greece
sbibi@uowm.gr, a.ampatzoglou@rug.nl, stamelos@csd.auth.gr

Abstract. Maintenance is one of the most effort consuming activities in the software development lifecycle. Efficient maintenance within short release cycles depends highly on the underlying source code structure, in the sense that complex modules are more difficult to maintain. In this paper we attempt to unveil and discuss relationships between maintenance productivity, the structural quality of the source code and process metrics like the type of a release and the number of downloads. To achieve this goal, we developed a Bayesian Belief Network (BBN) involving several maintainability predictors and three managerial indices for maintenance (i.e., duration, production, and productivity) on 20 open source software projects. The results suggest that maintenance duration depends on inheritance, coupling, and process metrics. On the other hand maintenance production and productivity depend mostly on code quality metrics.

Keywords: maintenance; productivity; software metrics; Bayesian networks.

1 Introduction

Software maintenance is, according to van Vliet [27], one of the most effort and time intensive activities in the software lifecycle, since it consumes 50-75% of the overall project resources. According to ISO 14764-2006 the most common maintenance activities include the adaptation of a system to new environments, the implementation of additional requirements, the improvement of run- or design-time quality properties, or the identification of latent defects. It is expected that the worse the internal quality of a system is, the more difficult to maintain the software will be [2]. However, there are several precautionary actions that managers can take (e.g., software refactorings [10]) so as to improve the structural quality of the software, which in turn will lead to decreased maintenance effort and cost. Nevertheless, the time budget that is usually allocated on such maintenance activities is usually limited. Thus, it should be cautiously allocated to the modules that suffer from the most important bad smells [10].

In this study we focus on the relationship between structural software quality characteristics (expressed through metrics), maintenance production (i.e., software units

maintained) and duration. Although we acknowledge the fact that structural quality is not the only parameter that should be used in such a model (i.e., a complete model should consider the number of changing requirements, number of defects, etc.), we isolate software structure, in order to explore its influence exclusively. In classical economics production and duration are combined through the measure of productivity, which is defined as an average measure of production in the unit of time [6]. In this line of thought, we define maintenance productivity as the average lines of code that are being maintained (*software units maintained*) in a given time unit. Although the exploration of the relationship between software units maintained and structural quality measures is not novel (see Section 2), to the best of our knowledge this is the first study that considers maintenance productivity. To achieve this goal we used a Bayesian Belief Network (BBN) to model the relationships between the input (*structural software metrics*) and output parameters (*maintenance production*, *maintenance duration*, and *maintenance productivity*). The network has been trained through a case study on 454 versions of 20 Open Source Software (OSS). The rest of the paper is organized as follows: Section 2 presents research efforts on studies related to maintenance production; Section 3 presents background information on BBN and the metrics that have been used in the developed network; Section 4 presents the case study design; in Sections 5, we present and discuss our results and the implications to the researchers and practitioners; in Section 6 we present threats to validity and finally, and in Section 7 we conclude the paper.

2 Related Work

Research on software maintenance effort can be categorized into two groups: (a) development of effort estimation models, and (b) development of methods that improve the maintenance outcomes.

Among popular effort estimation models we can find Estimation by Analogy (EbA) and estimation based on Regression. Shepperd et al. [24] suggested EbA for allocating staff in software maintenance projects based on similar completed historical maintenance projects. De Lucia et al. [16] use multiple regression analysis to construct corrective maintenance effort estimation models. The results show that the performance of the model is improved when the different types of maintenance tasks are included. The use of function points are proposed for estimating the effort of software maintenance projects [1]. Bayesian Networks are also used in Software Maintenance [26]. Van Koten and Gray [26] use Bayesian Networks to predict the maintainability measured as the change in software lines of code between subsequent releases. The results are improved compared to regression models. In [19] BBNs aim to predict delays in software maintenance tasks. Bayesian Networks have been also applied in software fault prediction [9] and cost estimation [25]. The application of the method indicated certain advantages regarding their ability to be combined with expert judgement and provide flexible and informative estimates.

On the other hand there are studies found in literature that examine the factors that affect software maintenance effort. System size, system age, number of input/output data items, application type, and programming language are among the parameters that can affect software maintenance effort according to [1] and [12]. Also there is a

vast literature on code metrics that affect the maintainability of software and consequently the corresponding effort required. According to Riaz et al. [22] the metric suites proposed by Li and Henry [15] and Chidamber and Kemerer [8] are the most accurate ones for assessing software maintainability. On the other hand there are studies that explore the specific parameters that affect open source project development, quality and evolution [14], [20], [21]. Developer participation, core team size, code ownership, productivity, defect density, and problem resolution intervals were examined in these studies leading to the conclusion among others that defect density and productivity benefit from the large open source community of testers and bug fixers. Speed of releases seems to require highly modularized software. To the best of our knowledge this is the first study that considers maintenance productivity.

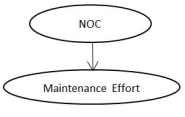
3 Background Information

In this section we present some background information that is needed to comprehend this study. In particular, we provide information related to *Bayesian Belief Networks*, and *structural quality metrics* associated to maintainability.

Bayesian Belief Networks are Directed Acyclic Graphs (DAGs), which are causal networks that consist of a set of nodes and a set of directed links between them, in a way that they do not form a cycle [13]. Each node represents a random variable that can take mutually exclusive values according to a probability distribution, which can be different for each node. Each link expresses probabilistic cause-effect relations among the linked variables and is depicted by an arc starting from the influencing variable (parent node) and terminating on the influenced variable (child node). The relation between the two nodes is based on Bayes' Rule:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \qquad (1)$$

A simple BBN example is presented in Figure 1. The model consists of two nodes. The first node (NOC) represents the number of classes in a software package and the second node (Maintenance Effort) represents the effort required for package maintenance measured in Person- Months widely reported in literature as Man-Months (MMs). We consider that the values of these two nodes fall into two discrete categories (Low and High). For the node NOC, let's suppose that *Low values* range between 1 class and 10 classes (Similarly for Maintenance Effort values). For example the first column of the Node Probability Table states that if the number of classes is low then there is 70% probability that the maintenance effort will be low and 30% percent probability that the maintenance effort will be high. The algorithm used for learning Bayesian Networks is analytically presented in [7].



NOC (Number of classes)		Low (1-10 classes)	High (10-30 classes)
Maintenance Effort	Low (1-3MM)	0.7	0.45
	High(>3MM)	0.3	0.55

Fig. 1. a. A BBN for maintenance effort **b**. Node Probability table for Fig 1a

The tool used for constructing the networks and the classifier can be found in the web (http://www.cs.ualberta.ca/~jcheng/bnpc.htm).

Maintainability Predictors: In this study we focus on metrics that are the most accurate software maintainability predictors. According to Riaz et al. [22] the metric suites proposed by Li and Henry [15] and Chidamber and Kemerer [8] are the most accurate ones, while the model of van Koten and Gray [26] is the most stable one [22]. The metric suite that is used from the aforementioned study is presented in Table 1. In order to automate the calculation of software quality metrics we used Percerons Client (http://www.percerons.com), i.e., a tool developed in our research group.

Table 1. Maintainability Predictors

Metric / Description	Phase	Quality Attribute (QA)
Depth of Inheritence Tree (DIT). Inheritence level number, 0 for the root class.	design	inheritance
Number of Children Classes (NoCC). Number of sub-classes that the class has.	design	inheritance
Message Passing Coupling (MPC). Number of send statements defined in the class.	code	coupling
Response For a Class (RFC). Number of local methods plus the number of methods called by local methods in the class.	code	coupling
Lack of Cohesion of Methods (LCOM). Number of disjoint sets of methods (number of sets of methods that do not interact with each other), in the class.	code	cohesion
Data Abstraction Coupling (DAC). Number of abstract types defined in the class.	design	coupling
Weighted Method per Class (WMC). Average cyclomatic complexity of all methods.	code	complexity
Number of Methods (NOM). Number of methods in the class.	design	size
Lines of Code (SIZE1/LoC). Number of semicolons in the class.	code	size
Number of Properties (SIZE2). Number of attributes and methods in the class	design	size
Coupling Between Objects (CBO). Number of classes that one class depends on	design	coupling
Average Method Size (AMS). Avereage number of semicolons in a method	code	complexity

4 Case Study Design

In order to investigate the influence of structural quality in several managerial maintenance indices (i.e., duration, effort and productivity), we performed a case study on 20 open-source software (OSS) projects. The case study is designed and reported according to the guidelines of Runeson et al. [23].

4.1 Objectives and Research Questions

The objective of this study is to analyze software metrics and investigate their influence on the maintenance production, duration and productivity. In particular we investigate three research questions:

RQ_1: *Which quality metrics are related to the duration of maintenance among two successive releases?*
This question aims at identifying the metrics that mostly influence the time required for developing a new version of an OSS project measured in days

elapsed from one release to another. Quick successive releases may indicate the level of readiness of the project community, the tendency to launch quick releases, and short time to fix bugs or add functionality.

RQ_2: *Which quality metrics are related to the production of maintenance among two successive releases?*
The objective of this question is to reveal which quality metrics are more associated to the production of the maintenance in OSS, measured as the number of lines of code added from previous release. The target of this analysis is to identify the quality characteristics whose levels enable a substantial change in a single version. We expect that low values of coupling and complexity, and high values of cohesion, will yield for projects in which maintenance production is large and efficient.

RQ_3: *Which quality metrics are related to the productivity of maintenance among two successive releases?*
This research question is relevant to productivity. Productivity is calculated as the ratio between production and duration. We note that this calculation of productivity deviates from the typical one, because duration in OSS development might include idle time, when developers are inactive. However, to the best of our knowledge, the actual development time cannot be retrieved from source code repositories. The objective is to confirm if high levels of quality lead to increased productivity, and to some extent eliminate the bias caused by change load.

4.2 Case Selection & Data Processing

To collect subjects for our case study we retrieved a set of popular and active OSS projects. In particular, we selected only projects that were active in 2014 (i.e., published at least one version), and when sorted by popularity (based on the default sourceforge.net measure) they were among the top ranked ones. Additionally, we selected projects containing more than 300 classes to ensure that sufficient units of analysis will participate in the study and that toy-applications would be excluded. Therefore, we downloaded and collected metrics data from 20 popular OSS projects so as to build a dataset that is adequate for statistical analysis. Some demographics for these projects are presented in Table 2.

Table 2. Case Study Subjects

Name	NOC	Versions	Cases	Name	NOC	Versions	Cases
AoI	749	31	27	iText	645	22	20
Azureus	3,888	24	22	Java Game Library	654	40	33
Checkstyle	1,186	32	16	ZDF MediaThek	617	40	33
Dr Java	3,464	55	43	Pixelator	827	33	33
File Bot	7,466	20	17	Mondrian	1,471	32	25
FreeCol	794	40	33	Open Rocket	3,018	26	21
FreeMind	443	41	35	Subsonic	4,688	40	11
Hibernate	3,821	48	31	Sweet Home 3D	341	24	17
Home Player	457	30	25	UMS	5,499	50	9
Html Unit	920	27	25	Tux Guitar	745	17	11

The case study is an embedded multiple-case study, in the sense that for every OSS project (i.e., *case*), we analyze every transition between two versions (i.e., *units of analysis*) [23]. For each release of the projects presented in Table 2, we collected three sets of data: the structural quality metrics presented in Table 1(see Section 3), the targeted maintenance managerial indices (see Section 4.1), and some process metrics. The obtained process metrics include the *date* each release was uploaded, the sequential *number of release* that indicates the maturity of the project (i.e., if we are in the beginning of the project lifecycle or not), and the *number of downloads* recorded for each release (considering the previous version each time) from the date launched to 01/01/2015. At this point the resulting project releases were further filtered to exclude releases that presented negative values in the production variable (deletion of functionality) and releases that presented zero duration time (releases uploaded all at once). After this pre-processing phase the remaining project releases were selected for analysis. Descriptive statistics for all dependent variables of our analysis are presented in Table 3.

Table 3. Descriptive Statistics for Maintenance Indices

Maintenance Index	N	Mean	Minimum	Maximum	Std. Dev.
Production (Δ_{LOC})	454	2629.16	1	162429	10748.11
Duration (in days)	454	79.85	1	1055	91.6
Productivity	454	76.03	0.01	9962	522.47

To apply the BBN analysis we had to further process the gathered metrics, since quality metrics present continuous arithmetic values. In particular, continuous values had to be transformed to categorical ones, so as to be utilized in BBN analysis. The transformation method used was the *equal frequency binning* [28]. This method automatically sets the boundaries of each bin (category), so as to ensure that all of them have an equal number of observations. Each metric that is represented in a categorical form has five distinct categories (VL—Very Low, L—Low, A—Average, H—High, and VH—Very High).

4.3 Data Analysis

On the completion of the aforementioned process we applied the Bayesian Network analysis and utilized heatmaps to graphically represent our data. Our goal was to better depict the influence of the independent variables on the dependent ones.

The first step of the analysis includes the selection and tuning of the variables that affect the structure of the BBN. In this step expert input is required to set the order of the variables that will affect the causal relationships that the model will define. Metrics order was selected taking into account two considerations: (a) the ***development phase***, when the values of the metrics will be available in the sense that we first define certain design metrics, while some time afterwards certain source code metrics are available, and (b) the ***relatedness*** of metrics, based on their calculation method. For example coupling metrics like CBO and MPC are closely related. In the model CBO precedes MPC, in the sense that the value of CBO "includes" the value of MPC.

After extracting the model we employed heatmaps to visualize the obtained outcome. Heatmaps are graphical representations that use color intensity in order to de-

note occurrence frequency. While applying heatmaps to our results set, we developed a matrix, in which row represents a value of the dependent variable (e.g., *low maintenance duration*) and a column a value of the independent variable (e.g., *systems with a very low level of polymorphism*). Each cell is a percentage that shows the probability that the independent variable will present a certain value according to the value of the dependent variable (e.g., *10% of the cases with a very low level of polymorphism are maintained very quickly— low maintenance duration*).

5 Results & Discussion

In this section we present the results obtained from the BBN analysis, organized by research question. The extracted network including all structural quality metrics, process measures, and managerial maintenance indices is presented in Figure 2. We note that since this study aims to investigate the interdependencies between the nodes of the graph, the estimation accuracy of the network is not explored. A discussion / interpretation of the most important relationships presented in the network is provided later in this section, while answering each research question.

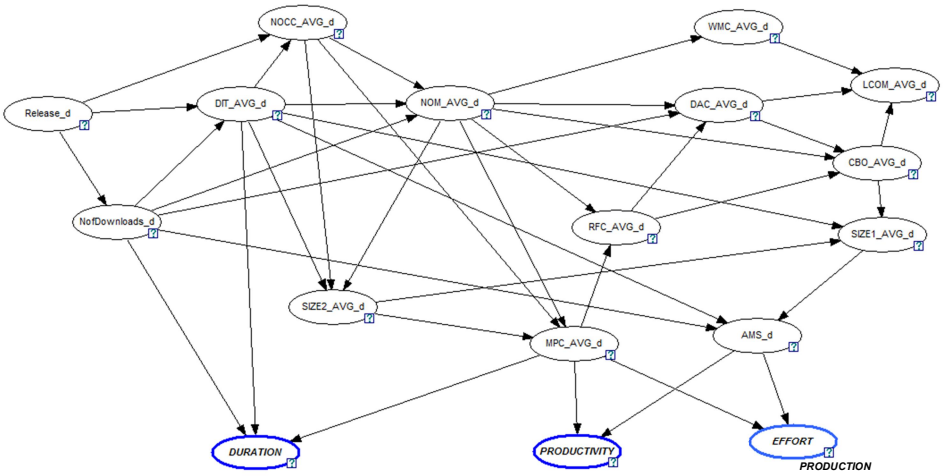


Figure 2. Maintenance Indices Bayesian Belief Network

5.1 Duration

In this section we identify the structural properties that are related to maintenance duration (RQ_1). From Figure 2, we can observe that variables that are directly connected to *DURATION* are *Number of downloads, Depth of Inheritance Tree* (DIT), and *Message Passing Coupling* (MPC). The heatmaps visualizing the relationships of these variables to the duration of the maintenance activities is presented in Table 4.
By interpreting the left-most part of the heatmap, we can observe that as the ***number of downloads*** *increases*, the ***maintenance duration*** becomes *larger*. This result can be intuitively interpreted by the fact that projects with many downloads are expected to serve larger communities, leading to the creation of more feature requests and bug fixing activities. The number of extra requirements that are requested in each mainte-

nance cycle increases their duration. Based on the central part of the heatmap, our results suggest that an *average* ***depth of inheritance tree*** offers *shorter* ***maintenance cycles***. On the other hand, *extensive or limited* ***inheritance*** leads to *larger* ***maintenance cycles***. Again, this can be characterized as an expected result, in the sense that: (a) using very large inheritance trees decreases software understandability since the full list of accessible variables and metrics from a class can spread to many classes, and (b) using limited inheritance does not make use of one of the most important benefits from using the object-oriented paradigm, and at the same time hinders the applicability of well-known object-oriented principles (e.g., open-closed principle [17]) and design patterns (e.g., template method, state/strategy, etc. [11]) that have been reported as beneficial to maintainability.

Table 4. Parameters influencing maintenance duration

	#Downloads					DIT					MPC				
Duration	VS	S	A	H	VH	VS	S	A	H	VH	VS	S	A	H	VH
Very Small	28	23	18	16	15	19	16	24	17	20	22	19	22	21	16
Small	21	21	18	21	18	18	18	24	21	18	17	19	25	20	19
Average	15	24	22	20	19	22	23	19	21	19	18	24	17	21	18
High	18	17	22	21	23	23	22	18	18	20	19	24	17	19	23
Very High	18	15	20	22	25	18	21	15	23	23	24	14	19	19	24

Finally, *very high* ***coupling*** leads to *high* or *very high* ***maintenance duration*** (see right-most part of the heatmap). This result is expected in the sense that according to the high coupling principle [17] dense inter-connection of classes reduces their understandability, increases the probability of initiating a ripple effect, and reduces their changeability. On the other hand, by focusing on *very small* ***coupling*** we get interesting results. Intuitively one would expect that very small coupling would increase modularity (low ripple effect, increase changeability) and therefore be connected to quick maintenance cycles, but this is the 2nd most frequent event. The most frequent event is that very small coupling is connected to large maintenance cycles. This result, although initially surprising, can be interpreted by the inherent relationship/trade-off between coupling and cohesion (as it is confirmed by the network of Figure 2). Specifically, it is expected that low coupling can also lead to bad design, in cases that specific classes are growing so large that they do not need to call methods from other classes (they are self-contained). This leads to reduced coupling, but probably these classes are assigned to more than one responsibility. This decision overrules the Single Responsibility Principle, which according to Martin [17], in turn leads to maintainability problems, especially focused on understandability and change proneness.

5.2 Production

In this section, we explore the BBN for mining relationship related to RQ_2, i.e., metrics that are related to maintenance production, measured as lines of code added from one version to the other. Similarly to Section 5.1, in Table 5, we present the heatmap for the two variables that are directly related to maintenance production. Interestingly, we can observe that the two most important parameters are: one complexity metric (namely AMS) and the same coupling metric as in Table 4 (namely MPC). This is considered important from two perspectives:

(a) Average Method Size (AMS) has, until now, not been explored as a possible maintainability predictor, although it appears to be related to maintenance production. However this result can be considered intuitive, in the sense that the size of the method is related to one the most persistent and frequently occurring bad smells, namely *Long Method* [10]. According to the literature methods of large size are very difficult to understand, modify and extend, leading to difficulties in increasing maintenance production.

(b) Message Passing Coupling (MPC) is consistently the coupling metric that can be used as the optimal maintainability index. The superiority of MPC with regard to assessing ripple effects (a significant aspect of maintainability) compared to the other examined coupling metrics is also discussed by Arvanitou et al. [5]. In particular MPC is the only (among the investigated metrics) that captures both coupling volume (number of relationships) and coupling intensity (how closely connected the two classes are). An additional important characteristic of MPC is that it counts coupling intensity using the discrete count function, and therefore is not biased from the number of times one method is being called [4].

Table 5. Parameters influencing maintenance production

Production	**AMS**					**MPC**				
	VS	S	A	H	VH	VS	S	A	H	VH
Very Small	16	27	25	20	13	15	22	23	17	22
Small	21	23	18	21	17	25	20	25	14	17
Average	21	17	15	21	26	23	21	19	21	16
High	24	15	17	22	22	20	19	15	23	24
Very High	19	16	24	20	20	18	18	19	24	21

5.3 Productivity

Similarly to RQ_1 and RQ_2, in Table 6, we present the heatmap that visualizes the parameters that are more closely related to maintenance productivity. By comparing the results of Table 6, with those of Tables 4 and 5, we can observe that productivity is more closely related to maintenance production rather than maintenance duration.

Table 6. Parameters influencing maintenance productivity

	AMS					**MPC**				
Productivity	VS	S	A	H	VH	VS	S	A	H	VH
Very Small	19	26	21	20	14	16	27	22	13	23
Small	19	22	19	19	21	23	23	22	14	18
Average	23	20	14	22	20	19	17	24	18	22
High	17	12	16	28	27	27	15	13	28	17
Very High	25	16	28	14	17	15	18	22	28	18

In particular we observe that the only metric that is consistently among the optimal maintenance predictors is MPC, whereas AMS is only related to maintenance production and productivity. Concerning the range of values for these variables we can reach the following conclusions: According to Martin [23], the average method size should not exceed 5 lines of code. Based on the findings of this case study, very small methods provide a maximum productivity in 42% of the cases, which is a rather intuitive result. The counter point, as discussed by Fowler [10] (i.e., long methods are hard to

maintain) cannot be discussed based on our results, since in order for such an investigation to be possible, it would be required to count the number of long method bad smells and not an aggregated/average measure. Concerning MPC, we can observe that tightly coupled systems are exhibiting small and very small productivity rates—a result that is expected based on the negative effect of coupling on maintainability.

5.4 Implications to Practitioners & Researchers

The results of our case study have provided some interesting insights and implications to both practitioners and researchers. Concerning practitioners, our study provides evidence that indeed structural quality metrics are affecting maintenance indices. However, from the vast amount of metrics that are available, only a limited number of them are highly influential. As expected the most significant metrics cover the most frequently quantified quality attributes, directly or indirectly, i.e., complexity, cohesion, coupling, and inheritance. In particular, the study pointed out that practitioners should pay attention on balancing the values of the following metrics: Average Method Size (AMS), Message Passing Coupling (MPC), and Depth of Inheritance Tree (DIT). Thus, practitioners should not only include these metrics in their quality dashboards, but also manage their natural trade-off (e.g., optimizing coupling on the expense of cohesion).

Furthermore, the outcome of this study indicated some interesting future research directions. First a replication of the case study can be performed including metrics of different categories and suites (e.g., architecture level) and also metrics that describe the communication and message exchange of the OSS developers' community. Secondly, the development of a complete estimation model that additionally takes into account factors other than software structure is expected to fully capture and predict all aspects of software maintenance productivity. Finally, tailoring the proposed model in an industrial context is necessary for the adoption of such approaches in closed source software development.

6 Threats to validity

In this section we discuss the threats to validity of our case study, based on the categorization described by Runeson et al. [23]. With regard to ***internal validity***, we need to clarify that the relationships obtained through the network are influenced by some confounding factors. For example, the load of high-level changes from one version of the system to another (e.g., new features, number of fixed bugs, etc.) is not considered in this study. However, we believe that the large size of our dataset imposed a normal distribution of actual changes that minimizes the effect of the confounding factor. Additionally, we remind that the investigation of the strength of relationships is out of the scope of this study, since it only aims at their identification.

Concerning ***external validity***, we have identified two threats to generalization. First, since all the examined projects come from one source code repository the results cannot be generalized to the OSS population. Second, since we only explored Java projects (due to the limitation of the used tool) the results cannot be generalized to other programming languages. However, we believe that projects popularity and size are indicative factors of successful OSS projects.

Regarding ***reliability***, based on the fact that our study is purely quantitative and the developed protocol (see Section 4) is thoroughly described, we believe that the process is easily replicable by other researchers. Though, concerning ***construct validity***, Bayesian network model construction includes by its nature many subjective decisions made by the modeler. The parameters of the model such as the selection of the binning method for the discretization of the values of the continuous variables and the ordering of the independent variables may also affect the model extracted. This provides an interesting direction for future studies that could include additionally sensitivity analysis regarding the parameters of the model.

7 Conclusions

Long-term monitoring of maintenance effort requires a measurement process of respective attributes that affect software maintenance and a model to represent the interrelations of the attributes gathered. Open source software is a suitable paradigm of continuous, sustainable maintenance efforts. Understanding why each system requires more or less maintenance effort is necessary for better monitoring software maintenance process and releases launch. In this study we have performed an empirical study on the factors that affect software maintenance duration, production and productivity on 20 open source software projects. We suggest Bayesian Networks as a tool to model all attributes and factors that can affect maintenance efforts. The created model included structural quality and process metrics, and managerial indices. The results indicate that maintenance *duration* is affected by the popularity of a project and *DIT* (a design metric) and *MPC* (a source code metric). On the other hand Production and Productivity are solely affected by quality metrics *AMS* and *MPC*.

References

1. A. Abran, H. Nguyenkim, "Measurement of the Maintenance Process from a Demand-based Perspective," *Journal of Software Maintenance: Research and Practice*, 5 (2), 1993.
2. A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The Financial Aspect of Managing Technical Debt: A Systematic Literature Review", *Information and Software Technology*, Elsevier, 64 (8), pp. 52 – 73, August 2015.
3. A. Ampatzoglou, O. Michou, and I. Stamelos, "Building and mining a repository of design pattern instances: Practical and research benefits", *Entertainment Computing*, Elsevier, 4 (2), pp. 131–142, April 2013.
4. A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou, and P. Avgeriou, "The Effect of GoF Design Patterns on Stability: A Case Study", *Transactions on Software Engineering*, IEEE Computer Society, 41 (8), pp. 781 – 802, August 2015.
5. E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Introducing a Ripple Effect Measure: A Theoretical and Empirical Validation", *9th International Symposium on Empirical Software Engineering and Measurement (ESEM' 15)*, IEEE Computer Society, 22-23 October 2015, Beijing, China.
6. B. W. Boehm. Software Engineering Economics (1st ed.). *Prentice Hall PTR*, 1981.
7. J. Cheng, R. Greiner, Learning Bayesian Belief Network Classifiers: Algorithms and System. *14th Canadian conference on artificial intelligence (AI'2001)*, 2001.
8. S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis", *Transactions on Software Engineering*,

IEEE Computer Society, 24 (8), pp. 629-639, August 1998.

9. N. Fenton, M., Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause and R. Mishra, "Predicting software defects in varying development lifecycles using Bayesian Nets", *Information and Software Technology*, 49 (1), January 2007,pp. 32-43
10. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code", *Addison-Wesley Professional*, 1st Edition, July 1999
11. E. Gamma, R. Helms, R. Johnson, and J. Vlissides, "Design patterns: elements of reusable Object-Oriented software", *Addison-Wesley Professional*, 1995
12. M. Ghods, K.M. Nelson, "Contributors to quality during software maintenance", *Decision Support Systems*, 23 (4), pp. 361–369, October 1998.
13. F. Jensen, "Bayesian Networks and Decision Graphs", *Springer*, 2002.
14. S. Koch, and C. Neumann. "Exploring the effects of process characteristics on product quality in open source software development." Principle Advancements in Database Management Technologies: New Applications and Frameworks (2009): 132.
15. W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, Elsevier, 23 (2), pp. 111-122, November 1993.
16. A. de Lucia, E. Pompella, S. Stefanucci, "Assessing effort estimation models for corrective software maintenance through empirical studies", *Information and Software Technology*, Elsevier, 47 (1), pp. 3–15, 2005.
17. R.C. Martin "Agile software development: principles, patterns and practices", *Prentice Hall*, New Jersey. 2003
18. R. C. Martin. 2008. "Clean Code: A Handbook of Agile Software Craftsmanship", *Prentice Hall PTR*, 1st Edition, Upper Saddle River, NJ, USA.
19. A. C. V. de Melo and A. de J. Sanchez. "Bayesian networks in software maintenance management", *31st International Conference on Theory and Practice of Computer Science (SOFSEM'05)*, pp. 394-398, 2005.
20. V. Midha and A. Bhattacherjee, "Governance practices and software maintenance: A study of open source projects", *Decision Support Systems*, 54 (1), pp. 23-32, December 2012.
21. A. Mockus, R. Fielding, J. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", *Transactions on Software Engineering and Methodology*, ACM, 11 (3), pp. 309–346, 2002.
22. M. Riaz, E. Mendes, and E. Tempero, "A systematic review on software maintainability prediction and metrics", *3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09)*, IEEE Computer Society, pp. 367-377 ,2009, USA.
23. P. Runeson, M. Host, A. Rainer, and B. Regnell, "Case Study Research in Software Engineering: Guidelines and Examples", *John Wiley & Sons*, 2012.
24. M. Shepperd, C. Schofield, B. Kitchenham, "Effort estimation using analogy", *18th International Conference on Software Engineering* (ICSE' 96), ACM, pp. 170–17, 1996.
25. I. Stamelos, L. Angelis, P. Dimou, E. Sakellaris, "On the use of Bayesian belief networks for the prediction of software productivity", *Information and Software Technology*, Elsevier, 45, pp. 51-60, 2002.
26. A. van Koten and A.R. Gray, "An Application of Bayesian Network for Predicting Object - Oriented Software Maintainability", *Information and Software Technology*, Elsevier, 48 (1), pp. 59 – 67, January 2006.
27. H. van Vliet, "Software Engineering: Principles and Practice", *John Wiley & Sons*, 2008.
28. I. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques", *Morgan Kaufmann*, 2nd Edition, 2005.