A Quantitative Approach for Evaluating Software Maintenance Services

Humberto Marques-Neto
Dept. of Computer Science
Pontifical Catholic University
of Minas Gerais (PUC Minas)
Belo Horizonte – Brazil
30.535-901
humberto@pucminas.br

Gladston J. Aparecido
Dept. of Computer Science
Pontifical Catholic University
of Minas Gerais (PUC Minas)
Belo Horizonte – Brazil
30.535-901
gladston.aparecido@
sga.pucminas.br

Marco Tulio Valente
Dept. of Computer Science
Universidade Federal de
Minas Gerais (UFMG)
Belo Horizonte – Brazil
31.270-010
mtov@dcc.ufmg.br

ABSTRACT

The ever-increasing representativeness of software maintenance in the daily effort of software team requires initiatives for enhancing the activities accomplished to provide a good service for users who request a software improvement. This article presents a quantitative approach for evaluating software maintenance services based on cluster analysis techniques. The proposed approach provides a compact characterization of the services delivered by a maintenance organization, including characteristics such as service, waiting, and queue time. The ultimate goal is to help organizations to better understand, manage, and improve their current software maintenance process. We also report in this paper the usage of the proposed approach in a medium-sized organization throughout 2010. This case study shows that 72 software maintenance requests can be grouped in seven distinct clusters containing requests with similar characteristics. The in-depth analysis of the clusters found with our approach can foster the understanding of the nature of the requests and, consequently, it may improve the process followed by the software maintenance team.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Measurement

Keywords

Software Maintenance, Quantitative Software Evaluation, Maintenance Management and Measurement, Maintenance Planning, Maintenance Process

SAC'13 March 18-22, 2013, Coimbra, Portugal.

1. INTRODUCTION

Software maintenance is an important, costly, and complex phase in the software lifecycle [10]. Some works [13, 17] point out that software maintenance activities can represent 40-80% of all effort spent during this lifecycle. Thus, planning and managing software maintenance is a challenging task for most software organizations because, while the initial phases of the software lifecycle target the development of new products, software maintenance is better described in terms of a service provided to the users of existing software [2, 11].

According to this view, the users are customers that request changes to software systems, including both corrections and enhancements. Such incoming requests are processed by the supplier organization (i.e. the maintainer), typically using an adaptation of a software development process. The ultimate goal is to deliver a new release of the target system that implements the changes requested by users.

Once maintenance is viewed as a service, we can leverage from the vast theory proposed to study queue systems in order to model and evaluate software maintenance processes. Particularly, in this article we present a quantitative approach that adapts to software maintenance a methodology originally proposed to characterize the workload of e-commerce services [9]. Our adaptation relies on concepts such as waiting time and service time (i.e. the queue time) to evaluate the quality of the services provided by software maintenance organizations.

More precisely, our approach requires the representation of each maintenance request as a state transition graph, called Maintenance Model Graph (MMG). This graph models the workflow followed by the maintainer to process the incoming change requests, including information about the starting and finishing dates of each maintenance activity (such as planning, design, implementation, testing etc). Finally, the proposed approach uses a clustering algorithm to generate a representative model for the requests processed by the maintainer in a given time frame. Therefore, instead of having to analyze a full array of requests, software maintenance managers can rely on a compact representation including only requests that are very similar in terms of some characteristics, such as their waiting and service times.

In order to illustrate the usage of the proposed approach, we present the results of a case study which shows its application in a real dataset of software maintenance requests of a

medium-sized organization. We analyzed seven MMGs clustered from a set of 72 software maintenance requests handled by the Information Technology Division of an Brazilian university in 2010. We show the similar characteristics of requests which were grouped together, i.e the average hours for the waiting time, service time, and queue time of each cluster and we also present the difference among the found clusters in order to help the software maintenance team to understand the nature of the requests posted by their users.

The remainder of this paper is organized as follows. We present the quantitative approach for evaluating software maintenance services in Section 2. A case study of its usage is discussed in Section 3. Related work is pointed out in Section 4 and the final remarks are offered in Section 5.

2. THE PROPOSED APPROACH

The proposed approach for evaluating software maintenance services has the following major steps: MMG Definition (Subsection 2.1), MMG Instantiation (Subsection 2.2), and MMG Clusterization (Subsection 2.3).

2.1 Defining the Maintenance Model Graphs

To apply the proposed approach, the first task is to define the Maintenance Model Graph (MMG). The MMG is a directed graph representing the workflow followed to process the requests of software maintenance. More specifically, the nodes in a MMG represent the possible states of a request and the edges represent the transitions between such states. Usually, there are three types of states: (a) states associated to the software engineering activities used to process a request (e.g. request under analysis/planning, request under implementation, request under validation etc); (b) waiting states (e.g. request waiting for analysis/planning, request waiting for implementation etc); and (c) final states (e.g. request successfully deployed or request canceled).

Figure 1 presents a typical MMG for a maintenance organization where the following engineering activities are sequentially performed to process a request: planning, implementation, validation, and deployment. More specifically, the MMG presented in this figure expresses that when a client requests a maintenance, this request remains in a waiting state (State 1). After the planning and analysis activities have been performed (State 2), the request can be suspended (State 3), canceled (State 4), or it can move to the implementation phase (State 5). After implementation and in case of failures, the request can be suspended (State 6) or canceled (State 4). Otherwise, it moves to validation (State 7). After validation, the request can return to the planning state (State 2) or move to deployment (States 9 and 10). Finally, the whole process finishes and the maintenance is considered deployed (State 11).

2.2 Instantiating the Maintenance Model Graphs

In the second step of our approach, a single MMG is created for each request considered in the time frame of the evaluation. The states of such MMG instances should be decorated with the following information (normally extracted from an issue tracking system):

- Starting and ending timestamps, i.e. information about the date and time the request has reached and exited each state.

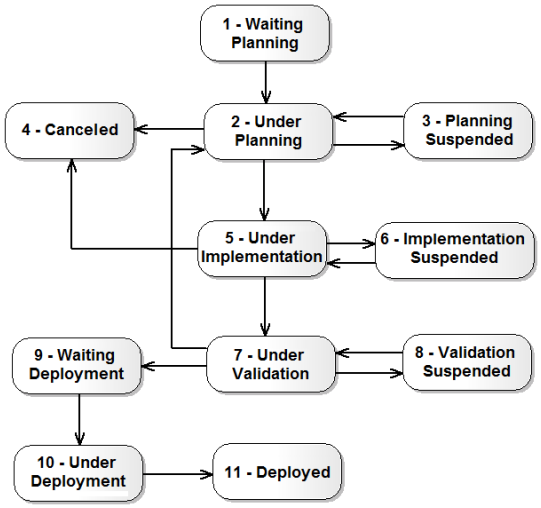


Figure 1: MMG Example

- Number of times the state has been visited when processing the request. As MMGs can have loops, this number can be greater than one for some states.

Next, using the MMG instances, the following characteristics are calculated for each request:

- *QueueTime*: time interval between the request registration in the issue tracking system and its corresponding conclusion or cancellation.
- *WaitingTime*: time interval the request remained in the issue tracking system, waiting for processing (for example, in the case of the MMG in Figure 1 the time spent in the states 1, 3, 6, 8, and 9).
- *ServiceTime*: time interval to process the request, including the execution of all software engineering activities modeled in the MMG (for example, in Figure 1, the time spent in the states 2, 5, 7, and 10).

For such characteristics, the following equation hold:

$$QueueTime = WaitingTime + ServiceTime$$

For the MMG presented in Figure 1, we also have the following equation:

$$\begin{aligned} ServiceTime = {} & PlanningTime + \\ & ImplementationTime + \\ & ValidationTime + \\ & DeploymentTime \end{aligned}$$

where (i) *PlanningTime* is the time interval demanded to understand, to plan and to schedule a software maintenance request, (ii) *ImplementaionTime* is the time interval to implement and to code the request, (iii) *ValidationTime* is the time interval for testing and (iv) *DeploymentTime* is the

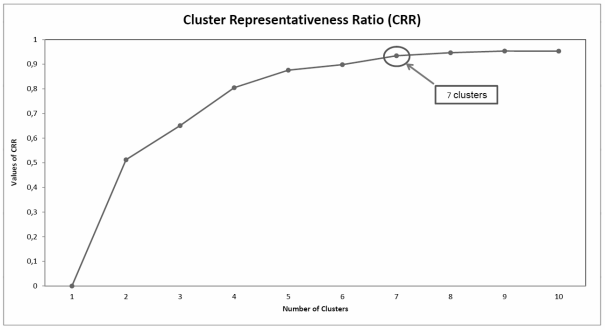


Figure 2: Cluster Representativeness Ratio (CRR)

time interval to deploy the request and to obtain the final approval of the involved users.

When calculating the time intervals associated to these characteristics and in case the organization follows well-defined working hours, we can consider only the hours the maintainers normally work (e.g. 9:00 AM to 5:00 PM) and therefore to exclude non-business hours, weekends, holidays etc. Anyway, it is important to note that the service time is not a measure of effort (for example, in terms of man-hours), but a measure of the number of hours required to process the request.

2.3 Clustering the Maintenance Model Graph Instances

In this step, for each MMG instance a vector is created with the waiting and service times calculated in the previous step. These vectors – called feature vectors – are used as the input of a clustering algorithm, which automatically classifies the vectors in groups (called clusters) so that the vectors in the same cluster are similar to each other. Following Menasce and Almeida [9], we recommend the use of the k-means clustering algorithm for this purpose. This algorithm partitions n data points into k clusters. In our particular case, k-means will provide as output the following information for each cluster:

- A set with the maintenance requests assigned to the cluster;
- The cluster's mean, i.e. a feature vector that represents the set of maintenance requests assigned to the cluster. Basically, each element of this vector is the mean of the correspondent elements in the feature vectors of the requests assigned to the cluster.

Defining the number of clusters: A critical decision when applying k-means is to set the number of clusters k, which is an input of the algorithm. Basically, to define this parameter it is important to have in mind the following goals: (a) the distance among the elements of the generated clusters – called intracluster distance – must be minimized, in order to provide homogeneous clusters; (b) the distance among the generated clusters – called intercluster distance – must be maximized, in order to ensure that there are clear differences among them [9]. In order to meet these goals, the following Cluster Representativeness Ratio (CRR) is usually employed:

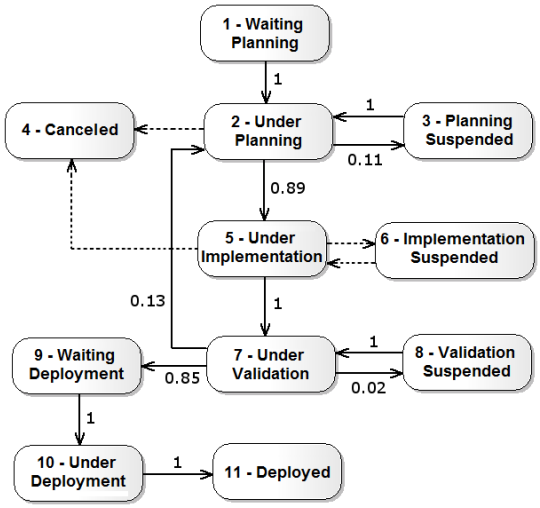


Figure 3: MMG representing a cluster

$$CRR = \frac{betweenss}{betweenss + \sum_{i=1}^{k} withinss[i]}$$

where *betweenss* and *withinss* are, respectively, measures for intercluster and intracluster distances. Such measures are automatically provided by most statistical packages that implement a k-means function. Therefore, detailed information on how to calculate such distances will not be reported in this paper and can be found elsewhere [16].

In general, the near CRR is to 1, the better is the representativeness of the generated clusters. However, maximizing CRR must not be pursued at all costs. For example, if k is defined as the size of the sample, each sample's element will be a cluster, and the intracluster distances will be zero. As a result, CRR is 1, but with trivial clusters. Therefore, we must maximize CRR, but without generating too many clusters. Based on this recommendation, a simple rule of thumb to determine the value of k is the following: (a) run the clustering algorithm with several values of k; (b) plot the respective CRR values, which will result in a monotonically increasing function towards the value 1; (c) supposing the last "significant rise" in this graph happens from $k-1$ to k, the recommended value for the number of clusters is k. In other words, after this value, increasing k will not result in significant gains in terms of CRR [16]. For example, in the example of Figure 2, the recommendation is to set $k = 7$ (because after this value the curve has a smoother growth).

Creating a MMG for the clusters: Finally, we must create a MMG representing the MMGs of each cluster. The edges in this MMG are decorated with a real value in the range $[0, 1]$ that denotes the probability of using the path represented by the edge. Suppose a cluster with requests $req[1], req[2], req[3], req[4], req[5], \ldots, req[n]$. Suppose also that $req[r].edge[S_i, S_j].visits$ denotes the number of times the edge connecting states S_i and S_j in the MMG associated to a given request r has been visited. Similarly, suppose that $req[r].state[S_i].visits$ denotes the number of times state S_i in the MMG associated to the request r has been visited. Using such definitions, the probability of taking an

Table 1: Clusters representing the considered maintenance requests

Clusters	# Requests	WaitingTime			ServiceTime			QueueTime	
		Mean (*hours*)	CV	%	Mean (*hours*)	CV	%	Mean (*hours*)	CV
Cluster #1	22	25.44	1.03	17.37	121.01	0.33	82.63	146.45	0.34
Cluster #2	22	29.96	1.06	11.08	240.46	0.17	88.92	270.42	0.16
Cluster #3	13	42.40	0.87	9.37	410.33	0.13	90.63	452.73	0.12
Cluster #4	7	84.86	1.21	12.67	585.05	0.07	87.33	669.91	0.16
Cluster #5	4	12.66	0.62	1.33	937.07	0.11	98.67	949.73	0.11
Cluster #6	3	271.42	0.39	61.58	169.33	0.46	38.42	440.75	0.39
Cluster #7	1	933.20	0.00	80.60	224.57	0.00	19.40	1,157.77	0.00

edge $[S_i, S_j]$ is defined as:

$$edge[S_i, S_j].prob = \frac{\sum_{r=1}^{n} req[r].edge[S_i, S_j].visits}{\sum_{r=1}^{n} req[r].state[S_i].visits}$$

Figure 3 shows an example of a MMG representing a cluster. This MMG reveals that the probability of canceling a request during planning is zero (the dashed edge connecting states 2 and 4 denotes a transition with probability zero). On the other hand, the probability to suspend a request under planning is 0.11 (as represented in the edge connecting states 2 and 3).

3. CASE STUDY

In order to evaluate the applicability of the proposed approach, we accomplished a case study using a real dataset of software maintenance requests, which were clustered and analyzed using the approach described in Section 2.

3.1 Dataset Overview

We analyzed 72 software maintenance requests handled by the Information Technology Division of an university. This division acquires, develops, and maintains all academic and administrative systems used by the university. Nowadays, it provides maintenance services to almost 40 systems, totaling more than four million lines of code in different programming languages (Java, Delphi, PHP etc). The division relies on a lightweighted process – called PASM (Process for Arranging Software Maintenance Requests) [1] – for handling maintenance as software projects [3, 15]. Therefore, the requests evaluated in our study are in fact software projects created by the software maintenance managers at IT Division to handle modifications in the systems of the university, as demanded by the users of those systems. The requests considered in the study have been opened and closed in 2010 (January to December). This period covers two semesters, i.e. a complete annual cycle of the university activities.

3.2 Methodology

We have initially defined a MMG expressing the workflow followed by this IT Division to provide maintenance services (which in fact is the MMG already presented in Figure 1, Section 2). An instance of this MMG has been created to provide information for each maintenance request considered in the study. The information about the starting and ending timestamps of each state has been retrieved from the issue tracking system used by this organization. Next, we have generated a feature vector for each request, including the following characteristics: service time and waiting time. Finally, we clustered the 72 MMGs using the k-means algorithm (as implemented by the R statistical and data analysis platform, http://www.r-project.org). We executed the algorithm 10 times, with k (the number of clusters) ranging from 1 to 10. For each execution, we calculated the Cluster Representativeness Ratio (CRR). Figure 2 – already presented in Section 2 – shows the CRRs achieved by each number of clusters we have tested. As mentioned before, this figure suggests that we should set $k = 7$.

3.3 Results and Discussion

Table 1 shows information on the seven clusters representing the maintenance requests considered in our study. The table presents the number of requests and the average values (in hours) for the waiting time, service time, and queue time of the requests in each cluster. Besides the cluster's mean, the table also provides an information on the variability of the requests in each cluster, represented by column CV (coefficient of variation), which shows the ratio between the standard deviation and the mean. Finally, for the waiting and service times, the table shows the percentage of the mean regarding the total queue time.

We briefly discuss our main finding after analyzing the generated clusters:

- Cluster #1 and Cluster #2 grouped together more than 60% of the considered requests (44 out of 72 requests). However, the maintainers spent almost twice as long to perform the software engineering activities (i.e. service time) in Cluster #2 , compared with Cluster #1. Therefore, these clusters reveal what we can classify as the typical and simple maintenance requests (Cluster #1, with a service time around 3 weeks) and the typical and medium-complexity requests (Cluster #2, with a service time around 6 weeks).

- Cluster #3 reveals what can be classified as the typical complex request (with a service time superior to 410 hours). This cluster accounts for around 18% of the considered requests.

- The remaining four clusters have captured requests with an outlier behavior (less than 21% of the requests). This behavior can be explained by high or extremely high service times (Cluster #4 and #5, respectively), or to high or extremely high waiting times (Clusters #6 and #7).

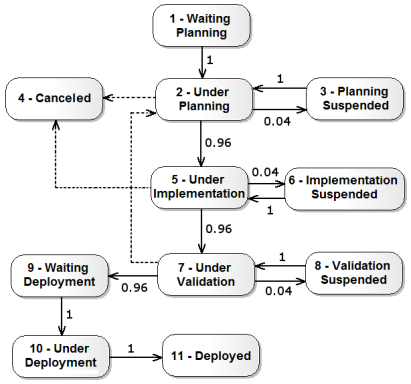
Besides evaluating the results at the cluster level, it is also possible to analyze the sub-components of the service time,

Table 2: Service time subcomponents

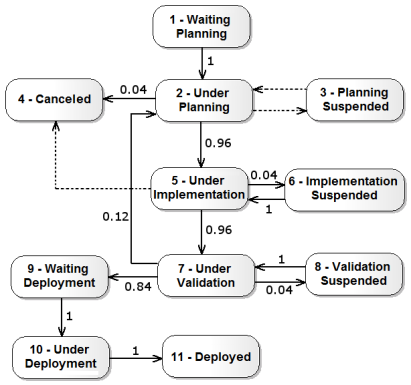
Clusters	Planning		Implementation		Validation		Deployment		ServiceTime
	Mean	%	Mean	%	Mean	%	Mean	%	
Cluster #1	19.06	15.75	28.02	23.16	38.65	31.94	35.28	29.15	121.01
Cluster #2	40.83	16.98	75.06	31.21	80.88	33.64	43.69	18.17	240.46
Cluster #3	59.71	14.55	127.20	31.00	143.96	35.08	79.46	19.37	410.33
Cluster #4	90.57	15.48	101.96	17.43	310.86	53.13	81.66	13.96	585.05
Cluster #5	178.04	19.00	140.56	15.00	543.50	58.00	74.97	8.00	937.07
Cluster #6	12.80	7.56	38.39	22.67	79.26	46.81	38.88	22.96	169.33
Cluster #7	71.15	31.68	147.72	65.78	5.67	2.52	0.03	0.02	224.57

i.e. the amount of the service time allocated to each software engineering activity. For each cluster, Table 2 presents the values of such sub-components. Among the typical clusters (Clusters #1, #2, and #3), we can observe for example that there are small variations in the percentage of time allocated to each activity. For example, the planning time accounts around 16%, 17%, and 15% of the service time for these clusters, respectively. This reinforces our initial claim that the differences among such clusters are due to the inherent complexity of the associated maintenance tasks. Moreover, the table also sheds light on the clusters with an outlier behavior due to their service times. For example, we can discover that the outlier behavior attributed to Clusters #4 and #5 was due to the number of hours allocated to validation (which has been superior to 50% of the service time of both clusters).

Finally, we can also evaluate the MMGs associated to each cluster in order to check the probabilities associated to the state transitions. Figure 4 shows the MMG for Clusters #1 and #2. We can observe that the only difference between such graphs are in the *Under Planning* state. In the Cluster #1, requests under planning have a small probability to be suspended (0.04). On the other hand, for Cluster #2, the requests under planning presented a small probability not to be suspended, but to be canceled (also 0.04). This difference is in line with our previous characterization of Cluster #1 as a cluster representing the simple maintenance requests (e.g. that may eventually be suspended to handle a more urgent task). Furthermore, we classified Cluster #2 as including complex requests (that eventually may be canceled).



(a) Cluster 1



(b) Cluster 2

Figure 4: MMG for Clusters #1 and #2

4. RELATED WORK

Banker and Slaughter have evaluated 129 software enhancement projects from a large financial organization [4]. Their empirical evaluation has been conducted using a non-parametric method commonly used to measure productivity, called Data Envelopment Analysis (DEA). They conclude that a cost reduction of up to 36% could have been achieved by clustering the evaluated requests.

The work of Kanellopoulos et al. [8] presents a model for depicting the software structure in conjunction with a set of metrics for assessing the maintainability of this software. The authors use the k-means algorithm over the source code for understanding the software structure. An empirical study presented by Robillard et al. [12] analyzes 4,200 software maintenance requests of seven different systems and shows a synthetic representation of software structure also created with clustering techniques. Understanding the software structure can help software maintainer when perform programming activities. Nevertheless, they do not investigate how a deep understanding of the requests' nature can contribute to improve the maintenance process.

A model to help the managers of a software team in the task of setting a timetable for handling maintenance requests is proposed by Tan and Mookerjee [15]. This model includes not only decisions about maintenance, but also decisions about replacement of a system. However, their mathematical model has not been validated in real-world software development organizations. In addition, it depends on several complex parameters such as entropy degree of the system (i.e. system degradation due to maintenance) and cost reduction due to reuse and due to the incorporation of new technologies.

Assuming that maintenance requests arrive in accordance to a Poisson distribution, Feng, Mookerjee and Sethi present a model that aim to help software teams to optimize the long-run total discounted cost of the system (i.e. the waiting cost of each pending maintenance requests plus maintenance costs) [5]. In their work, the authors assume that the wait-

ing cost per time unit of each outstanding request is known. They also assumed that the cost of maintenance consists of a fixed cost and a correction cost per request, which in general does not apply to maintenance process that foster the treatment of maintenance requests as software projects [1].

In a recent work, Gethers et al. [6] combine different techniques for predicting the impacts of a software change. These techniques make up a framework which uses information, such as (i) the register of the change request, (ii) the historic of software versions, and (iii) the dynamic analysis of the system behavior. A similar work have used Formal Concept Analysis (FCA) for identifying the impacts of software changes [14]. The authors also present a framework which obtain metrics based on elements of software structure in order to better estimate these impacts. However, these works do not consider the workflow followed for performing the maintenance requests.

Aparecido et al. present a software maintenance process named PASM (Process for Arranging Software Maintenance Requests) [1]. This process has three phases: registering, grouping, and processing. In order to evaluate the PASM the authors used a set of clustering techniques for analyzing the software maintenance requests. We believe that this approach can be used in any kind of process with focus on planning and managing activities for improving software maintenance.

5. FINAL REMARKS

We have presented a quantitative approach for evaluating maintenance services, based on cluster analysis techniques. As showed in our case study, the proposed approach can reveal quantitative data on the typical maintenance services provided by the organization (e.g. simple, medium-complexity, and complex requests) and also on the requests with an outlier behavior. First of all, this data provides a better understanding of the current status of the maintenance activities in a given organization. After that, it can be used to better manage and improve the maintenance process followed by the software team [7]. For example, programming teams with different skills and experience can be allocated to the different types of requests. Furthermore, it is possible to anticipate the costs, required resources, and processing times of upcoming requests. Finally, a detailed study of the requests with an outlier behavior may help to identify problems in the current maintenance process and therefore help the organization to improve and fix them.

Acknowledgments

This research has been supported by grants from FAPEMIG and CNPq.

6. REFERENCES

[1] Gladston Junio Aparecido, Marcelo Malta, Humberto Mossri, Humberto Marques-Neto, and Marco Tulio Valente. On the benefits of planning and grouping software maintenance requests. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 55–64, March 2011.

[2] Alain April, Jane Huffman Hayes, Alain Abran, and Reiner Dumke. Software maintenance maturity model (SM^{MM}): the software maintenance process model. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(3):197–223, May 2005.

[3] Rajiv D. Banker and Chris F. Kemerer. Scale economies in new software development. *IEEE Transactions on Software Engineering*, 15(10):1199–1205, October 1989.

[4] Rajiv D. Banker and Sandra A. Slaughter. A field study of scale economies in software maintenance. *Management Science*, 43(12):1709–1725, December 1997.

[5] Qi Feng, Vijay S. Mookerjee, and Suresh. P. Sethi. Optimal policies for the sizing and timing of software maintenance projects. *European Journal of Operational Research*, 173(3):1047–1066, September 2006.

[6] Malcom Gethers, Bogdan Dit, Huzefa Kagdi, and Denys Poshyvanyk. Integrated impact analysis for managing software changes. In *34th IEEE International Conference on Software Engineering (ICSE)*, pages 430–440, June 2012.

[7] Watts S. Humphrey. Characterizing the software process: A maturity framework. *IEEE Software*, 5:73–79, March 1988.

[8] Yiannis Kanellopoulos, Thimios Dimopulos, Christos Tjortjis, and Christos Makris. Mining source code elements for comprehending object-oriented systems and evaluating their maintainability. *ACM SIGKDD Explorations Newsletter*, 8(1):33–40, June 2006.

[9] Daniel Menasce and Virgilio Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, 2000.

[10] Radha Mookerjee. Maintaining enterprise software applications. *Communications of the ACM*, 48(11):75–79, November 2005.

[11] Frank Niessink and Hans van Vliet. Software maintenance from a service perspective. *Journal of Software Maintenance and Evolution: Research and Practice*, 12(2):103–120, March 2000.

[12] Martin P. Robillard and Barthélémy Dagenais. Recommending change clusters to support software investigation: an empirical study. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(3):143–164, April 2010.

[13] Harry M. Sneed and Stefan Opferkuch. Training and certifying software maintainers. In *12th IEEE European Conference on Software Maintenance and Reengineering (CSMR)*, pages 113–122, April 2008.

[14] Xiaobing Sun and Bixin Li. Using formal concept analysis to support change analysis. In *26th International Conference on Automated Software Engineering (ASE)*, pages 641–645, November 2011.

[15] Yong Tan and Vijay S. Mookerjee. Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, 31(3):238–255, March 2005.

[16] Stephane Tuffery. *Data Mining and Statistics for Decision Making*. Wiley, 2011.

[17] M. P. Ware, F. G. Wilkie, and M. Shapcott. The application of product measures in directing software maintenance activity. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):133–154, March 2007.