# A Probabilistic Assessment Method for Software Reliability in the Application of Safety Standards

**YANG Yaguang**[1]

*1. Office of Research, Nuclear Regulatory Commission, Rockville, 20850, USA (yaguang.yang@nrc.gov)*

**Abstract:** Safety standards for nuclear power plant design normally address the most critical requirements and general principles. These requirements and principles are important guidance in the practice of the engineering design for the nuclear power plants. However, verifying that some of these requirements are met can be difficult because of lacking consensus methods. One of the example is clause 5.76 of requirement 42 in IAEA "Safety of Nuclear Power Plant: Design", which states "The design shall take account of the probabilistic safety analysis of the plant for all modes of operation and for all plant states." To implement this requirement, one may use the more specific IAEA safety guide such as No. SSG-3 "Development and Application of Level 1 Probabilistic Safety Assessment for Nuclear Power Plants", which indicates that "at present it is not possible to derive a probabilistic model of software failure". In this paper, we try to close the gap by proposing a systematic and probabilistic method to estimate the software failure probability based on software test data.

**Keyword:** Reliability, software reliability, flow chart, Bayesian method

## 1 Introduction

IAEA "Safety of Nuclear Power Plant: Design" [1] is one of the most important IAEA standards which address the critical requirements and general principles. The implementation of the standard is normally based on more specific IAEA safety guide such as No. SSG-3 "Development and application of level 1 probabilistic safety assessment for nuclear power plants" [2] and/or more detailed IAEA technical reports such as "Application of probabilistic safety assessment (PSA) for nuclear power plants" [3]. However, gaps may exist between these documents. For example, clause 5.76 of requirement 42 in [1] states that "The design shall take account of the probabilistic safety analysis of the plant for all modes of operation and for all plant states", but a statement in [2] indicates that "at present it is not possible to derive a probabilistic model of software failure." In reality, new reactor designs and existing reactor system updates rely heavily on the equipment with DI&C components/systems which include software. There is increasing interest in reliability and safety analyses for safety-critical DI&C systems in international organizations, such as United States Nuclear Regulatory Commission.

To estimate the reliability for DI&C equipment with software is challenging because unlike hardware-only analog equipment, the software needs special consideration as its failure mechanism is unique; the reliability estimation method for a software system should be different from that used for hardware.

Office of Research, Nuclear Regulatory Commission, Rockville, 20850, USA.

Many attempts have been made to find practical ways to improve software reliability and to estimate the reliability. Dennis Lawrence [4] discussed activities that should be carried out throughout the software life cycle. Parnas, Asmis, and Madey [5] emphasized documentation requirements and quality control, including testing and reviews. Leveson and Harvey [6] proposed software fault tree analysis method. However, the most extensively investigated software reliability method is the software reliability growth model [7]. Many statistical models, such as the exponential distribution, Poisson distribution, Weibell distribution, and Gamma distribution models have been considered, and many different scenarios have been discussed [8-11]. Nonetheless, there is no formed consensus on an ideal software reliability estimation method. Because of this difficulty, in some nuclear plant design probability risk analysis (PRA), for example ESBWR (economic simplified boiling water reactor) PRA analysis described in [12], assumed that software reliability is some constant without giving a method to validate the number, which is far less than ideal.

In this paper, we restrict our discussion to single thread software, which, on one hand, makes the discussion simple, and on the other hand, should cover a class of software that is used in safety critical system, for which the design simplicity, the deterministic execution time, and the reliability estimation are likely required. We propose a refinement for a recently developed software reliability estimation method given in [13-14] to make it rigorously based on probability theory and test data rather than subjective judgement. The remainder of the paper is organized as follows. Section 2 presents a flow network model for software structure and a simple example is given to describe it. Section 3 introduces a software failure rate estimation method with a Bayesian estimation refinement. Section 4 discusses an automated tool for failure rate calculation. The last section gives the conclusions of this paper.

## 2 Software component failure model

Software failures are fundamentally different from hardware failures. Typical hardware failures are due to wear out or aging-related failures. Therefore, hardware failure models are based on a random failure time that can be described by exponential, Poisson, Weibull, or Gamma distributions, for example. However, typical software failures are due to undetected human errors in certain parts of the software and failures are triggered by combinations of specific events and input data sets. A failure occurs when triggering events direct software to execute a problematic part of the software and a triggering data set is in use. Therefore, software failure models based on a random failure time may be inappropriate. Instead, we should consider software-specific failure characteristics while developing a useful software reliability model and introducing a software reliability estimation method. In particular, we model the software failure probability using a binomial distribution. If a piece of software contains some error(s), then there is a probability that the software with error(s) will fail if some triggering event occurs and if a triggering data set is in use. Moreover, a test will catch the failure when this occurs. Therefore, the failure rate of a piece of software is then introduced according to the ratio of the number of times the software fail the test to the total times this piece of software is tested.

Though there is no consensus on a method to be used for software reliability assessments, we believe that a recently developed test-based method is well suited for software reliability estimations [13-14]. It was suggested in [13-14] that the structure of the software should be taken into consideration in software reliability assessments because (a) it reflects the individual software complexity, and (b) tests are not equally executed in every line of code. This lower level of detail

should give better reliability estimations than the black box model [15] because more information is used in the estimation. To simplify our presentation and save space, we focus on single-thread software, which is the most likely case for safety systems.

We use a flow network to model the software structure. Let *source* denote the start point of the software; *sink* denote the end of the software; n *nodes* $n_i \in N$ represent the logic branch or converging points; and *edges* $e_{ij} \in E$, $j = 1, \cdots, j_m$ denote the software code between node $i$ and the node next to $i$. If an edge is executed, then every line inside the edge is executed; i.e., no branch exists inside an edge. It is assumed that there is an infinite capacity in every edge, which means that each edge can have as many tests as desired. Using c/c++ language as an example, the nodes are collections of the beginning and the end of every function, the beginning and the end of every conditional block starting with 'if' or 'switch'; while the edges are collections of pieces of software between nodes that meet one of the following conditions: (a) between the lines of the start of each function and the first 'if', 'switch' or 'while'; (b) between the lines 'if' and the line 'else' or 'else if' or the end of 'if', or between the line 'else if' and the next 'else if' or the line that ends 'if'; (c) between the lines of 'case'; (d) between the lines after the end of 'if' or the line after the end of 'switch' and the line before the next 'if' or 'switch' or the line that ends the function. We use the following simple pseudo c/c++ code to describe the partition and the flow network concept.

```
Main(){                                          //node 1
Data initialization;                             //edge e11
    If condition A holds                         //node 2
    {
        Process data;                            //edge e21
        If data process success                  //node 3
        {
            Save result;                         //edge e31
        }
        Else if data process fail
        {
            Issue a warning;                     //edge e32
        }
    }
    Else if condition A does not hold
    {
        Print "condition fail";                  //edge e22
    }                                            //node 4
    Clean memories;                              //edge e41
}                                                //node 5
```

By applying the principles described above, the flow network model corresponding to the pseudo code is given in Fig. 1.

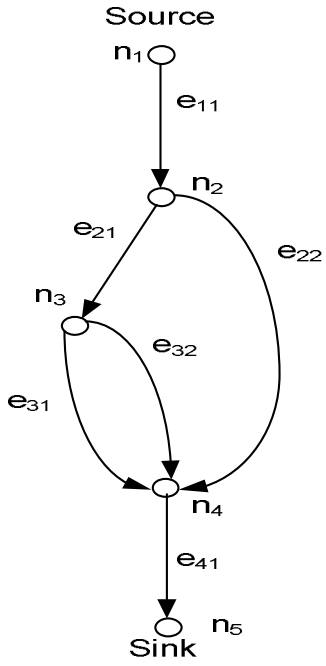


Figure 1. The flow network model of the pseudo code

# 3 Software Reliability Estimation

## 3.1 Reliability estimation for a single edge using Bayesian method

Let $T_{ij}$ be the average execution time of the edge $e_{ij}$ of the software and $h_{ij}$ be the total number of executions of $e_{ij}$ in the software test stage. If an edge $e_{ij}$ of the software is executed in a test scenario, we consider that the test scenario covers the edge $e_{ij}$. Let $e_{ij}=1$ denote an event in which edge $e_{ij}$ has been executed once and where the test result meets the expectations, and let $p_{ij}=P(e_{ij}=1)$ denote the probability of this event occurring. Let $e_{ij}=0$ denote an event in which edge $e_{ij}$ has been executed once and where the result fails to meet the expectations, and let $q_{ij}=P(e_{ij}=0)=1-p_{ij}$ be the probability of this event occurring. Therefore, $p_{ij}$ is the probability of having no error in $e_{ij}$ and $q_{ij}$ is the probability of having at least one error in $e_{ij}$. Clearly, the one-time test scenario follows a Bernoulli distribution. It is likely that some edges are executed more than once in the software test stage. These multiple-test scenarios follow a binomial distribution. Also, some edges may be tested more times than other edges. A main question in a multiple-test scenario is how to determine failure probability or reliability in these situations. The proposed failure rate estimation is based on the test result using the Bayesian method. Let $t_i$ be the outcome of the $i$ th test. The posterior density of $p_{ij}$ is sourced from earlier work, as follows [16]:

$$p(q_{ij}|h_{ij},k)=\frac{p(h_{ij},k|q_{ij})g(q_{ij})}{\int_0^1 p(h_{ij},k|q_{ij})g(q_{ij})dq_{ij}}, \tag{1}$$

where

$$f(t_1,...,t_{h_{ij}})=\int p(t_1\mid p_{ij})\cdots p(t_{h_{ij}}\mid p_{ij})g(p_{ij})dp_{ij}, \tag{2}$$

and $g(p_{ij})$ is a conjugate priori of distribution, which for binomial distribution, is a Beta distribution with a pair of parameters $(\alpha_{ij},\beta_{ij})$ [17] where $\alpha_{ij}>0$ and $\beta_{ij}>0$ are chosen to reflect any existing belief

or information about $e_{ij}$. If the piece of code $e_{ij}$ has been tested $n$ times, among these tests, $e_{ij}$ has passed the test $k$ times, then, the posterior density is also a Beta distribution with the parameter $(\alpha_{ij}+k, \beta_{ij}+n-k) > 0$ which is given as

$$p(p_{ij}) = \frac{1}{B(\alpha_{ij}+k, \beta_{ij}+n-k)} p_{ij}^{\,k+\alpha_{ij}-1}(1-p_{ij})^{n-k+\beta_{ij}-1}, \tag{3}$$

where $B(\alpha_{ij}+k, \beta_{ij}+n-k)$ is the Beta function. Given $n$ tests with $k$ success incident, the reliability estimation for a piece of code $e_{ij}$ is the mean of $p_{ij}$ which is given by

$$E(p_{ij} \mid k, n, \alpha_{ij}, \beta_{ij}) = \frac{\alpha_{ij}+k}{\alpha_{ij}+\beta_{ij}+n}. \tag{4}$$

In software testing practice, if a test does not meet the expectation, an investigation is conducted to find the error and a fix is made. The total test and successful times is reset. Therefore, we always have $n = k = h_{ij}$. But we may adjust parameters $(\alpha_{ij}, \beta_{ij}) > 0$ after a fix is made since we gain confidence as the software matures. To calculate failure probability, we simply use

$$q_{ij} = 1 - p_{ij}. \tag{5}$$

Therefore, at the end of the test stage, the software reliability can be modeled by a flow network whose structure is represented by nodes and edges, and each edge has its reliability determined by the test results and is estimated by (4). Based on this observation, it becomes possible to simplify the software reliability model by repeatedly combining either serial-connected edges or parallel-connected edges into a combined artificial edge. In the following discussions, we present the procedures and details pertaining to the combining of parallel-connected edges or serial-connected edges into a single artificial edge; we will also provide formulae to estimate the reliability of the combined artificial edge depending on whether these edges are serial-connected or parallel-connected edges.

### 3.2 Reliability estimation for parallel edge

First, for a block under node $n_i$ composed of $j_m$ parallel-connected edges, the total number of executions of all parallel-connected edges $e_{ij}$ during the test stage is

$$h_i = \sum_{j=1}^{j_m} h_{ij} \tag{6}$$

and the total execution time of all parallel-connected edges $e_{ij}$ under node $n_i$ during the test stage is the summation of the execution time multiplied by the number of executions of every edge, i.e.,

$$T_i = \sum_{j=1}^{j_m} T_{ij} h_{ij}. \tag{7}$$

Given that edge $e_{ij}$ has $h_{ij}$ executions in the test stage and each parallel edge with multiple tests follows a binomial distribution,

$$(p_{ij} + q_{ij})^{h_{ij}} = \sum_{k=0}^{h_{ij}} C_{h_{ij}}^{k} q_{ij}^{k} p_{ij}^{h_{ij}-k} = 1 \tag{8}$$

holds for every parallel-connected edge immediately under node $n_i$. As every edge in the parallel structure has its own binomial distribution $(p_{ij} + q_{ij})^{h_{ij}}$ and its own execution time $h_{ij}T_{ij}$, the

binomial distribution for the entire parallel structure which has a total execution time of $T_i$ should be a convex combination of the binomial distributions of individual edges weighted by $\frac{h_{ij}T_{ij}}{T_i}$; i.e., an edge that has longer execution time has a larger weight, and the summation of all of the weights is $\sum_{j=1}^{j_m}\frac{h_{ij}T_{ij}}{T_i}=1$. Therefore, the distribution of parallel edges should satisfy, considering (7) and (8), the following relationship:

$$\sum_{j=1}^{j_m}\frac{h_{ij}T_{ij}}{T_i}(p_{ij}+q_{ij})^{h_{ij}}=\sum_{j=1}^{j_m}\frac{h_{ij}T_{ij}}{T_i}=1. \tag{9}$$

Hence, immediately following (9), the reliability of the block composed of the parallel-connected edges under node $n_i$ is the event that every test has successfully passed. The probability of this event is given by

$$R_{i1}=\sum_{j=1}^{j_m}\frac{h_{ij}T_{ij}}{T_i}p_{ij}^{h_{ij}} \tag{10}$$

The number of executions of the combined artificial edge used in the next model reduction step is taken as the summation of the number of the executions of all of the edges in the parallel block,

$$H_{i1}=\sum_{j=1}^{j_m}h_{ij}\ . \tag{11}$$

The equivalent execution time for the combined artificial edge (from the parallel block) used in the next model reduction step is

$$T_{i1}=\frac{1}{H_{i1}}\sum_{j=1}^{j_m}T_{ij}h_{ij}\ . \tag{12}$$

The same method can be applied to the parallel-connected blocks, including blocks that are reduced to artificial edges.

### 3.3 Reliability estimation for serial edge

For a block under node $n_{i_1}$ composed of nodes $i_1,\ldots,i_s$ and serial-connected edges (there are no parallel-connected edges in all nodes $i_1,\ldots,i_s$), the total number of executions of all serial-connected edges $e_{ij}$ during the test stage is $h_i=h_{ij}$ for any $i\in i_1,\ldots,i_s$, and the total execution time of all serial-connected edges $e_{ij}$ under node $n_{i_1}$ during the test stage is the summation of the execution time multiplied by the number of executions of every edge, i.e.,

$$T_i=\sum_{i=i_1}^{i_s}T_{ij}h_{ij}\ . \tag{13}$$

Given that edge $e_{ij}$ has exactly $h_{ij}$ executions in the test stage and each serial edge with multiple tests follows a binomial distribution,

$$(p_{ij}+q_{ij})^{h_{ij}}=\sum_{k=0}^{h_{ij}}C_{h_{ij}}^{k}q_{ij}^{k}p_{ij}^{h_{ij}-k}=1 \tag{14}$$

holds for all serial-connected edges immediately under node $n_{i_1}$. For the serial-connected edges, the reliability of the entire block is the product of the reliabilities of the individual edges. Considering (14), the following relationship immediately holds:

$$\prod_{i=i_1}^{i_s}(p_{ij}+q_{ij})^{h_{ij}}=1. \quad (15)$$

Hence, the reliability of the block composed of serial-connected edges under node $n_{i_1}$ in the software is the event that every test has successfully passed. The probability of this event is given by

$$R_{i_1 j}=\prod_{i=i_1}^{i_s} p_{ij}^{\,h_{ij}}. \quad (16)$$

The number of executions of the combined artificial edge used in the next model reduction step is taken as the number of executions of any edge $h_{ij}$, where $i \in i_1,\ldots,i_s$, in the serial block, and

$$H_{i_1 j}=h_{ij}. \quad (17)$$

The equivalent execution time for the combined artificial edge (from the serial block) used in the next model reduction step is

$$T_{i_1 j}=\sum_{i=i_1}^{i_s} T_{ij}. \quad (18)$$

The same method can be applied to the serial-connected blocks, including blocks that are reduced to artificial edges.

### 3.4 Overall reliability estimation of the software

The overall reliability of the software is estimated as follows. First, construct the flow network as discussed in Section 2. As testing proceeds, repeatedly use (4) to update the reliability for each edge. The reliability of each edge is obtained when the test finishes. Given the reliabilities of all of the edges, one can use equations (10-12) to simplify parallel-connected edges into a single artificial edge and use equations (16-18) to simplify serial-connected edges into a single artificial edge. The software reliability is obtained by repeating the process until all of the edges are combined into a single artificial edge. We then obtain the total equivalent test time $T$ and the software reliability $R$. An example is used to describe the process in the next subsection.

### 3.5 An example

The pseudo c/c++ code example introduced in Section 2 is used to demonstrate how this software reliability estimation method works. The software partitioned as in Fig. 2 (a) has five nodes and six edges. Assume also that three tests are conducted. The first test path is $e_{11}e_{21}e_{31}e_{41}$, the second test path is $e_{11}e_{21}e_{32}e_{41}$, and the third test path is $e_{11}e_{22}e_{41}$. Assume further that the total test time is $T=.00011$ hours and $T_{ij}=.00001$ hours for every edge. Therefore, $h_{11}=h_{41}=3$, $h_{21}=2$, and $h_{22}=h_{31}=h_{32}=1$. Assume $\frac{\alpha_{11}+h_{11}}{\alpha_{11}+\beta_{11}+h_{11}}=\frac{\alpha_{41}+h_{41}}{\alpha_{41}+\beta_{41}+h_{41}}=0.999999$ for $e_{11}$ and $e_{41}$; thus, $p_{11}=p_{41}=0.999999$. Similarly assuming $\frac{\alpha_{31}+h_{31}}{\alpha_{31}+\beta_{31}+h_{31}}=\frac{\alpha_{32}+h_{32}}{\alpha_{32}+\beta_{32}+h_{32}}=\frac{\alpha_{22}+h_{22}}{\alpha_{22}+\beta_{22}+h_{22}}=0.99$ yields $p_{31}=p_{32}=p_{22}=0.99$, and assuming $\frac{\alpha_{21}+h_{21}}{\alpha_{21}+\beta_{21}+h_{21}}=0.99$ yields $p_{21}=0.9999$. The following

steps are used to obtain the reliability, starting from the blocks that are composed of only either parallel edges or serial edges:

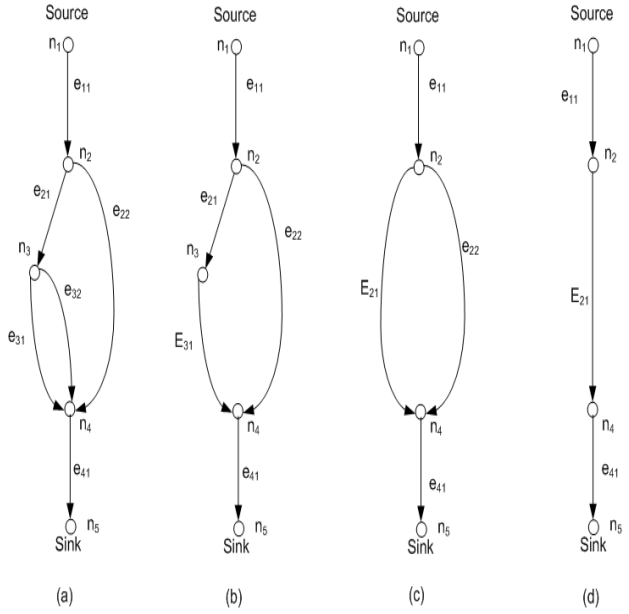


Figure 2. Reliability calculation procedures

- First, combining $e_{31}$ and $e_{32}$ gives $T_3 = h_{31}T_{31} + h_{32}T_{32} = 0.00002$, $\frac{h_{31}T_{31}}{T_3} = \frac{h_{32}T_{32}}{T_3} = \frac{1}{2}$; using (10) for the parallel edges $e_{31}$ and $e_{32}$, the flow network is reduced to Fig. 2 (b) with $R_{31} = 0.99$. Using (11) and (12), we obtain $H_{31} = \sum_{j=1}^{2} h_{3j} = 2$, and $T_{31} = \frac{1}{2}\sum_{j=1}^{2} T_{3j}h_{3j} = 0.00001$.

- Using (16) for the serial connection $e_{21}$ and $E_{31}$ to obtain the combined edge, the flow network is reduced to Fig. 2 (c) with $R_{21} = 0.9999^2 * 0.99^2$. Using (17) and (18), we have $T_{21} = \sum_{i=2}^{3} T_{i1} = 0.00002$, and $H_{21} = h_{21} = H_{31} = 2$.

- Considering the parallel connection in Fig. 2 (c), $T_2 = H_{21}T_{21} + h_{22}T_{22} = 0.00005$, $\frac{H_{21}T_{21}}{T_2} = \frac{4}{5}$, and $\frac{h_{22}T_{22}}{T_2} = \frac{1}{5}$, and using (7) for the parallel connection $E_{21}$ and $e_{22}$, we reduce Fig. 2 (c) to Fig. 2 (d) with $R_{21} = \frac{0.9999^2 * 0.99^2 * 4}{5} + \frac{0.99 * 1}{5}$. Using (11) and (12), we obtain $H_{21} = \sum_{j=1}^{2} h_{2j} = 3$ and $T_{21} = \frac{1}{3}\sum_{j=1}^{2} T_{2j}h_{2j} = 0.00001 * \frac{5}{3}$.

- Finally using (16) for serial connection $e_1$, $E_{21}$, and $e_{41}$, we have

$$R = 0.999999^3 * \left( \frac{0.9999^2 * 0.99^2 * 4}{5} + \frac{0.99 * 1}{5} \right) * 0.999999^3 = 0.981917$$

### 3.6 Software failure rate

For our analysis, the software failure rate can be obtained from the reliability assuming time dependency of the triggering conditions. Let $T$ be the total test time. Let the software reliability be $R$ during the total test time $T$; thus, the software failure probability is $(1-R)$ during the total test time $T$. Let $t$ be some unit time of the operational period, for instance one year, of continuous operation. Therefore, for the unit time of operational period $t$, the software failure rate is given by

$$(1-R)\frac{t}{T}. \tag{26}$$

In some cases, the software is not always running. It is executed only on demand. In this case, we need to modify the definitions of $T$ and $t$ slightly. Let $T$ be the total number of test runs and $t$ be the number of estimated demands in a unit time of the operational period (i.e., one year). In such a case, (26) continues to hold.

## 4 Automated Tool

It will be a tedious process if we directly apply the methods described in the previous sections without automated processes for software partition, data collection, edge reliability and total reliability estimations, as it is tedious to count and record all $n_i \in N$, $e_{ij} \in E$, $T_{ij}$, $T$, and $h_{ij}$ values manually; to update all $q_{ij}$ and $p_{ij}$ manually; and to estimate the overall software reliability manually using the model reduction method. However, with an automated tool, the estimation of the software reliability should be straightforward and free from human effort.

In many software development environments, the software structure, including the relationships between the calling and the called functions, is provided. For example, LabView provides this relationship in a tree structure. Therefore, it is possible, with some work, to develop a tool to generate the flow network structure.

Also, a number of popular operating systems and software development environments, such as Microsoft Visual Studio and vxWorks, can select different modes, such as debug and release modes. Different modes compile and run the software differently. For example, if the debug mode is selected, it can record the execution time for any part of the software under the test. Therefore,

techniques for determining the CPU times $T_{ij}$, $T$, and the number of executions $h_{ij}$ during the entire test stage are available.

We propose to add a test mode to the software development environment. It should have the following features:

1. When the software is compiled in the test mode, the tool should record *nodes* $n_i \in N$, *edges* $e_{ij} \in E$, and should create the flow network structure of the software.
2. When testing starts, in every test scenario run, the development environment should record $h_{ij}$, take average of $T_{ij}$, and accumulate $T$.
3. Software test engineers are required to examine and accept/reject the test result. If the engineer accepts the test result, the development environment should update $q_{ij}$ and $p_{ij}$, according to (4-5).
4. If a software defect is identified in edge $e_{ij}$, the defect should be fixed. For all edges $e_{ij}$ involved in the fix (they may belong to different threads), $h_{ij}$ should be reset to zero, after which the test will continue and all edge reliability estimation processes will be updated for $q_{ij}$ and $p_{ij}$ using (4-5) as before.
5. When all testing is complete, estimate the software reliability according to (10-12) and (16-18) use the procedure presented in Sections 3.2-3.4.
6. To improve the reliability of the software, the edges tested least should undergo more tests. Therefore, the information on these edges should be provided.
7. The information on the software reliability should be kept in the release mode. It should be available for reading if a request is made.

In summary, an automated tool in the software development environment is desirable, and it should have the features listed above to facilitate software reliability estimation. We believe, with some extra effort on top of the existing software development environment, that software development tool vendors should be able to provide all of the information to assess software reliability using the proposed method.

## 5 Conclusions

In this paper, we proposed a systematic and probabilistic method to estimate software reliability. The method is rigorously based on test data, flow network, binomial distribution, and Bayesian estimations rather than a subjective judgement. For any components or system involving both hardware and software, the reliability estimation has been discussed in [14].

## Acknowledgment

The author thanks his colleagues Mr. Bernard Dittman and Mr. Mauricio Gutierrez for their valuable comments that helped him to improve the presentation of the paper.

## References

[1] IAEA, Safety of nuclear power plant: design, IAEA Standard SSR-2/1, 2016.
[2] IAEA, Development and application of level-1 probabilistic safety assessment for nuclear power plant, IAEA Special safety guide No. SSG-3, 2010.
[3] IAEA, Application of probabilistic safety assessment (PSA) for nuclear power plants, IAEA technical report, IAEA-TECDOC-1200, 2001.
[4] J. Dennis Lawrence, "Software Reliability and Safety in Nuclear Reactor Protection Systems", US Nuclear Regulatory Commission, NUREG/CR-6001, 1993.
[5] David Lorge Parnas, G. J. K. Asmis, and Jan Madey, "Assessment of safety-critical software in nuclear power plants," Nuclear Safety, Vol. 32, No. 2 pp.189-198,1991.
[6] N. G. Leveson, P. R. Harvey, "Analyzing software safety," IEEE Trans. On Software Engineering, Vol. 9, pp. 569-579, 1983.
[7] W. Farr, "Software Reliability Modeling Survey", in Handbook of Software Reliability Engineering, Edited by Michael R. Lyu, IEEE Computer Society Press and McGraw-Hill Book Company, pp71-117, 1996.
[8] S. Kuo, C. Huang, and M. Lyu, "Framework for modeling software reliability, using various testing-efforts and fault-detection rate," IEEE Transactions on Reliability, Vol. 50, pp.310-320, 2001.
[9] H. Okamura, M. Ando, and T. Dohi, "A generalized gamma software reliability model," Systems and Computers in Japan, Vol. 38, pp81-90, 2007.
[10] W. Wang, T. Hemminger, and M. Tang, "A moving average Non-Homogeneous Poisson Process Reliability Growth Model to Account for Software with Repair and System Structure," IEEE Transactions on Reliability, Vol. 56, No. 3 pp. 411-421, 2007.
[11] C. Huang and C. Lin, "*Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag*," IEEE Transactions on Reliability, Vol. 55, No. 2 pp. 436-450, 2006.
[12] S. C. Bhatt and R. C. Wachowiak "ESBWR certification probabilistic risk assessment," GE-Hitachi Nuclear Energy, NEDO-33201, Revision 2, 2007.
[13] Y. Yang, A flow network model for software reliability assessment, Proceedings of 6th American nuclear society international topical meeting on nuclear plant instrumentation, control, and human-machine interface technologies, Knoxville, April 5-9, 2009.
[14] Y. Yang and R. Sydnor, Reliability estimation for a digital instrument and control system, Nuclear Engineering and Technology, 44(4), pp. 405-414, 2012.
[15] T. L. Chu, M. Yue, G. Martinez-Gruidi, and J. Lehner, Review of quantitative software reliability methods, BNL-94074-2010, Brookhaven National Laboratory, 2010.
[16] S. J. Press, *Bayesian statistics: principles, models, and applications*, John Wiley & Sons, New York, 1989.
[17] D. Navarro and A. Perfors, An introduction to the Beta-Binomial model, *Technical Report*, University of Adelaide, [online]. https://www.johndcook.com/CompendiumOfConjugatePriors.pdf