

The Theory of Software Reliability Corroboration

Bojan Cukic, Erdogan Gunel*, Harshinder Singh*, Lan Guo

Lane Department of Computer Science and Electrical Engineering

* Department of Statistics

West Virginia University

Morgantown, WV 26506-6109

E-mail: [cukic.lan}@csee.wvu.edu](mailto:{cukic.lan}@csee.wvu.edu), [egunel,hsingh}@stat.wvu.edu](mailto:{egunel,hsingh}@stat.wvu.edu)

Abstract

Software certification is a notoriously difficult problem. From software reliability engineering perspective, certification process must provide evidence that the program meets or exceeds the required level of reliability. When certifying the reliability of a high assurance system very few, if any, failures are observed by testing. In statistical estimation theory the probability of an event is estimated by determining the proportion of the times it occurs in a fixed number of trials. In absence of failures, the number of required certification tests becomes impractically large.

We suggest that subjective reliability estimation from the development lifecycle, based on observed behavior or the reflection of one's belief in the system quality, be included in certification. In statistical terms, we hypothesize that a system failure occurs with the hypothesized probability. Presumed reliability needs to be corroborated by statistical testing during the reliability certification phase. As evidence relevant to the hypothesis increases, we change the degree of belief in the hypothesis. Depending on the corroboration evidence, the system is either certified or rejected. The advantage of the proposed theory is an economically acceptable number of required system certification tests, even for high assurance systems so far considered impossible to certify.

1. Introduction

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* about θ . In practice, the hypothesized *failure probability* θ_0 and the *confidence level* C are application specific and predefined.

Input domain reliability assessment approach defines the reliability of a program as the probability of failure free operation for specific inputs [1]. The time component of reliability assessment in time domain models is replaced by the concept of test runs, which are activated by test cases from the *input domain* I . Program P is then a function that maps all the elements of the multidimensional space I into the output space O . The inputs that activate faults in the program are mapped into failures, i.e., incorrect outputs.

We argue that input domain based approach is theoretically more suitable for software reliability certification in the high assurance domain than time-domain based approach. Generally speaking, time domain approaches to software reliability assessment assume the observation of system failures. Occurrence of failures in certification testing of high assurance systems is unlikely since it implies significant rework and delays in the system release and deployment. In principle, system failures should occur prior to certification testing. Consequently, the main purpose of certification test is providing system developers, verification and validation agents, as well as (at this point in time nonexistent) the certification authority, with a necessary level of statistical confidence that the target software/system reliability level is reached prior to the deployment [18]. In the reliability certification phase, we consider software system as an entity, subjecting it to inputs and observing the outputs as being either correct or incorrect. This may be viewed as a stochastic process, where software executions result in failures in a stochastic manner [2].

Traditional theory behind input domain models of software reliability assessment assumes that certification testing process starts with "zero knowledge" about the system under test. In other words, all the observations collected during the development and verification and validation activities, such as inspections and reviews, module testing, formal analysis, engineering observations and judgments, etc.,

play no role in reliability certification. Consequently, the most important drawback of the input domain models is an enormous amount of testing (statistically independent executions of test cases) needed to attain a modest confidence that moderate software reliability has been achieved.

Process improvements and adherence to maturity models enable the production of software with repeatable quality [3, 19]. Ignoring process and product quality measurements collected throughout the development lifecycle in software reliability certification phase makes no sense. On the other hand, reliability prediction models based on process and product measurements alone may not be sufficiently accurate [10, 21]. These predictions, especially in the realm of high assurance systems, need *corroboration*. If during the certification testing we assume a limited belief in the accuracy of early assessment models then the main drawback of input domain reliability assessment models, the impractically large number of statistical tests, disappears. In other words, it is faster to confirm an existing belief (that the program is reliable enough) than to establish this belief by assuming nothing. In this paper, we describe statistical theory behind software reliability corroboration, the approach that has the long term potential to make software certification of high assurance systems practical. Early assessment models suitable for establishing a hypothesis about software reliability are out of the scope of this paper. However, these techniques remain the focus of our further investigation.

The rest of the paper is organized as follows. Section 2 overviews traditional input domain based software reliability assessment models and their limitations. In Section 3, we present a simple assessment methodology based on Bayesian estimation, which may take into account the prior knowledge about the system being tested. In Section 4, Bayesian estimation approach is replaced by the Bayesian hypothesis testing approach, which further simplifies the process of building statistically sound arguments pertinent to software reliability corroboration. The ultimate goal of these methodologies is to provide decision support for the very basic questions that all organizations involved in the development of high assurance applications face: "Is the system ready for release"? Section 5 overviews software reliability prediction models that try to predict software reliability in different stages of the lifecycle. Imminent improvement of these methodologies will allow their integration into the reliability corroboration framework. Section 6 summarizes our findings and provides directions for the further work.

2. Input Domain Based Models: Background and Limitations

An intuitive measure of software reliability is the proportion of test cases that result in correct outputs. If n represents the total number of test cases and n_f the number of detected failures, the estimated reliability according to the Nelson model [11], is

$$\hat{R} = \frac{n - n_f}{n} = 1 - \frac{n_f}{n}.$$

When an indefinite number of test runs is taken,

$$R = 1 - \lim_{n \rightarrow \infty} \frac{n_f}{n} = 1 - \theta,$$

the fraction n_f/n represents the estimated probability of failure in a single run of the program (frequently denoted by ϕ). This leads to the reliability prediction based on the probability of correct execution in each run. Thus, the estimated probability of correct execution over i runs is given by

$$\hat{R}(i) = (\hat{R})^i.$$

In the formula above, it is assumed that the inputs are selected independently according to the same probability distribution used to choose the random sample. A reference to the environment in which the program is executed, i.e., its operational profile, can be given explicitly. Thus, the probability of failure in a single run can also be estimated by

$$\phi = \sum_{i \in I} p(i)\alpha(i),$$

where $p(i)$ is the probability that input i is selected from I , and $\alpha(i)$ is 1 if input i leads to incorrect output, and 0 otherwise. Exhaustive testing can, in principle, provide an exact assessment of the probability of failure, since

$$\sum_{i=1}^{|I|} p(i) = 1,$$

where $|I|$ represents the total number of inputs. A sound foundation for reliability assessment in the input domain is provided by statistical sampling theory [1]. A program under test separates the input domain into two disjoint classes: inputs that are correctly and those that are incorrectly mapped into the corresponding outputs. Testing can be simulated by randomly drawing balls from an urn containing black balls (test cases in the input space I resulting in a failed execution) and white balls (correct executions). Consequently, from the statistical viewpoint, each test case is a Bernoulli trial, and the model is frequently called the *sampling model*. Since input domains are large, the likelihood of selecting the same test case i is small.

Thus, reliability estimation assumes *sampling with replacement* in which nothing precludes repetition of a test case, but it is simpler (and cheaper) to implement. Due to the use of statistical sampling, authors often introduce statistical terminology [1,12]. For example, set of points in the input space is called the population, while the number of executed test cases during program testing is called the *sample size*.

Reliability assessment of high assurance systems based on statistical sampling is performed in the certification phase of the software life-cycle. Faults are not removed when discovered. Rather, when ultra-high reliability is required, the program is rejected. Eventual assessment of the corrected program must be restarted from the beginning [13]. Often, it is unlikely that the program will fail during the certification testing, possibly due to the use of formal methods, fault prevention, and other techniques. Research has been performed concerning the difficult problem of estimating the probability of failure when a program does not fail. Classic statistical work dating back to Laplace states that when t white balls and no black balls are drawn from the urn containing an unknown proportion of black and white balls, the probability of drawing a black ball next (representing the probability of failure in a single run of the program) is $\frac{1}{t+2}$. The above result is known as

Laplace rule of succession [14]. It means that failure-free testing relates the estimated probability of failure to the number of test cases. In order to establish the probability of failure at less than, say, 10^{-9} failures per hour, one needs to test the program for 10^9 hours (approximately 114,000 years). Limited improvement is achievable through the acceleration of testing. The second possibility for improvement is making prior assumptions about the failure probability, as presented in later sections. The central question is how much testing should be conducted?

Let θ_0 denote the hypothesized probability of failure for the given program, the number of test cases U to observe at least one failure with probability C (confidence level) is given by [8]:

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

Tables 1 and 2 provide an insight into the required number of test cases as a function of reliability requirements. These results are well known from the literature [1, 2, 5, 8, 9, 12]. Table 1 indicates that an order of magnitude increase in 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 $\theta_0 = 10^{-4}$

3. Bayesian Estimation Approach

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 an 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 at least once.

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 hypothesized 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 [7]. 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.

Choice of the distribution that can accurately reflect prior beliefs is very important. In our study, we chose beta distribution [4, 6, 7]. There are two primary reasons for choosing this distribution.

- The Beta family of distributions is a rich and tractable family, by proper choice of the parameters, it is possible to depict different shapes of distributions that may be exhibited by the system.
- Beta distributions form a conjugate family to binomial distribution. 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].

While we find properties of Beta distribution the most appealing, other distributions with similar properties could be used too. 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

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

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 , as shown by the following formula

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

A practical way to incorporate prior information (gathered through software inspection, for example [8]) 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

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

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

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

When the specified confidence level C needs to be demonstrated,

$$Prob(\theta \leq \theta_0 | TestData) \geq C.$$

The total number of test cases, required to say with confidence C that the reliability requirement θ_0 is satisfied, assuming ignorance priors and failure free testing, turns out to be almost the same as the one reported in Section 2:

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

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.

3.1. Bayesian assessment using non-ignorance priors

Inclusion of the results of “qualitative” V&V activities is desirable. However, quantifying their effects on system reliability is difficult. Smidts et al. discuss techniques for including process information in reliability assessment in [10]. 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 [3]. 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 can be calculated.

Now, the complete beta function assumes the following form:

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

Repeated integration by parts and simplification yields

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

Taking $n=U$, and $r=0$ (no failures observed),

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

Consequently, when confidence level C is taken into consideration,

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

The following equation needs to be solved for U :

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

The above equation can be solved for U using one of the standard tools, such as 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 3 shows the number of tests needed for the corroboration of software reliability levels assumed in the leftmost column by failure free testing¹. Littlewood and Wright [6] extended this approach to cope with cases where $r \neq 0$.

A practical problem in using the values given in Table 3 in certification testing is a difficulty in deriving the "right" values for the parameters describing Beta distribution. Furthermore, this framework does not

Assumed Value of θ_0	Number of Test cases to corroborate θ_0
10^{-3}	275
10^{-4}	4,098
10^{-5}	42,323
10^{-6}	424,570
10^{-7}	~4,250,000

Table 3: Number of tests needed to corroborate assumed failure rate θ_0 , $C=0.99$

support expressing the level of trust in the prior distribution. For example, we would like to be able to express that development regimes and environments, which were repeatedly used by an organization in the past, result in more accurate priors than the new (untested) development regimes. Therefore, our next framework will include the level of trust in the "correctness" of the prior failure rate estimates.

4. Bayesian Hypothesis Testing

Let $0 < \theta_0 < 1$ be a sufficiently small number close to zero that represents the required system reliability. Let us consider the null hypothesis $H_0 : \theta \leq \theta_0$ and the alternative hypothesis $H_1 : \theta > \theta_0$. The null hypothesis states that the program's true reliability (which is unknown, i.e., being estimated) is higher than required, whereas H_1 states the opposite, i.e., the system should not be released. In classical statistics a statistical hypothesis testing procedure is evaluated in terms of the Type I and Type II error probabilities. Type I error occurs when H_0 is rejected when it is true and Type II error occurs when H_0 is accepted when it is not true. In Bayesian analysis the task of deciding between H_0 and H_1 is conceptually more appealing and, at the same time, more straightforward. We simply compute

$$P(\theta \leq \theta_0 | \text{Test data}),$$

the so called posterior probability of the null hypothesis H_0 . The conceptual advantage is that the posterior probability reflects the prior opinions (the opinions about θ prior to certification testing of the program) and the results of the actual certification test. Let $P(H_0)$ and $P(H_1)$, where $P(H_0) + P(H_1) = 1$, denote the prior probabilities assigned to null and the alternative hypothesis. In practice, it may be difficult to obtain these probabilities, as discussed in the next section. Then,

$$O(H_0) = P(H_0) / P(H_1)$$

is called the *prior odds of H_0 to H_1* and

$$O(H_0 | \text{Test Data}) = \frac{P(H_0 | \text{Test Data})}{P(H_1 | \text{Test Data})}$$

is called the *posterior odds ratio of H_0 to H_1* . The *Bayes factor* $F(H_0, H_1)$ is defined as the ratio of posterior odds to prior odds in favor of the null hypothesis,

$$F(H_0, H_1) = O(H_0 | \text{Test Data}) / O(H_0)$$

If the Bayes factor $F(H_0, H_1)$ is greater than one then we have data helped in increasing odds in favor of H_0 . The posterior probability of H_0 can be written in terms of the prior probability of H_0 and the Bayes factor,

$$P(H_0|Test Data) = \frac{P(H_0)F(H_0, H_1)}{[P(H_0)F(H_0, H_1) + (1 - P(H_0))]}$$

During certification testing, program executions either result in a success or in a failure. An important factor determining our ability to corroborate the null hypothesis is the number of failures in n tests of the program. We are interested in finding the overall number of certification tests, given the number of failures observed in certification, such that

$$P(H_0 | Test Data) = C_o$$

where C_o represents the required confidence level.

In Table 4, we give the number of required certification tests, assuming $C_o=0.99$, when no failures are encountered (column n_0), one failure is encountered (column n_1) and when two failures are encountered (column n_2), for some selected values of θ_o and $P(H_0)$.

A note of caution is appropriate here. This section presents a simple overview of the reasoning and justification behind the Bayesian hypothesis testing approach to determine the number of “software reliability corroboration tests”. In theory, the prior beliefs in the null hypothesis and its alternative need to be provided as probability distributions of software failures over intervals $(0, \theta_o)$ and $(\theta_o, 1)$, respectively. In Table 4, our assumption is that the distribution of θ under H_0 and H_1 is uniform. This by no means represents model limitation, since the theory behind this framework allows for any distribution. The detailed discussion of theoretical aspects of Bayesian hypothesis testing theory behind software reliability corroboration framework is given in Appendix A.

A closer look at the values in Table 4 reveals the strongest possible motivation for the software reliability corroboration approach. For $\theta_o=10^{-2}$ and $P(H_0)=0.4$, for example, if after 90 tests there is no failure then we are 99% confident that $\theta \leq 10^{-2}$. If after 128 tests there has been one failure observed then we are 99% confident that $\theta \leq 10^{-2}$. If after 167 tests there are 2 failures then we are, again, 99% confident that $\theta \leq 10^{-2}$. Table 4 may also be used in a “sequential way”, as follows. Suppose that $\theta_o=10^{-2}$, $P(H_0)=0.4$. Perform 90 tests. If one failure is encountered then continue testing. If after 128 tests there is one failure then stop. At this point, we are 99% confident that $\theta \leq 10^{-2}$. If after

128 tests two failures are observed (one before test case 90 and one after) then continue testing. If after 167 tests there are two failures observed then stop, we are 99% confident that $\theta \leq 10^{-2}$.

Table 4. The number of tests required for reliability corroboration according to bayesian hypothesis testing theory.

θ_o	$P(H_0)$	n_0	n_1	n_2
0.01	0.01	457	476	497
0.001	0.01	2378	2671	2975
0.0001	0.01	6831	10648	14501
0.00001	0.01	9349	33176	63649
0.000001	0.01	9752	101273	282007
0.01	0.02	388	410	433
0.001	0.02	1766	2098	2438
0.0001	0.02	3954	7549	11315
0.00001	0.02	4736	23037	49499
0.000001	0.02	4838	70800	221022
0.01	0.1	228	258	289
0.001	0.1	636	1017	1402
0.0001	0.1	853	3157	6150
0.00001	0.1	886	9646	27281
0.000001	0.1	890	30067	123725
0.01	0.4	90	128	167
0.001	0.4	138	411	739
0.0001	0.4	146	1251	3260
0.00001	0.4	147	3889	14724
0.000001	0.4	147	12222	67468
0.01	0.6	50	87	126
0.001	0.6	63	269	552
0.0001	0.6	65	827	2458
0.00001	0.6	65	2584	11173
0.000001	0.6	65	8139	51351

5. Predicting Software Reliability Prior to Certification

As mentioned earlier, inclusion of the results of “qualitative” V&V activities in the reliability corroboration framework is essential. But quantifying their effects on system reliability is difficult. Given reasonably low levels of trust one needs to have in the accuracy of their predictions (see the values of $P(H_o)$ in Table 4), we believe that candidate approaches that could be utilized in a software reliability corroboration framework already exist. However, significant research effort is needed to establish the levels of trust one may have in the precision and repeatability of their estimates.

There have been numerous attempts to predict software quality from design-level metrics such as cohesion and coupling. Bieman and Kang defined the design-level cohesion measures formally in [15], and used them to predict properties of implementations created from a given design. Their design-level cohesion (DLC) measure is similar to that used by Stevens et al. [16]. Later, Briand et. al. introduced and compared various high-level design measures for object-based software systems to identify fault-prone software parts in some high assurance systems [17]. Specifically, they defined a set of measures for cohesion and coupling and investigated the measures' relationship to fault-proneness.

Attempts to predict of software product quality based on the quality of software process are not rare. Jones hypothesized the relationship between CMM levels, “defect potentials” and “delivered defects” [19]. Fenton and Neil demonstrated that the use of a standard is likely either to deliver reliable and safe systems at an accepted cost or help predict reliability and safety accurately [20]. They examined a specific standard for safety critical systems (namely IEC 1508) and showed how it can be improved by applying their strategy.

Recently, Fenton et. al. provided a critical review of current software reliability prediction models and proposed Bayesian Belief Networks (BBNs) as the most promising research avenue [21]. According to them, defect counts cannot be used to predict reliability because they do not measure software quality according to its operational usage. Despite the reported high correlations between design complexity and defects, the relationship is not entirely causal. The same problem exists when size and complexity metrics are used as predictors of defects. Interestingly, BBN approach enables managing the causal relationship between product and process information and software defects.

BBN models have been developed for predicting software safety [22] software reliability engineering self-assessment [23] and component availability estimation [24].

Formal modeling and analysis is another active research area in software assurance. It is argued that improved software reliability results from the application of formal methods early in software life cycle. However, the exact effects of the application of formal methods are difficult to quantify [25, 26, 27]. Meanwhile, research in quantitative software reliability prediction has not considered including the impact of formal methods in software product and process information [28, 29].

6. Summary

We presented three different statistical frameworks for quantification of software reliability based on the input domain modeling. The quantification of reliability is obtained through the sampling model, Bayesian estimation and Bayesian hypothesis testing. The two Bayesian frameworks allow the inclusion of “qualitative” verification and validation activities performed during system’s development in terms of prior failure probabilities. These approaches are suitable for our "reliability corroboration" paradigm. To the best of our knowledge this is the first occasion that Bayesian hypothesis testing is proposed for the modeling of software reliability.

The significance of Bayesian hypothesis testing framework for software reliability corroboration is in the reasonable number of tests that it prescribes for software certification. This is especially obvious in the case of high assurance systems, which for all practical purposes, are considered impossible to certify by today's standards. While this methodology per se does not make systems more reliable than they already are, it provides a framework for quantification of otherwise qualitative software certification processes.

It comes as no surprise that programs developed in stable and mature development environments, which support measurement and process improvement feedback throughout the lifecycle, will require a fewer number of tests for certification. Ultimately, only such environments should be used for high assurance system development. But, as Table 4 indicates, a failure encountered in certification testing imposes a steep economic penalty, because tens of thousands of additional tests may become required.

Regardless of its elegance, Bayesian hypothesis testing framework for software reliability corroboration may make the practice of software certification risky if applied inappropriately. The basic current problem is the immaturity of methods for evaluating the precision and trust assigned to pre-certification software reliability beliefs. Overestimating the belief assigned to the achieved reliability level during the development may lead to a too few certification tests being prescribed and performed with potential catastrophic consequences. Any statistical model can be used inappropriately, and this one is not an exception.

Evaluating trust in the predictions of software reliability throughout the development lifecycle is the major thrust of our further research. Probably the most intriguing problem is the development of sound methodologies that can include the results of different verification and validation techniques, and not just testing, into the software reliability prediction.

Acknowledgements

The authors are very grateful for the comments and suggestions they received from the anonymous reviewers. This work was supported in part by US National Science Foundation grant CCR-0093315 and in part by NASA through cooperative agreement NCC 2-979.

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] 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.
- [6] 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.
- [7] Harry F. Martz, Ray A. Waller, *Bayesian Reliability Analysis*, John Wiley and Sons Inc., New York, 1982.
- [8] 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.
- [9] B. Cukic, D. Chakravarthy, "Bayesian Framework for Reliability Assurance of a Deployed Safety Critical System" *Proc. 5th International Symposium on High Assurance Systems (HASE 2000)*, Albuquerque, NM, October 2000.
- [10] 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.
- [11] T. A. Thayer, M. Lipow, E. C. Nelson, *Software Reliability*, North-Holland Publishing, TRW Series of Software Technology, Amsterdam, 1978.
- [12] J. M. Voas, C. C. Michael, K. W. Miller, "Confidentially Assessing a Zero Probability of Software Failure," *High Integrity Systems*, Vol. 1, No. 3, 1995, pp. 269-275.
- [13] D. L. Parnas, A. J. van Schouwen, S. P. Kwan, "Evaluation of Safety-Critical Software," *Communications of the ACM*, Vol. 33, No. 6, June 1990, pp. 636-648.
- [14] G. Cochran, *Sampling Techniques*, John Wiley & Sons, New York, NY, 1977.
- [15] J. M. Bieman and B. K. Kang, "Measuring Design-Level Cohesion", *IEEE Trans. Software Eng.*, Vol. 24, No. 2, Feb. 1998.
- [16] W. Stevens, G. Myers, and L. Constanine, "Structured Design", *IBM Systems J.*, Vol. 13, No. 2, pp. 115-139, 1974.
- [17] L. C. Briand, S. Morasca, and V. R. Basili, "Defining and Validating Measures for Objected-Based High-Level Design", *IEEE Trans. Software Eng.*, Vol. 25, No. 5, Sep./Oct. 1999.
- [18] J. M. Voas, C. C. Michael, and K. W. Miller, "Confidently Assessing a Zero Probability of Software Failure", *High Integrity Systems*, Vol. 1, No. 3, pp. 269-275, 1995.
- [19] C. Jones, "The Pragmatics of Software Process Improvements", *Software Engineering Technical Council Newsletter, Technical Council on Software Eng.*, IEEE Computer Society, Vol. 14, No. 2, Winter, 1996.
- [20] N. E. Fenton and M. Neil, "A Strategy for Improving Safety Related Software Engineering Standards", *IEEE Trans. Software Eng.*, Vol. 24, No. 11, Nov. 1998.

- [21] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models", *IEEE Trans. Software Eng.*, Vol. 25, No. 5, Sep/Oct 1999.
- [22] M. Neil, N. Fenton and L. Nielsen, "Building Large-Scale Bayesian Networks", *The Knowledge Engineering Review*, 15(3), pp257-284, 2000.
- [23] S. Donohue and Y. Yu, "Developing a Probabilistic SRE Self-Assessment Tool Using BBNs", *ISSRE 2001 Student Paper*, 2001.
- [24] Y. Yu, E. Stoker, "A Comparison of Probability and Bayesian Belief Networks Methods to Estimate Component Availability in Large, Distributed Networks", *ISSRE 2001 Student Paper*, 2001.
- [25] S. King, J. Hammond, R. Chapman, and A. Pryor, "Is Proof More Cost-Effective Than Testing?", *IEEE Trans. Software Eng.*, Vol. 26, No. 8, Aug. 2000.
- [26] C. Heitmeyer, J. Kirby, Jr., B. Labaw, M. Archer, and R. Bharadwaj, "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications", *IEEE Trans. Software Eng.*, Vol.24, No.11, Nov. 1998.
- [27] J. Rushby, "Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification", *Computer-Aided Verification, CAV'2000*, Vol. 1855 of Lecture Notes in Computer Science, pp. 508-520, Chicago, IL, Jul. 2000.
- [28] D. A. Peled, *Software Reliability Methods*, Springer-Verlag New York, New York, NY, 2001.
- [29] D. J. Richardson and L. A. Clarke, "Partition Analysis: A Method Combining Testing and Verification", *IEEE Trans. Software Eng.*, Vol. SE-11, No. 12, Dec. 1985.

Appendix A : Bayesian Hypothesis Testing: The Theory

Let r denote the number of failures in n tests and θ be the probability of a failure. We want to test the null hypothesis

$$H_0 : \quad \theta \leq \theta_0$$

against the alternative hypothesis

$$H_1 : \quad \theta > \theta_0$$

for some given constant $0 < \theta_0 < 1$.

In Bayesian analysis posterior probability $P(H_0 | r, n)$ is used to decide between H_0 and H_1 . It represents the probability of the null hypothesis in light of the data and prior knowledge. Let $P(H_0)$ and $P(H_1)$, where $P(H_0) + P(H_1) = 1$, denote the prior probabilities assigned to null and the alternative hypothesis,

$$O(H_0) = P(H_0) / P(H_1) \quad (1)$$

is called the prior odds of H_0 to H_1 and

$$O(H_0 | r, n) = P(H_0 | r, n) / P(H_1 | r, n) = P(\theta \leq \theta_0 | r, n) / P(\theta > \theta_0 | r, n) \quad (2)$$

is called the posterior odds ratio of H_0 to H_1 . The Bayes factor $F(H_0, H_1)$ is defined as the ratio of posterior odds to prior odds in favor of the null hypothesis,

$$F(H_0, H_1) = O(H_0 | r, n) / O(H_0) \quad (3)$$

The Bayes factor depends on the prior probability density function $g(\theta)$ of θ and $g_1(\theta)$ is

$$g(\theta) = \begin{cases} P(H_0)g_0(\theta) & \text{if } \theta \leq \theta_0 \\ P(H_1)g_1(\theta) & \text{if } \theta > \theta_0 \end{cases} \quad (4)$$

where g_0 and g_1 are proper probability density functions

$$(g_0(\theta) > 0, \int_0^{\theta_0} g_0(\theta) d\theta = 1, g_1(\theta) > 0, \int_{\theta_0}^1 g_1(\theta) d\theta = 1) .$$

They describe the distribution of θ over the two hypotheses. The probability mass function of r given n and θ is the binomial probability $f(r | \theta, n) = C_r^n \theta^r (1 - \theta)^{n-r}$, if $r = 0, 1, \dots, n$ and zero elsewhere,

$$C_r^n = \frac{n!}{r!(n-r)!} .$$

It can be shown that the Bayes factor is

$$F(H_0, H_1) = \frac{\int_0^{\theta_0} f(r | \theta, n) g_0(\theta) d\theta}{\int_{\theta_0}^1 f(r | \theta, n) g_1(\theta) d\theta} . \quad (5)$$

In the numerator, $f(r | \theta, n)$ is weighted by the prior distribution of θ under the null hypothesis and in the denominator it is weighted by the prior distribution of θ under the alternative hypothesis.

If we require $P(\theta \leq \theta_0 | r, n) = C_o$, this is equivalent to requiring

$$F(H_0, H_1) = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)} . \quad (6)$$

If θ has a prior distribution on the interval (a, b) where $a < b$, with the following probability density function:

$$f(\theta | \alpha, \beta) = \frac{1}{B(\alpha, \beta)(b-a)^{\alpha+\beta-1}} (\theta-a)^{\alpha-1} (b-\theta)^{\beta-1} \quad (7)$$

where $\alpha > 0, \beta > 0$, $B(\alpha, \beta) = \Gamma(\alpha) \Gamma(\beta) / \Gamma(\alpha+\beta)$, then it is said to have a Beta distribution on (a, b) with parameters α and β . Symbolically we write $\theta \sim \text{Beta}_{(a,b)}$

(α, β) . The tractable and rich family of probability distributions for θ under H_0 and H_1 are $Beta_{(0, \theta_0)}(\alpha_0, \beta_0)$ and $Beta_{(\theta_0, 1)}(\alpha_1, \beta_1)$ distributions. If $\alpha_0 = \beta_0 = 1$ then θ has a uniform distribution on $(0, \theta_0)$ under H_0 , $g_0(\theta) = 1/\theta_0$ if $0 < \theta < \theta_0$ and zero elsewhere. Similarly if $\alpha_1 = \beta_1 = 1$ then θ has a uniform distribution on $(\theta_0, 1)$ under H_1 , $g_1(\theta) = 1/(1 - \theta_0)$ if $\theta_0 < \theta < 1$ and zero elsewhere.

Suppose that no failures have been encountered during testing ($r=0$). We want to determine the number of tests required so that we are $100 * C_o\%$ confident that $\theta \leq \theta_0$. That is, we want to determine n such that $P(\theta \leq \theta_0 | r, n) = C_o$, for some C_o close to one. Suppose we take non-informative uniform prior distributions for θ under H_0 and H_1 , $\theta | H_0 \sim Uniform(0, \theta_0)$ and $\theta | H_1 \sim Uniform(\theta_0, 1)$. Bayes factor when $r=0$ is

$$F(H_0, H_1) = \frac{(1 - \theta_0)[1 - (1 - \theta_0)^{n+1}]}{\theta_0(1 - \theta_0)^{n+1}} \quad (8)$$

Since $P(\theta \leq \theta_0 | r, n) = C_o$ if and only if

$$F(H_0, H_1) = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)}, \text{ we need to solve}$$

$$\frac{(1 - \theta_0)[1 - (1 - \theta_0)^{n+1}]}{\theta_0(1 - \theta_0)^{n+1}} = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)}$$

for n . We have

$$n = - \frac{\ln \left[\frac{C_o \theta_0 P(H_1)}{(1 - C_o)(1 - \theta_0) P(H_0)} + 1 \right]}{\ln(1 - \theta_0)} - 1. \quad (9)$$

Furthermore if $P(H_0) = \theta_0$, $P(H_1) = 1 - \theta_0$ then

$$n = \frac{\ln(1 - C_o)}{\ln(1 - \theta_0)} - 1 \quad (10)$$

This result has been reported in Section 3, by taking a uniform prior distribution for θ on $(0, 1)$. If $g(\theta) = 1$ for $0 < \theta < 1$ and zero elsewhere then, this implies that $P(H_0) = \theta_0$, $P(H_1) = 1 - \theta_0$ and from (4), $g_0(\theta) P(H_0) = 1$ for $0 < \theta < \theta_0$ and, $g_1(\theta) P(H_1) = 1$ for $\theta_0 < \theta < 1$ or that θ has uniform distribution on $(0, \theta_0)$ under H_0 and a uniform distribution on $(\theta_0, 1)$ under H_1 . Hence (10) is a special case of (9). Since θ_0 is a number close to zero (such as 10^{-3} , 10^{-4} , etc.) and since taking a uniform distribution on $(0, 1)$ for θ implies $P(H_0) = \theta_0$ (very small prior probability is assigned to H_0), it will require a relatively large n to achieve $P(\theta \leq \theta_0 | r=0, n) = C_o$ when C_o is large, $C_o = 0.95, 0.99$ etc. However, note

the difference in the number of tests required by this methodology and the ones reported in earlier sections of this report.

In our derivation of the number of necessary tests given by (9), taking $Uniform(0, \theta_0)$ distribution for θ under H_0 makes sense especially when θ_0 is very small. Note that these values are provided in Table 4. However taking uniform $(\theta_0, 1)$ distribution for θ under H_1 may not be appropriate. If H_1 was true, we probably feel that expected value of θ under H_1 is close to θ_0 and not equal to $\theta_0 + (1 - \theta_0)/2$ as is the case with $Uniform(\theta_0, 1)$ distribution. If $\theta | H_1 \sim Beta_{(\theta_0, 1)}(\alpha_1, \beta_1)$ then the prior expected value of θ is

$$E(\theta | H_1) = \theta_0 + (1 - \theta_0) \frac{\alpha_1}{\alpha_1 + \beta_1}. \text{ If we take } \alpha_1 = 1 \text{ and}$$

for some small δ take $E(\theta | H_1) = \theta_0 + \delta$ then β_1 is $\frac{1 - \theta_0}{\delta} - 1$ and the necessary number of tests is given

by the solution of the following equation for n :

$$\frac{[(n - 1)\delta + (1 - \theta_0)][1 - (1 - \theta_0)^{n+1}]}{\theta_0(n + 1)(1 - \theta_0 - \delta)(1 - \theta_0)^n} = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)} \quad (11)$$

If no restrictions on $\alpha_0, \beta_0, \alpha_1, \beta_1$ are imposed, then we need to solve the following equation for n :

$$\frac{B(\alpha_1, \beta_1)}{B(\alpha_0, \beta_0)} (1 - \theta_0)^{\sum_{i=0}^n C_i^n (-\theta_0)^i B(i + \alpha_0 + \beta_0)} = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)} \quad (12)$$

In general, if r failures are encountered, and no restrictions on $\alpha_0, \beta_0, \alpha_1, \beta_1$ are imposed then we need to solve the following equation for n :

$$\frac{B(\alpha_1, \beta_1)}{B(\alpha_0, \beta_0)} \frac{\theta_0^r}{(1 - \theta_0)^{n-r}} \frac{\sum_{i=0}^{n-r} C_i^{n-r} (-\theta_0)^i B(i + \alpha_0, \beta_0)}{\sum_{i=0}^r C_i^r \theta_0^{r-i} (1 - \theta_0)^i B(\alpha_1 + i, n - r + \beta_1)} = \left(\frac{C_o}{1 - C_o} \right) \frac{P(H_1)}{P(H_0)} \quad (13)$$

The values presented in Table 4 are the solutions to different instances of equation (13), obtained by programs efficiently written in Matlab or Mathematica environments.