A Software Release Criteria

Ing.Pedro E. Colla
Motorola Argentina Center for Software (MACS) – Hipolito Yrigoyen 146 – Piso 9
X5000JHO – Córdoba – Argentina – E-mail: colla@motorola.com

Abstract

The purpose of the development is to create a product which meets the customer requirements as close as possible; a test phase is aimed to identify and remove both functional and requirements problems (dubbed as "faults") remaining after the development and preventing them to become a defect during the software operational lifetime. A significant body of theoretical work and empirical validation has been done to obtain models that provides a reasonable prediction of the software reliability. Practical issues are found to define criteria to perform the test planning and monitoring during the software development in order to manage the quality levels the software will show once its released into production. This paper briefly walkthrough the background of a comprehensive set of well known reliability models and then proposes from them a practical methodology to be used as a release criteria with planning and monitoring purposes. A simple project is used to gather metrics used to validate the concept and to identify areas which will require further work to be performed.

Overview

How to validate that a given piece of software functions correctly? How to manage its development till the point the software would comply with a given quality target? How to plan for it allowing a fair chunk of the schedule?

Musa et.al [3] has pointed out that the traditional answer has been that you perform a test process on the software which is composed by the exhaustive testing of its components against a given set of requirements.

But this answer will still left unanswered the question of how much test is actually needed and for how long on what amount of test and for how long it is needed, both factors usually involve a fair amount of guesswork.

More often than not the answer is contaminated by pragmatic decisions based on the committed calendar of the project rather than an objective evaluation of the quality of the software being tested.

Its often been said that since the schedule doesn't allow for the testing anyway you test until your calendar allows you to and that's it. Even if this criteria, somehow, is accepted as reasonable the need for a model still is in place to know what is being accepted in terms of expected future defects because of doing so.

A relatively large body of theoretically sound and experimentally validated statistical models *[1,2,3,6]* has been made available by different authors over time that enables the planning and monitoring of the testing processes until a given future defect profile is achieved that satisfies both the business as well as the engineering requirements of the project.

A decision criteria based on technical facts should allow both the planning and monitoring of the test stages to be performed in a much more manageable way.

Software Reliability Model

Any situation where a program fails to meet the user requirements must be considered a defect whereas a fault is the actual problem that produces the defect[1].

Both are obviously correlated and a software reliability model could be developed in order to predict the future total *number of defects* and the *intensity* of them (how often they will appear over time)

Errors are introduced on a software piece and exposed into a defect by the actual usage of the software.

Since errors are exposed as defects as the software is executed its reasonable to assume the remaining number of errors will decrease over time as fixes are introduced and the test time increases.

Software Reliability models will then express time not in terms of calendar but in actual continuous *test execution time* (τ) which takes into consideration the discontinuity introduced on the test process by the error fixing activities.

In terms of the execution time the elapsed between successive failures will increase as fixes are introduced reflecting the increase of the *Mean Time to Fail (MTTF)* of the software being tested.

Musa *[4]* introduced one of the first comprehensive models addressing the prediction of the software reliability, this model has been taken forward and refined by scores of authors into sophisticated representations of real world implementations, including most of the real world border effects not accounted for on the early models.

[1] Failure and Defect are usually referred in the bibliography as the same concept, the same for fault and error.

However, since most of the actual measurements required could be made with only marginal accuracy *[1]* on most software development, it is still very useful to realize qualitative results and quite a few basic quantitative ones without stepping into the intricacies of the model corrections.

The following basic conditions are assumed to be met:

- Tests are representative of the environment in which the program will be used.
- All failures are observed or the proportion of unobserved failures can be estimated.
- Failure intervals are independent of each other.
- The execution time between failures are piecewise exponentially distributed.
- The hazard rate is proportional to the expected value of the number of faults remaining.
- The fault correction occurrence rate is proportional to the failure occurrence rate (both rates with respect to exec time).

As defects are identified fixes are introduced on the software or the observation of the defect is disregarded on successive test runs.

Assuming that fixes does not introduces new faults into the code the *defect profile* (μ) experienced would be given by the following expression:

$$\mu(\tau) = \mu_0 (1 - e^{-(\frac{\lambda_0}{\mu_0}).\tau}) \text{ [Eq. 1]}$$

Where

μ_0 Total number of faults.

λ_0 Failure intensity at origin.

The *defect intensity* (λ) could be expressed as the derivate of the defect profile; Performing some substitutions in [Eq.1]

$$\xi(\tau) = -(\frac{\lambda_0}{\mu_0}).\tau \qquad \textit{[Eq. 2]}$$

$$\mu(\tau) = \mu_0 - \mu_0 e^{\xi(\tau)} \qquad \textit{[Eq. 3]}$$

Then

$$\frac{d\mu}{d\tau} = \lambda(\tau) = -\mu_0 e^{\xi}\frac{d\xi}{d\tau} \qquad \textit{[Eq. 4]}$$

and

$$\frac{d\xi}{d\tau} = -(\frac{\lambda_0}{\mu_0}) \qquad \textit{[Eq. 5]}$$

Therefore

$$\lambda(\tau) = \lambda_0 e^{-(\frac{\lambda_0}{\mu_0})\tau} \qquad \textit{[Eq. 6]}$$

This result represents some interest since the *defect intensity* ($\lambda(\tau)$) could readily be collected as a metric during the testing stage.

The *Reliability Model* would then allow the prediction of the software defect behavior based on the initial (static) values of two parameters $\{(\lambda_0, \mu_0)\}$ of the model that would define as a sort of signature the failure pattern over time of that software.

If both values could be assessed either during planning or at very early stages of the test process then the model could be used to both estimate the length of the testing process required to achieve a given goal and to monitor the actual test progress.

Metrics Collection

During the test process it might be relatively easy to collect up to n metrics related to the *defect intensity* (λ) as the instantaneous failure rate obtained when the test is performed as a function of the *execution test time* (τ).

$$\gamma(\tau) = \{(\tau_1, \lambda_1),..,(\tau_n, \lambda_n)\} \qquad \textit{[Eq. 7]}$$

The execution test time is the equivalent continuous net test time.

Complex models *[1,2,3,6]* has been developed to correlate execution test time with actual calendar time; however, a simple proportional correlation could be assumed with reasonably low distortions on the final results just taking net test sessions of equivalent effort investment (i.e. test days, or test hours).

Brettschneider Method

Brettschneider [5] developed a simple yet very powerful testing forecast and monitoring criteria based on the previous model and the data collected during the test.

The method aims to define whether or not the software has been tested long enough to be good for release; in other words if the quality goals have been met.

The key indicator is the *time since the last fail* (τ_z) measured in terms of test execution time.

The release maturity has been achieved when a given test execution time elapsed is achieved without experiencing any failure, or in other words, when the MTTF of the software has reached a given predefined value.

Since the MTTF is a function of the remaining errors in the software its an indirect way to measure that number had reach a given target value.

Given a target *defect density (ε)* the number of *target defects* (μ_ε) could be obtained from the *software size (L)* with:

$$\mu_\varepsilon = \varepsilon L \qquad \textit{[Eq. 8]}$$

also the total *number of defects (μ_c)* found so far would be:

$$\mu_c = \sum_{i=1}^{n} \lambda_i \qquad \textit{[Eq. 9]}$$

where λ_i are the individual defect intensities collected (see Metrics Collectionon page 3 , Eq 7. for further details), therefore applying the *BrettSchneider* [5] formula:

$$\tau_z = \tau_x \frac{Ln(\frac{\mu_\varepsilon}{\mu_\varepsilon + 0.5})}{Ln(\frac{\mu_e + 0.5}{\mu_e + \mu_c})} \qquad \textit{[Eq. 10]}$$

Where:

τ_x Test time to last problem

τ_c Test time at completion.

τ_z Target test time without errors

Which are related by:

$$\tau_z = \tau_c - \tau_x \qquad \text{[Eq. 11]}$$

In other words, at least a *zero problem test execution time (τ_z)* has to elapse for the MTTF of the software to be at the target levels which in turn reflects the number of remaining errors have achieved a given level.

This criteria could be used both to plan ahead of the test time requirement and to monitor the actual test progress in order to define when the target quality has been met (*Stop Criteria*).

Simple Defect Forecasting

We might also try to find which value pairs of $\{\lambda_0, \mu_0\}$ on the failure intensity equation produces a curve which fits the observed data within reasonable error margins.

In order to fit the observed data the equation is reformulated as [Eq.6]:

$$Ln(\lambda(\tau)) = Ln(\lambda_0) - (\frac{\lambda_0}{\mu_0}).\tau.Ln(e) \qquad \textit{[Eq. 12]}$$

Then some substitutions are performed:

$$Y(\tau) = Ln(\lambda(\tau)) \qquad \textit{[Eq. 13]}$$

$$\alpha = Ln(\lambda_0) \qquad \textit{[Eq. 14]}$$

$$\beta = -(\frac{\lambda_0}{\mu_0}) \qquad \textit{[Eq. 15]}$$

Then the equation [Eq.12] could be assimilated to the linear form:

$$Y(\tau) = \alpha + \beta.\tau \qquad \textit{[Eq. 16]}$$

Where the *slope (β)* and the *intercept (α)* could be computed using the linear regression least squares method.

$$\beta = \frac{\left[n\left(\sum \tau_i Y_i\right) - \left(\sum \tau_i\right)\left(\sum Y_i\right) \right]}{\left[\left(n \sum \tau_i^2\right) - \left(\sum \tau_i\right)^2 \right]} \qquad \textit{[Eq. 17]}$$

and

$$\alpha = \overline{Y} - \beta\tau \qquad \textit{[Eq. 18]}$$

Once suitable values are found for the *slope (β)* and *intercept (α)* values then the *correlation (ρ)* of the linear model obtained must be verified as to satisfy some reasonable correlation criteria, in the proposed method the following one were chosen as an initial reasonable value subject to further experimentation:

$$\rho \geq 0.75 \quad \textit{[Eq.19] (2)}$$

Finally from [Eq.14 & 15] the Reliability Model parameters could then be obtained as:

$$\lambda_0 = e^{\alpha} \qquad \textit{[Eq. 19]}$$

$$\mu_0 = -\left(\frac{\lambda_0}{\beta}\right) \qquad \textit{[Eq. 20]}$$

Planning and Monitoring

As with the *Brettschneider* method we start with the expected target *defect density* (ε) which could be transformed into the target number of defects for the entire software under test using [8]:

$$\mu_\varepsilon = \varepsilon L \qquad \textit{[Eq. 21]}$$

There is substantial evidence pointing to the fact that no process is perfect at removing faults, and in fact there is a very strong (albeit not surprising) correlation between the organizational process maturity and the *defect removal efficiency* (η) during the different development phases.

Table 1 Defect Removal Efficiency vs. SEI CMM Level

SEI CMM Level	Defect Removal Efficiency (%)
1	85%
2	89%
3	91%
4	93%
5	95%

then the minimum released defects starting from a given initial defect level would be:

$$\mu_{\min} = (1-\eta)\mu_0 \qquad \textit{[Eq. 22]}$$

Being the *defect removal efficiency* a factor depending on the testing process, organizational maturity and test tools being used; then the following relation must be verified:

$$\mu_{\min} \leq \mu_\varepsilon \qquad \textit{[Eq. 23]}$$

On any given moment of the test the *remaining number of errors* (δ) could be expressed as:

$$\delta(\tau) = \mu_0 - \mu(\tau) \qquad \textit{[Eq. 24]}$$

Replacing terms

$$\delta(\tau) = \mu_0 - \mu_0 (1 - e^{-(\frac{\lambda_0}{\mu_0}).\tau}) \qquad \text{[Eq. 25]}$$

Then

$$\delta(\tau) = \mu_0 . e^{-(\frac{\lambda_0}{\mu_0}).\tau} \qquad \textit{[Eq. 26]}$$

For planning and monitoring purposes it would be very interesting to know how long the test should last, this condition will occur when the following condition is met:

2 This limit is adopted as a judgement balance between how harshly we'll inspect the collected metrics before them to declare them usable for this criteria purposes and how reasonable is the convergence of the data collected into a model of reasonable accuracy for most practical purposes.

$$\delta(\tau_\varepsilon) = \mu_\varepsilon \quad \text{[Eq. 27]}$$

So, replacing terms

$$\mu_\varepsilon = \mu_0 e^{-(\frac{\lambda_0}{\mu_0})\tau_\varepsilon} \quad \text{[Eq. 28]}$$

$$(\frac{\mu_\varepsilon}{\mu_0}) = e^{-(\frac{\lambda_0}{\mu_0})\tau_\varepsilon} \quad \text{[Eq. 29]}$$

$$Ln(\frac{\mu_\varepsilon}{\mu_0}) = -(\frac{\lambda_0}{\mu_0})\tau_\varepsilon Ln(e) \quad \text{[Eq. 30]}$$

Then the test time to reach the target number of defects might be expressed as:

$$\tau_\varepsilon = -(\frac{\mu_0}{\lambda_0})Ln(\frac{\mu_\varepsilon}{\mu_0}) \quad \text{[Eq. 31]}$$

The above relation could then be used for both planning and monitoring purposes in order to forecast the test process.

Given the actual test time (τ_x) the remaining test time (τ_r) would then be:

$$\tau_r = \tau_\varepsilon - \tau_x \quad \text{[Eq. 32]}$$

This relation could then be used to monitor progress during the test as well.

Preliminary Verification

A very small project was used to preliminary verify the suitability of the process, the methodology to collect the relevant metrics and the practical application of the forecast method.

The *Champaqui project*[3] was made as part of a training curse and implemented a very simple test calculator tool.

The total *number of non-commented lines of code (L)* were

L=720 LOCS *[Eq. 33]*

During the testing of the software the following profile of defect intensity as a function of the normalized test execution time in days was collected:

Table 2 Testing Metrics Collected

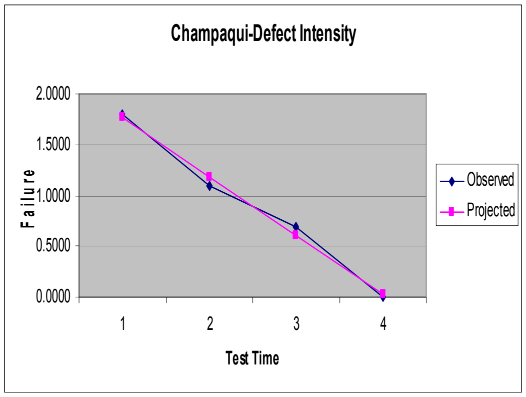
τ	$\lambda(\tau)$	$Ln(\lambda(\tau))$
1	6	1.79
2	3	1.09
3	2	0.69
4	1	0.00

Applying a linear regression [Eq.14 & 15]

$$\alpha = -0.5781 \quad \text{[Eq. 34]}$$

$$\beta = +2.3411 \quad \text{[Eq. 35]}$$

Yields a linear regression model depicted in the following figure:



[3] The Champaqui is the highest mount (2790 Mts) in the Cordoba state of Argentina where the software center is located; project names are often picked out of geo landmarks.

Figure 1 – Defect Intensity

And then [Eq.20 & 21]

$\lambda_0 = 10.39$ *[Eq. 36]*

$\mu_0 = 17.97 \cong 18$ defects *[Eq. 37]*

The defect profile (μ) could be seen in the following graph:

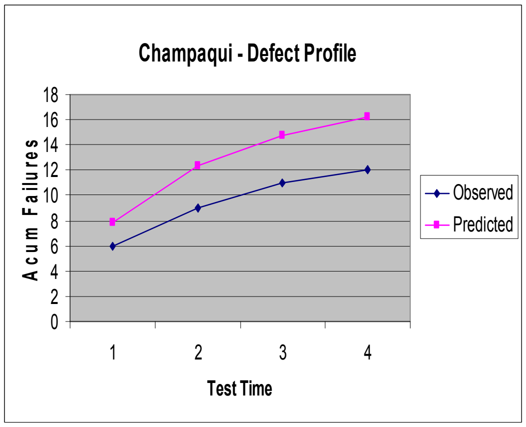


Figure 2 Defect Profile

Given that number of initial defects and assuming a high detection removal efficiency ($\eta \leq 0.95$ using an optimistic picking of the value) it yields

$\mu_{\min} \geq 18 * (1-0.95) \geq 0.9$ defects [Eq. 38]

Assuming the target would be μ_e=1 then the border condition [Eq.24]

$\mu_{\min \leq} \mu_{\varepsilon}$ *[Eq. 39]*

is met.

The remaining defects yet to be detected according to the model would be [Eq.27]

$\delta(4) = 1.78 \approx 2$ defects *[Eq. 40]*

and the required test time to achieve the goal would be [Eq.32]

$$\tau_{\varepsilon} = -(\frac{17.97}{10.39})Ln(\frac{1}{17.97})$$ *[Eq. 41]*

$\tau_{\varepsilon} = 4.99 \cong 5$ test execution days *[Eq. 42]*

Another full day of testing would be required to achieve the stated goal of remaining defects at release.

As a monitoring tool the above values could have been computed as the testing phase wnt, if so the results would has been:

Table 3 Projected Total Execution Time

τ	$\lambda(\tau)$	λ_0	μ_0	τ_ε
1	6	-*-	-*-	-*-
2	3	12.00	17.31	4.11
3	2	9.90	18.03	5.26
4	1	10.39	17.97	4.99

The results converge quite fast into stable (final) values as measured during the testing process itself, so its reasonably preliminary to consider using it as a monitoring tool.

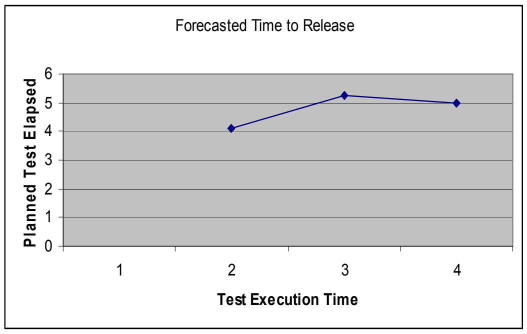


Figure 3 Forecasted Time To Release

Further Work

Further work to refine the proposed model is needed in the following areas:

- Collection of a set of metrics and establish a validation baseline for the method.
- Execution to Calendar Time model refinements.
- Test Compression validation under different project life cycle models.
- Quality of Fixes, model and baseline for defect removal behavior.
- Correlation between forecast and test progress.

References

[1] A critique of Software Defect Prediction Models – Norman E. Fenton, Martin Neil- IEEE Transactions on Software Engineering V25 N5 Set/Oct (1999)
[2] How Software Process Improvements helped Motorola – M.Diaz, J.Sligo – IEEE Software V14 N5 1997
[3] Quantifying Software Validation: When to Stop Testing? – J.D.Musa, A.F.Ackerman IEEE Software May 1989
[4] Software Reliability: Measurement, Prediction, Application – J.D.Musa, A.Iannino, K.Okumoto- McGraw-Hill NY 1987
[5] Simplified Method for Deciding if Software Quality Is Good Enough for Release - Ralph Brettschneider. Internal Motorola Paper 1998.
[6] Techniques for Modeling the Reliability of Fault-Tolerant Systems with the Markov State-Space Approach – R.W.Butler, S.C.Johnson – Langley Research Center – Hampton Virginia – NASA Reference Publication 1348.
[7] A Software Reliability Engineering practice- J.D.Musa, W.W.Everett, Naval Postgraduate school Code, March 1993.

About the Author

Pedro E. Colla *Graduated as MEE at the University of Buenos Aires in 1981, pursued BCS postgraduate studies in 1991 and earned a MBA degree in 1997. He has extensive background in the IT industry at both technical and management responsibilities.. Now he works at Motorola GSG – Argentina Center as Development Operations Manager and Chief Architect.*

Address:
Motorola GSG - Argentina
Hipólito Yrigoyen 146 – piso 9
X5000JHO – Córdoba
ARGENTINA
E-mail: colla@motorola.com