On the Benefits of Planning and Grouping Software Maintenance Requests

Gladston Aparecido Junio, Marcelo Nassau Malta,
Humberto de Almeida Mossri, Humberto T. Marques-Neto
Department of Computer Science, PUC MINAS
Belo Horizonte, Brazil
gladston.aparecido@sga.pucminas.br,
{nassau,hmossri,humberto}@pucminas.br

Marco Tulio Valente
Department of Computer Science, UFMG
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

***Abstract*—Despite its unquestionable importance, software maintenance usually has a negative image among software developers and even project managers. As a result, it is common to consider maintenance requests as short-term tasks that should be implemented as quick as possible to have a minimal impact for end-users. In order to promote software maintenance to a first-class software development activity, we first define in this paper a lightweighted process – called PASM (Process for Arranging Software Maintenance Requests) – for handling maintenance as software projects. Next, we describe an in-depth evaluation of the benefits achieved by the PASM process at a real software development organization. For this purpose, we rely on a set of clustering analysis techniques in order to better understand and compare the requests handled before and after the adoption of the proposed process. Our results indicate that the number of projects created to handle maintenance requests has increased almost three times after this organization has adopted the PASM process. Furthermore, we also concluded that projects based on the PASM present a better balance between the various software engineering activities. For example, after adopting PASM the developers have dedicated more time to analysis and validation and less time to implementation and codification tasks.**

***Keywords*-software maintenance process; software clustering; software economics.**

I. Introduction

Maintenance is one of the most important and costly phases in the life-cycle of a software system. For example, there are studies showing that up to 90% of the total resources consumed over the lifetime of a system are allocated to maintenance tasks [1]. Other studies estimate that at least 250 billion lines of source code were under maintenance worldwide at the beginning of the decade [2]. Finally, in the mid-90s, Sutherland has estimated in US$ 70 billion the annual spending that U.S. companies had with software maintenance activities [3]. In recent years, there are no indications that these costs have decreased. In fact, new systems are being developed every day for many domains. The result is a growing base of code that must be continuously maintained and evolved [4].

Despite its unquestionable importance, software maintenance usually has a negative image among software developers and even project managers [5], [6]. As a result, there is a tendency in not handling maintenance requests as software projects, i.e. projects of software maintenance with well-defined activities such as requirements specification, design, implementation, testing etc. Instead, it is common to consider software maintenance requests as short-term tasks that should be prioritized and implemented as quick as possible to have a minimal impact in end-users' activities [7]–[9].

The problems inherent to continuous policies for scheduling maintenance requests have already been pointed out in the literature. For example, Banker and Slaughter have investigated the scale economies achieved when maintenance is handled in batch mode, i.e. when individual requests are packaged and implemented as part of larger projects [7]. According to their models, this strategy can reduce maintenance costs by 36%. The benefits of a periodic maintenance policy – combined with policies to replace a system – have also been reported by Tan and Mookerjee [8]. However, both studies have been solely based on analytical models. Particularly, they have not checked if the results of their models are confirmed when maintaining real-world systems. In fact, to the best of our knowledge, we still lack studies that assess whether the theoretical gains described in the literature are actually observed in real-world software maintenance scenarios.

In this paper, we first define a lighthweighted process for handling maintenance requests as software projects. This process – called PASM (Process for Arranging Software Maintenance Requests) – includes three main phases: registering, grouping and processing. We have also applied the proposed process in the IT division of PUC Minas, one of the largest Brazilian universities. Until 2008, this division – named DATAPUC – has adopted a continuous policy for handling maintenance, with buffering and grouping performed in an ad hoc way. Since 2009, DATAPUC has embraced the proposed process for grouping software maintenance requests. Finally, we describe an in-depth evaluation of the benefits achieved with this adoption. For this purpose, we rely on a set of clustering analysis techniques originally proposed to evaluate the behavior of web application users [10], [11].

Our results indicate that the number of projects created to handle maintenance requests has increased almost three times after DATAPUC has adopted the PASM process (in 2008, 22 software maintenance projects have been delivered by DATAPUC; in 2009, this number increased to 62 projects). More important, the clustering analysis has showed that the PASM-based projects present a better bal-

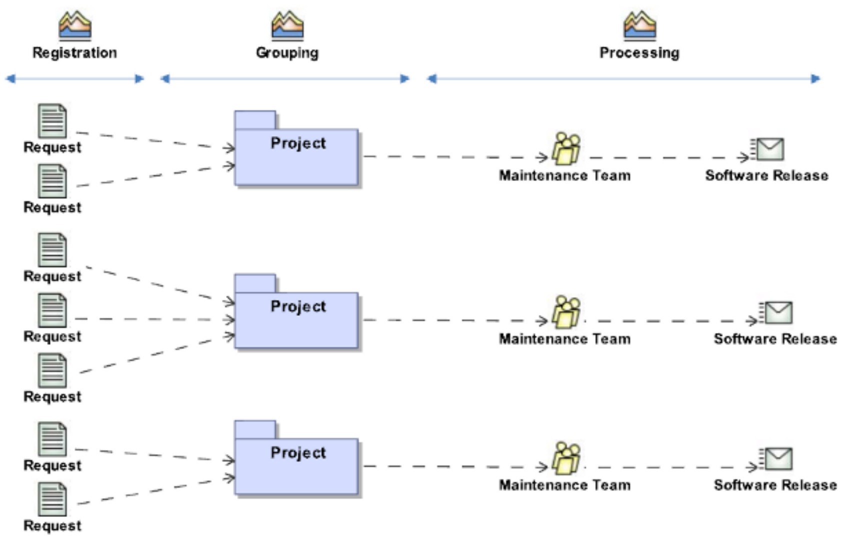

Figure 1. Process for Arranging Software Maintenance Requests (PASM)

ance between the various software engineering activities (for example, after adopting PASM, the developers have dedicated more time to analysis and validation and less time to implementation and codification tasks). Finally, we have observed on average a six hour reduction on the queue time of typical urgent maintenance requests, since the process contributes to reduce the chances of incorrectly handling a maintenance request as urgent.

The remainder of this paper is organized as described next. Section II starts by presenting the proposed process for planning and grouping maintenance requests, including a presentation of its main objectives and phases. Section III evaluates the benefits achieved by the proposed process at a real software development organization. Section IV discusses related work. Section V provides concluding statements and a list of future work.

II. The PASM Process

PASM is a lightweighted process that advocates that maintenance requests should be arranged in projects for the following reasons: (a) to facilitate planning and control activities; (b) to benefit of widely recognized software engineering best practices (such as requirements specification, design, testing etc); and (c) to take advantage of the economies of scale that manifest when smaller tasks are batched into larger projects.

As illustrated by Figure 1, the PASM process supports a periodic maintenance policy with three main phases: *Registration*, *Grouping*, and *Processing*. These phases are described next.

Registration: In this phase, users enter and describe their maintenance requests, preferably using an change management system. Basically, the existence of a registration phase – where requests are simply collected – ensures that the PASM process follows a periodic maintenance policy. Furthermore, the time frame allocated to this phase defines the frequency that new releases are periodically delivered to end-users. In order to set this time frame, the team deploying the PASM should consider the inevitable opportunity costs associated to delaying the processing of the collected requests. Basically, the higher this time frame implies the higher the associated opportunity (or wait) costs, considering the amount of requests which can be registered. On the other hand, smaller intervals can have a negative impact on the economies of scale provided by the periodic policy.

During the registration phase, end-users and software developers can request all categories of maintenance, including corrective, adaptive, perfective, and preventive [12]. Usually, both of them (end-users and software developers) can specify that an entered maintenance request is urgent. For example, urgency can be reported to errors that prevent the normal system behavior and that must be fixed as soon as possible. As another example, urgency can also be requested to a maintenance that due to legal constraints have to be implemented in a given time frame (e.g. a new tax created by the government). Maintenance requests classified as urgent – and having their urgent status ratified by a higher-level manager – are not subjected to waiting costs. Instead, such requests are immediately allocated to a maintenance team.

The output of the registration phase is a list of software maintenance requests placed during a given time frame.

Grouping: In this phase, the collected maintenance requests are grouped in larger software projects. In order to propose a software project P comprising maintenance requests $\{r_1, r_2, \ldots, r_n\}$ two variables should be considered: the project size and the functional similarity among requests r_i. First, the project size and complexity – estimated for example in functional points – should be compatible with the organization's available production capacity – estimated for example in man-hours per month.

Extremely larger projects will imply in waiting times not tolerable by end-users; moreover, larger projects delay the allocation of other projects to the available maintenance teams. Second, the requests r_i grouped in the project P should be functionally coherent. For example, they must require modifications in the same modules of the target system or they must require the implementation of new modules that are strongly-related.

When defining the PASM process, we deliberately decided to do not provide a complete and formal algorithm to automate the grouping phase. The main reason is that this phase is heavily based on decisions that depend on the semantics of the requests (e.g. is request r_i functionally coherent with request r_j?) or that depend on management level decisions (e.g. can the organization absorb the opportunity costs due to delaying a maintenance request?).

The output of the grouping phase is a list of groups of software maintenance requests that – as agreed by project managers and key-users representing the users that required the maintenance – must be implemented as a software development project.

Processing: In this phase, the software maintenance projects generated at the previous phase are allocated to a software development team. Such projects must follow the normal software development methodology and practices defined by the organization. In other words, the PASM process does not define a particular software development method.

The output of the processing phase is a new release of the target system that implements the maintenances requested by the end-users.

III. Evaluation

In this section we evaluate the benefits achieved by the PASM process at a real software development organization. The Information Technology Department of PUC Minas – called DATAPUC – is using the process proposed in this paper since November 2008. DATAPUC is responsible for the acquisition, development and maintenance of the academic and administrative systems used by PUC Minas. Currently, it maintains 40 systems, comprising more than four million lines of code in different programming languages (Java, Delphi, PHP etc). For this purpose, DATAPUC has 34 software developers (including programmers, testers, project managers etc). Before adopting the PASM process, most maintenance requests were handled by DATAPUC on demand, in a continuous way.

DATAPUC has configured the PASM process so that the registering and grouping phase take one month. In the first ten days of each month, end-users can place software maintenance requests, using DATAPUC's internal Change Management System. In the remainder twenty days of the month the requests are analyzed and evaluated by DATAPUC's developers and project managers, resulting in a set of software maintenance projects. The cycle is repeated in the next month.

Table I
Evaluated Software Maintenance Requests

Year	# of Requests	Man-hours
Before PASM (2008)	1073	3198
After PASM (2009)	1015	5820
Total	**2088**	**9018**

First, in this section we present the data source formed by the real software maintenance requests used for this evaluation. We also present the methodology used to characterize these requests in order to highlight the benefits achieved by the proposed process.

A. Data Source

In order to evaluate the PASM implementation, we have analyzed a set of 2,088 software maintenance requests processed by DATAPUC, comprising 9,011 man-hours. As summarized at Table I, the evaluated requests refer to two distinct time periods:

- Before PASM: this period includes requests initiated and completed between February and October 2008, i.e. before DATAPUC decision on starting to use the proposed process. During this time frame, 1,073 software maintenance requests have been processed by DATAPUC.

- After PASM: this period includes requests processed between February and October 2009, i.e. after the implementation of the PASM process. During this time frame, 1,015 maintenance requests have been processed by DATAPUC.

It is important to mention that the time periods under evaluation include the same sequence of months (February to October). In this way, we guarantee that seasonal activities in the life of a university – and that may demand maintenance in the systems under evaluation – are included in both periods. As an example of such activities, we can mention the preparation of the university annual budget or posting the students final grades at the end of each academic semester.

In order to evaluate the benefits achieved by the proposed process, several data about the maintenance requests considered in our study have been extracted from the Change Management System used by PUC Minas to track maintenance activities. For example, we have been able to collect the following information about each request: registering date (by end-users), description, priority, estimated work hours, and accomplished work hours. The Change Management System also provides detailed information about the workflow followed by each request, since its registration until the deployment of a new release of the system under maintenance. For example, it is possible to obtain information about the duration of each activity conducted by DATAPUC to handle the software maintenance and the employees responsible for it.

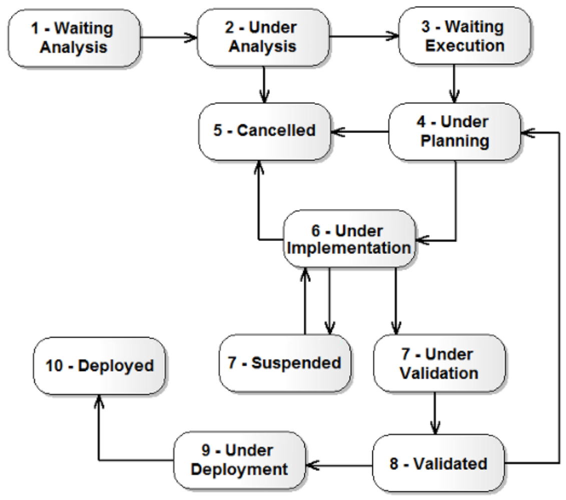

Figure 2. The SMRMG of Software Maintenance Project.

B. Characterization Methodology

In this section we present the methodology we followed to model the groups of software maintenance requests. This methodology is an adaptation of the characterization methodology proposed by Menasce et al. [10], [11] and by Marques-Neto et al. [13] to understand the behavior of Web applications users and the behavior of the users of broadband Internet services, respectively.

According to this methodology, each software maintenance request should be modeled as a directed graph, where the nodes represent the states of each request and the edges represent the transitions between such states. Particular, the states denote the software engineering activities that must be performed in order to handle a maintenance request. Furthermore, the edges are weighted with the difference between the end and the start time of the activities associated to the connected nodes. This graph is semantically rich specially because it allows the visualization of the various workflows that characterize each kind of maintenance request. We named this graph by SMRMG (*Software Maintenance Request Model Graph*).

For example, Figure 2 presents the SMRMG we have used to represent a software maintenance project. When the end-user registers a software maintenance project, this request remains in a waiting state (State 1). After the analysis activities have been performed (State 2), the request can be cancelled (State 5) or it can move to a state where it waits for execution (State 3). The planning activities associated to the request are associated to State 4. After that phase, the request can be cancelled (State 5) or it can enter the implementation phase (State 6). After implementation and in case of failures, the request can be suspended (State 7) or Cancelled (Sate 5). On the other hand, it moves to the validation phase (State 8). After validation, the request can return to the planning phase (State 4) or move to deployment (State 10). Finally, the whole process finishes and the maintenance is considered deployed (State 11).

To reason about software maintenance requests, we will rely on the following characteristics (calculated in hours):

- *QueueTime*: time interval between the registration of a software maintenance request and its corresponding conclusion or cancellation.
- *WaitTime*: time interval demanded to start the first activity for handling the maintenance request after its registration in the Change Management System.
- *ServiceTime*: time interval demanded to handle a software maintenance request (including analysis, implementation, validation, deployment etc).
- *PlanningTime*: time interval demanded to plan and schedule the processing of a software maintenance request, including detailing the activities needed to process the request, to estimate the effort demanded in its implementation, and to allocate the important resources.
- *AnalysisTime*: time interval used in activities of analysis of a software maintenance request.
- *ImplementaionTime*: time interval to implement and to code a software maintenance request.
- *ValidationTime*: time interval for testing the implementation of a software maintenance request.
- *DeploymentTime*: time interval to deploy a software maintenance request and to obtain the final approval of the involved users.

For such characteristics, the following equations hold:

$$\begin{aligned} QueueTime &= WaitTime + ServiceTime \\ ServiceTime &= PlanningTime + AnalysisTime + \\ &\quad ImplementationTime + \\ &\quad ValidationTime + \\ &\quad DeploymentTime \end{aligned}$$

In order to better understand the processing of software maintenance requests at DATAPUC, we have performed the following tasks:

1) We have generated a SMRMG graph for each software maintenance request from our data source.
2) For each generated SMRMG we have calculated the previous mentioned characteristics, generating a feature vector for each software maintenance request.
3) We have classified the maintenance requests from our data source according to the following criteria:

time period (i.e. before PASM or after PASM) and type (urgent maintenances or maintenances handled under software projects). Therefore, by combining such criteria, we have four maintenance groups. Table II provides detailed information about such groups

4) We have applied the k-means clustering algorithm over the feature vectors generated at Step 3. K-means is a widely used clustering algorithm for partitioning and automatic grouping of unlabeled datasets [14]. In this paper, we used the k-means implementation available in the WEKA tool[1].

5) We have evaluated the clusters generated by the k-means for each of the considered maintenance groups, i.e. (before PASM, urgent maintenance), (before PASM, maintenance projects), (after PASM, urgent maintenance), (after PASM, maintenance projects).

Table II
EVALUATED SOFTWARE MAINTENANCE GROUPS

	Before PASM (2008)		**After PASM (2009)**	
Group	**# Requests**	**Man-Hours**	**# Requests**	**Man-Hours**
Urgents	1051	2457	953	3089
Projects	22	741	62	2730

Since clustering techniques are useful to generate a set of representative elements for a given data source, our strategy is to evaluate the benefits and eventual drawbacks derived from the PASM process by comparing the clusters generated after and before its adoption. Our claim is that this strategy will make the comparison more clear, since we can concentrate only on the "most representative" maintenance from the considered data source. The results of this comparison are reported on the next subsection.

C. Results

This section reports the results obtained from clustering the following groups of software maintenance requests: maintenance projects (Subsection 1) and urgent maintenances (Subsection 2).

1) Maintenance Projects: Clustering the SMRMGs for these projects has resulted in two clusters for the period before the PASM adoption and two clusters for the period after the adoption of the proposed process by DATAPUC. Table III shows the details of such clusters (in hours). In this table, CV denotes the ratio between the standard deviation and the respective mean.

An initial observation of the results presented at Table III reveals a considerable increase in the number of projects handled after PASM (62 projects), compared to the number of projects handled before PASM (22 projects). Clearly, this results is expected when moving from a continuous to a periodic policy for handling maintenance requests. However, the results described in this table also support a detailed analysis on the internals of the projects handled before and after the PASM adoption.

[1]Available at http://www.cs.waikato.ac.nz/ml/weka

First, we describe the two clusters generated for the period before the PASM adoption:

- *Before PASM, Cluster 1*: This cluster contains 19 software maintenance projects handled between February and October 2008. By analyzing the results at Table III, we can observe that on average the *WaitTime* of the requests in this cluster corresponds to 6.6% from their *QueueTime* (12.16 / 185.00). Furthermore, the *ImplementationTime* corresponds to 43.2% from the *ServiceTime* (74.63 / 172.84). Finally, the *ValidationTime* corresponds to 22.7% from the *ServiceTime* (39.34 / 172.84). In summary, we can characterize this cluster as the typical software projects that have emerged at DAPATUC before the organization embraced a systematic process to group maintenance requests. Generally, such projects have been intuitively proposed by the own developers responsible for the maintenance tasks.

- *Before PASM, Cluster 2*: This cluster has only three maintenance projects that are characterized by their long *ServiceTime* (more than 754 hours, against 172.84 hours from Cluster 1). Analyzing the details of these three projects, we have discovered that two of them are internal projects proposed by the own developers to improve the internal quality and the maintainability of the systems under their responsibility. Therefore, they can be characterized as non-typical software projects.

Finally, the results for the 2008 clusters show a high variability of some characteristics, such as *ImplementationTime* and *ValidationTime* for Cluster 1 and *AnalysisTime* and *PlanningTime* for Cluster 2. This variability is expressed on the high values for the CV of these characteristics, which indicates the existence of values quite different from the mean time. It points out that the software maintenance projects identified before the deployment of PASM could benefit from a systematic software development process, in order for example to increase the time of software analysis and design, to decrease the time allocated to implementation, and to reduce the likelihood of project cancellation.

Next, we describe the two clusters presented on Table III generated for the period after the PASM adoption:

- *After PASM, Cluster 1*: This cluster contains 52 software maintenance projects. Therefore, we can refer to such projects as the typical software projects handled after the PASM adoption. By analyzing

Table III
CLUSTERS FOR SOFTWARE MAINTENANCE PROJECTS.

Characteristics	**Before PASM (2008)**				**After PASM (2009)**			
	Cluster 1 (19 projects)		*Cluster 2* (3 projects)		*Cluster 1* (52 projects)		*Cluster 2* (10 projects)	
	Mean	*CV*	*Mean*	*CV*	*Mean*	*CV*	*Mean*	*CV*
QueueTime	185.00	0.57	771.87	0.21	245.04	0.66	694.91	0.51
WaitTime	12.16	1.01	17.50	0.85	26.85	1.97	11.95	1.11
ServiceTime	172.84	0.61	754.37	0.20	218.19	0.68	682.97	0.51
AnalysisTime	31.51	1.26	7.48	1.60	48.71	1.05	141.61	1.70
PlanningTime	4.98	1.32	4.86	1.65	1.37	2.94	57.35	0.79
ImplementationTime	74.63	1.30	348.27	0.73	50.39	0.80	122.56	0.92
ValidationTime	39.34	1.28	377.46	0.83	94.19	1.29	249.70	1.10
DeploymentTime	21.01	0.85	16.14	0.84	21.04	1.10	104.49	1.38

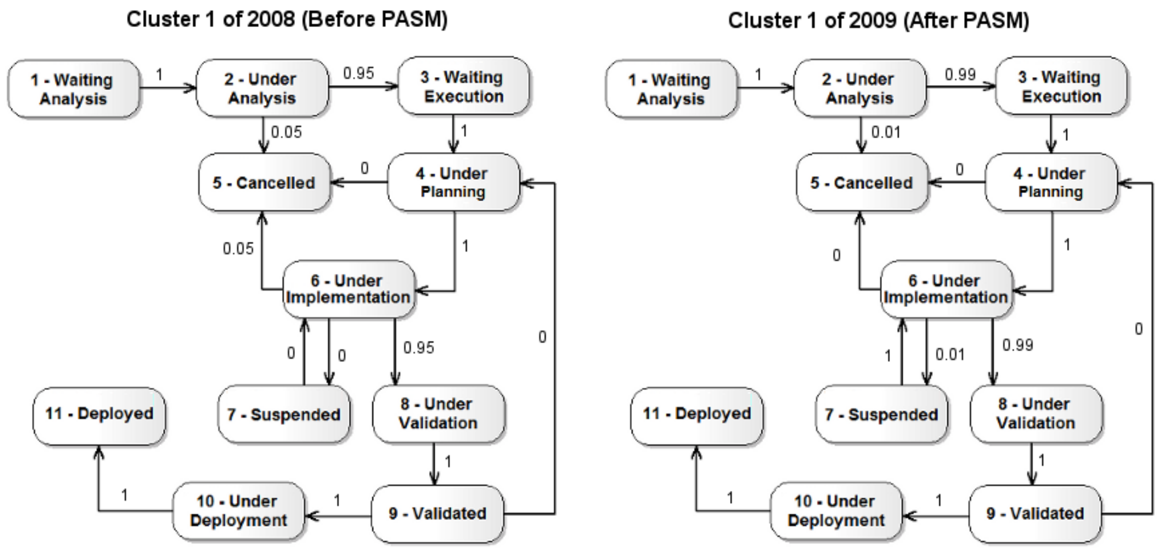

Figure 3. SMRMG of Software Maintenance Projects for Cluster 1 (2008 and 2009)

the results at Table III, we can observe that for the projects in this cluster the *ImplementationTime* corresponds to 23.1% from the *ServiceTime* (50.39 / 218.19). Furthermore, the *ValidationTime* corresponds to 43.1% from the *ServiceTime* (94.19 / 218.19). Therefore, by constrasting this cluster with the typical projects before PASM (Cluster 1, Before PASM) we observe an increase in the time dedicated to validation (from 22.7% to 43.1%) and a decrease in the time allocated to implementation (from 43.2% to 23.1%). Such results are an indication of the benefits in terms of software quality achieved by the PASM adoption: developers have dedicated less time to implementation and more time to validation.

On the other hand, we can observe that on average the *WaitTime* of the requests in this cluster corresponds to 10.9% from their *QueueTime* (26.85 / 245.04). This represents a small increase regarding the typical projects before PASM (from 6.6% to 10.9%). However, this increment represents the price – in terms of waiting time – end-users must pay when a organization moves from a continuous to a periodic policy for handling maintenance requests.

- *After PASM, Cluster 2*: This cluster includes 10 complex projects that demanded almost 695 hours of *QueueTime*. By verifying the details of such requests, we observed that they are projects proposed to renovate a complete subsystem. It is also important to mention that the CV mean time value for *AnalysisTime* is the highest among the clusters presented at Table III), which points out the difficulty of the developers in dealing with long-term and complex projects.

To conclude our analysis, Figure 3 presents the SMRMGs associated to the clusters classified as including typical maintenance projects (Cluster 1, before and after the PASM). Essentially, such graphs reproduce the SMRMG describing the workflow followed by DATAPUC to handle maintenance requests (Figure 2), but with a probability in the edges. This value denotes the probability of following the associated transition

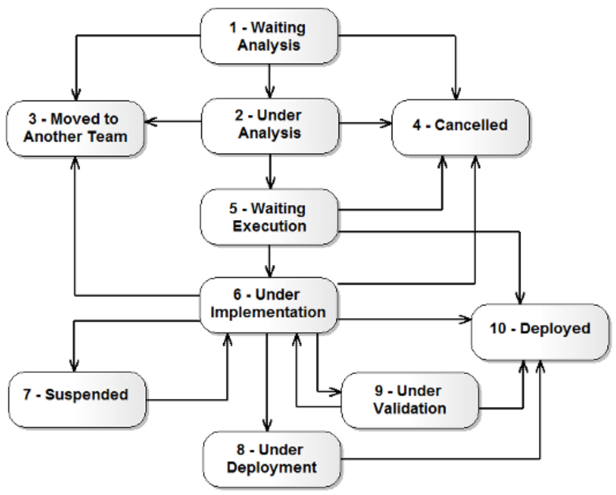

Figure 4. SMRMG for Urgent Maintenance Requests

(considering the universe of maintenance requests represented by the cluster). We can observe two differences between the presented SMRMGs. First, the probability to cancel a request was 0.05 before the PASM adoption (transitions from states 2-5 and 6-5). After adopting the PASM, this probability has been reduced to 0.01. Second, after the PASM adoption, the probability to suspend a project has been 0.01 (it was zero before the process adoption). However, only one project has been suspended in 2009, after DATAPUC has migrated to our process. Therefore, this single project does not have statistical significance (we can consider it as an outlier).

2) Urgent Maintenance: We have also clustered the urgent maintenance requests handled before and after the PASM adoption. Our intention was to investigate whether the proposed process has had influence in the workflow usually followed to handle such maintenances.

Figure 4 shows the SMRMG we have used to cluster urgent requests. Basically, this SMRMG is similar to the graph proposed for maintenance projects (showed at Figure 2) with two important differences: (a) we do not have anymore a *Under Planning* state, since urgent requests are by nature short-term tasks and planning can be performed during the analysis phase; (b) there is a new state called *Moved to Another Team* (State 3) that denotes the situation whether a urgent request is assigned to a wrong development team, which is usually discovered during the analysis phase. In this case, the request must be reallocated to another team and a new SMRMG is created to denote its processing by the new team.

Clustering the SMRMGs for urgent maintenance requests has generated five clusters for each time period: before PASM (clusters showed at Table IV) and after PASM (clusters showed at Table V). We will start by evaluating the clusters found for the period before the PASM adoption. The two clusters with more requests in this time interval are the following:

- *Before PASM, Cluster 2*: This cluster contains 651 urgent maintenance requests, with a *QueueTime* of 16.79 hours and both *ValidationTime* and *DeploymentTime* equal to zero. Therefore, due to their urgency, such requests have been deployed right after their implementation has finished. We will refer to this cluster as the typical urgent maintenance requests.

- *Before PASM, Cluster 5*: This cluster has 222 urgent maintenance requests, with a *QueueTime* superior to the previous described cluster (25.82 hours). This increase is due to the fact such requests – probably due to their potential do compromise the correct behavior of the system – have demanded 10.63 hours for *ValidationTime*. We will refer to this cluster as the error-prone urgent maintenance requests.

Next, we have reasoned about the clusters generated for the period after the PASM adoption. The two clusters with more requests in this time interval are the following:

- *After PASM, Cluster 5*: This cluster contains 383 urgent maintenance requests, which can be classified as typical according to our previous criteria (both *ValidationTime* and *DeploymentTime* equal to zero). The *QueueTime* for the requests in this cluster was 12.82 hours, therefore almost four hours less than the typical and urgent requests handled before the PASM adoption (Cluster 2, before PASM).

- *After PASM, Cluster 2*: This cluster has 212 urgent maintenance requests and the highest *QueueTime* among the clusters found for the period after the PASM adoption (26.51 hours). Basically, this is explained by the number of hours allocated to validation tasks (13.89 hours). Therefore, this cluster can be denoted as including the error-prone and urgent maintenance requests, according to our previous definition. Furthermore, we have observed a slightly increase in the *WaitTime* when compared to the error-prone and urgent requests clustered before the PASM adoption (from 25.82 hours to 26.51 hours).

To summarize, after adopting the PASM process DATAPUC has achieved on average a six hour reduction on the delivering of typical urgent maintenance requests. Probably, such gain is explained by the fact that the PASM has introduced in the organization a systematic process to filter urgent maintenances and to buffer maintenance requests that are not urgent. In other words, the process contributes to reduce the chances of incorrectly handling a maintenance as urgent. Finally, we have observed a small increase on the average time dedicated to more complex and error-prone urgent maintenance requests. However, this increase was due to more time allocated to validation tasks, which can be considered a positive side-effect (since

Table IV
CLUSTERS FOR URGENT MAINTENANCE REQUESTS (BEFORE PASM)

Characteristics	**Before PASM (2008)**									
	Cluster 1 (36 requests)		*Cluster 2* (651 requests)		*Cluster 3* (30 requests)		*Cluster 4* (112 requests)		*Cluster 5* (222 requests)	
	Mean	*CV*	*Mean*	*CV*	*Mean*	*CV*	*Mean*	*CV*	*Mean*	*CV*
QueueTime	31.73	1.98	16.79	3.67	29.42	2.06	16.92	1.03	25.82	2.35
WaitTime	2.56	1.72	2.80	1.89	4.08	1.24	2.25	1.76	2.46	1.97
ServiceTime	29.16	2.11	13.98	4.38	25.34	2.33	14.67	1.14	23.18	2.61
AnalysisTime	1.53	2.71	4.46	9.30	11.79	4.41	2.60	1.88	8.33	6.56
ImplementationTime	4.74	1.57	7.02	5.17	12.10	2.73	1.64	2.34	2.28	2.57
ValidationTime	7.02	2.57	0.00	0.00	0.00	0.00	0.49	8.30	10.63	2.24
DeploymentTime	0.00	0.00	0.00	0.00	0.00	0.00	9.82	1.34	0.00	0.00

Table V
CLUSTERS FOR URGENT MAINTENANCE REQUESTS (AFTER PASM)

Characteristics	**After PASM (2009)**									
	Cluster 1 (201 requests)		*Cluster 2* (212 requests)		*Cluster 3* (96 requests)		*Cluster 4* (61 requests)		*Cluster 5* (383 requests)	
	Mean	*CV*	*Mean*	*CV*	*Mean*	*CV*	*Mean*	*CV*	*Mean*	*CV*
QueueTime	20.87	1.23	26.51	1.73	10.44	1.84	22.15	1.02	12.82	1.95
WaitTime	3.28	1.48	3.05	1.69	2.73	1.55	3.27	1.28	3.22	1.72
ServiceTime	17.59	1.43	23.46	1.91	7.38	2.54	18.88	1.14	9.60	2.48
AnalysisTime	2.54	2.65	3.23	3.79	4.72	2.85	3.07	1.80	2.45	3.66
ImplementationTime	4.05	2.41	4.27	4.15	0.98	6.06	3.80	2.12	3.94	3.64
ValidationTime	3.98	3.47	13.89	1.51	0.16	9.80	5.81	1.55	0.00	0.00
DeploymentTime	6.78	1.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

it is well-known that validation activities contribute to increase the final quality of a software product).

IV. RELATED WORK

Related work can be arranged in two groups: (a) works proposing models to evaluate and optimize costs of software maintenance and (b) works proposing process and methods to guide software maintenance.

Cost Models: Banker and Slaughter were among the first to investigate the benefits provided by the treatment of maintenance as software projects [7]. In order to support their claims, they have evaluated 129 software enhancement projects from a large financial organization. Their empirical evaluation has been conducted using a non-parametric method – called Data Envelopment Analysis (DEA) – commonly used to measure productivity. They conclude that a cost reduction of up to 36% could have been achieved by clustering the evaluated requests into larger projects. However, they have not conducted subsequent field experiments in order to confirm such gains.

Tan and Mookerjee have proposed a fairly complete model to help project managers to set a timetable for handling maintenance requests [8]. Their model includes not only decisions about maintenance, but also decisions about replacement of a system. Sometimes, the internal quality of a system reaches a level of degradation that recommends its completely replacement (to avoid the costs of its maintenance). Thus, the analytical model proposed estimates not only gains due to scale economies but also gains derived from the complete renovation of a system. However, their mathematical model has not been validated in real-world software development organizations. In addition, it depends on several parameters such as cost reduction due to reuse (in dollars/month), cost reduction due to the incorporation of new technologies (in dollars/month), entropy degree of the system (i.e. system degradation due to maintenance) etc. In general, finding reliable values for these parameters is a complex task.

Feng, Mookerjee and Sethi have presented a dynamic programming formulation for the software maintenance problem [15]. Assuming that maintenance requests arrive in accordance to a Poisson distribution, their model aims to optimize the long-run total discounted cost of the system (i.e. the waiting cost of each pending maintenance requests plus maintenance costs). They assume that the waiting cost per time unit of each outstanding request is known. It also assumed that the cost of maintenance consists of a fixed cost and a correction cost per request. Therefore, their dynamic programming equations does not consider that fixed costs can be saved when two or more maintenance requests are grouped (i.e. that it is possible to achieve economies of scale in software maintenance).

The Software Maintenance Project Effort Estimation Model (SMPEEM) uses function points to calculate the effort required to deliver a set of maintenance requests [16]. The model is based on three main factors: the engineer's skill (people domain), technical characteristics (product domain) and the maintenance environment (process domain). However, the proposed model does not consider the benefits in terms of scale economies that can be obtained when requests are grouped in larger software development projects.

Processes and Methods: MANTEMA is a methodology for managing the software maintenance process. It proposes a iterative set of activities and tasks to handle modification requests (MR) [17]. However, MANTEMA does not include an explicit activity where MRs are buffered, evaluated and then grouped in larger projects, i.e. the methodology assumes that MRs are handled in a first-come, first-served basis. MANTOOL is a tool for managing the software maintenance process according to the MANTEMA methodology [18].

The Software Maintenance Life Cycle Model (SMLC) recognizes four non-stationary stages in the life of an application software system: introduction, growth, maturation, and decline [19]. The first three stages are dominated by user support, repair, and enhancement requests, respectively. In the last stage, users and project managers start to plan the replacement of the system. Later, Kung have proposed a quantitative decision method to detect changes from a stage to the next one [20]. The goal is to help managers to forecast the demands of maintenance. However, SMLC does not tackle the central goal of the PASM process, i.e. the benefits that can be achieved by grouping maintenance requests.

Niessink and Van Vliet have investigated the differences between software maintenance and software development [21]. Their central argument is that maintenance should be seen as providing a service, whereas development is concerned with the construction of products. Consequently, maintenance should be guided by different processes than those normally used during the development phase. Based on such conclusions, they proposed an IT Service Capability Maturity Model, i.e. a maturity model that focuses on services rather than on products. Donzelli have described the challenges faced on the integration of different software development methods when evolving an aircraft mission system [22]. However, both works do not aim to handle maintenance requests under a periodic policy, as proposed by the PASM process.

By using data from a real project, Donzelli has showed that different approaches and techniques can be combined to implement a software maintenance and evolution process that contributes to reduce maintenance costs [22]. However, among the suggested techniques, he does not consider periodic maintenance policies, as those proposed by the PASM process.

V. Conclusions

This study has proposed and evaluated a process to support the planning and organization of maintenance requests, named by PASM. The proposed process provides a set of guidelines to support the adoption of a periodic maintenance policy rather than a continuous policy, commonly practiced by organizations with in-house maintenance teams. Our results derived from deployment PASM at a real-world setting have demonstrated that grouping software maintenance requests can be performed without compromising the maintenance and evolution of a software system. Moreover, our results have corroborated the conclusions of previous studies, which argue that grouping maintenance requests helps to reduce maintenance costs due to the gains by scale economies. Moreover, our study has showed other benefits provided by the PASM process, such as increase in the time allocated to understand the software maintenance requests and to validate the new release of the system under maintenance.

As future work we intend to foster and to evaluate the adoption of the PASM process by other organizations and to assess the quality of the software released after the adoption of the proposed process (for example, using classical metrics for object-oriented systems [23], [24]).

Acknowledgment

This research was supported by grants from CNPq and FAPEMIG.

References

[1] L. Erlikh, “Leveraging legacy system dollars for e-business,” in *IEEE IT Pro*, 2000, pp. 17–23.

[2] I. Sommerville, *Software Engineering*, 7th ed. Addison Wesley, 2004.

[3] J. Sutherland, “Business objects in corporate information systems,” *ACM Computing Surveys*, vol. 27, no. 2, pp. 274–276, 2000.

[4] G. V. Neville-Neil, “The meaning of maintenance,” *ACM Queue*, vol. 7, no. 7, pp. 20–22, 2009.

[5] B. Boehm, “The economics of software maintenance,” in *Software Maintenance Workshop*, 1983, pp. 9–37.

[6] P. Stachour and D. Collier-Brown, “You don’t know jack about software maintenance,” *Communications ACM*, vol. 52, no. 11, pp. 54–58, 2009.

[7] R. D. Banker and S. A. Slaughter, “A field study of scale economies in software maintenance,” *Management Science*, vol. 43, no. 12, pp. 1709–1725, 1997.

[8] Y. Tan and V. S. Mookerjee, “Comparing uniform and flexible policies for software maintenance and replacement,” *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 238–255, 2005.

[9] R. D. Banker and C. F. Kemerer, “Scale economies in new software development,” *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1199–1205, 1989.

[10] D. Menasce and V. Almeida, *Scaling for E Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, 2000.

[11] D. A. Menasce, V. Almeida, R. C. Fonseca, and M. A. Mendes, “A methodology for workload characterization of e-commerce sites,” in *ACM Conference on Electronic Commerce*, 1999, pp. 119–128.

[12] IEEE, "IEEE standard for software maintenance, IEEE std 1219-1998," in *IEEE Standards Software Engineering, Volume Two: Process Standards*. New York: IEEE Press, 1999, vol. 2.

[13] H. Marques-Neto, J. Almeida, L. Rocha, W. Meira, P. Guerra, and V. Almeida, "A characterization of broadband user behavior and their e-business activities," *SIGMETRICS Performance Evaluation Review*, vol. 32, no. 3, pp. 3–13, 2004.

[14] J. Hartigan, *Clustering Algorithms*. John Wiley and Sons, 1975.

[15] Q. Feng, V. S. Mookerjee, and S. P. Sethi, "Optimal policies for the sizing and timing of software maintenance projects," *European Journal of Operational Research*, vol. 173, no. 3, pp. 1047–1066, 2006.

[16] Y. Ahn, J. Suh, S. Kim, and H. Kim, "The software maintenance project effort estimation model based on function points," *Journal of Software Maintenance*, vol. 15, no. 2, pp. 71–85, 2003.

[17] M. P. Usaola, M. P. Velthuis, and F. R. Gonzalez, "Using a qualitative research method for building a software maintenance methodology," *Software Practice and Experience*, vol. 32, no. 13, pp. 1239–1260, 2002.

[18] ——, "Mantool: a tool for supporting the software maintenance process," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, no. 2, pp. 77–95, 2001.

[19] H.-J. Kung and C. Hsu, "Software maintenance life cycle model," in *14th IEEE International Conference on Software Maintenance (ICSM)*, 1998, pp. 113–121.

[20] H.-J. Kung, "Quantitative method to determine software maintenance life cycle," in *20th IEEE International Conference on Software Maintenance (ICSM)*, 2004, pp. 232–241.

[21] F. Niessink and H. van Vliet, "Software maintenance from a service perspective," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 12, no. 2, pp. 103–120, 2000.

[22] P. Donzelli, "Tailoring the software maintenance process to better support complex systems evolution projects," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 1, pp. 27–40, 2003.

[23] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[24] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, "Predicting maintenance performance using object-oriented design complexity metrics," *IEEE Transactions on Software Engineering*, vol. 29, no. 1, pp. 77–87, 2003.