

Managing Resource Allocation in Software Maintenance

Sriram Narayanan*

Jayashankar M. Swaminathan**

Sridhar Balasubramanian***

DRAFT VERSION: PLEASE CITE ONLY WITH PERMISSION

* Department of Supply Chain Management, Eli Broad School of Business, Michigan State University, East Lansing, MI 48834. Tel: (517) 432-6432; Email sriram@msu.edu

**Department of Operations, Technology, and Innovation Management, Kenan-Flagler Business School, University of North Carolina at Chapel Hill, Campus Box 3490, McColl Building, Chapel Hill NC 27599-3490. Tel: 919-843-8341; Fax: 919-843-4897; Email: msj@unc.edu

***Department of Marketing, The Kenan-Flagler Business School, The University of North Carolina at Chapel Hill, Campus Box 3490, McColl Building, Chapel Hill NC 27599-3490. Phone: (919) 962-3194; Fax: (919) 962-7186; Email: Sridhar_Balasubramanian@unc.edu

Managing Resource Allocation in Software Maintenance

We examine the optimal allocation of engineering resources in software maintenance operations. We first present competing theoretical models of software defect resolution. Next, we estimate these models using field data on defect resolution for 12503 defects that occurred over 97 months in a large systems software product. We then apply the most appropriate model – one based on a beta-geometric distribution – to estimate the level of engineering resources required to manage debugging efforts. We propose heuristics based on temporal cut off policies to maximize the rate of successful resolution of bugs. We empirically demonstrate that such policies can be applied to enhance productivity in this complex environment. In particular, we show that the proposed cut off policies minimally impact rates of successful resolution while substantially reducing waiting times for incoming bugs in the system, thereby improving overall productivity and resource utilization. Finally, we extend our model to consider two alternative scenarios where (a) incoming bugs differ on priority levels assigned for resolution and (b) incoming bugs are from different populations.

Keywords: Software maintenance; Resource allocation; System productivity; Heuristics; Queuing approximations

1. Introduction

Software maintenance is costly. Maintenance costs comprise between 50% and 80% of the overall information systems (IS) budget (Nosek and Palvia 1990) and can outweigh development costs by a factor between 2 and 10 (Scacchi 1994). A recent report by the National Institute of Standards and Technology (NIST) estimates that the annual cost of software bugs to the US economy is a staggering \$59.5 billion (NIST 2002). A considerable amount of the maintenance expense – close to 75% – is expended on labor (Amoribieta et al. 2001). Further, the software sector also suffers from a combination of labor scarcity and high employee turnover (Sudhakar 2002). At the same time, except for simple pieces of code, developing bug-free software is a laudable but unrealistic goal (Lieberman 1997).

In this context, efficient and effective utilization of scarce engineering resources is a key challenge for managers who are responsible for software maintenance operations. Software maintenance can be divided into corrective, perfective, adaptive and preventive maintenance (Leintz et al. 1978). Corrective maintenance involves fixing bugs in the software. On the other hand, perfective, adaptive and preventive maintenance tasks may involve adding new features to improve functionality of the software or modifying the software specifications to meet future needs. Considerable research in the software maintenance arena has been oriented towards understanding software enhancement (e.g., Banker et al.

2002; 1991; 1998; Banker and Slaughter 1997; Kemerer and Slaughter 1997; Krishnan et al. 2004). Thus far, research on software debugging has mainly focused on individuals and their working environment. For example, research has aimed to understand the cognitive behavior of engineers (individuals) undertaking the debugging task (Hale and Haworth 1991; Hale et al. 1999). Other researchers have explored how productivity can be enhanced by increasing engineers' rates of learning (Boh et al. 2007; Narayanan et al. 2006) and by managing team dynamics, including new employee joining and employee turnover (Narayanan et al. 2006). In this paper, we study how the attributes of the debugging process can be exploited to craft and implement policies to effectively manage the maintenance process. To this end, we focus our attention on the estimation, allocation, and utilization of scarce resources in software maintenance operations.

To the best of our knowledge, there is no existing work which examines resource allocation and management issues in the software debugging context. Our work addresses this gap and contributes to the literature by examining the following research questions: How should managers determine the required engineering capacity to manage debugging tasks? How should this capacity be allocated to efficiently manage the debugging tasks? Finally, given that managers may need to allocate resources to multiple types of incoming bugs, what are the implications of capacity choice on system performance in terms of bug queue lengths and resolution rates?

Our decision model involves two stages. In the first stage, we draw from theory and propose alternative statistical models of bug resolution. The models are then estimated and validated using data sourced from a large software company – this data pertains to the resolution of 12503 bugs over a period of 97 months. Based on this analysis, we find that the longer the bug stays in the system, the lower is the probability of its successful resolution during each successive period. We use the beta-geometric distribution to model this phenomenon – the estimated model serves as an input to the second stage. In the second stage, we present a queuing model for bug resolution where we consider engineers as parallel servers and use the estimated resolution probability from the first stage to determine expected service times for bug resolution. We then formulate an optimization model that maximizes the expected rate of successful resolution and propose temporal cut-off policies for resolution based on the time that bug has been with an engineer. We demonstrate that such policies minimally impact rates of successful bug

resolution and simultaneously reduce waiting times for the assignment of incoming bugs to engineers, thereby improving overall system productivity. Finally, we extend our analysis to two alternative scenarios where (a) incoming bugs differ on priority levels assigned for resolution and (b) incoming bugs are from different populations. Using field data and interviews, we ensured throughout that our modeling approach closely reflected the realities and practical constraints of the software maintenance operation that we study.

In §2, we describe the research setting and the data. In §3, we present and estimate competing statistical models of bug resolution times and demarcate the best-fitting model. In §4, we present the optimization model and explore the implications of temporal cut off policies. In §5, we use queuing approximations to analyze the tradeoff between the imposition of temporal cutoff policies and the waiting time of bugs in the system for two different queuing systems – First Come First Serve (FCFS) and Pre-Emptive Resume priority (PRI). In §6, we extend our analysis of temporal cutoff policies to multiple groups. Finally, in §7, we outline the limitations of the analysis, discuss managerial implications, and present some generalizations.

2. Research Setting and Data

Research Site

The data for the study was sourced from a large, India-based, export-oriented software services provider. The firm employs over 20,000 engineers and develops and maintains both application and system software. Exports comprise more than 90% of the firm’s revenue. The research effort included on-site field work conducted over several months at the firm’s operating sites in India. The data used in this study covers 12503 debugging tasks over a period of 97 months. The bugs originated from a large piece of system software and were assigned to individuals in 31 different project teams.

Resolution Process

New bugs were identified in numerous ways: (1) from products deployed in the field (found by customers); (2) during integration testing on the completed software release (found by regression testers); (3) during testing of individual modules before system integration (found by unit testers); (4) during pre-deployment testing of software in simulated environments in actual or near actual environments (found

during alpha and beta testing); and (5) during activities such as the fixing of existing bugs, new code development, code compilation and code walk through (found by maintenance engineers or developers). Figure 1 describes the bug resolution process. Each incoming bug entered a waiting line. As and when an engineer with workflow capacity became available, the bug was assigned to him or her.¹ The first task was to determine the validity of the bug. Once it was determined that the problem was worth debugging, the engineer spent time and energy towards this task. Valid bugs were then reproduced in a laboratory setting. Valid bugs could ultimately reach four possible end states – *closed* (i.e., the bug report was valid, but a conscious decision was made not to fix the bug)², *duplicate* (i.e., the bug report described the same problem as in another report encountered earlier), *irreproducible* (i.e., the bug could not be reproduced in laboratory conditions by the evaluating engineer) and *resolved* (i.e., the bug had been resolved and fixed). The duplicate and resolved states of the bug were classified as *successfully resolved* while the closed and irreproducible bugs were classified as *unsuccessfully resolved*. The successful resolution of a bug contributed to improvement in product quality because the bug had either been fixed or was tagged to a known problem.

Managerial Dilemma

Despite employing large numbers of engineers, management found it very difficult to keep the average resolution times within reasonable limits. A key problem brought to our attention was that some bugs tended to absorb a significant fraction of a typical engineer’s time. This not only increased average resolution times but also delayed the assignment of incoming bugs – both these outcomes negatively impacted customer satisfaction. The management team was interested in exploring policies that would improve resolution times and overall system productivity.

In order to more completely understand the situation, we first collected data related to 12503 bugs that spanned over 97 months (see Table 1). Note that Table 1 presents the resolution data with the time span of resolution divided into months (or *periods*) taken to resolve a given bug. Stated differently, all bugs are assumed to have arrived at period 0. In each period, some bugs are successfully resolved, some

¹ In case of a large available capacity, the pre-allocation waiting time could be zero.

² The rights to close a bug are available to the component owner (a senior manager). Examples of reasons to close a bug may include: (a) the occurrence of the bug represents a “one off” case that is not expected to repeat because the customer used the software in conditions that were not in conformance to specifications; or (b) the technical changes needed to fix the bug could potentially create problems in other features of the software.

are designated as closed or irreproducible – i.e., unsuccessfully resolved – and the remaining bugs are carried forward into the next period for the resolution process to be repeated. As seen in Figure 2, the proportion of successful resolutions decreases with time. If this decrease evidenced in Table 1 and Figure 2 is significant, then the population of bugs (tasks) in an engineer’s portfolio over time will increasingly consist of bugs that have a low probability of successful resolution. In such a situation, the effort invested in addressing these bugs may come at the expense of effort that could be invested in resolving new incoming bugs that have a greater probability of being resolved. Focusing on unresolved bugs for lengthy periods of time could also lead to significant waiting times for entering bugs to be allocated to engineers. In order to manage such a situation, we design cut off policies for resolution based on a threshold time that a bug spends in the system without successful resolution. Towards this objective, we first present alternative models of the bug resolution process.

----- Insert Figure 1, Figure 2, and Table 1 about here -----

3. Alternative model formulations

Table 1 summarizes the data for the 25362 bug-months that the 12503 bugs spent in the system. As a naïve approach, one could assume that the conditional probability (p) of successful bug resolution in each time period is homogeneous – this corresponds to a geometric distribution. The corresponding maximum likelihood estimate (evaluated using data given in table 1) is $p = 0.381$, with a standard error of 0.003. Note that this is identical to the maximum likelihood estimate of the resolution probability provided by Table 1 ($9672/25362=0.381$). A plot of the expected and the actual number of resolutions is shown in Figure 3, and the corresponding log likelihood and chi-square statistics are shown in Table 2. A plot of expected and actual resolution rates shown in Figure 3 does not suggest a good model fit.

Further, note that the reduction in the ratio of bugs resolved over time suggests that at any given instance of time, the population of bugs is heterogeneous – some bugs are more likely to be resolved earlier than others. Interestingly, a similar heterogeneity has been observed in fertility studies (Kaplan et al. 1992; Weinberg and Gladen 1986; Suchindran et al. 1974). In that setting, some women undergoing

fertility treatment are more likely to become pregnant than others, and some fraction of women will never become pregnant regardless of the number of trials.³

We now capture heterogeneity in incoming bugs using three competing specifications and assess model fits along multiple dimensions (graphical comparison, pearson chi-square and log likelihood values – AIC criterion). All the models share one characteristic – as time goes by, the probability of successful bug resolution decreases. However, each model differs in its assumptions about the source of heterogeneity.

----- Insert Table 2 and Figure 3 about here -----

The Beta-Geometric Model

To capture the underlying heterogeneity one could assume that the probability of successful resolution p in a geometric model is drawn from a general density function specified by $f(p)$. In this case, the probability of a successful resolution during time t can be written as:

$$p_t = \int_0^1 p(1-p)^{t-1} f(p) dp \quad (1)$$

Assuming that p is drawn from a beta distribution with parameters α and β , the conditional probability of successfully resolving a bug in period t given that it has not been successfully resolved until $t-1$ is:

$$p_t = \frac{\alpha}{\alpha + \beta + t - 1} \quad (2)$$

Based on eqn.2, we can see that p_t – the probability that the bug is resolved in a given time period – is monotonically decreasing with time, and that $p_t \rightarrow 0$ as $t \rightarrow \infty$. The parameter estimates of α and β and the standard errors of estimates for the parameters for this model are presented in Table 2 (Beta-Geometric hazard case). Figure 4 shows the relative fit of the actual and the expected values.

----- Insert Figure 4 about here -----

Split Population Geometric Model

³ Such heterogeneity has important operational implications for managing capacity and the flow of patients in service systems such as in-vitro fertilization centers in hospitals (Kaplan et al. 1992). As we will show, heterogeneity in bugs will influence resource allocation decisions in the software maintenance context as well.

In the split population model, heterogeneity is captured by assuming that incoming bugs are split into two fractions – one fraction can be resolved and the other fraction can *never* be resolved. Specifically, we assume that a fraction θ of the population of incoming bugs cannot be resolved, and in the other fraction the probability of successful resolution at time t (p_t), follows geometric distribution given by $p_t = p(1-p)^{t-1}$. Accordingly, the conditional probability of successfully resolving the bug during time t given that the bug has not been resolved until $t-1$ is given by:

$$p_t = \frac{\theta p (1-p)^{t-1}}{1-\theta + \theta(1-p)^{t-1}} \quad (3)$$

Similar to eqn. 2, eqn. 3 has the property that $p_t \rightarrow 0$ as $t \rightarrow \infty$. Figure 5 shows the relative fit of the actual and the expected values. The parameter estimates of p and θ , standard errors, and fit measures are described in Table 2.

----- Insert Figure 5 about here -----

Trinomial Model

The split population model and the beta-geometric model do not explicitly parameterize the bugs that are declared *unsuccessfully resolved* in each time period. This can be addressed by splitting the bugs into three groups at each time period – successfully resolved, unsuccessfully resolved, and carried forward. We assume that the probability of resolution p in time t is given by the beta-geometric hazard (see equation 2) and that the fraction of bugs that are unsuccessfully resolved in each time period is q . We estimate three parameters – namely α , β , and q to characterize the distribution of successfully resolved bugs across periods. Parameter estimates, standard errors, and fit measures are described in Table 2. The graph describing the relative fit between the actual and expected values is in Figure 6.

----- Insert Figure 6 about here -----

The performance of the three models can be compared using Akaike Information Criterion (AIC), calculated as $(-2LL + k)$ where k is the number of model parameters, and LL is the Log-Likelihood. The beta-geometric hazard model exhibits the lowest AIC. Further, based on the Chi-Square statistic, the beta-geometric hazard model exhibits the smallest deviation between actual and expected values among the

competing models.⁴ A comparison of the plots of the predicted and actual values of successfully resolved bugs across the models (Figures 3, 4, 5 and 6) also indicates that the beta-geometric distribution performs well. Finally, one may expect the three-parameter trinomial model to outperform the two parameter beta-geometric model, but this may not be the case here because the models are non-nested. Based on the above rationale, we choose the beta-geometric model for our subsequent analysis.

The proportion of bugs with a low probability of successful resolution in the set of carried-over bugs increases with time. These bugs absorb resources that could be productively deployed to address new, incoming bugs that have a higher probability of resolution. Further, an excessive focus on the carried over bugs can lengthen the waiting times for incoming bugs to be assigned. Therefore, in our analysis we consider imposing a threshold policy on time until which the bugs can stay in the system and assess the impact of the policy on resource allocation.

4. Impact of temporal cutoff policies

We now investigate the implications of imposing a threshold time to cut off working on a bug (denoted by a) and move it to an *unsuccessfully resolved* state. We first estimate the implications of this policy for the combined dataset covering 12503 bugs. We assume that all incoming bugs originate from a single source. In part, the methodology below follows Kaplan et al. (1992).⁵ Later, in §5, we extend the analysis to a multiple population setting where we differentiate the bugs based on the originating source.

Let U be the time period when a bug exits the system as *unsuccessfully resolved*. Let S be the time period when a bug exits the system as *successfully resolved*. Let X indicate the actual number of periods taken to resolve the bug. During each period the bug is active, the engineer may discover some knowledge that prompts the decision to declare it as unsuccessfully resolved, or may end up successfully resolving it, or may carry it forward to the next period. Therefore:

⁴ Despite the good fit displayed in the graphs, the p-values of the likelihood ratio chi-square tests are all significant in Table 3, i.e., the tests reject the null hypotheses that the models predict the data well. However, this is not surprising because the chi-square test is sensitive to sample size (Segars and Grover 1993). As a robustness check, we ran models with smaller data samples corresponding to a sub-sample of engineers. The beta-geometric model continued to perform well and, as expected, the chi-square test failed to reject the premise that the model fits the data well.

⁵ Kaplan et al. (1992) use a split-population geometric model to study pregnancy outcomes from in vitro fertilization.

$$X = \text{Min} \{U, S\}$$

$$P\{X \geq x\} = P\{U \geq x\}P\{S \geq x\}, \quad x = 1, 2, 3, \dots \quad (4)$$

The expected number of time periods a bug lasts in the system is:

$$e = \sum_{x=1}^{\infty} P\{X \geq x\} \quad (5)$$

When bugs are moved to an *unsuccessfully resolved* state after a threshold of a time periods, $P\{U(a) > a\} = 0$ and $P\{S(a) > a\} = 0$, where the probability of successful and unsuccessful resolution are functions of the cut off time period a . If a bug enters period a , it implies that it did not reach either resolved or unresolved states until period $a-1$. The probability that a randomly chosen bug is successfully resolved under a threshold policy of a time periods is denoted by:

$$P(a) \Rightarrow P(X = a) = \sum_{x=1}^a P\{X(a) \geq x\} p_x \quad (6)$$

where p_x is the probability of a bug being successfully resolved in period x conditional on the fact that it lasts until period $x-1$ (described in eqn. 2). Assuming that the arrival process is Poisson with rate λ per month consistent with related literature in software reliability (van Pul 1994), n engineer slots are available each month for assignment of incoming bugs, and a utilization rate of $\rho(a)$, $n\rho(a)$ bugs will be worked on at any point in time.⁶ The exit rate of bugs from the system is $n \frac{\rho(a)}{e(a)}$ (see Figure 7). In

steady state, the condition for stability of the queue is:

$$\rho(a) = \frac{\lambda e(a)}{n} < 1 \quad (7)$$

Therefore, the optimization problem to maximize the overall number of *successful resolutions* given an available capacity of n engineer slots as shown below:

$$\text{Max}_{a=1,2,3,\dots} \lambda P(a) \quad (8)$$

$$\text{Subject to } \lambda e(a) < n \text{ where, } a = 1, 2, 3, 4.. \quad (9)$$

⁶ An engineer-bug association is called a slot. An engineer could be assigned multiple bugs in a month. For example, if five engineers could each work on 4 bugs simultaneously, there are 20 available slots.

To maximize eqn. 8, we need to find the maximum a that meets the constraints in eqn. 9. In addition, the maximum number of time periods a bug is allowed to stay in the system under a threshold policy of a months before being declared as unsuccessfully resolved can be written as:

$$P\{U(a) \geq x\} = \begin{cases} \prod_{t=1}^{x-1} (1 - d_t), & a = 1, 2, 3, \dots \\ 0 & x > a \end{cases} \quad (10)$$

where, d_t is the conditional probability of moving the bugs to *unsuccessfully resolved* at the end of month t given that no decision has been reached on the state of the bug until month $t-1$. Using eqns. 4 and 10, we can rewrite eqn. 6 as:

$$P(a) = \sum_{x=1}^a \left(\prod_{i=1}^{x-1} (1 - d_i) P\{S(a) \geq x\} \left(\frac{\alpha}{\alpha + \beta + x - 1} \right) \right) \quad (11)$$

where $P\{S(a) \geq x\} = \prod_{j=1}^{x-1} (1 - p_j)$ and $p_j = \frac{\alpha}{\alpha + \beta - j + 1}$.

Given that the resolution can be cut off at $t = a$, the expected time a bug stays in the system as a function of the threshold time period a is:

$$e(a) = \sum_{x=1}^a P\{X(a) \geq x\} \quad (12)$$

In eqn. 12, $e(1) = 1$, and $e(\infty)$ is the expected number of time periods a bug stays in the system when a cut off policy is not imposed. Figures 8 and 9 demonstrate how $e(a)$ and $P(a)$ change with a . As seen in Figure 9, by the end of month 7, $P(a)$ reaches 97.3 % of its peak value of 0.77.

Assuming steady state arrival of bugs, given that the 12503 bugs were filed across 97 months, the arrival rate of bugs is, $\lambda = 12503/97 = 128.9/\text{Month}$. The maximum likelihood estimate for d_x can be computed from Table 1 based on the proportion of bugs that were moved to *unsuccessfully resolved* state in period x conditional on no decision being made regarding the disposal of the bug until period $x-1$. The maximum likelihood values of α and β are shown in Table 2. The expected number of months the bug stays in the system when there is no threshold policy enforced is, $e(\infty) = 2.047$. Based on actual data in Table 1 the expected number of months the bugs stay in the system is $25362/12503 = 2.028$. Further,

$P(\infty)$ – the proportion of bugs successfully resolved – as obtained from the model is 0.77. From Table 1, we can see that this fraction is $9672/12503= 0.77$. The total number of engineer slots required is given by $\lambda e(\infty)=128.9 * 2.204 \cong 264$ engineer slots per month. The average number of engineers available per month based on the dataset is 35.79. Therefore, an engineer should be concurrently working on 7-8 bugs. This number is in broad agreement with the data available. All these imply that the data fits the model quite well.

----- Insert Table 3, Figure 7, Figure 8 and Figure 9 about here -----

5. Waiting time tradeoffs

We next analyze the effects of imposing temporal cut off policies on the waiting times of bugs in the system. In the studied operation, incoming bugs were assigned a severity rating from 1 (most severe) to 6 (least severe) by the bug filer, based on its impact on software functionality. Maintenance engineers were generally expected to address the more severe bugs first. Using approximations developed in prior literature, we analyze two different cases. First, we analyze the case where bugs are handled on a First Come First Serve (FCFS) basis. Next, we assume that the incoming population of bugs follows a preemptive resume priority queue with higher severity bugs preempting the lower severity bugs.

Non-preemptive FCFS policy

Let $W_{NP}(a)$ be the total time a bug spends in the system when a threshold policy of a time periods is enforced under the non-preemptive FCFS policy. Therefore:

$$W_{NP}(a) = e(a) + W_Q(a) \quad (13)$$

Here, $W_Q(a)$ is the waiting time in the queue (before the bug is assigned to an engineer) and $e(a)$ is the expected time that the bug is in the system. We assume that the arrival process of the bugs is Poisson – with rate λ and assume a general service process and approximate the queue to an M/G/n system. The waiting times in queue for the non-preemptive FCFS case can be computed for the M/G/n queue using the approximation suggested by Whitt (1983). $W_Q(a)$ can be approximated by:⁷

⁷ We can also employ other approximations by Boxma et al. (1979) and by Nozaki and Ross (1978). Our waiting times under all three approximations were very close to each other.

$$W_Q(a) = \frac{1}{2} \left(1 + \frac{\text{Var}(X(a))}{e(a)^2} \right) Q(\lambda, P(a), n) \quad (14)$$

Here, $Q(\lambda, P(a), n)$ is the exact delay for a M/M/n system with arrival rate λ . Table 4 shows the mean time a bug spends in the system. Note that the computation has an underlying assumption that each incoming bug can be assigned to any individual. However, the waiting time can increase more sharply if there are restrictions on assignment to particular individuals who may not be able to work on a specific kind of bug. Therefore, the gains from imposing cut off policies are likely to be higher in real life. Finally, Figure 10 displays the sensitivity of the waiting times to capacity additions.

----- Insert Table 4 and Figure 10 about here -----

Preemptive resume priority case

Our field interviews with the managers suggested that the managers followed the policy of preempting bugs of lower severity with bugs of higher severity. In this section, we understand how the preemption of the lower severity bugs by the higher severity bugs has an impact the overall waiting times in the system. To simplify our analysis of the pre-emptive queues and use the existing approximations for pre-emptive multi class queues, we divided the incoming bugs into three broad categories. This categorization is consistent with the way managers at the software firm demarcated and allocated the bugs in the resolution process. The categories were: *Highest Severity* bugs consisting of Severity 1 and Severity 2 bugs, *Medium Severity*: bugs consisting of Severity 3 and Severity 4, and finally, *Lowest Severity* bugs consisting of Severity 5 and Severity 6 bugs. To estimate the waiting times in queue under the pre-emptive resume priority scheme, we use the method suggested by Bondi and Buzen (1984):

$$W(PRI, \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, n) \approx W(PRI, n \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1) \frac{W(FCFS, \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, n)}{W(FCFS, n \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1)} \quad (15)$$

Here, $W(PRI, \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, n)$ is the waiting time for the system with n servers under a pre-emptive resume priority (PRI) discipline for the p priority classes, with an arrival rate vector $\underline{\lambda}_{(p)} = (\lambda_1, \lambda_2, \dots, \lambda_p)$ and a service rate vector $\underline{\mu}_{(p)} = (\mu_1, \mu_2, \dots, \mu_p)$. Further, $W(FCFS, \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, n)$ is the waiting time for the M/G/n system with a FCFS priority. This can be computed using the approximations in Nozaki and Ross

(1978) or Boxma et al. (1978).⁸ Further, $W(FCFS, n\underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1)$ is the waiting time for the M/G/1 system with FCFS priority with an average service rate of $n\bar{\mu}_{(p)}$. Here, $\bar{\mu}_{(p)}$ is the mean service rate weighted by the p priority levels, and is computed as $\bar{\mu}_{(p)} = \left(\sum_{j=1}^p \frac{\lambda_j}{\mu_j} \right)^{-1} \sum_{j=1}^p \lambda_j$. The waiting time of an M/G/1 queue is obtained using the *Pollaczek-Khinchin* formula:

$$W(FCFS, n\underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1) = \frac{\lambda_{(p)} E[X(a)^2]}{2n^2(1 - \rho_{(p)})} \quad (16)$$

Here, $\lambda_{(p)} = \sum_{j=1}^p \lambda_j$ for the p priority classes and $\rho_{(p)} = \sum_{j=1}^p \rho_j$. The condition of stability of the queue is $\rho(r) = \sum \lambda_i / n\mu_i < 1$. In addition, $W(PRI, n\underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1)$ indicates the average waiting time for an M/G/1 pre-emptive priority queue with p priority classes and is computed as:

$$W(PRI, n\underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, 1) = \frac{1}{\lambda_{(p)}} \sum_{i=1}^p \lambda_i \bar{W}_i \quad (17)$$

Here \bar{W}_i is the expected waiting time for individual priority class i given by:

$$\bar{W}_i = \frac{\sum_{j=1}^i \frac{\lambda_j E[X_j(a)^2]}{2}}{\left(1 - \sum_{j=1}^i \rho_j\right) \left(1 - \sum_{j=1}^{i-1} \rho_j\right)} \quad (18)$$

The approximation given in eqn. 15 as suggested by Bondi and Buzen (1984) works better when the queue of high priority customers is not very long and has been verified with simulation for three priority classes (Bondi and Buzen 1984). In our data, the High Severity group constituted about 26% of the population. Finally, total time a bug spends in the system $W(a)$ is:

$$W_{PP}(a) = W(PRI, \underline{\mu}_{(p)}, \underline{\lambda}_{(p)}, n) + 1/\bar{\mu}_{(p)} \quad (19)$$

⁸ Bondi and Buzen (1984) find that the approximation suggested by Boxma et al. (1978) gives values of waiting times closer to simulated values as opposed to the approximations suggested by Nozaki and Ross (1978). Therefore, we computed the waiting times based on Boxma et al. (1978).

Here, $W_{pp}(a)$ is the overall waiting time for the PRI case.

Based on the computations shown in Table 4, we find that an application of a preemption policy increases the overall delay significantly. This increase in delay is on account of bugs of lower priority classes piling up in the system. To some extent, the delay can be reduced by imposing more stringent cut off policies.

6. Multiple Group Model

In this section, we relax the assumption that all the bugs come from a single source and segregate the data based on the bug source. The consideration of multiple sources is important from a managerial standpoint. The average resolution time period for bugs may systematically vary across sources. Therefore, a manager may prioritize resources based on relative resolution rates of bugs from each source.

We identified six different sources of bugs that were filed. These sources differed based on the stage of the development cycle in which the bugs are filed and consequently on the knowledge of the bug filer about the specifications and the code base of the software. Bugs could be classified into those found during (i) the core code development process (henceforth called *Core-development* bugs), (ii) activities that are not directly related to the code development, but support code development such as compilation and code inspection (henceforth called *Development-support* bugs), (iii) actual deployment at the customer site (henceforth called *Customer-found* bugs), (iv) testing in an environment that simulates actual deployment within the firm premises or at customer premises before full deployment including alpha testing and beta testing (henceforth called *Pre-deployment* bugs), (v) integration testing on the product before release (henceforth called *Regression testing* bugs) and (vi) testing of individual modules for functionality of specific software components prior to integration (henceforth called *Unit testing* bugs). We then calculated the estimates of α and β for bugs from each source and the percentage of successfully resolved bugs arriving from each of the six sources (see Table 3). Any rule for allocating resources must specify how available capacity is allocated across bugs from the different sources.

The overall allocation of resources across k bug sources can be modeled by extending the formulation in eqns. 8 and 9 as follows:

$$\text{Max}_a \sum_{i=1}^k \lambda_i P_i(a_i) \quad (20)$$

$$\text{Subject to } \sum_{i=1}^k \lambda_i e_i(a_i) < n \quad \begin{array}{l} \forall i=1 \dots k \\ \forall a=1 \dots t \end{array} \quad (21)$$

Solving the above problem involves enumerating all the possible combinations of the cut off time periods and choosing the combination that has the maximum resolution rate. Further, note that based on the above formulation, if even one bug from that source is admitted then all the bugs from that source should be admitted – henceforth, this is referred to as the *equality rule*. This requires evaluating a potential 32^6 (1,073,741,824) combinations (based on 6 sources and a possible maximum of 32 time periods for bug resolution for each source). Clearly, this is computationally extensive due to the number of permutations that need to be examined. Also note that this formulation restricts the full utilization of capacity available in the system to maximize the resolution rates since capacity has to be increased in discrete blocks to accommodate incoming bugs or there is underutilization of capacity.

This disadvantage can be overcome by filling up the remaining capacity with bugs of higher resolution probability. The highest resolution rates can be obtained by assigning any available resource to the source that has the highest marginal probability of successful resolution – henceforth, called the *marginal probability rule*. Based on this insight, we can develop a greedy heuristic that allocates only a fraction of bugs that enters in any period t for a group i . To do this, (1) we calculate the marginal probability of resolution for bugs from each source across time periods (p_{it}) – i.e. we compute the conditional probability of successful resolution for group i in period t based on the

equation, $p_{it} = \frac{\alpha_i}{\alpha_i + \beta_i - t + 1}$. For example, based on data shown in Table 3, in the first period, this

probability is highest for Core-development bugs (0.655).⁹ Thus, any available capacity should first be directed to these bugs. This probability (p_{it}) will decrease for each group with time. Note that based on this equation, working on an unresolved bug from any source i for a second time period may possibly yield a higher likelihood of resolution compared to working on a new bug from a different source for the first time. Next, (2) we sort the data in decreasing order of p_{it} . Beginning with the group that has the highest p_{it} , we fix the threshold value of cut off at period t for the corresponding group i and fill the

⁹ The conditional probability (p_{it}) based on equation 2 is given by $(1.031)/(1.031+0.543+1-1)=0.655$

available capacity with all the arriving bugs of type i in that period. If we can assign all the incoming bugs in period t of type i , we move to the next lower p_{it} until all the available capacity is utilized. If we cannot accept all the incoming bugs in period t of type i , then we accept only a fraction of incoming bugs in order to fill the capacity. In the end, for any given system capacity, this heuristic provides the values of the maximum cut off time period for each group (a_i^{\max}) and the fraction of bugs carried over from $a_i^{\max} - 1$ to a_i^{\max} that should be worked on for resolution.

Creating meta-groups

Even after the above greedy heuristic the actual implementation of the marginal probability rule could be cumbersome with six groups. Therefore, we further aggregate the bugs into three meta-groups. The first meta-group comprises bugs discovered during the coding process (Core-development) and the activities that are closely related to the coding process including compilation, and code inspection tasks (Development-support). These activities are performed by development/maintenance engineers who work closely with the software code. On account of their good understanding of the code, these engineers can provide relatively precise information and guidance to the maintenance engineers who work on bug resolution. Therefore, the bugs filed within this meta-group can be expected to have a high fraction of successfully resolved bugs as compared to those filed by other entities. The second meta-group comprises bugs discovered during the integration testing phase of the product (Regression testing) and those that are found when individual functionality of the module is being tested (Unit testing). Bugs discovered during such testing may span multiple components, or multiple code bases within a component. Therefore, bugs filed by these engineers can be relatively more difficult to reproduce and resolve. Finally, the third meta-group comprises bugs filed by customers (Customer found) and bugs found during the full scale pre-deployment testing of the product (Pre-deployment). Customers are least familiar with the software code. Further, during the alpha and beta testing stages and during the live deployment of the software at customer locations, the software may be put to work in diverse environments not encountered during the formal testing process. This may lead to the discovery of bugs which are either not well specified or not easily reproducible in the lab environment. Thus, these bugs are least likely to be successfully resolved.

We now empirically verify this three-way grouping. Consider the percentage of successfully resolved bugs across the meta-groups in Table 5. As expected: (a) Core-development and Development-

support bugs have the highest percentage of successfully resolved bugs, (b) Customer-found and Pre-deployment bugs have the lowest fraction of bugs resolved, and finally (c) Regression and Unit Testing bugs fall between the other two meta-groups. Further, we can empirically demonstrate the equivalence of the pairs of sources which are combined within each meta-group by comparing parameter estimates (α and β) across the source-pairs. To do this, following Weinberg and Gladen (1986), we first transform the beta-geometric distribution described in eqn. 2 as follows (here, parameters a and b are functions of α and β):

$$\frac{1}{p_t} = a + b(t - 1) \quad (22)$$

Next, we can introduce a covariate Z into eqn 22:

$$\frac{1}{p_t} = a + b(t - 1) + Z(c + d(t - 1)) \quad (23)$$

Here Z is the dummy variable which indicates that the bug is derived from one source (out of the two bug sources being compared within a meta-group). The regression equation described in eqn. 23 was estimated using a maximum likelihood approach. Significant estimates for c and/or d would indicate that the two sources are distinct. Our tests of parameter equivalence confirmed that the source-pairs that were combined into the three distinct meta-groups were indeed homogenous. Accordingly, we computed the revised estimates of α and β for each of the meta-groups (see Table 5). Table 5 also presents the data for the predicted and actual times the bug stays in the system, and the predicted and actual probability of a bug being successfully resolved, for each meta-group. The closeness of the predicted and actual estimates in Table 5 suggests that the beta-geometric model of successful resolution probability is appropriate for each meta-group as well. Note that, based on equality rule, the maximum number of possible enumerations reduces to a more manageable 32^3 (32,768) once the six bug sources are aggregated into three meta-groups.

Figure 11 shows the optimal integer cut-off schedule for individual groups with increasing slot size for the marginal probability rule. Table 6 shows the slot allocations to each meta-group under the marginal probability rule. We see that with limited capacity, we assign all slots to core-development and development-support bugs. This is because the marginal rate of successful resolution of these bugs is the

highest. Note that for 150 slots, the optimal policy would be to impose a cut off period of two months on core-development and development-support bugs, two months on regression test and unit test bugs and one month on customer found and pre-deployment bugs. Further, the policy indicates that engineers should work on *all* core-development and development-support bugs that enter the system until the end of second period, work on *all* regression test and unit test bugs that enter the system until the end of first period *and only* carry forward only 76% of the leftover bugs from the first period in this meta-group to be worked on in the second period, and finally, work on all customer found and pre-deployment bugs entering the system till the end of the first period (no bugs in this last meta-group are carried forward into the second period). Figure 12 displays the cut off policy across different slot sizes for the equality rule.¹⁰ The resolution rates under the equality rule and the marginal probability rule for the same number of slots are contrasted in Table 6. Note that for reasonable capacity levels beyond 150 slots – at these levels all three meta-groups of bugs can be fully accommodated for at least one period – the resolution rates are similar. However, when capacity is tight (slots less than 100), the benefits from the marginal allocation rule are significant in terms of expected resolutions. Therefore, utilizing marginal probability rule is more beneficial when the capacity is low.

Finally, Figure 13 displays the relative percentage of bugs resolved – computed as $\left(\frac{\lambda_1 P_1(a_1) + \lambda_2 P_2(a_2) + \lambda_3 P_3(a_3)}{\lambda_1 P_1(\infty) + \lambda_2 P_2(\infty) + \lambda_3 P_3(\infty)} * 100 \right)$ – with increase in number of slots available. The insight here is that the marginal rate of successful resolution of bugs reduces with increasing capacity. Specifically, capacity addition beyond 220 slots yields negligible increases in rates of successful bug resolution.

----- Insert Table 5, Table 6, Figure 11, Figure 12 and Figure 13 about here -----

7. Concluding Remarks

Maintenance is one of the most important and costly phases of the software product life cycle. Therefore, increasing productivity in software maintenance is a priority for many companies. Despite considerable research in the area of software engineering, much of this literature is based on principles of software engineering and deals with the software development process. Further, research on software

¹⁰ The algorithm for choosing the optimal cut off combination and the fraction of bugs resolved for each meta-group that broadly follows Kaplan et al. (1992) is shown in Appendix 1.

maintenance productivity has focused on the individuals that perform the maintenance task and has investigated how individual productivity increases on account of learning (Boh et al. 2007; Narayanan et al. 2006). In this research, we presented a different perspective by motivating the idea that productivity can also be improved by developing a deeper understanding of the debugging process and exploiting the process attributes to effectively allocate and utilize scarce resources.

Towards this objective, first, we examined three plausible statistical models of the bug resolution process, and empirically demonstrated that a beta-geometric model of successful resolution probability best describes the bug resolution patterns in the data. Building on the insights gained from this model, we derived cut off policies which indicate the threshold time beyond which a maintenance engineer must stop working on a bug and move to new bug. We showed that such cut off policies can reduce overall bug queues and minimize resolution delays without significantly impacting the overall probability of successful bug resolution. From a service provider's perspective, this not only improves the utilization of resources, but also enhances customer satisfaction by reducing waiting times for bug resolution.

Second, we applied an approach that yields a heuristic estimate of engineering capacity required in debugging environments. The estimates provided by the model closely matched the raw data, lending credence to the adopted approach.

Third, when bugs come from multiple sources, we analyzed (a) how engineering resources must be allocated across these sources and (b) how cut off policies must be set across bugs from these sources towards maximizing the overall bug resolution rate. We further analyzed how capacity constraints impacted both the bug resolution rates and the bug queue lengths under these conditions.

We demonstrated a methodology that that can be applied to help managers in software maintenance operations make the appropriate tradeoffs between the choice of capacity levels, the allocation of that capacity, and the improvement of system performance (in terms of lowering waiting times and enhancing bug resolution rates). We showed that the temporal cut-off policies we derive for the resolution of bugs can efficiently discipline the bug resolution process by forcing engineers in a capacity-constrained situation to move on to more promising tasks at appropriate points of time. Importantly, we demonstrate that these policies, when appropriately designed, minimally impact rates of successful bug resolution and yet substantially reduce waiting times for incoming bugs in the system. Such approaches

can help the managers utilize their capacities better and improve customer service at the same time. This approach can also be applied to derive resource allocation policies in other resource-constrained service and operations environments – such as new product development – that are characterized by similar uncertainty about successful task completion and task completion time.

Our analysis has the following limitations. We demonstrate how the heterogeneity in the population of bugs can be exploited to gain operational advantages. Specifically, we demarcate the bug filer as a source of heterogeneity and assess the corresponding impact on resource allocation. However, we do not directly examine what factors are responsible for introducing such heterogeneity in the first place.¹¹ Explicitly incorporating the sources of heterogeneity can potentially yield even more fine-grained policies. Second, we empirically model the software maintenance operations within a single company. A replication of this methodology across multiple companies would provide additional evidence regarding its robustness and validity. Third, we do not explicitly consider learning that happens within the system. Future models can consider the effect of learning. Finally, one may imagine that one bug fix may cause an impact on the software and create more bugs. This may create dependencies of quality of bug fix on the arrival rates of new bugs. However modeling this phenomenon may reduce the analytical tractability of this approach. Despite these limitations, we believe that our research makes a significant contribution to the area of resource allocation in software maintenance. Further, the tight link between our findings and the reality on the ground lends support to the practical insights that emerge from our analysis. That said, much remains to be explored in the area, and we hope our work catalyzes further research on productivity enhancement in software maintenance operations.

References

Amoribieta, I., K. Bhaumik, K. Kanakamedala, A. D. Parkhe. 2001. Programmers abroad: A primer on offshore software development. *McKinsey Quarterly* 2

Banker, R. D., S. M. Datar, C. F. Kemerer. 1991. A model to evaluate variables impacting the productivity of software maintenance projects. *Management Science* 37(1) 1-18.

Banker, R., S. Slaughter. 1997. A field study of scale economies in software maintenance. *Management Science* 43(12) 1709-1725.

¹¹ For insights regarding the sources of such heterogeneity, see Kitchenham et al. (1999) and Kemerer and Slaughter (1997).

- Banker, R., D. Gordon, S. Slaughter. 1998. Software development practices, software complexity, and software maintenance performance. *Management Science* **44**(4) 433-451.
- Banker, R. D., S. M. Datar, C. F. Kemerer, D. Zweig. 2002. Software Errors and Software Maintenance Management. *Information Technology and Management* **3** 25-41.
- Boh, W. F., S. A. Slaughter and J. A. Espinosa (2007). Learning from Experience in Software Development: A Multilevel Analysis. *Management Science*, *53*(8), 1315-1331.
- Bondi, A. B., J. P. Buzen. 1984. The response times of priority classes under preemptive resume in M/G/m queues. *ACM SIGMETRICS Performance Evaluation Review* **12**(3) 195-201.
- Boxma, O. J., J. W. Cohen, N. Huffels. 1979. Approximations of the Mean Waiting Time in an M/G/s Queueing System. *Operations Research* **27**(6) 1115-1128.
- Hale, D. P., D. A. Haworth. 1991. Toward a model of programmers' cognitive processes in software maintenance: a structural learning theory approach for debugging. *Journal of Software Maintenance* **3**(2) 85-106.
- Hale, J. E., S. Sharpe, D. P. Hale. 1999. An Evaluation of the Cognitive Processes of Programmers Engaged in Software Debugging. *Journal of Software Maintenance: Research and Practice* **11** 73-91.
- Kaplan, E. H., A. Hershlag, A. H. Decherney, G. Lavy. 1992. To be or not to be? That is Conception! Managing In Vitro Fertilization Programs. *Management Science* **38**(9) 1217-1229.
- Kemerer., C. F., S. A. Slaughter. 1997. Determinants of Software Maintenance Profiles: An Empirical Investigation. *Software Maintenance: Research & Practice* **9** 235-251.
- Kitchenham, B. A., G. H. Travassos, A. v. Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang. 1999. Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice* **11** 365-389.
- Krishnan, M. S., T. Mukhopadhyay, C. H. Kriebel. 2004. A Decision Model for Software Maintenance. *Information Systems Research* **15**(4) 396-412.
- Lientz, B. P., E. B. Swanson, G. E. Tompkins. 1978. Characteristics of Application Software Maintenance. *Communications of ACM* **21**(6) 466-471.
- Liberman, H. 1997. The Debugging Scandal and What to do about it. *Communications of ACM* **40**(4) 27-29.
- Narayanan, S., S. Balasubramanian, J. M. Swaminathan. 2006. Individual Learning and Productivity in Software Maintenance Environment: An Empirical Analysis, *Working Paper*: Kenan-Flagler Business School, University of North Carolina at Chapel Hill.
- Nozaki, S. A., S. M. Ross. 1978. Approximations in Finite-Capacity Multi-Server Queues with Poisson Arrivals. *Journal of Applied Probability* **15**(4) 826-834.
- NIST. 2002. Software Errors Cost U.S. Economy \$59.5 Billion Annually, Vol. 2006. National Institute of Standards and Technology

Nosek, J. T., P. Palvia. 1990. Software maintenance management: changes in the last decade. *Journal of Software Maintenance: Research and Practice* **2**(3) 157 - 174.

Scacchi, W. 1994. Understanding Software Productivity. *Advances in Software Engineering and Knowledge Engineering*, **4** 37-70.

Segars, A. H., V. Grover. 1993. Re-Examining Perceived Ease of Use and Usefulness: A Confirmatory Factor Analysis. *MIS Quarterly* **17**(4) 517-525.

Suchindran, C. M., P. A. Lachenbruch. 1975. Estimates of Fecundability from a Truncated Distribution of Conception Times. *Demography* **12**(2) 291-301.

Sudhakar, G. P. 2002. Why do young engineers hop companies?, December 17 ed., Vol. 2006. The Hindu

Teasley, S. D., L. A. Covi, M. S. Krishnan, J. S. Olson. 2002. Rapid software development through team collocation. *IEEE Transactions on Software Engineering* **28**(7) 671 - 683.

van Pul, M. C. 1994. A general introduction to software reliability. *CWI Quarterly* **7**(3) 203-244

Weinberg, C. R., B. C. Gladen. 1986. The Beta-Geometric Distribution Applied to Comparative Fecundability Studies. *Biometrics* **42**(3) 547-560.

Whitt, W. 1983. The Queueing Network Analyzer. *Bell System Technical Journal* **62**(9) 2779-2815.

Tables, Graphs and Figures

Figure 1: Bug resolution process

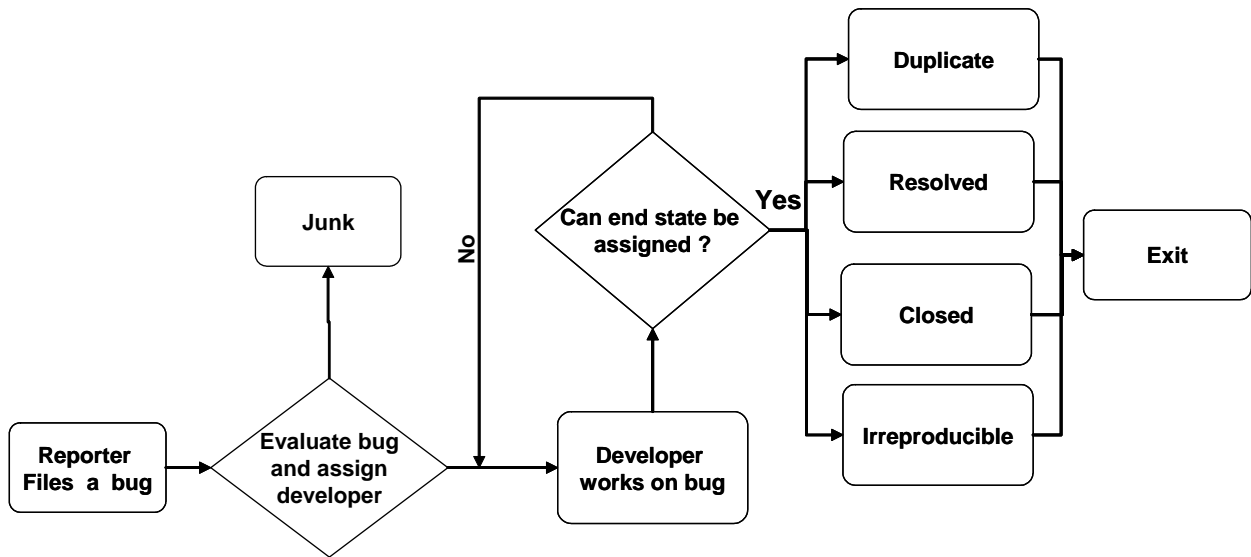


Figure 2: Proportion of bugs successfully resolved in any time period

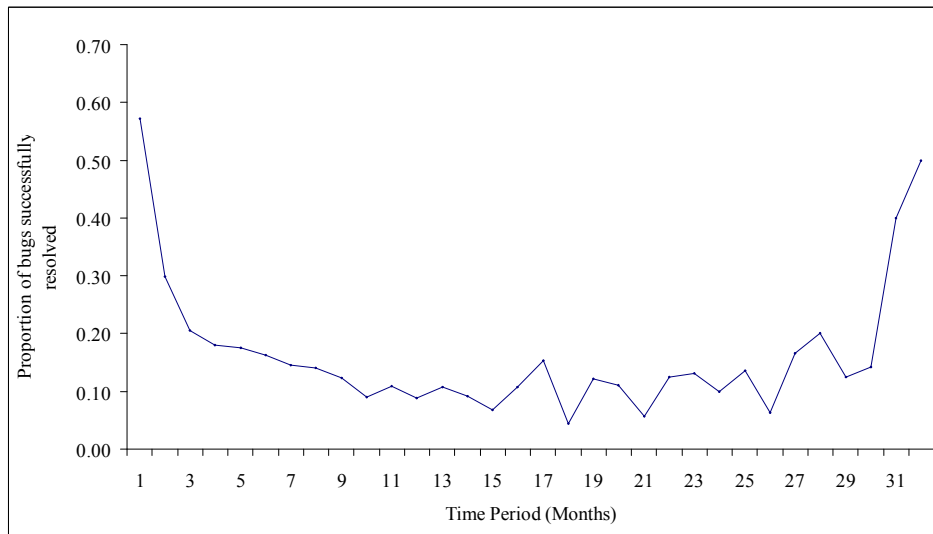


Figure 3: Actual versus expected outcomes for the Homogeneous model

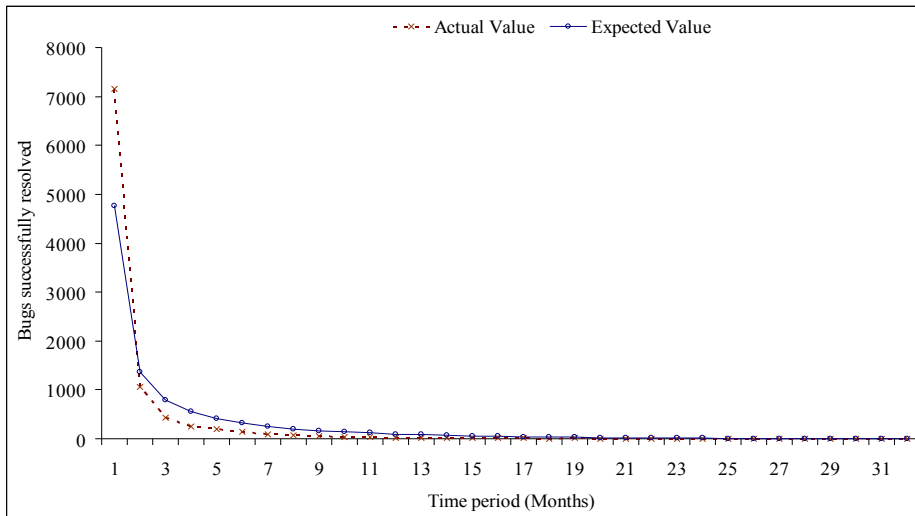


Figure 4: Actual versus Expected successes for the Beta-Geometric model

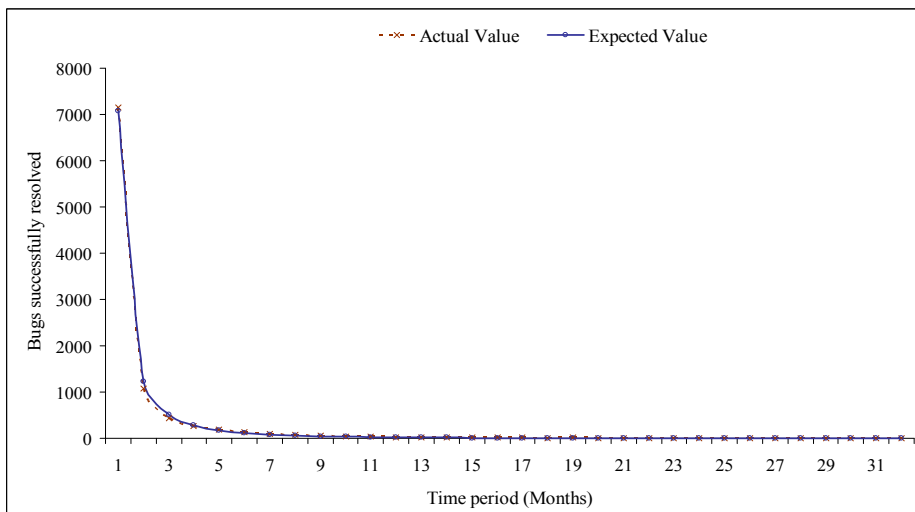


Figure 5: Actual versus Expected successes for the Split Population Geometric model

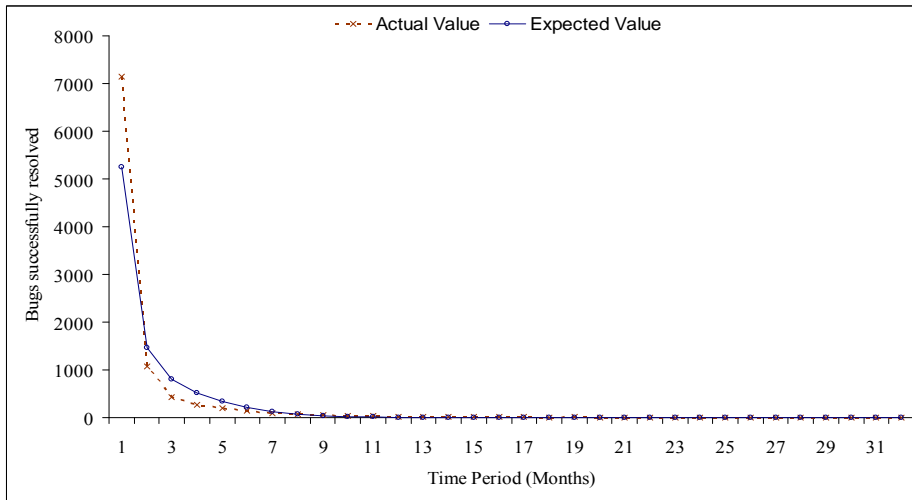


Figure 6: Actual versus Expected successes for the Trinomial model

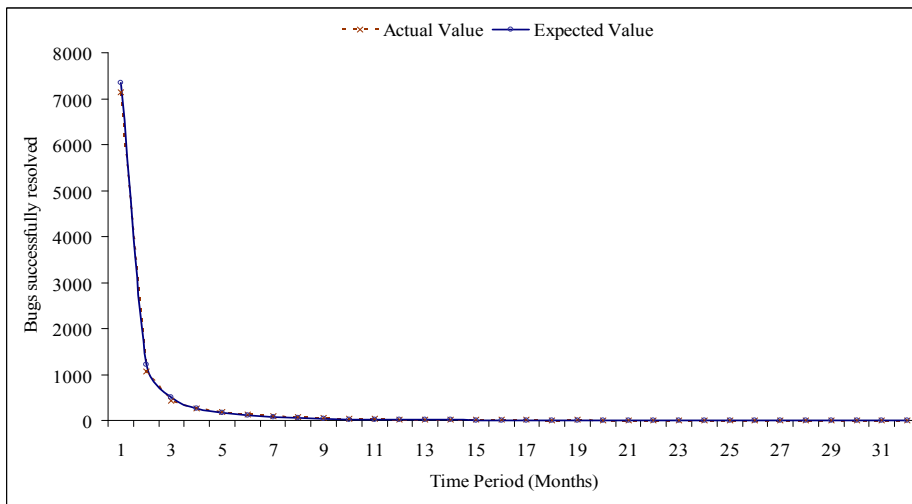


Figure 7: The queuing process

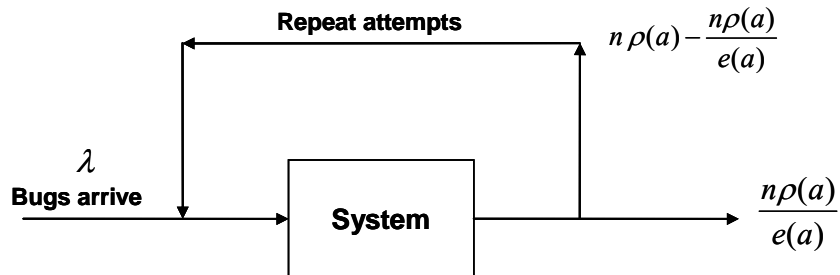


Figure 8: Expected resolution time

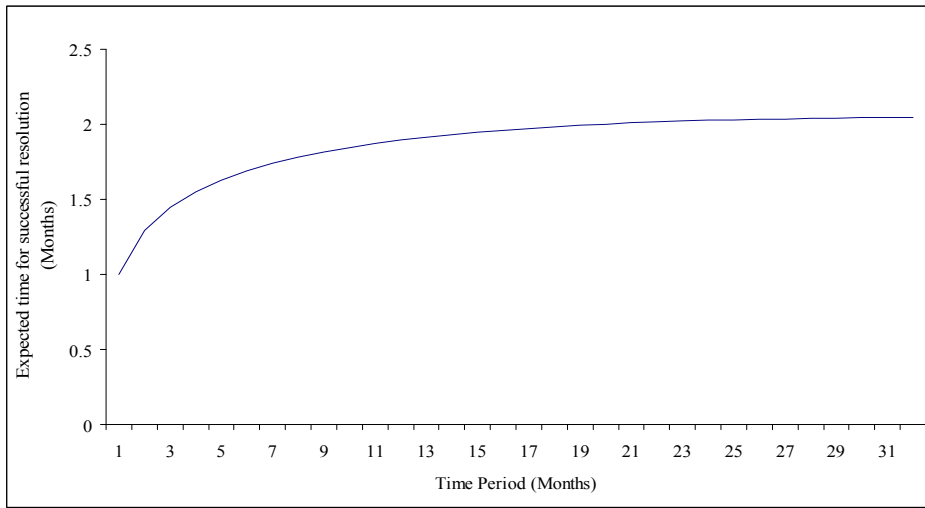


Figure 9: Expected probability of successful resolution

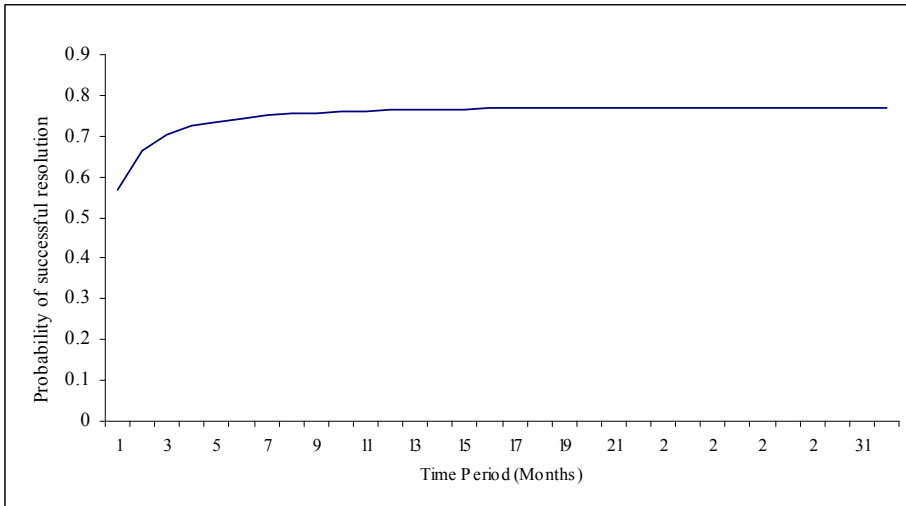


Figure 10: Sensitivity analysis of the waiting times to increase in slot size

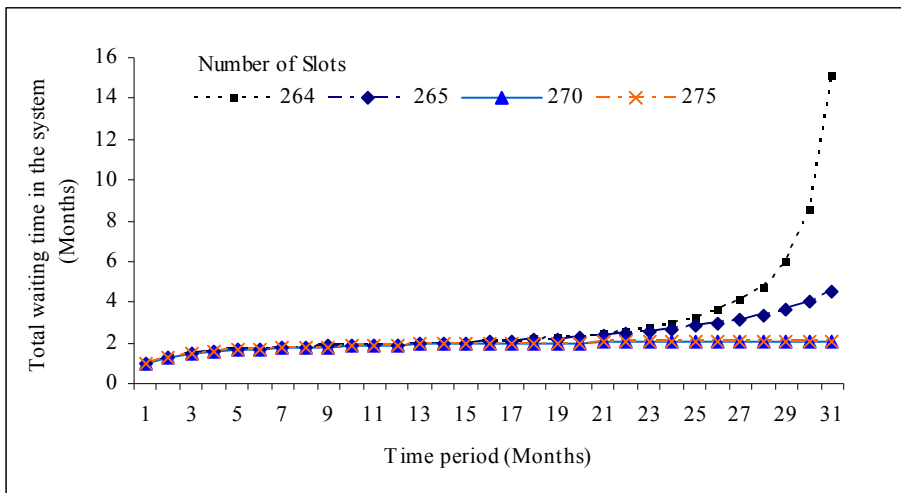


Figure 11: Cut off combinations for different groups given slot size – Marginal probability rule

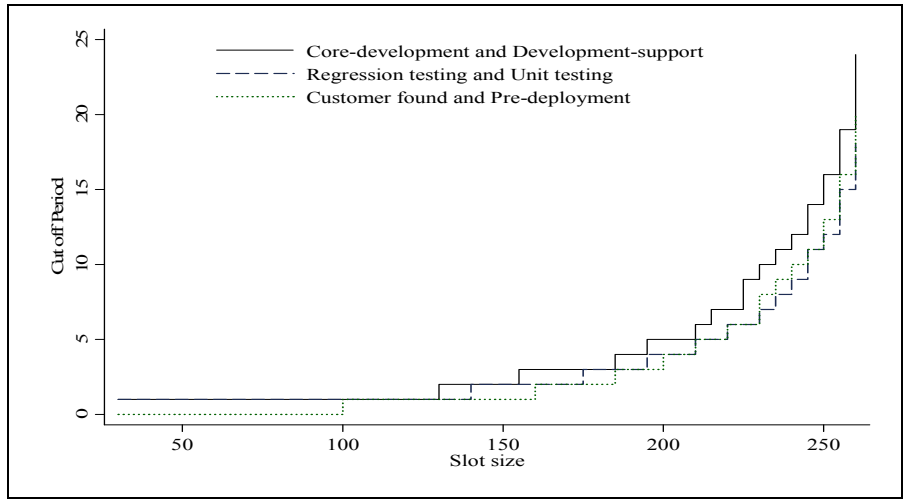


Figure 12: Optimal cut off combinations for different groups given slot size – Equality rule

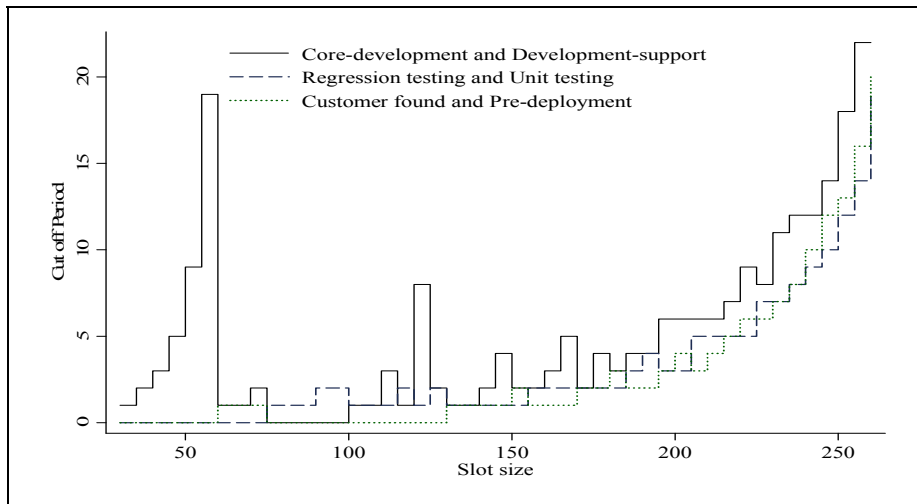


Figure 13: Relative resolution rates for the three groups based on slot size

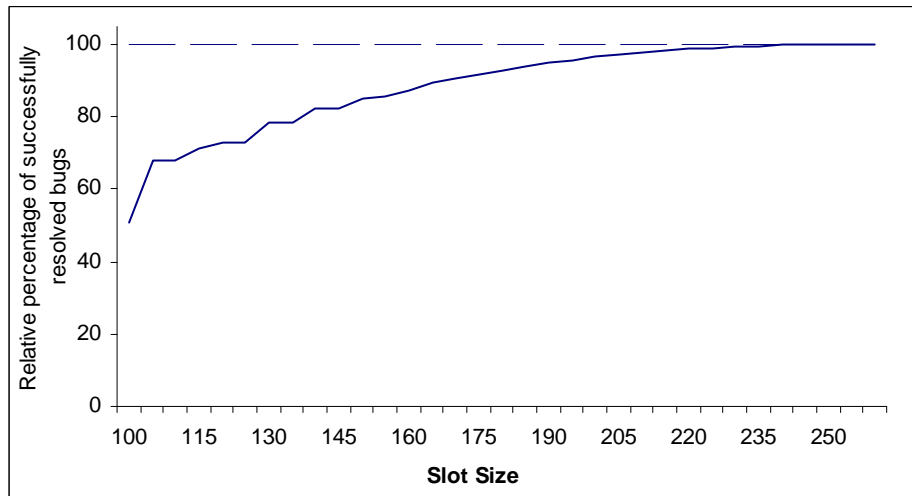


Table 1: Successful resolutions in each time period

Time period (Months)	Successful Resolutions	Total	Closed/ Irreproducible	Ratio Successfully Resolved
1	7147	12503	1774	0.572
2	1067	3582	431	0.298
3	428	2084	215	0.205
4	260	1441	103	0.180
5	189	1078	51	0.175
6	136	838	40	0.162
7	96	662	42	0.145
8	74	524	26	0.141
9	52	424	15	0.123
10	32	357	14	0.09
11	34	311	27	0.109
12	22	250	13	0.088
13	23	215	5	0.107
14	17	187	8	0.091
15	11	162	12	0.068
16	15	139	7	0.108
17	18	117	7	0.154
18	4	92	6	0.043
19	10	82	9	0.122
20	7	63	3	0.111
21	3	53	2	0.057
22	6	48	4	0.125
23	5	38	3	0.132
24	3	30	5	0.100
25	3	22	3	0.136
26	1	16	3	0.063
27	2	12	0	0.167
28	2	10	0	0.200
29	1	8	0	0.125
30	1	7	1	0.143
31	2	5	1	0.400
32	1	2	1	0.500
Total	9672	25362		0.381

Model	Chi-Square	LL	Parameter	Estimates	SE
Homogeneous	2517.17	-16858.75	p	0.381	0.003
Beta-geometric	223.96	-14777.40	α	1.025	0.024
			β	0.543	0.023
Split Population Geometric	4,976,65	-16308.15	θ	0.963	0.003
			p	0.436	0.002
Trinomial	258.38	-21658.10	α	0.904	0.026
			β	0.426	0.018
			q	0.11	0.001

Main Group	Sub Group	Sample Size	α	β	Chi-Square (DF)	Log-Likelihood	% Successfully Resolved
Grouping by the Filer of the Bug	Common Group	12503	1.025	0.543	223.96 (31)	-14777.40	77.30%
	Standard Error		0.024	0.023			
	Core-development	2245	1.031	0.543	73.58 (31)	-2751.54	91.63%
	Standard Error		0.066	0.046			
	Predicted Probability	0.911					
	Regression Testing	1818	0.836	0.616	40.23 (22)	-1860.52	72.28%
	Standard Error		0.084	0.073			
	Predicted Probability	0.720					
	Customer Found	2119	0.871	1.198	75.69 (31)	-2964.77	69.80%
	Standard Error		0.067	0.120			
	Predicted Probability	0.694					
	Unit Testing	5023	0.795	0.575	115.71 (31)	-5445.33	74.74%
	Standard Error		0.043	0.038			
	Predicted Probability	0.744					
Development-support	729	1.207	0.663	40.80 (20)	-864.035	87.40%	
Standard Error		0.154	0.107				
Predicted Probability	0.870						
Pre-deployment	512	0.970	1.087	27.99 (31)	-698.409	74.41%	
Standard Error		0.369	0.145				
Predicted Probability	0.740						

Table 4: Tradeoff between mean time a bug spends in the system and the probability of a successful resolution

Cut off time period (a)	Resolution Probability	W_{NP} (264 slots)	W_{NP} (265 slots)	W_{PP} (264 slots)	W_{PP} (265 slots)
1	0.566	1.00	1.00	1.00	1.00
2	0.665	1.29	1.29	1.29	1.29
3	0.704	1.45	1.45	1.44	1.44
4	0.724	1.55	1.55	1.55	1.55
5	0.736	1.63	1.63	1.63	1.63
6	0.744	1.69	1.69	1.70	1.70
7	0.750	1.74	1.74	1.78	1.77
8	0.754	1.78	1.78	1.90	1.90
9	0.757	1.82	1.82	2.13	2.12
10	0.760	1.85	1.85	2.54	2.52
11	0.762	1.88	1.88	3.24	3.18
12	0.763	1.91	1.91	4.28	4.17
13	0.764	1.94	1.94	5.81	5.60
14	0.765	1.97	1.97	8.06	7.68
15	0.766	2.01	2.00	11.25	10.59
16	0.767	2.05	2.03	15.55	14.45
17	0.768	2.09	2.07	21.37	19.57
18	0.768	2.15	2.12	29.16	26.24
19	0.769	2.22	2.18	39.53	34.87
20	0.769	2.30	2.24	52.40	45.21
21	0.769	2.41	2.31	69.63	58.45
22	0.769	2.54	2.41	93.59	75.87
23	0.770	2.72	2.52	125.70	97.62
24	0.770	2.97	2.66	169.92	124.94
25	0.770	3.26	2.81	224.68	155.25
26	0.770	3.63	2.97	294.42	189.31
27	0.770	4.08	3.14	377.95	224.91
28	0.770	4.76	3.36	507.44	271.47
29	0.770	5.95	3.64	733.76	335.51
30	0.770	8.52	4.02	1226.60	432.68
31	0.770	15.12	4.47	2485.65	658.29
32	0.770	37.66	4.87		

Table 5: Maximum Likelihood estimates and related data for the meta-groups

Meta-group	Item	Estimate	Log-Likelihood	Chi-Square (DF)
Core-development and Development-support	Arrival Rate	27.12/Month		
	α (S.E)	1.062 (0.060)		
	β (S.E)	0.565 (0.042)		
	Predicted expected time to successful resolution	2.08 Months	-3636.525	101.68 (31)
	Actual expected time to successful resolution	2.074 Months		
	Predicted probability of successful resolution	0.900		
	Actual fraction successfully resolved	0.905		
Regression testing and Unit testing	Arrival Rate	70.52/Month		
	α (S.E)	0.804 (0.038)		
	β (S.E)	0.585 (0.033)		
	Predicted expected time to successful resolution	1.805 Months	-7305.968	143.88 (31)
	Actual expected time to successful resolution	1.783 Months		
	Predicted probability of successful resolution	0.738		
	Actual fraction successfully resolved	0.740		
Customer found and Pre-deployment	Arrival Rate	31.24/Month		
	α (S.E)	0.887 (0.061)		
	β (S.E)	1.172 (0.105)		
	Predicted expected time to successful resolution	2.62 Months	-3665.813	81.30 (31)
	Actual expected time to successful resolution	2.61 Months		
	Predicted probability of successful resolution	0.703		
	Actual fraction successfully resolved	0.706		

Table 6: Integer cutoffs and fraction considered for resolution based on marginal probability rule

Slots	Maximum Integer Cut offs			Fraction of Bugs considered for resolution in the “Maximum Integer Cut off period”			Resolution Rate - Marginal Probability rule	Resolution Rate - Equality rule
	Development	Testing	Customer	Development	Testing	Customer		
30	1.00	1.00	0.00	1.00	0.04	0.00	19.37	17.71
35	1.00	1.00	0.00	1.00	0.11	0.00	22.27	20.88
40	1.00	1.00	0.00	1.00	0.18	0.00	25.16	22.17
45	1.00	1.00	0.00	1.00	0.25	0.00	28.06	23.25
50	1.00	1.00	0.00	1.00	0.32	0.00	30.95	23.96
55	1.00	1.00	0.00	1.00	0.40	0.00	33.85	24.37
60	1.00	1.00	0.00	1.00	0.47	0.00	36.74	31.17
65	1.00	1.00	0.00	1.00	0.54	0.00	39.64	31.17
70	1.00	1.00	0.00	1.00	0.61	0.00	42.53	34.35
75	1.00	1.00	0.00	1.00	0.68	0.00	45.43	40.83
80	1.00	1.00	0.00	1.00	0.75	0.00	48.32	40.83
85	1.00	1.00	0.00	1.00	0.82	0.00	51.22	40.83
90	1.00	1.00	0.00	1.00	0.89	0.00	54.11	46.62
95	1.00	1.00	0.00	1.00	0.96	0.00	57.01	46.62
100	1.00	1.00	1.00	1.00	1.00	0.08	59.55	58.54
105	1.00	1.00	1.00	1.00	1.00	0.23	61.71	58.54
110	1.00	1.00	1.00	1.00	1.00	0.39	63.86	63.00
115	1.00	1.00	1.00	1.00	1.00	0.55	66.02	64.33
120	1.00	1.00	1.00	1.00	1.00	0.71	68.17	64.69
125	1.00	1.00	1.00	1.00	1.00	0.87	70.32	67.50
130	2.00	1.00	1.00	0.12	1.00	1.00	72.45	72.00
135	2.00	1.00	1.00	0.35	1.00	1.00	72.45	72.00
140	2.00	2.00	1.00	1.00	0.18	1.00	76.29	75.18
145	2.00	2.00	1.00	1.00	0.47	1.00	77.97	77.13
150	2.00	2.00	1.00	1.00	0.76	1.00	79.66	78.93
155	3.00	2.00	1.00	0.21	1.00	1.00	81.30	80.97
160	3.00	2.00	2.00	1.00	1.00	0.12	82.76	82.25
165	3.00	2.00	2.00	1.00	1.00	0.50	84.21	83.33
170	3.00	2.00	2.00	1.00	1.00	0.89	85.66	84.71
175	3.00	3.00	2.00	1.00	0.39	1.00	86.93	86.67
180	3.00	3.00	2.00	1.00	0.95	1.00	88.11	87.64
185	4.00	3.00	3.00	1.00	1.00	0.13	89.25	88.79
190	4.00	3.00	3.00	1.00	1.00	0.88	90.34	89.85
195	5.00	4.00	3.00	1.00	0.34	1.00	91.30	91.12
200	5.00	4.00	4.00	1.00	1.00	0.16	92.21	92.00
205	5.00	4.00	4.00	1.00	1.00	0.82	93.07	92.80
210	6.00	5.00	5.00	1.00	1.00	0.05	93.82	93.69
215	7.00	5.01	5.00	1.00	0.01	1.00	94.54	94.43
220	7.00	6.00	6.00	1.00	1.00	0.57	95.17	95.05
225	9.00	6.00	6.00	1.00	1.00	0.64	95.76	95.65
230	10.00	7.00	8.00	1.00	1.00	0.28	96.29	96.19
235	11.00	8.00	9.00	1.00	1.00	0.34	96.76	96.65
240	12.00	9.00	10.00	1.00	1.00	0.90	97.19	97.08
245	14.00	11.00	11.00	1.00	0.63	1.00	97.56	97.47
250	16.00	12.00	13.00	1.00	1.00	1.00	97.89	97.79
255	19.00	15.00	16.00	1.00	1.00	0.36	98.17	98.07
260	24.00	18.00	20.00	1.00	1.00	0.27	98.41	98.31

Appendix 1: Algorithm to evaluate optimal cut-off combinations for three groups

The key idea underlying the algorithm, which broadly follows Kaplan et al. (1992), is that, for any given capacity, we enumerate all combinations of cut off time periods (a_i) for each meta-group which can be accommodated subject to the capacity constraint. The algorithm ultimately yields the combination of a_i which maximizes the rate of successful resolution (RR).

Initialize: Set LOWERBOUND $RR = 0$, $a_i^*(n) = 0$, $a_j^*(n) = 0$, $a_k^*(n) = 0 \forall i, j, k=1, 2, 3$

Set UPPERBOUND $a_i^{\max}(n)$, $a_j^{\max}(n)$, $a_k^{\max}(n) \forall i, j, k=1, 2, 3$

OUTER LOOP: FOR $i = 1, 2, 3$ $a_i(n) = 1, 2, 3, 4, \dots a_i^{\max}(n)$

MIDDLE LOOP: FOR $j = 1, 2, 3$ and $j \neq i$, $a_j(n) = 1, 2, 3, 4, \dots a_j^{\max}(n)$

INNER LOOP: FOR $k = 1, 2, 3$, $k \neq i$, $k \neq j$, $a_k(n) = 1, 2, 3, 4, \dots a_k^{\max}(n)$

If $\lambda_i e_i(a_i(n)) + \lambda_j e_j(a_j(n)) + \lambda_k e_k(a_k(n)) < n$

if $\lambda_i P_i(a_i) + \lambda_j P_j(a_j) + \lambda_k P_k(a_k) > RR$ then

Set $a_i^*(n) = a_i(n)$, $a_j^*(n) = a_j(n)$, $a_k^*(n) = a_k(n)$

Set $RR = \lambda_i P_i(a_i) + \lambda_j P_j(a_j) + \lambda_k P_k(a_k)$

Filename: 12-02-2007-Resource Allocation in software Maintenance-
Final.doc
Directory: C:\Documents and Settings\hbsuser\My Documents
Template: C:\Documents and Settings\hbsuser\Application
Data\Microsoft\Templates\Normal.dotm
Title: Optimal Resource Allocation in Software Maintenance
Subject:
Author: sriku
Keywords:
Comments:
Creation Date: 12/19/2007 11:47:00 AM
Change Number: 2
Last Saved On: 12/19/2007 11:47:00 AM
Last Saved By: swaminaj
Total Editing Time: 15 Minutes
Last Printed On: 3/12/2008 9:04:00 AM
As of Last Complete Printing
Number of Pages: 34
Number of Words: 9,111 (approx.)
Number of Characters: 51,939 (approx.)