A quantitative software testing method for hardware and software integrated systems in safety critical applications

Hai Tang[a], Lixuan Lu*[a]
[a] University of Ontario Institute of Technology, Oshawa, ON, Canada

Abstract: Most of today's Safety Instrumented Systems (SIS) are hardware and software integrated systems. In these systems, failures can occur in both hardware and software. Hardware failures and their effects have been studied extensively in the literature. However, the methods and results dealing with hardware failure are not directly applicable for software reliability modeling, due to the difference of nature between hardware and software. This is especially of concern when the SIS is used for safety critical applications. In this paper, a hardware and software integrated reliability model is proposed to model the reliability of the integrated system. The requirement on software reliability is then determined based on the hardware reliability and the requirement on the Safety Integrity Level (SIL) of the integrated system. Following this, a Bayesian stopping rule is used to determine the minimal number of successful software runs, in order to provide a certain level of confidence that the reliability requirement on the software is achieved.

Keywords: Probabilistic risk assessment (PRA)/probabilistic safety assessment (PSA), Safety instrumented systems (SIS), Hardware and software integrated reliability model, Bayesian stopping rule, Software reliability demonstration test (SRDT).

1. INTRODUCTION

Most of today's Safety Instrumented Systems (SIS) are hardware and software integrated systems, used in safety critical applications, such as in nuclear power plant control, chemical processes and machine guarding. They are used to detect hazardous events and perform safety-related functions to reduce the risk of human injury and fatality. Although there have been many methods proposed in literature for probabilistic risk assessment (PRA)/probabilistic safety assessment (PSA) of instrumentation and control systems, most of the analyses are geared towards hardware failures and their effects. Quantitative reliability modeling and analysis of hardware and software integrated systems still pose great challenges, especially in the situation when the a quantitative software reliability assessment is taken into consideration.

Software and hardware fail differently. This is caused by many fundamental aspects. Hardware is the physically connected components that perform or support a function within the system. It normally has a time-related failure rate because of the hardware wear out process. Hardware failure occurs when some form of stress exceeds the associated strength of the product. On the other hand, software is a collection of instructions which enables a controller to perform a specific task on the hardware platform, and software cannot work alone without hardware. Software failure mechanisms are different from hardware failures in that all software failures are inadvertently designed into the system [1]. Software does not wear out as hardware, thus software failure is time-independent.

When building reliability models for hardware and software integrated systems, there are many factors to consider. First of all, due to the fundamental difference between hardware and software failure mechanisms, it is obvious that the methods developed to model the hardware reliability are not suitable for software reliability modeling, and vice versa. Software failures are the design defects highly related to design process management and developers' skills and experience. It is not easy to quantitatively incorporate all these factors with existing reliability models, although many software

*Contact Author: Lixuan.Lu@uoit.ca

reliability models have been developed for reliability assessment of different phases of the software lifecycle [2, 3]. All of these models have their own weakness, and none of them can cover the whole software lifecycle. Moreover, general purpose software reliability models always require failure data during the development, testing and operating phase of software lifecycle, which can be difficult to obtain for safety critical software. Software used in SIS requires an extremely high reliability, whose failure is very rare or may be never found from the software testing.

An integrated reliability model is developed in this paper to analyze the Safety Integrity Level (SIL) of an integrated SIS, in which the time-independent software reliability model is coupled with the time-related hardware model to calculate the average probability of failure on demand (PFDavg). A reliability demonstration test is designed with Bayesian stooping rules, which gives a proof with certain level of confidence that the software is able to respond successfully when a SIS is required to perform a safety function on demand.

2. SAFETY INTEGRETY LEVEL

2.1. Average Probability of Failure on Demand

IEC 61508 [4] is an international standard for functional safety of systems. It specifies the requirement on the entire safety lifecycle management of a SIS, including system design, development and certification stages. According to the standard, the safety analysis procedure requires a quantitative determination of the risk reduction by using the safety system as a protection layer to the Equipment Under Control (EUC).

Safety Integrity level (SIL) is used as a measure of the risk reduction achieved by the safety system when a hazardous event occurs in the EUC. The probability of a SIS failing dangerously is the Probability of Failure on Demand (PFD). This represents the risk that the safety system fails to perform safety functions as designed when a demand for the safety function occurred. Assuming a single channel safety system has a dangerous failure rate λ_D, the PFD is the probability of a dangerous failure and is shown in Eq. (1):

$$PFD(t) = 1 - e^{-\lambda_D t} \tag{1}$$

Safety systems usually have an extremely low failure rate, which allows the PFD to be calculated by the approximated Eq. (2) in real applications.

$$PFD(t) = \lambda_D t \tag{2}$$

PFD average (PFDavg) is the average of probability of failure on demand, which is defined in Eq. (3):

$$PFDavg = \frac{1}{T}\int PFD(t)dt \tag{3}$$

2.2. Safety Integrity Level (SIL)

The *PFDavg* quantitatively represents the probability that a safety system fails performing the designed safety function when a demand occurs. The SIL is an index number uesed in real applications to indicate system risk reduction levels according to system *PFDavg* values. As shown in Table 1, four SILs, ranking form SIL 1 to SIL 4, are specified to show different ranges of system PFDavg. Systems with a higher SIL have a better risk reduction to the EUC. A lower *PFDavg* corresponds to a higher SIL, which means the system has a higher probability to perform a safety

function correctly when a hazardous event occurs. Safety systems usually use redundancy on the component level or system level to achieve a certain SIL.

Table 1: Safety Integrity Level

Safety Integrity Level	Probability of failure on demand (PFDavg)	Risk Reduction factor (RRF)
1	10^{-1} to10^{-2}	10 to 100
2	10^{-2} to10^{-3}	100 to 1000
3	10^{-3} to10^{-4}	1000 to 10000
4	10^{-4} to10^{-5}	10000 to 100000

3. SOFTWARE RELIABILITY

3.1. Input Domain Based Model

Software plays a more and more important role in modern control systems, due to the fact that most control functions generated by the programmable logic controllers are software based functions. Safety functions performed by SIS require an interaction between hardware and software, in which the software processes the reading of the sensors and provides an output as the control signal to the EUC. SIS will fail into a dangerous condition if the software has an incorrect response to the hazardous event when the SIS is expected to perform a safety function.

Software reliability is a quantitative measurement of how well the software system work according to the design specifications. Unlike hardware, software does not wear out over time. There are many software reliability models in the literature to model software reliability in different stages of the software lifecycle. Reliability growth models are widely used in industrial applications. They provide a rudimentary estimation on system reliability and can be used to support project management. However, the assumptions made to use these models are still questionable and the characteristics of the software under evaluation are insufficiently accounted for. In addition, the techniques used to estimate the software reliability require a failure history. This can be difficult to obtain for safety critical systems. Because software in these systems normally ha a much higher reliability level than general purpose software and it rarely or never fails during testing [5].Input domain based models are more advantageous over the reliability growth models when modeling the reliability of safety critical software, because the method does not rely on empirical assumptions and the result is acquired through a direct statistical approach according to the probability of successful execution in software testing processes.

In an input domain based model [6], *n* inputs are randomly selected from the input data set $E=(E_i: i=1, 2, \dots, N)$, where E_i is the subset of software input domain. The inputs are sampled with the input distribution of operational profile $P=(P_i: i=1, 2, \dots, N)$; where P_i is the probability of choosing E_i as the input. If *f* failures are found by the execution of *n* inputs, then the software reliability can be estimated as:

$$R = 1 - f / n \tag{4}$$

When using the input domain based model, a prior knowledge of the system operational profile is required for testing purposes. The operational profile is a quantitative characterization of how the software is used in the real applications. A step by step approach to develop the operation profile for software testing can be found in [7]. For safety critical software test, the operational profile could be developed according to the operational history of a plant or by expert knowledge. If the input distribution of the software is unknown, then the operational profile can be developed by assuming a uniform input distribution over the software input domain.

3.2. Application of Input Domain Based Model for Safety Critical Software

Numerically, if no fault is detected from a software test, then the reliability of software can be estimated as 1. However, a realistic software product will never achieve 100% reliability unless all combinations of all possible inputs are tested and they all give the correct results as expected. To achieve this goal, an infinite number of tests are required. This is obviously impossible to use in real applications due to the time and cost for testing. There must be a stopping rule for software test to determine the minimal number of successful runs to provide a certain level of confidence that the software achieves its reliability goal.

In this paper, a hardware and software integrated reliability model is developed to calculate the requirement of software reliability when the software and software are integrated as a system to meet a probability of failure on demand for a certain SIL. Afterwards, a Bayesian stopping rule for safety critical software testing is applied to calculate the minimal executions before the termination of the test.

4. HARDWARE AND SOFTWARE INTEGRATED RELIABILITY MODEL

4.1. Hardware and Software Integrated System

An integrated reliability model is developed to model the system reliability of hardware and software integrated systems. The hardware subsystem is the physically interconnected devices including sensors, controllers and actuators. The software subsystem is the programs running on the hardware platform which enable the system to perform a specific task. The difficulties of reliability modeling of hardware and software integrated systems are caused by the natures of how hardware and software fail. Hardware fails over time. The reliability of hardware usually follows time-related distributions, such as exponential distribution or Weibull distribution. On the other hand, software failures are time-independent. Software does not wear out. The failures of software are the errors designed into the systems. When doing quantitative reliability assessment of hardware and software integrated systems, a model is needed to couple the time-related hardware reliability model and time-independent software reliability model together.

4.2. Integrated Reliability Modeling

To have the SIS respond successfully when a safety function is on demand, the hardware should work in a failure-free condition and the software should process the output properly as designed and give a correct control command to the hardware. Failures in either hardware or software could cause a failure on demand for the SIS.

With a dangerous failure rate $\lambda_{D,hw}$, the probability of failure on demand of hardware ($PFD_{hw}(t)$) can be calculated by Eq. (5) below:

$$PFD_{hw}(t) = 1 - e^{-\lambda_{D,hw}t} = \lambda_{D,hw}t \tag{5}$$

Software reliability R_{sw} is a time-independent value and it can be calculated by Eq. (6):

$$R_{sw} = 1 - F_{sw} \tag{6}$$

where F_{sw} is the probability of failure of the software. The reliability target of the software (R_{sw}) needs to be calculated from the integrated system reliability model as the reliability goal for software testing.

Use PFD_{sys} to represent the probability of failure on demand of the integrated system when the hardware and software are working together to respond to a safety function on demand. The system will fail when "hardware fails" OR "software fails". Therefore, the probability of failure on demand of the integrated system is calculated as shown in Eq. (7):

$$PFD_{sys}(t) = PFD_{hw}(t) \cup F_{sw} = PFD_{hw}(t) + F_{sw} - PFD_{hw}(t) \times F_{sw} \tag{7}$$

The average probability of fail on demand of the hardware and software integrated system ($PFDavg_{sys}$) can be calculated as Eq. (8):

$$PFDavg_{sys} = \frac{1}{T}\int PFD_{sys}(t)dt \tag{8}$$

Substituting Eq. (7) into Eq. (8), Eq. (9) is obtained:

$$PFDavg_{sys} = \frac{1}{T}\int (PFD_{hw}(t) + F_{sw} - PFD_{hw}(t) \times F_{sw})dt \tag{9}$$

where T is usually referred to as the interval of proof test, which brings the system back to its original state after operating continuously for a period of time. Since the software reliability is time-independent, F_{sw} can be treated as a constant value when doing the integration, thus Eq. (9) is rearranged as:

$$PFDavg_{sys} = \frac{1}{T}\int PFD_{hw}(t)dt + F_{sw} - F_{ws} \times \frac{1}{T}\int PFD_{hw}(t)dt \tag{10}$$

Substituting Eqs. (6) and (8) into Eq. (10) gives:

$$PFDavg_{sys} = PFDavg_{hw} + (1 - R_{sw}) - (1 - R_{sw}) \times PFDavg_{hw} \tag{11}$$

4.3. Software Reliability Requirement

The reliability of the software can be calculated by solving Eq. (11), which yields:

$$R_{sw} = \frac{1 - PFDavg_{sys}}{1 - PFDavg_{hw}} \tag{12}$$

To meet a given requirement on SIL n (n=1, 2, 3, 4), the *PFDavg* of the hardware and software integrated system must be lower than the maximal allowed average probability of failure on demand ($PFDavg_{sys,max,n}$) for that SIL. Thus, the software reliability must be higher than the critical software reliability ($R_{sw,critical,n}$), which is the software reliability value that makes the $PFDavg_{sys}$ equals to $PFDavg_{sys,max,n}$. According to Eq. (12), the software reliability requirement of a SIL n hardware and software integrated system is:

$$R_{sw,critical,n} = \frac{1 - PFDavg_{sys,\max,n}}{1 - PFDavg_{hw}} \tag{13}$$

5. BAYESIAN STOPPING RULE FOR SOFTWARE TESTING

5.1. Bayesian theory for software testing

After the software reliability requirement for the system is determined, the Software Reliability Demonstration Test (SRDT) is performed to verify that the software meets the reliability requirement with an acceptable degree of confidence. A stopping rule for the software reliability testing is required to make a decision of whether the test should be continued or terminated. When testing a safety critical system, the stopping decision could be made based on Bayesian analysis of the testing results [8].

There are two possible results for each test: fail or pass. If the SIS performs correctly when a safety function is on demand, the test is passed. If the system fails to response properly for a safety function demand, the test is failed. Assuming the value of probability of failure for software is *p*, the number of failures *f* from *n* tests follows a Binomial distribution:

$$P(F = f) = C_f^n \; p^f (1-p)^{n-f} \tag{14}$$

A prior conjugate is used to represent the changes of the parameter of interest *p* with extra information gathered from the test. The conjugate distribution follows a Beta(*a*,*b*) distribution:

$$f(p) = \frac{p^{a-1}(1-p)^{b-1}}{B(a,b)} \tag{15}$$

Where *B(a,b)* is the Beta function, and *a* and *b* are parameters chosen by the assessor to represent the prior knowledge of the parameter *p*. casein the case where, there is no information available for selecting *a* and *b*, a uniform prior can be used with *a*=*b*=1.

If *f* failures are found from *n* tests, the posterior distribution of *p* is Beta(*a*+*f*,*b*+*n*-*f*):

$$f(p \mid f, n, a, b) = \frac{p^{a+f-1}(1-p)^{b+n-f-1}}{B(a+f, b+n-f)} \tag{16}$$

For uniform prior, Eq. (16) becomes:

$$f(p \mid f, n, 1, 1) = \frac{p^f (1-p)^{n-f}}{B(1+f, 1+n-f)} \tag{17}$$

5.2. Stopping rule based on Bayesian decision theory

If F_{sw} is used to represent the maximum probability of failure for software, and C is used to represent the confidence level of the test result, the reliability requirement for a software test could be expressed as:

$$P(p < F_{sw}) \geq C \tag{18}$$

To meet the requirement on reliability and confidence, the smallest value of successful execution without failure n_1 is the smallest value of *n* which satisfies Eq. (19):

$$\int_0^{F_{sw}} \frac{(1-p)^n}{B(1,1+n)} dp \geq C \tag{19}$$

If one failure occurs after s_1 (s_1<n) executions, the posterior distribution for *p* becomes:

$$f(p \mid 1, s_1, 1, 1) = \frac{p(1-p)^{s_1-1}}{B(2, s_1)} \tag{20}$$

This is the new prior distribution of p for the following testing stage. The posterior distribution after n_2 failure free tests are observed is:

$$f(p \mid 1, s_1 + n_2, 1, 1) = \frac{p(1-p)^{s_1+n_2-1}}{B(2, s_1 + n_2)} \tag{21}$$

For a given requirement on F_{sw} and C, the smallest value n_2 for failure free execution in the following stage of test after the fault is fixed can be solved from Eq. (22):

$$\int_0^{F_{sw}} \frac{p(1-p)^{s_1+n_2-1}}{B(2, s_1 + n_2)} dp \geq C \tag{22}$$

To continue this process, if the jth failure occurs on the s_jth test executions, the number of failure free execution for the next test stage (n_{j+1}) can be computed by solving the general Eq. (23):

$$\int_0^{F_{sw}} \frac{p^j (1-p)^{\sum_{i=1}^{j} s_i + n_{j+1} - j}}{B(j+1, \sum_{i=1}^{j} s_i + n_{j+1} - j + 1)} dp \geq C \tag{23}$$

In real applications, a simplified stopping rule developed below can be used to reduce the calculation for the process as described above. If j failures occurred, let the total number of executions until n_{j+1} failure free tests observed in the $j+1$th stage of test be N.

$$N = s_1 + s_2 + \cdots + s_j + n_{j+1} \tag{24}$$

N is the minimum number of executions required to successfully demonstrate the required level of reliability with predetermined acceptable confidence level. Regardless of when these failures are happened during the test, this test process can be treated equivalently as a single test process in which j failures are observed out of N executions. To meet the requirement of software reliability, the total executions N that contains j failures should be the minimum value of N which satisfies Eq. (25):

$$\int_0^{F_{sw}} \frac{p^j (1-p)^{N-j}}{B(j+1, 1+N-j)} dp \geq C \tag{25}$$

According to the analysis above, the stopping rule for the software testing only depends on the total number of executions and the total number of failures out of these executions. Given a reliability requirement F_{sw} and C, the stopping rule for reliability test can be calculated before the test is carried out.

6. NUMERICAL EXAMPLE

This part of paper will give a numerical example to demonstrate the application of the methodology developed above.

Assuming a hardware and software integrated system is used as an emergency shutdown system for a chemical plant to stop the process in EUC when a hazardous event occurs. The dangerous failure rate λ_D for the hardware system is evaluated as 1.6×10^{-7}/h and proof testing will perform every year (8760 hours), which brings the safety instrumented system back to its original state. According to the hazard analysis of the chemical plant, a SIL 3 is required for the emergency shutdown system.

The software reliability requirement for the given SIL can be calculated from the SIL requirement and hardware failure rate. From Eq. (5), the probability of failure on demand of the hardware subsystem is:

$$PFD_{hw}(t) = 1.6\times10^{-7}t \tag{26}$$

The $PFDavg_{hw}$ of hardware is the integral average of hardware PFD over the time of proof testing interval T, and is calculated as below:

$$PFDavg_{hw} = \frac{1}{T}\int PFD_{hw}(t)dt = \frac{1}{T}\int_0^T \lambda_D t dt = \frac{\lambda_D T}{2} = \frac{1.6\times10^{-7}\times8760}{2} = 7\times10^{-4} \tag{27}$$

For a SIL 3 application, the maximal allowed $PFDavg_{sys}$ for the system is 10^{-3}, thus the software reliability requirement for the system is calculated from equation (13) as:

$$R_{sw,critical,3} = \frac{1-PFDavg_{sys,\max,3}}{1-PFDavg_{hw}} = \frac{1-10^{-3}}{1-7\times10^{-4}} = 0.9997 \tag{28}$$

The probability of failure for software is therefore:

$$F_{sw} = 1-R_{sw,critical,3} = 1-0.9997 = 3\times10^{-4} \tag{29}$$

A uniform prior distribution is used for the Bayesian analysis, where a=b=1. If a 99% confidence on the software reliability is required by the software test, the minimal number of executions with j failures is calculated by solving Eq. (30) below based on Eq. (25).

$$\int_0^{3\times10^{-4}} \frac{p^j(1-p)^{N-j}}{B(j+1,1+N-j)}dp \geq 0.99 \tag{30}$$

This equation is solved by using numerical method. Based on different numbers of system failures, the solution of Eq. (30) gives the stopping rule for the system safety test. The results are shown in Table 2. As can be seen, when the number of failures during test increases, the total number of required executions increase as well, in order to be 99% confident that the required software reliability level is achieved.

7. CONCLUSION

In this paper, a quantitative method is developed to determine the minimum number of testing required on software in hardware and software integrated systems, given the required Safety Integrity Level (SIL) on the overall system and hardware failure data. Due to the fundamental differences between hardware and software failure mechanisms, hardware and software are usually analyzed separately in practice by using distinct methodologies. Hardware failures are typically time-related with a certain failure rate λ, while the software failures are time-independent, since errors are inevitably designed into the final software product. To better model hardware and software integrated systems for

Table 2: Stopping rule for software testing

Numbers of failures, j	Total number of executions, N
0	15347
1	22124
2	28016
3	33479
4	38677
5	43690
6	48563
7	53328
8	58003
9	62604
10	67143
11	71627
12	76063

reliability analysis and to provide a practical method for reliability assessment, a reliability model for hardware and software integrated systems is developed in this paper to model the effect of both hardware and software on system safety and to produce a quantitative assessment result. Using this model, the requirement for software reliability is first determined based on hardware reliability and system SIL. A Software Reliability Demonstration Test (SRDT) based on a Bayesian stopping rule is then used to calculate the minimum number of executions that is required to prove the reliability requirement is achieved. One advantage of the method proposed in this paper compared with existing methods is that it couples both hardware and software reliability within the system. In addition, unlike the software reliability growth models based on empirical assumptions, the method proposed here measures the software reliability via the input domain based model, with which the result of the assessment could come directly from the observation of the testing. Finally, as shown in the numerical example, this method is practical and easy to use in real life applications.

References

[1] William M. Goble, "*Control systems safety evaluation and reliability*", International Society of Automation, 2010.

[2] C.V. Ramamoorthy and F.B. Bastani, "*Software Reliability—Status and Perspectives*", IEEE Transactions on Software Engineering, vol.SE-8, pp.354-371, (1982).

[3] A.L. Goel, "*Software Reliability Models: Assumptions, Limitations, and applicability*", IEEE Transactions on Software Engineering, vol.SE-11, pp. 1411-1423, (1985).

[4] International Electrotechnical Commission, "*IEC 61508 Second Edition: Functional Safety of Electrical/Electronic/Programmable Electronic Systems*", (2010).

[5] A. Pasquini, E. De Agostino and G.D. Di Marco, "*An input-domain based method to estimate software reliability"*, IEEE Transactions on Reliability, vol. 45 , no. 1, pp. 95-105, (1996).

[6] E. Nelson, "*Estimating software reliability from test data*", Microelectronics Reliability, vol. 17, pp. 67-73, (1978).

[7] J.D. Musa, "*Operational profile in software-reliability engineering*", IEEE Software, vol. 10, pp. 14-32, (1993).

[8] B. Littlewood and D. Wright, "*Some conservative stopping rules for the operational testing of safety critical software*", IEEE Transactions on Software Engineering, vol. 23, pp. 673-683, (1997).