Abstract

This paper explores a novel application of queuing theory to the corrective software maintenance problem to support quantitative balancing between resources and responsiveness. Initially, we provide a detailed description of the states a defect traverses from find to fix and a definition and justification of mean time to resolution as a useful process metric. We consider the effect of queuing system structures, priority levels and priority disciplines on the differential mean times to resolution of defects of different severities. We find that modeling the defect resolution capacity of a software engineering group as n identical M/M/1 servers provides a flexible and realistic approximation to the queuing behavior of four different organizations. We consider three queuing disciplines. Though purely preemptive and non-preemptive priority disciplines may be suited for other groups, our data was best fit by a mixed discipline, one in which only the most severe defects preempt ongoing service activities of lesser severities. We provide two examples of the utility of such a model: Given the reasonable assumption that the most severe defects have the highest impact on reliability, we find that the reduction of the resolution time for these defects must come from changes reducing the service time. On the other hand the effect of additional engineering resources on the resolution time of less severe defects is easily computed and can be significant.

1 Introduction and motivation

Corrective maintenance, which consists of fixing defects after a product is released into the field is significantly more expensive than fixing the defects prior to release [10]. These maintenance activities are vital to ensuring customer satisfaction and normally incur a sizeable portion of the costs associated with a software product. Typically, a defect reported by a customer is associated with a severity level based on its impact on the services provided by the application. The higher the severity level of a defect, the higher is its impact and the quicker the expected resolution time. To guide the cost-effective allocation of resources, it is helpful to understand and capture the key aspects of the defect resolution process into a mathematical model. The very creation and fitting of a model requires a careful and beneficial review of the existing processes. Once derived, the model can be used to estimate how resolution times of defects of different severities vary as a function of the allocated resources and the arrival rates. In particular one can determine which strategies are most important to improve reliability or the level of staffing needed to achieve a target resolution time.

In this paper we first describe a typical life cycle of a customer-reported defect. Based on this life cycle, we define a new metric, namely, Mean Time to Resolution (MTTR) which characterizes the defect resolution process. Typically, a group in a software development organization consists of a pool of engineers, where each engineer is responsible for resolving the defects assigned to him/her. Further, each engineer handles the higher severity defects with higher priority than the lower severity defects. We first hypothesize that the defect resolution process of each (full time equivalent) engineer can be modeled using one of the two multi-priority queuing models: head-of-the-line, non-preemptive priority queue and preemptive priority queue. These two models are then used to analyze the customer defect data obtained from the defect tracking system at Cisco Systems. The results of the analysis indicate that: (i) for the most severe defects the observed mean resolution time is very close to the mean resolution time of the preemptive queuing model, and (ii) for the defects of lower severities, the observed mean resolution times are closer to the mean resolution times of the non-preemptive queuing model.

The following inferences can be drawn from these results for the data under consideration. First, the defect resolution process is neither completely preemptive nor completely non-preemptive. Rather, it follows a combination of the preemptive and non-preemptive schemes. Second, the most severe defects preempt the service of all the lower severity defects. Based on these observations, we reject our initial
hypothesis that the defect resolution process follows either the non-preemptive or preemptive multi-priority model and consider an alternate hypothesis, a mixture of preemptive and non-preemptive multi-priority queues. Expressions for mean times to resolution for the mixed model are derived to enable its application to the customer defect data. Comparison of the mean resolution times of the mixed model with the observed mean resolution times validates the hypothesis that the defect resolution process indeed follows the mixed model. For the defect data considered, the mixed queuing model suggests that the mean resolution time of the most severe defects can be improved by reducing the service time. Further, the mean resolution times of lower severity defects and hence the overall mean resolution time can be improved by increasing the level of staffing.

The layout of the paper is as follows: Section 2 summarizes the related research. Section 3 describes the life cycle of a customer reported defect. Section 4 discusses the queuing models for the defect resolution process. Section 5 illustrates how the models can be used to analyze the actual customer defect data and discusses the results of the analysis. Section 6 offers concluding remarks and directions for future research.

2 Related research

In this section we summarize the related research, which provides motivation for the work reported in this paper.

Incorporating the impact of repair process to provide realistic estimates of software reliability has been the focus of many research efforts. Stutzke et al. [13] incorporate repair time into a software reliability model that considers human errors. Levendel [7] and Kremer [6] develop a birth-death model which takes into consideration repair time. Dalal [2] assume that the software debugging follows a constant repair rate and incorporate repair into an exponential order statistics software reliability model. Schneidewind [12] incorporate a constant repair rate into the Schneidewind software reliability model. Gokhale et al. [3] incorporate explicit repair into SRGMs using a numerical solution. Jones et al. [4] consider imperfect repair in the context of infinite failures models using simulation. However, their objective is to examine the appropriateness of the infinite failures models. Imperfect repair has also been considered by other researchers [3, 6, 7]. The primary motivation of most of the above efforts is to relax the assumption of instantaneous and perfect repair underlying the software reliability models. They augment the software reliability models, require failure and repair data for their application and do not capture the factors of the repair/resolution process including classification of defects based on severities, queuing, and distribution of defects among the available engineers.

A very few efforts have used concepts from queuing theory to model maintenance activities. Antoniol et al. [1] use queuing concepts to model massive maintenance projects in a virtual software factory. Their research seeks to evaluate alternative configurations of a maintenance center. Di Penta et al. [9] model web-centric service centers using a $M/M/1$ queue. Ramaswamy et al. [11] use a $M/M/n$ queue to characterize business-critical maintenance projects to determine the appropriate level of staffing. A major drawback of these efforts is that they ignore defect severities. Further, they assume that the engineers in a pool are interchangeable. In most software organizations, defects are prioritized according to their severities, to determine the order in which they are handled. Also, all the engineers are not interchangeable, in fact, each engineer has unique expertise, skills and experiences. Our approach described in this paper considers the impact of these factors on the mean resolution times of software defects.

3 Defect life cycle

In this section we describe the life cycle of a typical defect, shown in Figure 1, as it makes its way through the defect tracking system. A defect in the tracking system originates in the $Start$ state, where a customer service representative is writing/recording the defect into the tracking system. A defect resides in the $Start$ state only for a few seconds. Once the defect is written completely, it transitions to the $New$ state. In the $New$ state, attributes such as the component/feature the defect is associated with and the manager responsible for the defect are identified.

There are several paths a defect can follow from the $Start$ state. We first describe the path which leads to its resolution by actually fixing it. On this path, the defect transitions from $New$ to $Assigned$ state, where the defect has been assigned to an engineer by the manager. From the $Assigned$ state, the most typical or straightforward transition is to the $Open$ state, where the engineer acknowledges the defect assignment. The defect transitions to the $Fixed$ state when the engineer fixes the underlying error. Ultimately, the defect transitions to the $Verify$ state, after the engineer has verified the correctness of the fix.

We now describe deviations from the typical path described above and the reasons that cause these deviations. From the $New$ state, the defect may transition to the $Wait$ state, if the manager determines that no decision about who the defect should be assigned to can be reached based on the available information. This may arise due to lack of adequate information about the defect or the unavailability of engineer(s) with the required expertise. In the $Wait$ state, once adequate information is available to assign the defect, it transitions to the $Assigned$ state. A transition from $New$ to $Junk$ occurs if the manager determines that the defect is spurious, from $New$ to $Duplicate$ occurs if the defect is a
duplicate of another defect that already exists in the system, and from New to Unreproducible occurs if after many attempts to reproduce the problem the manager believes that it is impossible to do so. The defect may also transition to Junk, Duplicate and Unreproducible states from the Assigned state.

In the Open state, if the responsible engineer determines that the defect is incorrectly assigned, he/she may send the defect back to the manager by changing its status to Assigned. The engineer may also determine that fixing the defect is not the best possible option. This situation may arise for example, if the product is obsolete or will be retired in the near future. The engineer changes the status of the defect to Assigned in this case as well, and subsequently, the manager changes the status to Closed. For some defects, the engineer may also determine that the association of the defect with a particular feature/component was inappropriate, and hence changes the status of the defect to New, which involves associating it with a new feature/component and possibly a new manager. From the Open state, the defect may transition to the Info or Hold states. The transition to the Info state occurs if additional information is needed from the internal customer service representative or the test engineer who wrote the defect. In the Hold state, the defect is held waiting for information from an external vendor. Upon or during the process of fixing, the engineer may discover that the defect impacts several product releases which require separate fixes, due to which it transitions from Open to More. A defect may transition from Fixed to More for the same reason. After possibly multiple More – Fixed transitions, the defect finally transitions to the Verify state.

Informally, our goal is to measure and model the time taken by a group to process a defect to the first point where it is considered “resolved”, that is, either it is fixed or it is decided that a fix is unlikely. The resolved states include More, Fixed and Verify as well as Junk, Unreproducible, Duplicate and Closed. The Fixed state is included in the resolved states, because although a standard practice is to verify fixes, this transition may not be recorded by all the groups within an organization. In actual practice it is important that the remainder of the states are the ones for which an organization clearly accepts ownership and are on main path to resolution; these states include Start, New Assigned, Open and Wait. The states Info and Hold are debatable since exiting them is out of the engineers’ hands. Since a very small percentage of defects end up in states Info and Hold, we also consider a defect as resolved in these states, in order to define a metric that is solely under the control of the development organization.

Thus, the set of states in Figure 1 can be divided into two subsets, namely, Resolved and Outstanding. The Resolved subset consists of Junk, Unreproducible, Duplicate, Closed, Info, Hold, Verify and Fixed, and these states are marked with a solid/darker line in Figure 1. The states belonging to the other subset, namely, Outstanding include Start, New, Assigned, Open and Wait and these states are encompassed into a single oval in the figure. The time spent by a defect in these Outstanding states is considered as the time in the system and includes
both waiting and service time. We note that the Resolved states, as defined, are not terminal in a Markovian sense. There are rare transitions (not shown) from any of them back to the Assigned state on the basis of new information. Transitions among the Resolved states, such as between More, Fixed, and Verify are not uncommon, but do not affect the metric at all.

We define a metric Mean Time to Resolution (MTTR) as the ratio of the number of defects being processed or in any one of the Outstanding states by the average rate at which defects transition to the set of Resolved states. Informally, if in a certain group, on an average there are ten defects that are not in any of the Resolved states and on an average two defects transition to any of the Resolved states per week, then the above definition would produce an estimate of five weeks as the average MTTR for defects for that group. In practice, both the queue length and the resolution rate over thirteen weeks are averaged.

The MTTR metric as defined above has several desirable properties. Within an organization, it can compare groups of different sizes since it normalizes queue length by traffic to determine the delay in units of time. Senior management can easily relate to this metric since accounting calculations like days-accounts-receivable or days-inventory are also computed on a quantity divided by flow basis. Finally, the definition matches Little’s law [14] which holds very generally and is given by Equation (1). Cisco has been using the MTTR metric to goal and monitor the defect resolution process for over five years.

$$\text{Mean time in system} = \frac{\text{Ave number in system}}{\text{Ave arrival rate}} \quad (1)$$

We note that the definition of MTTR does not include the time to either write a defect or to distribute a fix. It is also worth mentioning that many defect encounters are re-occurrences [8] and the fix may be already available for download. Also, the effort to transition to states such as Junk, Unreproducible, Duplicate and Closed has been observed to be non-trivial because these are difficult defects, due to which these states are included in the MTTR definition.

4 Queuing models

In this section we first discuss the characteristics of the defect resolution process, followed by the rationale for our choice of the queuing models. Finally, we present the mathematical formulation of the two queuing models.

There are two aspects of the defect resolution process which influence the mean time to resolution of the defects. The first factor is concerned with the distribution of the defects to the engineers, which determines the queue structure. The second factor is concerned with the order in which the defects are handled, which determines the queue discipline. These two aspects are discussed next.

4.1 Queue structure

A group in a software organization typically consists of a pool of engineers responsible for resolving customer defects. One possible approach is to regard the entire group as a “black-box” and model it using a single server queue, for example, a $M/M/1$ system. The service rate of the single server will be given by the sum of the service rates of the individual engineers. In the single-server model, the average time spent in the system by higher severity defects would be unrealistically short, even for the high server utilization needed to match the average mean time in the system across defects of all severities. Another approach would be to model the group using a single multi-server queuing station, with the number of servers equal to the number of engineers, for example, a $M/M/n$ system. These multiple servers are assumed interchangeable and they work off of a common queue. This assumption is easily and commonly violated in practice. In fact, each engineer is unique in terms of the skills, involvement and familiarity with the features/components, the level of experience and the expertise. When the reported defects are distributed among the engineers, these factors must be considered.

These characteristics can be captured by modeling each engineer as a separate single server queue. We propose a structure of independent queues of similar capacity and loading, for example, $n$ distinct $M/M/1$ queues. It is clearly more realistic than either a queue with a single server or a single queue for multiple servers. Although it may seem improbable that the loading on the engineers is relatively even, it is in fact the role of the management to manage the process to that end. We are not assuming that real-world engineers are of identical capacity, rather we are hypothesizing that a model with $n$ “full time equivalent” maintenance engineers is sufficient to model significant aspects of real-world queuing behavior.

4.2 Queue discipline

The second aspect is concerned with the order in which the defects assigned to a single engineer are resolved, which determines the queuing discipline of each engineer. In general, to maximize customer satisfaction, the higher the severity level of a defect, the higher is the priority with which it should be resolved by the engineer to which it is assigned. This will ensure that the time taken to resolve a defect will be consistent with its priority, with the average time taken to resolve higher severity defects will be lower than the average time taken to resolve lower severity defects. Thus, the first-come-first-serve (FCFS) service dis-
cipline, which would essentially handle the defects in the order in which they were assigned is inappropriate to model this process. As a result, we consider queuing models which classify requests into classes, define priorities among the classes and then handle the requests according to these priorities. In general, two types of prioritized handling of requests is possible, namely, preemptive and non-preemptive and one model of each type is considered. A brief overview of each of these models along with the rationale for using it to model the defect resolution process is described below.

4.2.1 Head-of-the-line, non-preemptive priority queue

In head-of-the-line priority queuing or non-preemptive priority queuing, requests queue according to priority groups and are strictly separated according to their group membership. The highest priority group is labeled 1. An arrival from group \( j \) joins the "torso" of the queue behind all the requests of higher priorities from groups 1 through \( j \), and in front of all the requests from priority groups \( j + 1 \) and lower. Thus in this queuing model, the server picks a request from group \( j + 1 \), only if there are no outstanding requests from groups 1 through \( j \).

An engineer who follows head-of-the-line priority queuing begins fixing a defect of severity \( j \) only if there are no outstanding defects with severities 1 through \( j - 1 \). Even if a higher severity defect (1 through \( j - 1 \)) is reported when a severity \( j \) defect is being fixed, the engineer undertakes the new defect of higher severity only after the current defect is completely resolved.

4.2.2 Preemptive priority queue

In preemptive priority queuing, if a new request is of higher priority than the request that is currently being serviced, the current service request is preempted. Thus, a new request belonging to classes 1 through \( j - 1 \) preempts the ongoing service of a request belonging to category \( j \).

If a preemptive priority queuing discipline is employed by an engineer, then the engineer preempts the resolution of a defect of severity \( j \) if a defect of higher severity (1 through \( j - 1 \)) is reported and undertakes the newly reported defect immediately. We consider work conserving discipline, that is, the engineer can leverage the time and effort already expended in resolving the current defect before it is preempted, when the engineer resumes its resolution.

4.3 Mathematical formulation

In this section we present the mathematical formulation and expressions for the mean resolution times of the two queuing models.

We assume that the customer-reported defects are classified into \( m \) categories, according to their severities, which capture their impact on the services provided by the system. The defects belonging to category 1 are of highest severity, while the defects in category \( m \) are of the lowest severity. A new defect is classified at severity level \( j \) with probability \( p_j \). The defects are fixed by a pool of \( n \) engineers in a group. Based on our experience, we assume that the customers report defects according to a Poisson process with rate \( \lambda \). Further, the service time of a defect for each engineer is exponentially distributed with the parameter \( \mu \), irrespective of the defect severity. As discussed above, a management policy ensures that these defects are evenly distributed to the \( n \) engineers in the pool. Thus, the rate at which defects are assigned to each one of the \( n \) engineers is given by \( \lambda/n \). The defect distribution and queuing for the pool of engineers can thus be modeled as \( n \) independent \( M/M/1 \), multi-priority queues and is shown in Figure 2.

![Figure 2. Defect distribution and queuing](image)

The rate at which defects of severity \( j \) are assigned to each engineer, denoted \( \lambda_j \) is then given by:

\[
\lambda_j = \frac{\lambda p_j}{n}, j = 1, \ldots, m
\]  

(2)

The utilization of each engineer for severity \( j \) defects, denoted \( \rho_j \) is given by Equation (3), while the overall utilization (across all defects) is given by the sum of the \( \rho_j \)'s.

\[
\rho_j = \frac{\lambda p_j}{n \mu}
\]  

(3)

In the subsequent subsections, we present expressions for the mean resolution times for the two queuing models discussed in Section 4.2. We offer an intuitive explanation for the expressions to provide a feel for the processes and tradeoffs involved. The detailed derivations can be obtained from Kleinrock [5]. In the subsequent subsections, the superscripts \( np \) and \( p \) denote non-preemptive and preemptive queuing models respectively.

4.3.1 Head-of-the-line, non-preemptive priority queue

We let \( t^{np} \) denote the mean time to resolution of defects of severity \( j \) across all engineers. We let \( t^p \) denote the mean
time to resolution of defects regardless of their severity and across all engineers. Since the parameters of the queuing model for each engineer are identical, the mean resolution times for each engineer will be the same. Thus, the average mean resolution times across all engineers will be the same as the mean resolution times for a single engineer.

The resolution time of a defect is composed of two parts. The first part is the service time, which is the inverse of the service rate. The second part is the time in the queue or the waiting time which is determined by the product of the service time and the average number of defects of the same or higher severity already in the queue when the defect arrived or the average number of defects of higher severity that may arrive when the defect is waiting. Based on this reasoning, the mean time to resolution of severity \( j \) defects for a single engineer and across all engineers is given by:

\[
t_j^{np} = \frac{1}{\mu} + \frac{1}{\mu} \left( 1 - \rho_k \right) \sum_{k=1}^{j} p_k
\]

The overall mean time to resolution, regardless of the severities, across all engineers, denoted \( t^{np} \) is given by:

\[
t^{np} = \sum_{j=1}^{m} p_j t_j^{np}
\]

4.3.2 Preemptive priority queue

We let \( t_j^P \) denote the mean time to resolution of severity \( j \) defects across all the engineers. We let \( t^P \) denote the mean time to resolution of defects regardless of their severities for all engineers.

Intuitively, in a preemptive queue, the mean time to resolution of defects of severity \( j \) is the service time expanded by the fraction of the time the engineer spends resolving defects of severities \( 1 \) through \( j - 1 \). Thus, the mean time to resolution of each engineer and across all engineers to resolve defects of severity \( j \) is given by:

\[
t_j^P = \frac{1}{\mu} \left( 1 - \rho_k \right) \frac{1}{\prod_{k=1}^{j} (1 - \rho_k)}
\]

The overall mean time to resolution, regardless of the defect severities, and across all engineers, denoted \( t^P \) can be obtained using Equation (5), by using \( t_j^P \) instead of \( t_j^{np} \).

5 Model application

In this section we illustrate how the queuing models described in Section 4.3 could be applied to actual customer defect data. Towards this end, we initially describe the characteristics of the customer defect data under consideration, followed by a discussion of how the model parameters could be estimated from the data. We then present and discuss the results obtained from the models.

5.1 Data description

We obtained the customer defect data from the defect tracking system used in Cisco Systems. The defect tracking system records the time of each defect creation and all the subsequent state transitions. Using these logs special history files are created from which we determined the number of defects that remain unresolved each week, as well as the number of defects that transition to any one of the resolved states as defined in Section 3. Defects are assigned one of six severities. Severities four through six are lightly used, and are often either cosmetic defects or requests for additional features. They are handled by other processes, not prioritized in the same way as severities one through three, and are thus beyond the scope of this paper. In our analysis we use the final (i.e. current) severity to group each defect.

The customer defect data for four software groups is obtained from the defect tracking system. For each group, the defect data is obtained on a weekly basis and consists of the number of defects fixed and outstanding for each one of the three severities. The total number of fixed and outstanding defects across all the severities is also obtained. Several weekly samples of defect data are obtained.

A 10-week sample of the defect data for a single group is shown in Table 1. In the table, we let \( F_{j,i} \) denote the number of defects of severity \( j \) that were fixed during week \( i \). \( U_{j,i} \) denote the number of unresolved defects of severity \( j \) during week \( i \) and \( F_i \) and \( U_i \) be the total number of fixed and unresolved defects (regardless of the severities) during week \( i \). Using the weekly defect data, the aggregate defect statistics are obtained. For each defect severity, and across all severities, the aggregate defect statistics consists of the average number of fixed and unresolved defects and the change in the number of unresolved defects between the first and the last weeks of the interval. We let \( AF_j \) and \( AU_j \) denote the average number of fixed and unresolved defects of severity \( j \) (all severities). These aggregate defect statistics, along with the total number of weekly samples used to compute the statistics (\( L \)) are reported in Table 2 for four groups.

The aggregate defect statistics reported in Table 2 are processed further to obtain the number of defects reported over the entire reporting interval. For severity \( j \) defects \( (j = 1, 2, 3) \), the number of defects reported \( D_j \) is given by Equation (7). Equation (7) can also be used to compute the total number of reported defects \( D \) across all the severities, by using \( AF \) and \( \Delta \) in place of \( AF_j \) and \( \Delta_j \). For each group, the number of defects reported over the entire reporting interval for each severity level and across all severity levels are reported in Table 3.

\[
D_j = (AF_j) \ast L + \Delta_j
\]

Using the aggregate statistics, the observed mean time
Table 1. Sample weekly defect data for a group

<table>
<thead>
<tr>
<th>Week #</th>
<th>Sev. #1 Fix. $F_{i,1}$</th>
<th>Unr. $U_{i,1}$</th>
<th>Sev. #2 Fix. $F_{i,2}$</th>
<th>Unr. $U_{i,2}$</th>
<th>Sev. #3 Fix. $F_{i,3}$</th>
<th>Unr. $U_{i,3}$</th>
<th>All Fix. $F_i$</th>
<th>Unr. $U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>12</td>
<td>9</td>
<td>68</td>
<td>54</td>
<td>349</td>
<td>64</td>
<td>429</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>16</td>
<td>13</td>
<td>68</td>
<td>54</td>
<td>345</td>
<td>68</td>
<td>429</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>19</td>
<td>19</td>
<td>64</td>
<td>47</td>
<td>353</td>
<td>70</td>
<td>436</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>17</td>
<td>18</td>
<td>58</td>
<td>44</td>
<td>349</td>
<td>67</td>
<td>424</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>14</td>
<td>13</td>
<td>63</td>
<td>45</td>
<td>353</td>
<td>62</td>
<td>430</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>11</td>
<td>14</td>
<td>62</td>
<td>43</td>
<td>345</td>
<td>63</td>
<td>418</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>11</td>
<td>12</td>
<td>65</td>
<td>43</td>
<td>347</td>
<td>57</td>
<td>423</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>12</td>
<td>14</td>
<td>60</td>
<td>49</td>
<td>350</td>
<td>66</td>
<td>422</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>10</td>
<td>14</td>
<td>59</td>
<td>46</td>
<td>363</td>
<td>62</td>
<td>432</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>10</td>
<td>8</td>
<td>60</td>
<td>42</td>
<td>367</td>
<td>53</td>
<td>437</td>
</tr>
</tbody>
</table>

Table 2. Aggregate customer defect statistics

<table>
<thead>
<tr>
<th>Gr.</th>
<th>Sev. level #1</th>
<th>Sev. level #2</th>
<th>Sev. level #3</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
<td>Fix. $L_{AF}$</td>
<td>Unr. $L_{AU}$</td>
<td>Ch. $L_{\Delta}$</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>52</td>
<td>1.33</td>
<td>4.10</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>52</td>
<td>0.29</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>52</td>
<td>0.38</td>
<td>1.19</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>52</td>
<td>0.40</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Table 3. Reported defects and overall mean resolution time

<table>
<thead>
<tr>
<th>Gr.</th>
<th>Sev. #1 $D_1$</th>
<th>Sev. #2 $D_2$</th>
<th>Sev. #3 $D_3$</th>
<th>All $D$</th>
<th>$t^o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>69</td>
<td>673</td>
<td>2497</td>
<td>3239</td>
<td>60.62</td>
</tr>
<tr>
<td>C</td>
<td>15</td>
<td>142</td>
<td>471</td>
<td>628</td>
<td>60.06</td>
</tr>
<tr>
<td>E</td>
<td>21</td>
<td>220</td>
<td>892</td>
<td>1133</td>
<td>62.22</td>
</tr>
<tr>
<td>H</td>
<td>23</td>
<td>331</td>
<td>1494</td>
<td>1848</td>
<td>39.23</td>
</tr>
</tbody>
</table>

A relatively uniform defect resolution process is used in these four organizations, due to which the average service time of a defect for each engineer across all the groups is 20 days. The average service time was obtained by consulting with experts familiar with the processes in these groups. We regard all the days to be equivalent, that is, we do not distinguish between regular business/work days and week ends/holidays, both in the consideration of the service time and in the computation of the observed mean resolution times using Equation (8).

5.2 Parameter estimation

The reported defect data was used to estimate the parameters of the queuing models as described in this section. The arrival rate $\lambda$ of the defects is estimated as:

$$\lambda = \frac{D}{L}$$  \hfill (9)

The probability of classifying a defect at severity level $j$ is estimated as:

$$p_j = \frac{D_j}{D}, \quad j = 1, 2, 3$$  \hfill (10)

The service rate $\mu$ for each engineer is 0.05/day, the inverse of the average service time. The reported defects are distributed equally among all the $n$ engineers in the pool available to fix defects. Thus, the effective rate at which defects are assigned to each of the $n$ engineers is given by $\lambda/n$. To estimate the rate at which defects are assigned to each engineer, we need $n$ which is the size of the pool of engineers. Since the pool size of a group is not available from the defect tracking system, we estimate $n$ using the following rationale.
For a work-conserving service discipline, the time spent in the system by a defect or the mean time to resolution of a defect across all severities is independent of the queuing discipline. Thus, based on Little’s law [14], using the observed average time to resolution $t^o$ across all the severities and all the engineers, an estimate of $\rho$ for each engineer is given by Equation (11). This estimate of $\rho$ is such that the observed and the predicted mean resolution times of defects averaged over all severities are the same.

$$\rho = 1 - \frac{1}{\mu t^o} \quad \text{(11)}$$

Based on $\rho$, the arrival rate of each engineer is given by $\mu \times \rho$. The number of engineers $n$ is then given by the ratio of $\lambda$ to the arrival rate of each engineer.

The estimates of the model parameters obtained from the aggregate defect statistics using the equations presented in this section are summarized in Table 4.

<table>
<thead>
<tr>
<th>Gr.</th>
<th>$\lambda$ (l/day)</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$n$</th>
<th>$\lambda/n$ (l/day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>62.29</td>
<td>0.021</td>
<td>0.208</td>
<td>0.771</td>
<td>1859</td>
<td>0.034</td>
</tr>
<tr>
<td>C</td>
<td>12.08</td>
<td>0.024</td>
<td>0.226</td>
<td>0.750</td>
<td>362</td>
<td>0.033</td>
</tr>
<tr>
<td>E</td>
<td>21.79</td>
<td>0.019</td>
<td>0.194</td>
<td>0.787</td>
<td>642</td>
<td>0.034</td>
</tr>
<tr>
<td>H</td>
<td>35.54</td>
<td>0.012</td>
<td>0.179</td>
<td>0.808</td>
<td>1450</td>
<td>0.025</td>
</tr>
</tbody>
</table>

5.3 Results and discussion

Using the parameter estimates summarized in Table 4, the mean resolution times for both the preemptive and non-preemptive queuing models were computed for each defect severity across all the engineers. These mean resolution times are summarized in Table 5. The observed mean times to resolution for each defect severity, denoted $t_j^o$, computed using Equation (8) by using $AO_j$ and $AF_j$ instead of $AO$ and $AF$, are also reported in Table 5.

It should be noted that the overall observed mean resolution time is used to estimate the $\rho$ for each engineer as described in Section 5.2. Thus, the overall predicted mean resolution times (across severities) of both the preemptive and non-preemptive models match exactly with the overall observed mean resolution time in Table 3, due to which the predicted overall resolution times are not reported. In summary, the estimated model parameters are such that the overall (across severities) predicted and the observed mean resolution times are identical. The predicted and the observed mean resolution times for each severity are then compared.

The results reported in Table 5 indicate that the mean resolution time for severity #1 defects is well predicted by the preemptive queuing model. For severity #2 defects, however, the observed mean resolution time is closer to the mean resolution time predicted by the non preemptive model than the mean resolution time predicted by the preemptive model. For severity #3 defects, the mean resolution times predicted by both the preemptive and non-preemptive models are similarly close to the observed. Based on these results, it seems that the repair process of an individual engineer is not completely consistent with the hypothesis of either the preemptive priority or the non-preemptive priority schemes. Therefore we consider an alternate hypothesis in which a mixed model combines preemptive handling for severity #1 defects with non-preemptive head-of-the-line priority queuing for severity #2 and #3 defects.

5.4 Mixed preemptive/non-preemptive model

In this section we present expressions for the mean times to resolution for the defects of all severities for the mixed model. The expressions, derived based on queuing theory concepts [5], are for the case when the defects are prioritized into three categories, leaving the general case for the future. Similar to the earlier models, a detailed derivation is not presented, instead an intuitive explanation is offered.

Since severity #1 defects are handled preemptively, the defects of lower severities introduce no delay in the handling of a severity #1 defect. Thus, the mean time to resolution of each engineer and hence across all engineers for severity #1 defects in the mixed model, denoted $t_{1m}^o$ is the same as the mean time in the preemptive model. $t_{1m}^o$ is thus obtained by using $j = 1$ in Equation (6) and is given by:

$$t_{1m}^o = \frac{1}{\mu} \left(1 + \frac{\rho_1}{1 - \rho_1}\right) \quad \text{(12)}$$

Intuitively, the time to resolution of a severity #2 defect is the sum of three parts, namely, the service time inflated by the probability that it is preempted by a severity #1 defect, the time spent waiting for the resolution of any defect (severity #1, #2 or #3) in process when it arrives and the time spent waiting for the other severity #2 defects already in the queue. In the mixed model, the resolution time of a severity #2 defect is expected to be even worse than the pure non-preemptive model, because a severity #2 defect is preempted by a severity #1 defect but does not get the benefit of preempting a severity #3 defect. Thus, the mean time to resolution of a severity #2 defect is given by:

$$t_{2m}^o = \frac{1}{\mu(1 - \rho_1)} \left(1 + \frac{\rho}{1 - \sum_{j=1}^{2} \rho_j}\right) \quad \text{(13)}$$

The time to resolution of a severity #3 defect in the mixed model will be worse than the non-preemptive model because it is preempted by a severity #1 defect, but it may
not be as bad as the purely preemptive model because it is not preempted by a severity #2 defect. In short, it is the same as in the case of the non-preemptive model with its service time expanded due to the preemption by a severity #1 defect. The time spent waiting in the queue is unchanged since all the defects which are ahead of a new one need to be resolved even in the non-preemptive case. Thus, the mean time to resolution of a severity #3 defect in the mixed model is given by:

\[
t_{3}^{m} = \frac{1}{\mu} \frac{1}{(1-\rho_{1})} + \frac{\rho}{(1-\sum_{j=1}^{2}\rho_{j})(1-\sum_{j=1}^{3}\rho_{j})}
\]  

The average resolution time across defects of all severities and all engineers is given by Equation (5), for \( m = 3 \) and \( t_{j}^{m} \) instead of \( t_{j}^{np} \).

Using these expressions, the mean resolution times for the mixed model are computed and are reported in Table 5. The results in the table indicate that the mean resolution times for all severities obtained from the mixed model are very close to the observed mean resolution times. Further, as expected, the mean resolution time of severity #2 defects in the mixed model is slightly worse than the pure non-preemptive model. Also, the mean resolution time of severity #3 defects in the mixed model lies in between the mean resolution times of the non-preemptive and preemptive models, which is also expected. Thus, the mixed model accurately captures the characteristics of the defect resolution process from which the data are generated.

### 5.5 Reliability implications

A severity #1 defect has a serious impact on the services provided by a system. When a severity #1 defect is encountered, in some cases a repeated loss of service can be prevented by providing the customer with a workaround. The workaround, however, often involves some compromise and inconvenience and there is a risk that the defect may be encountered again by the same or another customer. A severity #1 defect should thus be resolved as quickly as possible to improve the field system reliability. For the data under consideration, because of the preemptive handling of the severity #1 defects, coupled with the fact that they account for a small percentage of the overall defect population, their mean time to resolution is primarily influenced by the service time. Thus, the mean resolution time of severity #1 defects, and hence the system reliability can be improved by reducing the service time.

The impact of lower severity defects on system reliability may not be as severe as the severity #1 defects. However, it may be still desirable to resolve them as fast as possible. For a given service time, which may be tailored to ensure that the resolution time of severity #1 defects is acceptable, the mean resolution time of severity #2 and #3 defects can be improved by adding more engineers to the pool. The queuing models presented in this paper can facilitate predictive or “what-if” analysis, to determine the impact of increasing the pool size. Table 6 summarizes the mean resolution times for severity #2 and #3 defects when the size of the engineer pool is increased by 10% and 20%. These resolution times are computed using the expressions for the mixed model in Section 5.4. The results reported in the table indicate that increasing the pool size has a negligible impact on the mean resolution time of severity #1 defects, which is expected. Further, the impact on the mean resolution time of severity #3 defects is more than the impact on severity #2 defects. This is because the percentage of severity #2 defects is much lower than the percentage of severity defects (20% severity #2 and 77% severity #3). As the percentage of severity #2 defects increases, the impact of increasing the engineer pool size on their resolution time will be higher. The mean resolution time across all defects reduces as the pool size increases. Thus, in this case, to improve the responsiveness, which is perceived as the mean resolution time of the defects regardless of their severity, the size of the engineer pool must be increased.

It is important to note that increasing the pool size does
not change the total amount of work required to fix the defects. It simply redistributes the work among more engineers, which drives down the load of each. Usually, the pool size is increased not by hiring new staff, but by reallocating a portion of the time of some engineers to the defect resolution activity. These engineers may be originally involved with other activities such as developing new features and these activities may be preempted and delayed due to such reallocation. Thus, the decision to increase the pool size must be made after careful consideration of the associated tradeoffs. The models in this paper can provide a systematic, quantitative guidance to enable such tradeoffs.

6 Conclusions and future research

We have shown the possibility and the benefits of applying multi-priority queuing models to the software defect resolution process. In particular, we have shown the utility of structuring the system as $n$ independent $M/M/1$ queues. Further, for the defect data considered in the paper we found that is necessary to use a priority system which is mixed, that is, it is neither purely preemptive nor non-preemptive. Finally we have shown that to the extent defects of different severities affect reliability to different degrees, understanding and management of queuing behavior can precisely and quantitatively identify the impact of different aspects on the resolution times of different severities and hence suggest methods to improve the delivered reliability.

There are several opportunities to further advance this research. Data from additional groups, or from the processing of internal defects, could validate the approach and extend our catalogue of possible behaviors. It should be possible to perform a detailed Markovian analysis of the state transitions in the defect life cycle with an eye to identifying process bottlenecks. It would be useful to study the exact distribution of service times, to understand more than average behavior. Finally, it should be possible to integrate multi-priority repair models with models of software reliability growth or defect re-occurrences.

Acknowledgments
We would like to thank John Intintolo, Cisco Systems for his encouragement.

References


