

FY2001 UNIVERSITY SOFTWARE INITIATIVE PROPOSAL
for the
NASA SOFTWARE IV&V FACILITY

Initiative Title: *Integrating Formal Methods and Testing in a Quantitative Software Reliability Assessment Framework*

WHITE PAPER

Bayesian Framework for Reliability Assurance of Safety Critical Systems

Bojan Cukic

NASA/WVU Software Research Laboratory
Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26506-6109
E-mail: cukic@csee.wvu.edu

Abstract

The existence of software faults in safety-critical systems is not tolerable. Goals of software reliability assessment are estimating the failure probability of the program, θ , and gaining statistical confidence that θ is realistic. While in most cases reliability assessment is performed prior to the deployment of the system, there are circumstances when reliability assessment is needed in the process of (re)evaluation of the fielded (deployed) system. Post deployment reliability assessment provides reassurance that the expected dependability characteristics of the system have been achieved. It may be used as a basis of the recommendation for maintenance and further improvement, or the recommendation to discontinue the use of the system.

The white paper presents a statistical framework which allows the incorporation of rigorous verification and validation activities into the reliability assessment. These verification and validation activities may include informal as well as the formal methods in software engineering. The net effect of combining various V&V techniques with random testing is a drastic reduction (several orders of magnitude) in the number of required test cases for reliability assessment. The framework also permits post deployment reliability assessment, thus providing reassurance that the expected dependability characteristics of the system have been met. NASA's Day-of-Launch I-Load Update (DOLILU II) system is used as a case study.

1. Introduction

Quality assurance of software is a notoriously difficult problem. Due to the complexity of software systems and to the issue of scalability, most techniques that are sufficiently precise to ensure quality are too cumbersome to scale up. Therefore, the use of software in safety-critical systems remains a somewhat paradoxical issue. The properties of software that can be exploited to improve the control and safety functions in critical applications, are exactly the qualities that lead to our concern about the reliability of the systems that are being produced.

In this study, we present practical problems and challenges encountered in an experiment to assess software reliability of NASA's *Day-of-Launch I-Load Update (DOLILU II)* system. The Day-of-Launch I-Load Update system for the Space Shuttle program has been developed to allow modification of the Shuttle's first stage guidance commands based on actual wind conditions measured in hours preceding the launch. This system consists of the trajectory validation software required to generate and verify the new I-Loads, to evaluate wind and trajectory

conditions, and to recommend decisions to fly (or not to fly) with the new I-Loads. From this short description, it is apparent that DOLILU II is a high consequence system, i.e., a very high cost is associated with eventual occurrence of a failure.

Interestingly, this system has been in use for several years and no failures have been observed. Reliability assessment of DOLILU II has been prompted by the increasing organizational pressures to provide meaningful quantification of software assessment process. DOLILU system has undergone independent software verification and validation (IV&V) process [3] throughout its development life-cycle. However, the results of IV&V are qualitative in nature, and cannot be easily quantified. Therefore, one of the goals of our study is to assess the feasibility of software reliability measurement for mission critical systems.

Software reliability is a quantitative measure of software quality. It is defined as a probability of failure free execution given a specific environment and a fixed time interval. The goal of software reliability assessment is not just to estimate the *failure probability* of the program, θ , but to gain *statistical confidence* that θ is realistic. In practice, the required *failure probability* θ_0 and the *confidence level* C are application specific and predefined. In most life critical applications, one has to establish that software reliability is indeed high. Butler and Finelli [2] classify software systems with respect to their reliability into three types. These are

- Ultrahigh reliability: $< 10^{-7}$ failures/hour,
- Moderate reliability: $10^{-3} - 10^{-7}$ failures/hour,
- Low reliability: $> 10^{-3}$ failures/hour.

Unlike hardware, where reliability is associated with physical faults and aging of materials, the leading cause of software unreliability are design faults introduced throughout the development and maintenance life cycle, starting with the requirements documents. In the reliability certification phase, we consider software system as an entity, subjecting it to inputs and observing the outputs. In each execution, the system either produces a correct or an incorrect result. This may be viewed as a stochastic process, where software executions result in failures in a stochastic manner [2].

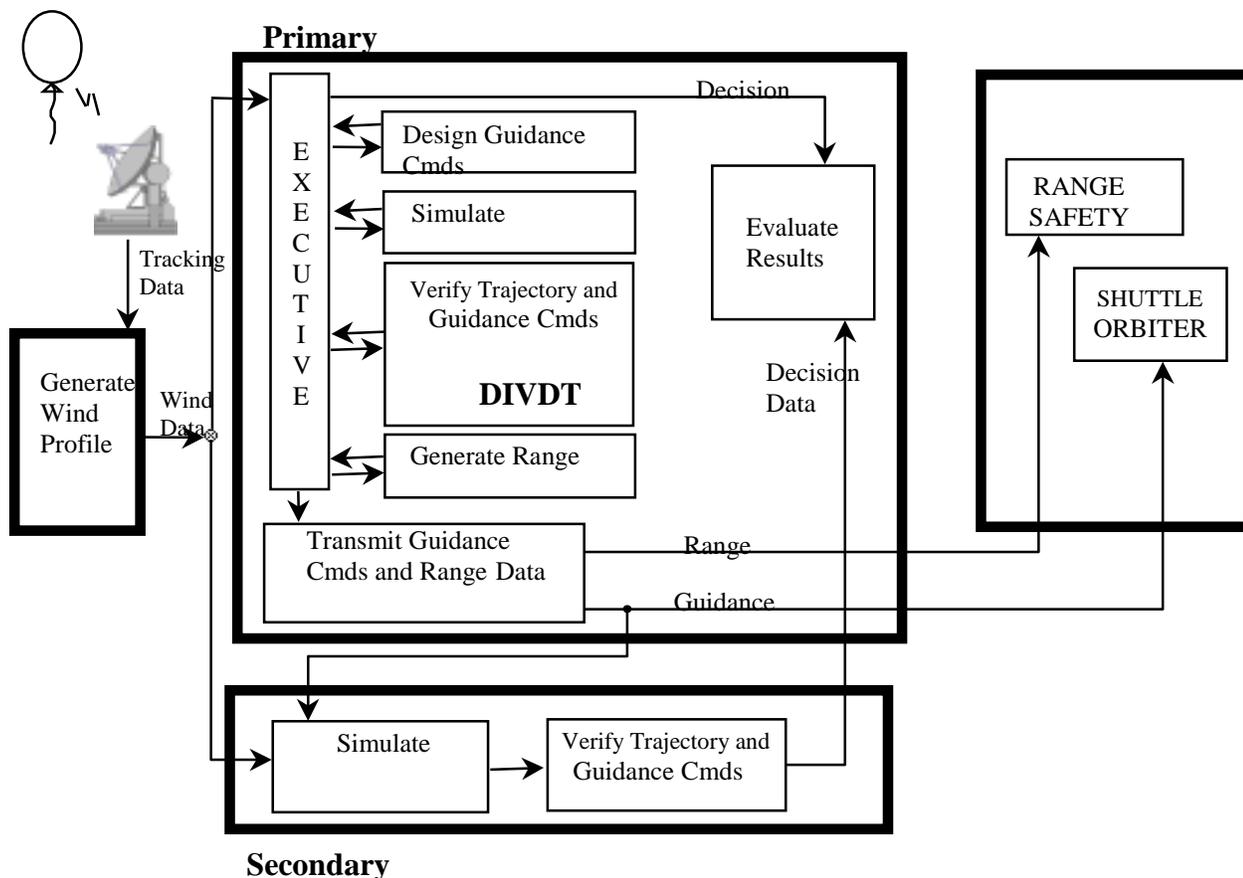
In principle, software reliability can be quantified through program verification or statistical testing. The requirements specification for DOLILU system is written in English and no attempt has been made to formalize it with any form of mathematical notation. The size and the complexity of the specification documents make formal program verification virtually impossible at this point in time. Therefore, the reliability assessment of DOLILU system is to be obtained from program testing.

Traditionally, software creation involves a development life-cycle, where the software is created, then sufficiently tested. When a failure is observed, the fault causing it is fixed and the software is tested again. This forms the basis for Reliability Growth Models (RGM). The goal for these models is to fit mathematical functions through the inter-failure time observations and predict the post-testing reliability of the final version of the software. There are numerous RGMs, the reader may refer to [11] for a concise description of some of them¹.

There are several problems in potential application of reliability growth models in the assessment of DOLILU II. When considering RGMs, the mean time between failures needs to increase and, consequently, the failure probability decreases as more and more faults are fixed. However, one should ask what should be an acceptable time frame before the model can achieve ultrahigh reliability? Keiller and Miller [5] observe that the necessary time is prohibitively long. Another problem arises when the software itself doesn't exhibit failures in system level (acceptance) testing. This is not rare for safety-critical software, and, indeed, is the case with testing of DOLILU II. How could one fit an interfailure time distribution with no observed failures during acceptance testing? Therefore, instead of RGMs the applicable models of software reliability assessment are input domain models, such as the sampling model [1]. Repeated failure-free execution of the program provides a certain level of confidence that the required reliability level has been successfully achieved. The practicality of the sampling model has been questioned, due to the very large number of required tests. Nevertheless, in the case of DOLILU II, we demonstrate that test requirements are realistically achievable.

Interestingly, prior to our reliability assessment effort, DOLILU II system has been in operational use by NASA for several years. This fact distinguishes our study from most other reported case studies in software reliability engineering. Rigorous quality assurance procedures, described in [12], had been performed prior to the deployment of DOLILU. These include fault based testing, stress testing, and several other white-box testing techniques. No

¹ Detection of a failure and identification of its cause, the fault, is usually far from trivial. Most reliability growth models assume that faults are successfully identified and removed.



assumptions were drawn concerning the distribution of the remaining faults in the program, since any such prediction would have been very difficult to justify. One of our long-term goals is to build the theory allowing the incorporation of various verification and validation techniques into the reliability prediction.

The central question investigated in this report is the estimation of the amount of testing, represented by the number of statistically independent test cases, needed to establish with sufficient confidence that the required reliability level is achieved. We start the rest of the report by briefly describing the features of the application that prompted this study, DOLILU II. Section 3 defines the strengths and weaknesses of the sampling model and provides an estimate on the number of tests needed for reliability assessment of DOLILU II. Section 4 introduces Bayesian statistics framework for reliability assessment, which allows counting of failure free system executions during its operational usage as tests for the purpose of reliability assessment. Section 5 expands the Bayesian framework by allowing non-ignorance prior assumptions, allowing for the inclusion of the evidence that reflects extensive verification and validation activities performed during system development. Summary and brief description of our current and future research efforts are given in Section 6.

2. Description of DOLILU System

The DOLILU II system diagram is shown in Figure 1. In order to increase fault detection capabilities, DOLILU II system consists of two independently developed computational lanes (*Primary* and *Secondary System*). Guidance commands are considered valid only if the outputs of both computational lanes are the same. For the purpose of this study, we are interested in the reliability assessment of the primary system.

The DOLILU II system functions as follows. When provided with the wind and atmospheric data, the executive module invokes the *Day-of-Launch Ascent Design System* (DADS). DADS module generates guidance commands for the day of launch condition. Guidance commands are passed to the *Space Vehicle Dynamic Simulation* (SVDS)

Processor. It generates trajectories for reference winds and atmospheric conditions. Only successfully simulated trajectories are forwarded to the verification module.

Near real-time verification is the most critical function of the DOLILU II system. Successfully simulated trajectories and their corresponding I-Loads are verified for conformance with safety related rules, called envelopes. The envelopes have been derived from previous experience (*experience envelopes*), and known system constraints (*system envelopes*). If any of the system constraint rules is violated, the violation must be reported and the trajectory must be dismissed. *The Day of Launch I-Load Verification Table (DIVDT) Processor* performs trajectory verification, and it should detect all potentially unsafe flight conditions [13]. Therefore, the reliability quantification of the DIVDT processor is highly desirable. Since it verifies the outputs of all the other modules of DOLILU II system, our study is limited to the testing of DIVDT processor.

3. Assessment of reliability: The random sampling model

Dynamic testing is a conventional method of program checking. A program under test is executed with different combinations of input data, and the results are compared with the expected values. The outcome of testing is used to predict the reliability of the program. Note, however, that software reliability depends on the software quality as well as on its operational usage. Since testing cannot guarantee the absence of faults, exposing the program to the operations anticipated to be the most frequently used should catch the failures that are the most likely to appear in field use. It is assumed that these failures, if detected, are the ones that count the most to the user. The quantitative approximation of the system's field use is called the *operational profile* [10].

A sound foundation for reliability assessment in the input domain is provided by statistical sampling theory [1]. Practical drawbacks of the sampling model of software reliability assessment are the following:

1. A large number of test cases is required,
2. Reliability estimation depends upon the ability to closely approximate/predict the operational profile of the field use, and
3. The existence of a test oracle is assumed.

Our discussions with NASA IV&V (Independent Verification and Validation) personnel revealed that DOLILU II system requires demonstration of failure probability to be under 10^{-4} . Due to the criticality of the program, required confidence level should surpass 0.99. In other words, reliability assessment process needs to provide evidence that there will be less than one in a hundred chance of encountering a system failure in ten thousand executions. The state of the practice in software reliability engineering states that these reliability levels are practically achievable [2, 6].

Let θ_0 denote the required probability of failure for the given program, and let C represent the confidence level that is. Since the system in its present usage requires a demonstration of failure probability of under 10^{-4} , this can be denoted as

$$P(\theta < 10^{-4} \mid TestData) \geq C. \quad (3.1)$$

The remaining question is how much testing needs to be done to satisfy these reliability requirements. In case of the sampling model, assuming statistical independence of test cases and failure free executions, the number of tests U can be calculated from the following expression [9]:

$$U = \frac{\ln(1-C)}{\ln(1-\theta_0)}. \quad (3.2)$$

Substituting in (3.2) the values specific for the assessment of DOLILU II, we get

$$U = \frac{\ln(1-0.99)}{\ln(1-10^{-4})} \Rightarrow U = \frac{\ln(0.01)}{\ln(0.9999)} = 4.6 \times 10^4. \quad (3.3)$$

Tables 1 and 2 provide a better insight into the required number of test cases as a function of reliability requirements. Table 1 indicates that an order of magnitude increase in required reliability causes an order of magnitude increase in the number of tests. The confidence level in Table 1 is constant, set to $C=0.99$.

Value of θ_0	Number of Test cases
10^{-2}	458
10^{-3}	4,602
10^{-4}	46,048
10^{-5}	460,514
10^{-6}	4,605,167

Table 1: Number of test cases as a function of required failure rate, with $C=0.99$

In Table 2, the required failure rate of the program is set to a constant, $\theta_0 = 10^{-4}$, while the confidence level varies between 0.92 and 0.999.

Value of C	Number of Test Cases
0.92	25,255
0.94	28,132
0.96	32,187
0.98	39,118
0.99	46,048
0.999	69,074

Table 2: Number of test cases as a function of required confidence level C, with $\theta_0 = 10^{-4}$

In case of DOLILU II system, while very expensive, the amount of testing defined above is not prohibitive. But, the sampling approach does not take into account successful use of the deployed system. This can be addressed in the Bayesian approach, described below.

4. Bayesian approach with ignorance priors

The cornerstone of Bayesian inference is the notion of subjective probability. Such a notion contrasts with the well-perceived notion of frequency for probability estimation. The axiom of probability states that the probability of a binary event has to be estimated by determining the success ratio. To attest toward this empirical estimation, one has to conduct trials in which the event occurs repetitively.

Subjective probability deals not only with the events but with propositions as well. A proposition is formulated from a collection of events that contribute towards the estimation based on observed behavior, or the reflection of one's belief in the system. In statistical terms, we *hypothesize* that the event does occur with the estimated probability. As evidence relevant to the hypothesis increases, we may change the *degree of belief* in the hypothesis. Interestingly, some argue that subjective probabilities assigned to a particular hypothesis may indeed be quite individualistic [8]. In other words, the probabilities assigned by different individuals would reflect different beliefs yielding different results. Bayesian inference theory circumvents this in the posterior analysis where our degree of belief changes with the observations made. However, egregious probability assumptions are not permissible.

Bayesian approach to software reliability prediction, described by Miller, et. al. [9] and Littlewood [8], forms an effective framework for the quantification of software quality. It allowed us to take into account failure-free executions of DOLILU II system that were observed in its operational environment. Due to its ability to take into account prior information, the Bayesian reliability assessment framework allowed reduction in the number of required test cases for reliability reassessment, when compared with the number of test cases inferred from the statistical sampling theory, for the same reliability goal.

Choice of the distribution that can accurately reflect prior beliefs is very important. In our study, we chose beta distribution [4, 8]. There are two primary reasons for choosing this distribution.

- By proper choice of the parameters, it is possible to depict any type of distribution that may be exhibited by the system.
- Beta distributions form a conjugate family. The conjugate family has the property that both the prior and posterior distributions will be members of the same parametric family of distributions. Intuitively, this represents a kind of homogeneity in the way in which our beliefs are represented, and how they change as we receive extra information [7].

Within the Bayesian framework *prior* knowledge about the parameter of interest, here the probability of failure on demand denoted as θ , is represented by the prior distribution. The probability density function $f(\theta)$ is

$$f(\theta) = \frac{\theta^{p-1} (1 - \theta)^{q-1}}{B(p, q)} \quad (4.4)$$

where $B(p, q)$ is the complete beta function with $p > 0, q > 0$. Parameters p and q can be adjusted to reflect prior beliefs about the reliability of software under test. The assumption of ignorance prior implies that measured failure rate θ is uniformly distributed within the range $[0.0, 1.0]$. Setting the values of p and q to the constant 1 ($p=q=1$), reflects ignorance prior [4]. Thus, a rectangular function represents the distribution of θ (θ can assume any value between 0.0 and 1.0 with equal probability), as shown by the following formula

$$f(\theta) = \frac{\theta^{1-1} (1 - \theta)^{1-1}}{B(1,1)} = 1. \quad (4.5)$$

A practical way to incorporate prior information (gathered through software inspection, for example [9]) into Bayesian reliability estimation is through the assumption that given information corresponds to a certain number or successfully executed random tests. In general, any prior assumption that is based on justifiable prediction of a mean and a variance for θ can be converted into values of parameters p and q , as will be discussed later.

The posterior distribution can be formed after the results of software tests become known. If the system has executed n demands encountering r failures, the posterior distribution $f(\theta)$ is

$$f(\theta | n, r, p, q) = \frac{\theta^{p+r-1} (1 - \theta)^{q+n-r-1}}{B(p + r, q + n - r)}. \quad (4.6)$$

Assuming the ignorance prior ($p=q=1$),

$$f(\theta | n, r, 1, 1) = \frac{\theta^r (1 - \theta)^{n-r}}{B(1 + r, 1 + n - r)}. \quad (4.7)$$

If during system testing U statistically independent demands are executed without encountering a failure then $n=U$ and $r=0$. Consequently,

$$f(\theta | U, 0, 1, 1) = \frac{(1 - \theta)^U}{B(1, 1 + U)}. \quad (4.8)$$

$B(1, 1+U)$ is the complete beta function. The generalized form is

$$B(a, b) = \int_0^1 \theta^{a-1} (1 - \theta)^{b-1} d\theta. \quad (4.9)$$

Therefore, $B(1, 1+U)$ is given by

$$B(1,1+U) = \int_0^1 (1-\theta)^U d\theta = \frac{1}{1+U} \quad (4.10)$$

From the reliability assessment point of view, we are interested in bounding the system failure rate θ to the interval between 0 and θ_0 , where θ_0 denotes the known (required) upper bound of the probability of failure the system under test may exhibit in the operational environment. The probability that the real failure rate, θ is lower than the required probability of failure θ_0 , $Prob(\theta < \theta_0)$, is the cumulative density function given by

$$Prob(\theta \leq p_0) = \int_0^{\theta_0} f(\theta | U, 0, 1, 1) d\theta. \quad (4.11)$$

When the specified confidence level C needs to be demonstrated,

$$Prob(\theta \leq \theta_0) \geq C. \quad (4.12)$$

With some substitutions, the following relation must hold:

$$\int_0^{\theta_0} f(\theta | U, 0, 1, 1) d\theta \geq C. \quad (4.13)$$

Substituting for $f(\theta | U, 0, 1, 1)$ from equation (4.8),

$$\int_0^{\theta_0} \frac{(1-\theta)^U}{B(1,1+U)} d\theta \geq C. \quad (4.14)$$

By combining equations (4.10) and (4.14), and integrating the Beta function, we get

$$\int_0^{\theta_0} (1+U)(1-\theta)^U d\theta \geq C. \quad (4.15)$$

When the above equation is solved for U as an equality, the result has the following form

$$(1+U) \left[\frac{(1-\theta)^{1+U}}{(1+U)(-1)} \right]_0^{\theta_0} = C. \quad (4.16)$$

When boundary conditions are included,

$$1 - (1-\theta_0)^{1+U} = C. \quad (4.17)$$

The total number of test cases, required to say with confidence C that the reliability requirement p_0 is satisfied, is

$$U = \frac{\ln(1-C)}{\ln(1-\theta_0)} - 1. \quad (4.18)$$

It comes as no surprise that Equations (3.2) and (4.18) are effectively the same. Clearly, since $p=q=1$ provides no

prior knowledge they do not influence the interpretation of the test results. Hence, both the sampling model and Bayesian theory require the same number of cases in the absence of prior knowledge about the failure rates. But the reason for introducing Bayesian statistics is to allow incorporating assumptions gathered through the application of extensive verification and validation activities and the successful runs of the deployed system.

Let us first consider incorporating the successful runs of the deployed system. The easiest way to allow this is by setting parameters p and q to appropriate values, reflecting the observations from the fielded system. Assuming the system has been executed T_0 times with no failures observed, as recommended in [9], parameter p can be set to 1, q can be set to T_0 . These values of p and q are approximations. Nevertheless, they provide a simple and intuitively correct estimate of the observed system behavior. The corresponding posterior density function is

$$f(\theta) = \frac{(1-\theta)^{U+T_0-1}}{B(1, T_0+U)}. \quad (4.19)$$

Hence, the corresponding complete Beta function is

$$B(1, T_0+U) = \int_0^1 (1-\theta)^{T_0+U-1} d\theta = \frac{1}{T_0+U}. \quad (4.20)$$

After substitution in (4.14) and simplification, the number of failure free test runs U needed to achieve confidence level C in software reliability, upon execution of T_0 successful trials, is given by the following expression:

$$U = \frac{\ln(1-C)}{\ln(1-\theta_0)} - T_0 \quad (4.21)$$

In addition to executions of the deployed system, the acceptance tests chosen randomly according to the operational profile of system usage are counted in T_0 . However, hand-crafted stress tests, for example, do not satisfy the statistical independence criterion, and cannot be included in this reliability estimation framework. In essence, the strength of the prior evidence in reliability of software may reduce the testing effort, proportional to the number of statistically independent system tests already performed and field executions completed. DOLILU II system usually undergoes about a hundred executions per year. Having in mind the number of reliability tests indicated in Tables 1 and 2, the amount of extra testing needed to ensure compliance with the reliability requirement is not significantly reduced. Thus, the next step is to look at utilization of Bayesian approach when prior knowledge about software quality includes more than just ignorance.

5. Bayesian assessment using non-ignorance priors

Now we shall introduce the non-ignorance prior belief into the Bayesian framework. In case of DOLILU II, several main reasons encourage non-ignorance priors:

- The software has been in use for over five years. As already discussed, several methodologies, including partial correctness proofs, inspection, code walk-throughs, etc., were performed during the development life cycle. Rigorous development process was followed, including an independent contractor performing independent verification and validation activities.
- If only the successful executions of the deployed system are included in reliability assessment, the number of additional tests remains in the order of tens of thousands.

Considering the above factors, inclusion of the results of “qualitative” V&V activities is desirable. However, quantifying their effects on system reliability is difficult. Smidts at al. discuss techniques for including process information in reliability assessment in [15]. For the reason of space limitation, the attention in the remainder of this paper is limited towards derivation of a probabilistic framework that accepts prior distributions reflecting the application of V&V activities. Exactly how to derive meaningful prior beliefs from qualitative V&V activities is subject to substantial further research. In general, qualitative V&V activities should inspire belief in the mean value of system failure probability, θ , and its variance. These are represented in the Bayesian framework through the values of parameters p and q of a Beta distribution. Having μ , representing the guess of the mean value of system

failure probability and σ^2 , its variance the appropriate values of p and q are calculated as follows. Since

$$\mu = \frac{p}{p+q} \quad (5.22)$$

and

$$\sigma^2 = \frac{pq}{(p+q)^2(p+q+1)}, \quad (5.23)$$

solving for p and q , we get

$$p = \mu \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right) \quad (5.24)$$

$$q = (1-\mu) \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right)$$

Now, the complete beta function assumes the following form:

$$B(p, q+U) = \int_0^1 \theta^{p-1} (1-\theta)^{q+U-1} d\theta. \quad (5.25)$$

Repeated integration by parts and simplification yields

$$B(p, q+U) = \frac{(p-1)!}{(q+U)(q+U+1)\dots(q+U+p-1)}. \quad (5.26)$$

Taking $n=U$, and $r=0$ (no failures observed), equation 4.6 can be rewritten as

$$f(\theta|U, 0, p, q) = \frac{\theta^{p-1} (1-\theta)^{q+U-1}}{B(p, q+U)}. \quad (5.27)$$

Consequently, when confidence level C is taken into consideration,

$$\int_0^{\theta_0} f(\theta|U, 0, p, q) d\theta \geq C. \quad (5.28)$$

The following equation needs to be solved for U :

$$\frac{(q+U)(q+U+1)\dots(q+U+p-1)}{(p-1)!} * \int_0^{\theta_0} \theta^{p-1} (1-\theta)^{q+U-1} d\theta = C \quad (5.29)$$

Integrating equation (5.29) by parts, following some simplifications, we get

$$\begin{aligned}
& 1 - \frac{(q+U+p-1)(q+U+p-2)\dots(q+U+1)}{(p-1)!} \theta_0^{p-1} (1-\theta_0)^{q+U} - \\
& \frac{(q+U+p-1)(q+U+p-2)\dots(q+U+2)}{(p-2)!} \theta_0^{p-1} (1-\theta_0)^{q+U+1} - \\
& \frac{(q+U+p-1)(q+U+p-2)\dots(q+U+3)}{(p-3)!} \theta_0^{p-3} (1-\theta_0)^{q+U+2} - \\
& \frac{(q+U+p-1)(q+U+p-2)\dots(q+U+4)}{(p-4)!} \theta_0^{p-4} (1-\theta_0)^{q+U+3} - \\
& \dots = C
\end{aligned} \tag{5.30}$$

The above equation can be solved for U using one of the standard tools, such as MatCad or Mathematica. Depending on the values of variables, the number of tests U needed to ascertain required reliability levels can be obtained. Depending on the rigor of the applied V&V techniques, reflected in (hopefully reasonable) assumptions on μ and σ^2 , the number of additional tests required for system reliability assessment can be several orders of magnitudes less than in case of the ignorance prior. Table 2 shows the number of tests needed for corroborating software reliability levels assumed in the leftmost column.

Assumed Value of θ	Number of Test cases to corroborate θ
10^{-3}	950
10^{-4}	2678
10^{-5}	9436
10^{-6}	20,796
10^{-7}	51,987

Table 2: Number of tests needed to corroborate assumed failure rate θ , $C=0.99$

In case of the reassessment of DOLILU II, we assumed that rigorous V&V techniques were responsible for already achieving the required reliability of less than 10^{-4} failures per execution. With this assumption, corroboration of the required software reliability was achieved by running approximately 2600 statistically independent new tests.

5.1 Bayesian reliability estimation with failure occurrences during testing

The fact that during acceptance testing and field use of DOLILU II failures were not observed makes this system an exception rather than the rule. In general, when failures are observed in reliability testing, assuming their root causes are successfully corrected, the question remains how many future test cases need to be performed. Bayesian reliability assessment framework can be easily extended to handle this situation. Let us assume that a failure occurred after the execution of s demands. The posterior for θ immediately following the failure on the s^{th} demand is

$$f(\theta | s, 1, p, q) = \frac{\theta^p (1 - \theta)^{s-2}}{B(p + 1, s + q - 1)} \tag{5.31}$$

This forms the prior distribution for θ for further testing that needs to be conducted following the removal of the fault that caused the failure. We now need to compute U_1 , the total number of failure-free executions required for the software to demonstrate required level of reliability following the occurrence of a failure. Hence,

$$f(\theta|U_1 + s, 1, p, q) = \frac{\theta^p (1 - \theta)^{U_1 + s + q - 2}}{B(p + 1, U_1 + s + q - 1)} \quad (5.32)$$

Notice that this is simply the posterior distribution after seeing both $(s-1)$ failure free executions followed by a failure (on demand s), and then seeing U_1 more further failure free demands. This posterior distribution will be the same whenever the single failure occurred among $s+U_1$ demands. In other words, it depends only upon the total number of demands, and the number of failures. Now we may compute U_1 for which

$$\int_0^{\theta_0} \frac{\theta^p (1 - \theta)^{U_1 + s + q - 2}}{B(p + 1, U_1 + s + q - 1)} d\theta \geq C \quad (5.33)$$

In general, if we observe r failures occurring at the s_1^{th} , $(s_1 + s_2)^{\text{th}}$, $\dots, (s_1 + s_2 + s_3 + \dots + s_r)^{\text{th}}$ demands, we should require a further of U_r demands executed failure-free, such that

$$\int_0^{\theta_0} \frac{\theta^{p+r} (1 - \theta)^{\sum_0^r s_a + U_1 + q - r}}{B(p + r, U_1 + \sum_0^r s_a + q - r)} d\theta \geq C \quad (5.34)$$

Interested readers will find the solution of this equation and the discussion about its practical usage in [8].

6. Summary

We presented a framework for quantification of software reliability of NASA's DOLILU II system. The quantification of reliability is based on the sampling model and Bayesian inference. Since DOLILU II system has been in operational use for several years, one of the important features of the reliability assessment methodology is the ability to incorporate information on deployed system's (failure free) performance. Furthermore, the Bayesian framework has been extended so that it can account for the use of "qualitative" verification and validation activities performed during system's development.

There are numerous venues for extending the results reported here. Probably the most intriguing problem is the development of a sound methodology for including the results of different verification and validation techniques, and not just testing, into the software reliability prediction. While this methodology per se does not make systems more reliable than they already are, it provides a framework for quantification of otherwise quantitative software assurance processes. As concerns our effort to assess the reliability of DOLILU II, we are currently investigating automated generation of a large number of test trajectories [14]. These trajectories represent weather conditions that might realistically appear in the field, since the existing data is not sufficient for the number of tests needed in reliability assessment. Another important open research issue is the availability of testing oracles for DOLILU II, currently also under development.

Acknowledgements

This work was supported in part by NASA, through cooperative agreement #NCC 2-979. Profs. Erdogan Gunel and Harshinder Singh of the Department of Statistics, West Virginia University, provided valuable comments and suggestions on improving the early drafts of this paper.

References

- [1] F. B. Bastani, A. Pasquini, "Assessment of a Sampling Method for Measuring Safety-Critical Software Reliability," *Proc. ISSRE'94*, Monterey, CA., Nov. 1994.
- [2] R. W. Butler, G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Trans. Software Eng.*, Vol. 19, No. 1, Jan. 1993, pp. 3-12.

- [3] M. S. Deutsch, *Software Verification and Validation: Realistic Project Approaches*, Prentice Hall Series in Software Engineering, Prentice Hall, Englewood Cliffs, NJ, 1982.
- [4] N. L. Johnson, S. Kotz, *Distributions in Statistics: Continuous Univariate Distributions*, Wiley Series in Probability and Mathematical Statistics, 1969.
- [5] Keiller P.A., Miller D.R., "On the use and the performance of software reliability growth models", *Reliability Engineering and System Safety*, 1991, pp. 95-117.
- [6] J. D. Lawrence, W. L. Persons, G. G. Preckshot, J. Gallagher, "Evaluating Software for Safety Systems in Nuclear Power Plants," in *Proc. COMPASS '94*, Gaithersburg, MD, June 1994.
- [7] Bev Littlewood, Lorenzo Strigini, "Validation of Ultrahigh Dependability for Software-based Systems", *Comm. of the ACM*, Vol. 36, No. 11, Nov. 1993, pp. 69-79.
- [8] B. Littlewood, D. Wright, "Stopping Rules for the Operational Testing of Safety-Critical Software," in *Proc. FTCS'25*, Pasadena, CA, June 1995, pp. 444-451.
- [8] Harry F. Martz, Ray A. Waller, *Bayesian Reliability Analysis*, John Wiley and Sons Inc., New York, 1982.
- [9] K. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, J. W. Voas, "Estimating the Probability of Failure When Testing Reveals no Failures," *IEEE Trans. on Software Eng.*, Vol. 18, No. 1, Jan. 1992, pp. 33-44.
- [10] J. D. Musa, "Operational Profiles in Software Reliability Engineering," *IEEE Software*, Mar. 1993, pp. 14-32.
- [11] J. D. Musa, *Software Reliability Engineering*, McGraw Hill, New York, 1998.
- [12] NASA, "Space Shuttle: DOLILU-II, Definition and requirements document", Volume VI, Quality Assurance Rules, May 1993.
- [13] Rockwell International, "Software Requirements Specifications, Flight design and Dynamics ascent discipline Ascent Subsystem Day of launch function (DOLILU-II) DIVDT program function: Ver: 4.2 FD", March 1993.
- [14] B. Taylor, B. Cukic, "Evaluation of Regressive Models for Automated Test Trajectory Generation," *Proc. 11th International Symposium on Software Reliability Assessment (ISSRE 2000)*, San Jose, CA, October 2000.
- [15] C. Smidts, M. Stutzke, R. W. Stoddard, "Software Reliability Modeling: An Approach to Early Reliability Prediction", *IEEE Transactions on Reliability*, Vol. 47, No. 3, September 1998, pp. 268-278.