A Split Poisson Process Model for the Occurrence of Defects and Change Requests during User Acceptance Testing

Kevin McDaid and Simon Wilson

***Abstract*—Software developed for a specific customer under contract typically undergoes a period of testing by the customer before acceptance. This is known as user acceptance testing and the process can reveal both defects in the system and requests for changes to the product. This paper uses nonhomogeneous Poisson processes to model a real user acceptance data set from a recently developed system. In particular a split Poisson process is shown to provide an excellent fit to the data. The paper explains how this model can be used to aid the allocation of resources through the accurate prediction of occurrences both during the acceptance testing phase and before this activity begins. Furthermore the paper proposes a decision theoretic solution to help the customer decide how long to allow for this phase. The method allows for the cost of defects occurring during operation and the delayed introduction of the system. It also allows for the difference in cost of implementation of change requests during acceptance testing and during operation.**

***Index Terms*—User acceptance testing. Software reliability growth modelling. Split Poisson process. Decision Theory. Optimal software release. Bayesian methods. Utility.**

I. Introduction

THE testing of modern software systems to remove remaining defects is a difficult and costly business. Under the traditional Waterfall development methodology, it begins with unit testing and progresses to integration testing to ensure the separate components interface correctly. It is usual to finish with a prolonged amount of system testing where a combination of testing methods can be used. These seek to remove remaining defects and to establish that the system complies with the requirements for the software.

In the case of a system developed for a single customer, known as bespoke software, it is also standard to include a final testing phase immediately before release, conducted collaboratively by the customer and the developing organization, to ensure that the product complies with the requirements from the customer's perspective. This is known as User acceptance Testing. A detailed description of this activity can be found in [1] and [2]. This phase can reveal defects or errors in the software. Unlike traditional system testing, it can also reveal issues that are effectively additional to or changes to the requirements. These are known as change requests and the developing organization would normally charge the customer the cost of implementing these changes to the system.

The best practice of Software Reliability Engineering, as detailed in [3], has developed a range of probabilistic models to predict the reliability of a system during operation using the occurrence time of defects during testing. These are known as Software Reliability Growth models (SRGM's) and they have focussed almost exclusively on defect data form the system testing phase. This paper examines the application of a particular class of models, known as nonhomogeneous Poisson processes, to defect and change request data occurring during user acceptance testing of a recently developed commercial bespoke distributed database system. Building on these models the paper proposes a split Poisson process model to represent the data.

In addition to predicting the achieved reliability of a software system, test and project managers can use SRGM's to predict the number of defects that will occur over a future time period. This can allow for more accurate resource allocation and possibly, depending on the exact release criteria used by the firm, support the decision when to terminate the system testing phase of the lifecycle ([4], [5]).

Of course, accurate predictions early in the testing process are of most value to project and test managers. Unfortunately, during the initial stages of system testing the amount of defect data available is limited and standard approaches to fitting SRGM's, such as maximum likelihood methods, can be problematic ([6], [7], [8]). This paper addresses this problem in the context of user acceptance testing using Bayesian methods that combine the limited existing defect data with the expert opinion of key personnel. In this way improved estimates for the parameters of the SRGM's can be obtained, which in turn can lead to better defect and change request occurrence predictions.

The traditional approach to software development relies heavily on the system testing phase to ensure the released product is highly reliable. Experience has shown that without substantial testing at this stage it is likely that the product will most likely contain a large number of faults with the potential to lead to regular failure during operation. This in turn will result in high post-release costs and a significant loss of consumer confidence. Thus, it is important to allocate a significant amount of time to the activity of system testing. However, it is also important to avoid over testing the product. This can result in a piece of software which, although very reliable, may

Dr K. McDaid is with the Department of Computing and Mathematics, Dundalk Institute of Technology, Co Louth, Ireland. Phone: +353 42 9381760, e-mail: kevin.mcdaid@dkit.ie

Dr S. Wilson is with the Department of Statistics, Trinity College Dublin, Ireland. Phone: +353 1 6081759, e-mail: swilson@tcd.ie

be overpriced and possibly obsolete. On the whole it is crucial for the test manager to achieve a balance between under testing and over testing. This is also an issue for the customer organization who must decide how long to allow for the user acceptance testing phase. In fact the decision is more complicated in their case as they must also allow for the difference between the cost of implementing change requests before and after acceptance of delivery of the product.

This optimal release decision is addressed in a fundamental way in Software Reliability Engineering. Within this best practice it is well understood that the software should be tested and repaired to achieve a necessary level of reliability, and that valuable resources should not be expended in an attempt to go beyond the predetermined level. Similarly, this is also an issue for the customer receiving a bespoke solution as they must make a decision when the product is reliable enough to accept delivery. This research takes a broader view of the problem with the assumption that the final decision should depend on a range of factors including the cost of defects during operation, the loss in competitiveness due to delayed acceptance, the cost of involving personnel in testing, as well as the costs of implementing changes before and after delivery of the product.

Specifically, the work adopts a decision-theoretic approach that includes a decision strategy to determine the stopping rule, a utility to describe the consequences to the customer organization of accepting the product at each time and a probability model to describe the occurrence of defects and change request over time. The optimal time to stop the user acceptance testing phase is that which maximizes the expected utility.

This paper contributes to the existing knowledge base in a number of ways. It develops and assesses, based on real data, an original model for an under-researched problem, namely the occurrence of defects and change requests during user acceptance testing. It demonstrates its application to prediction before and during the testing process through the combination of defect data with expert knowledge. Finally, it develops an original utility function to support the manager in the decision as to when to accept delivery of the software. This is used to apply both a single and multiple stage decision strategy.

The paper is summarized as follows. Section II introduces the area of Software Reliability Engineering and explains briefly nonhomogeneous Poisson process models and their application to defect data. The rationale for their use with defect and change request data arising from user acceptance testing is also explained. A split Poisson process models is developed and compared with independent Poisson process models for defects and change request. The comparison is made on the basis of real data which is also detailed in this section. Section III explains how to apply the models to the prediction of occurrences. A Bayesian model is developed to ensure that stable predictions can be made throughout the user acceptance testing phase. The quality of these predictions are assessed. The method used to elicit expert opinion is also explained. Section IV develops a utility function to assess how long to allow for user acceptance testing. A number of decision strategies are explained and applied to the real data. Section V concludes the work with a short summary.

II. A Probability Model for Occurrences during User Acceptance Testing

This section introduces the data under examination and introduces a number of nonhomogeneous Poisson processes to model the occurrence of change request and defects.

A. User Acceptance Testing Data

The system under investigation was developed using a V-model approach by a large software company for a government department. The on-line system, supported by significant batch and database facilities, included a range of functionality from basic detail capture to complex rule-based calculations. Development required approximately 3,500 person days of effort over a period of eighteen months involving 29 different personnel. A more detailed description is provided in [9] and [10].

The available data for the software system is taken from the pre-release system testing and user acceptance testing phases and the post-release operational phase. In this paper we examine the occurrences over the user acceptance testing period only. Before considering the application of software reliability growth models we must establish a suitable unit of time. A difficulty arises here due to the lack of information on the time spent by personnel testing the system. The analysis in this paper uses calendar days as the unit of time, with days where UAT testing did not take place excluded from the analysis. While weekend and Christmas periods were clear cases of inactivity, every effort, including examination of time sheets and discussions with senior staff, has been made to exclude days when testing activity did not take place.

The data gathered during the user acceptance testing phase distinguishes between defects and change requests, where change requests are user-suggested changes to the system. Defects and change requests when taken together are termed issues. Figure 1 shows a plot of the occurrence of defects and change requests over time, where time is the standardized time as described earlier. Note that in the first 50% of the time (45 days) 72% of the issues were found (237 out of 329). Also note that the data fails to show any continuity between system and user-acceptance testing. The rate of defect occurrence at the end of the system testing phase is more than twice that at the start of the user acceptance testing phase reflecting the different testing strategies. In system testing, effort is focused on the revelation of the highest possible number of failures whereas in user acceptance testing the emphasis is on the verification of the system through the application of real inputs by potential users.

To finish this subsection we note that the user acceptance testing was conducted by the customer with the help of the developers. This raises the question as to whether

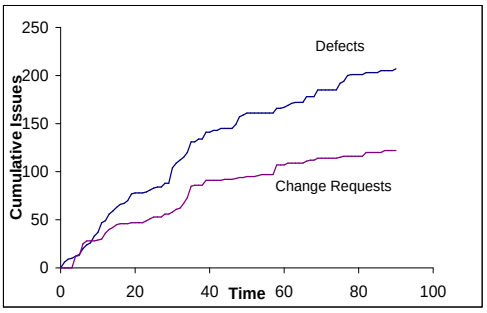

Fig. 1. Cumulative occurrence of software defects and change requests over time during the user acceptance testing phase.

Model	Mean Value Function, $m(T)$
Goel-Okumoto	$a(1-e^{-bT})$
Yamada S-shaped	$a(1-(1+bT)e^{-bT})$
Weibell	$a(1-e^{-bT^c})$
Log-logistic	$a\left(\frac{(bT)^c}{1+(bT)^c}\right)$

TABLE I
MEAN VALUE FUNCTIONS FOR GOEL-OKUMOTO, S-SHAPED, WEIBULL AND LOGLOGISTIC MODELS

the testing can be considered as statistical or based on a more structured coverage approach. While there is little information available on the exact inputs applied, the fact that this testing was driven by the customer using real data would indicate that the method may be closer to statistical rather than a structured coverage-based approach. That said, other research, [11], supports the use of the class of models we consider in Section B for defect data arising through testing strategies that involve the selection of inputs to maximize some measure of achieved coverage.

B. Nonhomogeneous Poisson Process Model

Numerous probabilistic models are available in SRE for the purpose of reliability and defect occurrence prediction. These models are usually classified as data-domain or time-domain models, ([12] and [13]), with each category containing several sub-categories. A widely accepted group of these models are the class of non-homogeneous Poisson processes (NHPP) where the total number of defects expected is finite. These were adopted from hardware reliability models between the early 1970s and the mid 1990s. Of these models four have achieved prominence, namely the Goel and Okumoto [14] model and Yamada S-Shaped model [15], each based on two parameters, and the three parameter Weibull and Log-logistic models [11]. The mean value functions for each of these is shown in Table I.

A relatively simple explanation of the Goel-Okumoto model is possible through the mean value function presented, which gives the cumulative number of defects that would be expected to occur before each time point, T. For the Goel-Okumoto model the mean value function has two parameters (a and b). Typically, these are estimated from the occurrence times of defects during testing. For this form the average number of defects discovered by time T, $m(T)$, is of course increasing over time but the rate of increase, representing the rate at which new defects come to light, slows with the highest rate value at the beginning of testing. The other models differ from the Goel-Okumoto in shape as the rate of occurrence initially increases before then decreasing. For the Goel-Okumoto model the a parameter represents the number of defects that would be found were testing to continue at infinitum. The b parameter represents the rate at which defects come to light. More specifically, it gives (approximately) the proportion of the remaining defects that would be discovered in a single unit of testing.

The NHPP software reliability models contain many assumptions, some of which are unrealistic, including the instant and perfect repair of defects. However, in practice, [16] is a good example, the models has been found to provide a good mechanism for modelling software failure data. Estimation of the parameters of these models is normally performed by maximizing the likelihood or density function which, in the case of ungrouped data, can be written as

$$f(t_1, t_2, \ldots, t_n) = e^{-m(T)} \prod_{i=1}^{n} \Lambda(t_i), \tag{1}$$

where $\Lambda(t)$ is the rate function found by differentiating the mean value function and T is the time at which testing is censored with n occurrences at $t_1, t_2, \ldots, t_n$ before this point.

In the case of the Goel-Okumoto model with parameters a and b the specific form is

$$(ab)^n e^{-a(1-e^{-bT})} e^{-b\sum_{i=1}^{n} t_i}. \tag{2}$$

The maximum likelihood value for the a parameter is given by

$$a = \frac{n}{1-e^{-bT}}, \tag{3}$$

where the value for b is found by numerically solving the following:

$$\frac{n}{b} = \sum_{i=1}^{n} t_i + \frac{nTe^{-bT}}{1-e^{-bT}}. \tag{4}$$

However, as other authors have documented ([6] and [17]), there can be significant issues with this approach. Specifically, there are combinations of defect occurrence values for which the maximum likelihood estimates for the Goel-Okumoto model do not exist. This situation occurs most often during the early stages of testing. However, a further problem arises where, although the maximum likelihood estimates for the parameters can be derived, the resulting values are unrealistic. This again predominantly occurs early in testing and often yields very high values

for the a parameter in the case of the Goel-Okumoto and the Yamada S-shaped model. We highlight this problem in Section III through the real industry example. Note that this drawback also applies to the Weibull and Log-logistic models.

Importantly, the mean value functions for the NHPP models discussed above can all be written in the form a[C(t)] where a is the number of defects that may be discovered were testing to continue at infinitum and C(t) can be thought of as the percentage coverage achieved over time. It is this structure that is used in [11] to argue that these models are suitable for application in the case where the defect data arises through testing that is coverage-based rather than operational profile based. We now turn our attention to the fitting of these models.

C. Independent Poisson Process Model

The simplest approach to modelling the occurrence of defect and change request data would assume that the processes were independent and fit nonhomogeneous Poisson process models to the defect and change request data separately. We have fitted the four models to each of the series. The Goel-Okumoto, Weibull and Log-logistic models result in very similar maximum likelihood values with the S-shape model performing relatively poorly. Of these, the Goel-Okumoto model requires one fewer parameters and should thus be considered as the best choice to model each of the defect and change request data sets. A formal calculation of the Akaike Information Criterion (AIC) supports this conclusion. It is not surprising that the Goel-Okumoto model is chosen given the shape of the curves. Figure 2 shows the data and the fitted Gole-Okumoto models. It is clear that the model represents the actual data reasonably well with some problems with the fit to the data during the period from day 20 to 60.

Based on the maximum likelihood methods the model values were found to be a=266 and b=0.017 for the defects and a=143 and b=0.022 for the change requests. The relative values for the rate parameters indicate that the user acceptance testing phase may be more successful at revealing change requests than revealing failures. The question we next consider is whether a simpler model using three rather than four parameters can be found to fit the data.

D. Split Poisson Process Model

Previously we fitted independent Poisson process models to the defect and change request data. A closer examination of the user acceptance testing process may reveal clues as to a potential alternative model. Within the process it seems that when an issue arose it was later classified as a defect or a change request. This would indicate an underlying process where the occurrences are separated into two types. We assume a probability, θ, that the occurrences are of type I (defects) and $1-\theta$ that the occurrences are of type II (change requests). If this value of θ does not change over time then this is the case of a split Poisson process. Suppose defects occur at times $t_1, t_2, \ldots, t_n$ and change requests at times $s_1, \ldots, s_m$ before censoring time

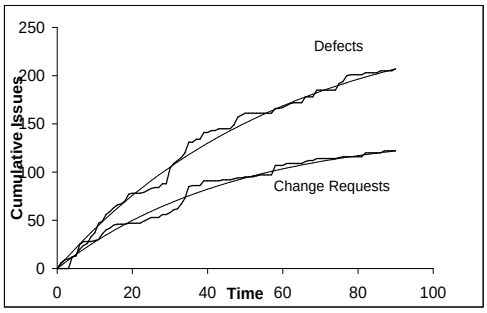

Fig. 2. Cumulative occurrence of software issues over time during the user acceptance testing phase with fitted independent Goel-Okumoto models.

T then the density function, $f(t_1, t_2, \ldots, t_n, s_1, \ldots, s_m)$, is given by

$$\left[(ab)^{n+m} e^{-a\left(1-e^{-bT}\right)} e^{-b\left(\sum t + \sum s\right)}\right] \left[\theta^n (1-\theta)^m\right], \quad (5)$$

which can also be written in the following form

$$\left[(\theta ab)^{n} e^{-\theta a\left(1-e^{-bT}\right)} e^{-b\left(\sum t\right)}\right] \times$$

$$\left[((1-\theta)ab)^{m} e^{-(1-\theta)a\left(1-e^{-bT}\right)} e^{-b\left(\sum s\right)}\right]. \quad (6)$$

This shows that the occurrence of defects and change requests can be treated as independent nonhomogeneous Poisson processes with mean value functions given by $\theta a\left(1-e^{-bT}\right)$ and $(1-\theta)a\left(1-e^{-bT}\right)$ respectively. This is a well-known result from Poisson process theory and implies in this study that the occurrence of defects follows a Goel-Okumoto model with parameters $a\theta$ and b, and the occurrence of change requests follows an independent (given model parameters) Goel-Okumoto model with parameters $a(1-\theta)$ and b.

The structure of the density function in Equation 5 implies that the maximum likelihood estimates for the a and b parameters can be found using the solution presented in Equations 3 and 4. The maximum likelihood estimates for θ is the proportion of defects in the data, namely $\frac{n}{n+m}$. We fit the split Poisson process model to the data and display the result in Figure 3. The common rate parameter is estimated as 0.0185 with the overall a parameters given by 406 with $\theta = 0.63$ yielding a values for the defects and change requests given by 255 and 150 respectively. Crucially, the AIC value for this model is 173.75 which is less than the value of 174.76 for the four parameter Goel-Okumoto models presented earlier. This provides some indication that the benefit of a reduced parameter model (three as opposed to four) outweighs the reduction in fit of the model. Having established the model we will show shortly how it can be use for prediction purposes.

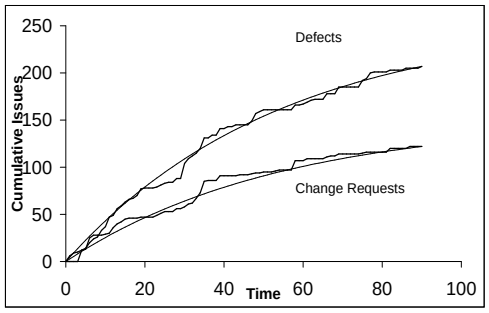

Fig. 3. Cumulative occurrence of software issues over time with fitted split Poisson process model.

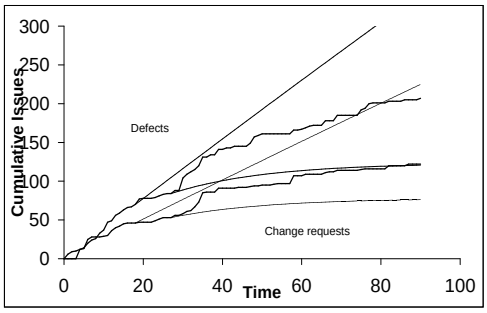

Fig. 4. Prediction at 20 and 30 percent points using maximum likelihood

As the mean value function for the four models presented in Table I can be written as $a[C(t)]$ it is relatively straightforward to develop a split process in the case of each of these models.

III. Prediction of Defect and Change Request Occurrence

The power of a model is in its potential to provide accurate predictions for future occurrences. In this section we examine how this can be achieved both before and after the user acceptance testing phase.

A. Issues with Early Prediction

As mentioned earlier, estimation of the parameters of these models is normally performed using maximum likelihood methods. It is well known ([6], [17]) that there can be significant issues with this approach as there are combinations of defect occurrence values for which the maximum likelihood estimates for the Goel-Okumoto model do not exist. A further problem arises where, although the maximum likelihood estimates for the parameters can be derived, the resulting values are unrealistic. These difficulties are most likely to occur early in testing and often result in very high values for the a parameter. We illustrate this process in Figure 4 where the split Poisson process model is fitted based at the 20 and 30 percent time points using the data available up to that point. The figure shows the poor performance of the predicted cumulative number of defects and change requests following that point. Note that it is not possible to fit the data using occurrences up to the 10 % point.

A further criticism of the maximum likelihood approach is the fact that it ignores the expert knowledge of key project personnel accumulated over possibly years of involvement with the industry and organization in question. This knowledge should influence the selection of the parameters and should in particular protect against the selection of extreme values for the parameters. The approach also suffers from the lack of a mechanism to predict the occurrences prior to the commencement of testing, when predictions are of most value to the firm. To counter both these drawbacks we next present a Bayesian version of the split Poisson process model and apply it to the prediction problem.

B. Bayesian Split Poisson Process Model

Assuming the split Poisson process model in the previous section, we adopt a Bayesian model by placing prior distributions on the model parameters, a, b and θ. Independent prior gamma distributions are placed on the a and b parameters with a prior beta distribution placed on the θ parameter. These assumptions are flexible and allow us to incorporate a wide variety of prior knowledge.

Specifically, we assumes that the a is represented by a prior gamma distribution with parameters τ and λ, b by a prior gamma distribution with parameters α and μ and θ by a prior beta distribution with parameters ω and ρ. The choice of a prior gamma distribution is consistent with [18] and [19]. Thus the prior structure is as follows

$$f(a,b,\theta|\tau,\lambda,\alpha,\mu,\omega,\rho) = f(a,b|\tau,\lambda,\alpha,\mu)f(\theta|\omega,\rho)$$

$$= \frac{a^{\tau-1}e^{-\lambda a}\lambda^{\tau}}{\Gamma(\tau)}\frac{b^{\alpha-1}e^{-\mu b}\alpha^{\mu}}{\Gamma(\alpha)}\frac{\Gamma(\omega)\Gamma(\rho)\theta^{\omega-1}(1-\theta)^{\rho-1}}{\Gamma(\omega+\rho)} \quad (7)$$

As the prior and density functions for the proportion parameter, θ and the Goel-Okumoto model parameters a and b can be separated we can use available results to generate prediction methods for the number of defects and change requests based on the posterior distribution.

We next illustrate how to evaluate the number of defects that would be expected to occur after T_1 units of acceptance testing, assuming $T(T \leq T_1)$ units of testing has already taken place resulting in n defects at times $t_1, t_2, \ldots, t_n$ and m change requests at $s_1, \ldots, s_m$. This is developed as

$$E(\bar{N}_D(T_1)|\tilde{t},\tilde{s},\tau,\lambda,\mu,\alpha,\omega,\rho)$$
$$= E_{a,b,\theta}(E(\bar{N}_D|a,b,\theta))$$
$$= [E_{\theta}(\theta)]\left[E_{a,b}(\bar{N}_{D,CR}(T_1))\right]$$
$$= [E_{\theta}(\theta)]\left[E_{a,b}(ae^{-bT_1})\right]$$

$$= \left[\int_\theta \theta f(\theta|\omega,\rho)\right]\left[\int_{a,b} f(a,b|\tilde{t},\tilde{s},\tau,\lambda,\mu,\alpha)(ae^{-bT_1})dadb\right]$$
$$= \left[\frac{\omega+n}{\omega+\rho+n+m}\right]\times$$
$$\left[\frac{1}{(1+\lambda)Y}\sum_{i=0}^{\infty}\frac{\Gamma(n+m+\tau+i+1)}{i!(1+\lambda)^i(\sum t+\sum s+\mu+T_1+iT)^{n+m+\alpha}}\right], \quad (8)$$

where

$$Y=\sum_{j=0}^{\infty}\frac{\Gamma(n+m+\tau+j)}{j!(1+\lambda)^j(\sum t+\sum s+\mu+jT)^{n+m+\alpha}},$$

and $\bar{N}_{D,CR}(T_1)$ represents the combined number of defects and change requests occurring after time T_1. The prediction for the number of defects before testing begins at which point only prior information is available can be shown to be

$$E(\bar{N}_D(T_1)|\tau,\lambda,\mu,\alpha,\omega,\rho)=\left[\frac{\omega}{\omega+\rho}\right]\frac{\tau}{\lambda}\left(\frac{\mu}{\mu+T_1}\right)^{\alpha} \quad (9)$$

The prediction for the number of change requests, corresponding to Equation 8, has an identical second term with the first term given by $\frac{\rho+m}{\omega+\rho+n+m}$. Note that the expansion of the $E_{a,b}(\bar{N}_{D,CR}(T_1))$ term in Equation 8 follows from work in [18]. The fact that the posterior distribution for the proportion, θ, follows a beta distribution with parameters $\omega+n$ and $\rho+m$ is used to expand the $E_\theta(\theta)$ term

C. Assessment of Predictions

We next apply the Bayesian split Poisson process model to the data presented previously. We assume two separate sets of prior values for the model parameters, one which represents the situation where the expert opinion is very close to the true values and the second where the prior opinion yields estimates for a and b which are higher than the ideal values and the prior values for the beta distribution represent a uniform distribution. The second set illustrates the situation where the expert provide poor, yet not completely unrealistic, prior information. Table II shows the two selected parameter sets. The mean and standard deviation of the prior distributions are also presented as, for comparison purposes, are the maximum likelihood estimates based on all the available defect and change request data.

The predictions for the remaining number of defects and change requests based on the 20 and 30 percent points are presented in Figure 5. These predictions, for both sets of parameters, are a significant improvement on those presented in Figure 4.

D. S-Shape and other Models

We have, to this point, concentrated on the Goel-Okumoto model for which a closed form for the prediction

	Set 1	Set 2
τ	400	60
λ	1	0.1
Mean a	400	600
SD a	20	77
MLE a	406	406
α	20	3
μ	1000	100
Mean b	0.02	0.03
SD b	0.0045	0.017
MLE b	0.0185	0.0185
ω	6	1
ρ	4	1
Mean θ	0.6	0.5
SD θ	0.15	0.29
MLE θ	0.63	0.63

TABLE II
PRIOR PARAMETERS SELECTIONS COMPARED WITH MAXIMUM LIKELIHOOD ESTIMATES

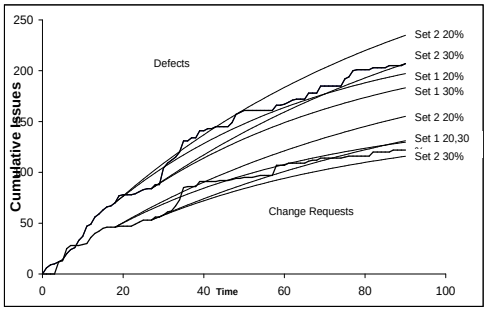

Fig. 5. Predictions at 20 and 30 percent points using Bayesian split Poisson process.

of the posterior expected number of occurrences is available. A similar prediction for the S-shaped model follows from Equation 8 with the $E_{a,b}(\bar{N}_{D,CR}(T_1))$ term given by

$$\frac{\sum_{i=0}^{\infty}\left[(i+1)g(i+1,T_1-T)\sum_{j=0}^{i+1}h(j,i+1,T_1-T)\right]}{\sum_{i=0}^{\infty}\left[g(i,0)\sum_{j=0}^{i}h(j,i,0)\right]}, \quad (10)$$

where $T_1 \geq T$ and

$$g(i,x)=\frac{(n+m+\tau+i-1)!}{(1+\lambda)^{i-1}(\mu+\sum_{i=1}^{n}t_i+\sum s+x+iT)^{2n+2m+\alpha}},$$

and

$$h(j,i,x)=\frac{(2n+2m+\alpha+j-1)!T^j}{j!(i-j)!\,(\mu+\sum t+\sum s+x+iT)^j}.$$

For the other two models, namely the Weibull and the Log-logistic models, the corresponding estimates must be

obtained using Markov Chain Monte Carlo methods. Some work in this area is available in [7] and [19]. We now look at how the split Poisson process model can be used to help the customer decide how long to allow for user acceptance testing.

IV. How long to Allow for User Acceptance Testing

The decision as to how long the customer should allow for user acceptance testing is not a well-researched one. This contrasts with the extensive work done to help producers of software decide when to terminate the system testing phase and to release the product (see [5], [12] and [20] for examples). In practice, many firms that employ the best practice of Software Reliability Engineering tend to base this release decision on either the achieved reliability of the product or the estimated number of defects in the system remaining to be discovered.

However, the state of the art approach, as developed in [21], is to develop a utility function that gives the value to the software firm of accepting the product at every possible time point. This function would include terms that measure the number of defects remaining and the cost to the firm of their discovery post acceptance. It would balance this against the cost of continued testing and loss of business opportunity in delaying the acceptance decision. In this section we present an analogous method for the decision when to terminate the user acceptance testing phase.

In general the decision-theoretic approach includes a testing plan to dictate the decision process, a utility to describe the consequences of accepting the product at each time and a probability model to describe the occurrence of defects and change requests over time. The optimal time to test is that which maximizes the expected utility. Decision plans that describe the process and basis of release decisions for software under development can take a variety of forms. The single-stage plan simply uses information available prior to testing to determine the release time, regardless of the number or pattern of occurrences during testing. Singpurwalla [12], McDaid and Wilson [18] and Okumoto and Goel [22] have investigated the solution for this plan using a variety of models.

The most desirable testing plan is the sequential strategy, in which the test failure data informs the decision process on a continuous basis, to enable an infinite sequence of decisions eventually culminating in a decision to terminate testing. However, the solution is in most cases computationally intractable. Other testing plans that are more reasonable and flexible have been developed in the literature. van Dorp et al [20] introduced the one stage look-ahead plan which, based on a fixed set of decision times, compares the expected utility associated with stopping testing immediately with the expected utility of proceeding to the next decision time and then stopping. It is this decision plan that is used later in Section C

However, this strategy suffers in that it fails to take account of the possible future decisions beyond the current time point. McDaid and Wilson [18] conducts a study to compare this plan with a number of other plans including the single stage and the more flexible two-stage decision plan. Dalal and Mallows ([5] and [23]) first proposed the sequential solution which they derived under certain asymptotic assumptions. Chavez [4] proposed a decision theoretic solution that allowed for the severity of defects. Other related work can be found in [24], [25], [26] and [27].

A. Utility Function

The utility function describes the costs and benefits to the software company of testing a software system to a time T during user acceptance testing and then accepting the product. A utility function of the following form is proposed.

$$\mathcal{U}(T, \bar{N}_D(T), N_{CR}(T), \bar{N}_{CR}(T)) = A - F(T) - C_D\bar{N}_D(T) - C_{CR1}N_{CR}(T) - C_{CR2}\bar{N}_{CR}(T) \tag{11}$$

where

- There is a value A representing the economic value of receiving a perfect system requiring no changes and thus removing the need for any user acceptance testing.
- The cost of staff engaged in testing the program to time T and the lost opportunity, in terms of lost initiative, of delayed acceptance of the product are expressed by a function $F(T)$. A simple form for $F(T)$ would be linear,

$$\begin{aligned} F(T) &= (S+R)T, \\ &= fT \end{aligned}$$

  with S representing the staff cost and R the lost opportunity (possibly lost revenue) per unit of time.
- The cost of each defect found after the system is accepted in terms of inefficiencies is a constant C_D. The actual cost of repairing these defects should be borne by the developing organization, at least over the initial period when most defects will come to light.
- The cost of implementing each change request found during acceptance testing is a constant C_{CR1}.
- The cost of implementing each change request found during operation is a constant C_{CR2}; typically, C_{CR2} should be much larger that C_{CR1} for the same reasons as the cost of resolving defects found during operation is substantially large than the cost of resolving defects found during testing.

The proposed form for the utility function is similar to that used by other authors ([4], [5] and [21]) when considering the problem of how long to allow for system testing. Although utility is not equivalent to monetary profit, the utility function could be interpreted as the benefit in financial terms over the entire lifetime of the software, and, given this, our approach would perhaps be more accurately called a cost benefit analysis.

It is not difficult to adapt the above utility in a number of ways. First, it is possible to incorporate a maximum lifetime for the system and a time beyond which the developer will not be contractually required to fix defects. It is also possible to restrict the number of change requests that are implemented through a total allowable spend on this activity or through a time point after which it is not possible or feasible to make changes.

While the basic approach assumes that the occurrences follow a Goel-Okumoto model during user acceptance testing, the occurrences during operation can follow any model provided that the total number of defects and change requests discovered over time is consistent with the predicted number (a in the case of the Goel-Okumoto model). However, to include the additional time parameters mentioned in the previous paragraph it would be necessary to explicitly consider the pattern of defect and change request occurrences during operation.

If it is believed that this pattern differs from the pattern during user acceptance testing than this will have to be allowed for. This may involve an adjustment of the rate parameter of the Goel-Okumoto model or the adoption of a different model and the use of expert opinion and possibly defect and change request occurrences to provide information on the parameters. In any case the framework proposed can deal with these approaches.

With regard to the costs, the fixed C_D, C_{CR1} and C_{CR2} values could be replaced by random quantities that allow for the cost of defects and the implementation of change requests to vary. In this case the mean of the distribution for each of the costs would replace the fixed values. Also, a term to allow for possible penalty clauses to be invoked in the case of delivery of seriously unreliable product could be added. The application of decision theoretic approaches to determining the time to stop system testing in the presence of penalty clauses is discussed in [28].

The probability models developed in Section II assume time to be execution or cpu time whereas, in the utility function, calendar time is more reasonable for the opportunity loss term. If cpu time is a constant fraction of calendar time then one can divide those parts of the utility defined with calendar time by that fraction to transform them to cpu time. We implicitly assume that this has been accounted for in the assignment of utility parameter values.

The elicitation of the utility and model parameters is an important issue in the practical application of the work. McDaid and Wilson [18] describe a method to deduce values for the prior parameters of a similar model based on the predicted number of occurrences supplied by an expert along with a measure of the likely variance in the number. The utility parameters are directly associated with financial costs and benefits of testing, and so it should be possible for the customer to make reasonable estimates of their values. It may be easier to fix arbitrarily one of the costs, say C_{CR1}, to be 1 and then think of the other costs in terms of multiples of this cost. For the value of f, one may choose a time in the future (T' is a candidate) and assess the cost X of testing and lost opportunity to that point.

As an illustration, suppose that the customer is deciding on the length of time to test a piece of software. First, it assesses the utility parameters. Fixing $C_{CR1} = 1$, it decides that the cost of implementing change requests after release is 10 times C_{CR1}, so $C_{CR2} = 10$, and that the cost to the firm of each defect found is half C_{CR1} giving $C_D = 0.5$. It concludes that the utility of the perfect software is $A = 3000$. Finally, for the value of f, it looks at the opportunity and staff cost of testing to a time $T' = 20$, and decides that it is $200C_{CR1}$; thus f is 10. As regards the prior probability we will use the two sets quoted already in Section II.C. It is interesting to note that the accurate specification of the prior and utility parameters will require the assistance of the company developing the system.

We have described the probability model and profit function central to this work. The final component is the decision strategy which describes the structure of the decision making mechanism in the context of when the decision is taken and how much information is included in the decision making process. In this section we describe the single and one stage look-ahead plans and apply each to the data described earlier.

B. Single Stage Decision Method

A piece of software is to undergo user acceptance testing for time T with the decision taken before testing commences. When issues are encountered the associated defect repairs and change requests are implemented immediately. After testing for T units of time the software is then accepted. Once accepted the program enters the service phase where further defects and change requests are discovered, each one contributing to a reduction in the utility of the program to the customer company. This is the case of single stage testing and the goal for the procuring company is to pick an optimal value for T.

Given knowledge of $\bar{N}_D(T)$, $N_{CR}(T)$ and $\bar{N}_{CR}(T)$ for all T, it would be a simple matter to determine which time maximizes the utility of Equation 11. However, at the time that we must decide T, they are not known, and so we must look to the expected utility, $E[\mathcal{U}(T, \bar{N}_D(T), N_{CR}(T), \bar{N}_{CR}(T))]$ where the expected value is with respect to the prior parameters alone. This can be expanded as follows:

$$A - fT - C_D E[\bar{N}_D(T)] - C_{CR1} E[N_{CR}(T)] - C_{CR2} E[\bar{N}_{CR}(T)]$$

$$= A - fT - C_D \frac{\tau\omega}{\lambda(\omega+\rho)} \left(\frac{\mu}{\mu+T}\right)^{\alpha}$$

$$- C_{CR1} \frac{\tau\rho}{\lambda(\omega+\rho)} \left[1 - \left(\frac{\mu}{\mu+T}\right)^{\alpha}\right] - C_{CR2} \frac{\tau\rho}{\lambda(\omega+\rho)} \left(\frac{\mu}{\mu+T}\right)^{\alpha}.$$

The value of T that maximizes this expression is

$$T^* = \mu \left[\left(\frac{\alpha\tau(\omega C_D + \rho(C_{CR2} - C_{CR1}))}{\mu\lambda f(\omega+\rho)} \right)^{\frac{1}{\alpha+1}} - 1 \right]. \quad (12)$$

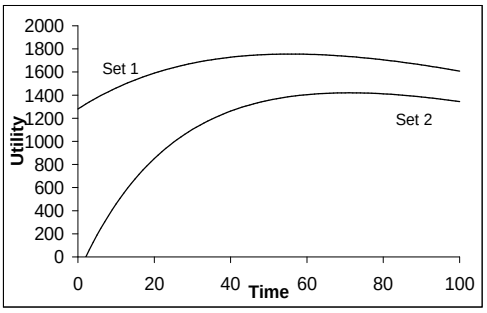

Fig. 6. Expected utility for the parameter values elicited in Section IV.A

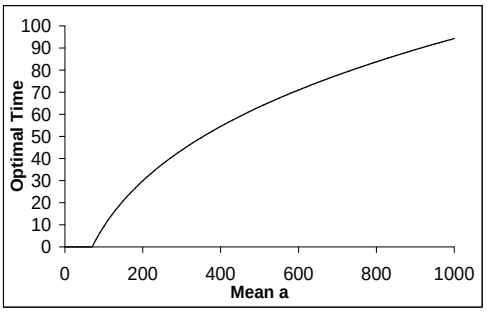

Fig. 7. Sensitivity of single stage optimal time to changes in mean of $a = \frac{\tau}{\lambda}$

There are some intuitive properties of T^*. First, it is a function of the distribution of a through its mean, $\frac{\tau}{\lambda}$, and increases as the prior mean for a increases. It also increases as a function of the difference in the costs of implementing change requests before and after release $C_{CR2} - C_{CR1}$, and decreases as the penalty cost and cost of testing f increases. Finally, the value of T^* exceeds 0 when

$$\frac{\alpha\tau(\omega C_D + \rho(C_{CR2} - C_{CR1}))}{\mu\lambda(\omega + \rho)} > f. \tag{13}$$

In Figure 6 we plot the utility function for the utility parameters and both sets of prior parameters specified in Section IV.A. The optimal stopping times for Set 1 and Set 2 are 56 and 71 days respectively.

In practice it is imperative to consider the optimal choice of release time when the values of the parameters differ from those initially selected. Since T^* takes a simple closed form, the sensitivity of the solution to all prior and utility parameters is straightforward. Figure 7 shows a plot of the optimal release time for varying values of the prior parameters for the rate a. The shape of the graph indicates that there is a cutoff rate below which it is disadvantageous to test and that the optimal time to test approaches 0 as a approaching infinity.

C. Multi Stage Decision Method

Although relatively easy to solve, the single stage plan is flawed, as the determination of T^* relies on prior information alone and there is no opportunity to modify the testing time in the light of the test data. The ideal decision strategy, as given by the *Sequential Plan*, would allow an unlimited number of stages of testing, and would base the decision on whether to release or test for a further stage on all the information available up to that point. This means that the derivation of the expected utility at a stage is done with respect to the posterior distribution given the data up to that point rather than the prior.

However, there are computational problems with such a test. Allowing an unlimited sequence of stages means that one cannot solve the decision problem, which requires one to solve recursively from the last testing stage. Even if one were to restrict testing to a reasonable finite number of stages, the optimization is computationally extremely demanding, involving a nested sequence of maximization and expectations. That said, [5] and [26], have developed sequential solutions for a simplified version of the Goel-Okumoto model.

In practice it is not necessary to include an unlimited sequence of stages as management will have a predefined set of points at which the continue/stop testing decision will be taken. In practice the times may well represent the end of each calendar week during user acceptance testing.

D. One Stage Look-Ahead Plan

The application of the sequential strategy using established reliability growth models has not, even for moderate J, been possible to date. Instead, work ([20] and [29]) has concentrated on the *One Stage Look-ahead Plan* which compares the utility of stopping the testing activity immediately with the utility of stopping at the next stage in the future. This model has the drawback that, when making a decision, the only choices presented are to accept the system immediately or to continue testing to the next time point and then accept. The possibility of testing beyond the next time point is ignored. Work in [18] has shown that the sequential decision strategy cannot decide to accept before the one stage look-ahead strategy. Work by the author under review suggests that the use of the one stage look-ahead plan gives almost identical results to the sequential test except where the prior expert information is exact but highly inaccurate.

At the completion of each stage, the decision to accept or continue testing is taken by comparing the expected utility of accepting the software immediately with the expected utility of testing to the next stage and then accepting the program. The evaluation of each of these utilities is based on all the information available to date, including the prior expert information and the occurrence time for defects and change requests. It is calculated by finding the expected values of each of the terms on the right hand side of Equation 11 using the expression developed in Equation 8.

We now demonstrate the application of the strategy using the utility and prior data sets specified earlier. We

Decision Time	Utility Now	Utility Next	Decision
0	1280	1378	Test
5	1158	1233	Test
10	1334	1405	Test
15	1478	1543	Test
20	1632	1674	Test
25	1629	1661	Test
30	1727	1747	Test
35	1646	1670	Test
40	1675	1691	Test
45	1721	1730	Test
50	1750	1752	Test
55	1778	1775	Stop

TABLE III
One Stage Look-ahead Decision Plan for Set 1 parameters

Decision Time	Utility Now	Utility Next	Decision
0	-150	188	Test
5	125	201	Test
10	432	512	Test
15	607	684	Test
20	894	944	Test
25	932	972	Test
30	1055	1085	Test
35	698	740	Test
40	802	836	Test
45	1022	1046	Test
50	1099	1116	Test
55	1336	1344	Test
60	1294	1301	Test
65	1456	1455	Stop

TABLE IV
One Stage Look-ahead Decision Plan for Set 2 parameters

select decision time points at $0, 5, \ldots, 100$. Table III shows the utility values calculated at each time point for the usual utility parameters values and Set 1 of the prior parameter values. The table shows that testing continues until $T = 55$ at which point the decision is to terminate the user acceptance testing phase as the utility of accepting the system now (1778) exceeds the utility of testing until time 60 and then accepting the system (1775). As expected, the acceptance time (55) matches the single stage estimate (56) based on accurate prior information. For Set 2 table IV shows that the decision to stop testing takes place at time $T = 65$. This is earlier than the corresponding single stage estimate of $T = 71$ which illustrates that the occurrence data (both defect and change request) serves to correct the inaccuracy of the prior knowledge.

To this point Section IV has concentrated on the customer and their goals in deciding when to terminate the user acceptance testing phase. There are of course situations where the producing firm will determine the length of the user acceptance testing period. The appropriate utility function for them will differ from the one described in that it will include a term to cover the cost of repairing defects during and after release. It may well also include terms (contributing positively to the utility) for the profit associated with implementing change requests before and after the user acceptance testing phase stops.

V. Conclusion

This work develops a relatively simple model for the occurrence of change requests and defects during the user acceptance testing phase for a bespoke piece of software. This can be of benefit to development firms in their efforts to forecast the required resources to support the repair of defects and the implementation of change requests. We illustrate the issues with prediction using the model and develop a Bayesian model to overcome these problems and at the same time allow for the incorporation of expert opinion. Similarly, the split nonhomogeneous Poisson process model can also be of benefit to the customer organization who can use it to predict the total cost and to decide how long to conduct user acceptance testing before accepting the product for operation. The work shows how this can be done through a decision-theoretic solution using both a single stage and a multiple stage decision plan.

VI. Acknowledgements

K. McDaid acknowledges the support of Dundalk Institute of Technology through its secondment programme.

References

[1] B. Hetzel. *The Complete Guide to Software Testing (second edition)*, Wiley, 1988.
[2] G.J. Myers. *Software Reliability, Principles and Practices*, Wiley, 1976.
[3] J.D. Musa. *Software Reliability Engineering*, McGraw-Hill, New York, 1998.
[4] T.A. Chavez. *Decision-Analytic Stopping Rule for Validation of Commercial Software Systems*. IEEE Transactions on Software Engineering, 26: 907-918, 2000.
[5] S.R. Dalal and C.L. Mallows. *When should one stop testing?* Journal of the American Statistical Association, 83: 872-879, 1988.
[6] S.A. Hossain and R.C. Dahiya. *Estimating the parameters of a non-homogeneous Poisson process for software reliability*. IEEE Transactions on Reliability, 42(4):604-612, 1993.
[7] L. Yin and K.S. Trivedi. *Confidence Interval Estimation of NHPP-Based Software Reliability Models*. Proceedings of the International Symposium on Software Reliability Engineering, 1999.
[8] M. Xie, G.Y. Hong and C. Wohlin. *A Practical Method for the Estimation of Software Reliability Growth in the Early Stage of Testing*. Proceedings of the International Symposium on Software Reliability Engineering, 1997.
[9] H. Hoffmann and K. McDaid *An Analysis of Failure Occurrence Patterns during Testing and Operation* Supplemental Proceedings of the International Symposium on Software Reliability Engineering, 2004.
[10] H. Hoffmann and K. McDaid *Using Software Reliability Growth Models to Predict the Occurrence of Defects during Testing* Proceedings of the International Conference on Software Development, Iceland, May 2005.
[11] S. Gokhale and K. S. Trivedi. *A Time/Structure Based Software Reliability Model*. Annals of Software Engineering , vol. 8, pp. 85-121, 1999.
[12] N.D. Singpurwalla and S.P. Wilson. *Statistical Methods in Software Engineering*. Springer, 1999.

[13] N.D. Singpurwalla and S.P. Wilson. *Software Reliability Modeling*. International Statistical Review, 62 3:289-317, 1994.
[14] A.L. Goel and K. Okumoto. *Time dependent error-detection rate models for Software Reliability and Other performance Measures*. IEEE Transactions on Software Engineering, R-29, pp206-211, 1979.
[15] S. Yamada, M. Ohba and S. Osaki. *S-shaped Reliability Growth Modeling for Software Error Detection*. IEEE Trans. on Reliability, R-32, pp. 475-485, 1983.
[16] S. Gokhale. *Analysis of Software Reliability and Performance*, Ph.D. thesis, Duke University, 1998.
[17] G. Knafl and J. Morgan. *Solving ML equations for 2-parameter poisson-process models for ungrouped software failure data*. IEEE Transactions on Reliability, 45(1):42-53, 1996.
[18] K. McDaid and S. Wilson. *Deciding how long to test software*. Journal of the Royal Statistical Society, Series D, R-50. part 2: 117-134, 2001.
[19] L. Kuo and T.Y. Yang. *Bayesian Computation for Nonhomogeneous Poisson Processes in Software Reliability*. Journal of American Statistical Association, 91 434: 763-77, 1995.
[20] J.R. van Dorp, T.A. Mazzuchi and R. Soyer. *Sequential inference and decision making for single mission systems development*. Journal of Statistical Planning and Inference, 64: 289-317, 1998.
[21] N.D. Singpurwalla. *Determining an optimal time interval for testing and debugging software*. IEEE Transactions on Software Engineering, 17: 313–319, 1991.
[22] K. Okumoto and and A.L. Goel. *Optimum release time for software systems, based on reliability and cost criteria*. Journal Systems and Software, 1: 315–318, 1980.
[23] S.R. Dalal and C.L. Mallows. *Some graphical aids for deciding when to stop testing software*. Journal on Selected Areas in Communications, 8: 169–175, 1990.
[24] E.H. Forman and N.D. Singpurwalla. *An empirical stopping rule for debugging and testing computer software*. Journal of the American Statistical Association, 72: 750–757, 1977.
[25] P.J. Boland and H. Singh. *Determining the Optimal Release Time for Software Reliability in the Geometric Poisson Model*. International Jounral of Reliability, Quality and Safety Engineering, 9, 3, pp 201-213, 2002.
[26] K. McDaid. *How Long to Test Software*. PhD dissertation, Trinity College Dublin, 1998.
[27] P. Randolph and M. Sahinoglu. *A Stopping Rule for a Compound Poisson Random Variable*, Applied Stochastic Models and Data Analysis., Vol. 11, 135-143, 1995
[28] K. McDaid. *When to suffer the penalties for late delivery of software*. Proceedings of the European Forum on Software Measurement, 2004.
[29] S. Zacks. *Sequential procedures in software testing*, In Recent advances in life-testing and reliability, CRC Press, 1995.