Design of a platform for reliability analysis of safety-critical digital I&C software

YANG Ming, ZOU Bowen, and YOSHIKAWA Hidekazu

College of Nuclear Science and Technology, Harbin Engineering University No. 145 Nantong Street, Harbin 150001, China (yangming@hrbeu.edu.cn; zoubowen@hrbeu.edu.cn; yosikawa@kib.bigloge.ne.jp)

Abstract: This paper introduces a platform for reliability analysis of safety-critical digital I&C software. A hierarchical structure model (HSM) method is proposed for modeling the structure of software at different layers of abstraction during software development life cycle. A software reliability estimation method based on Flow Network Model (FNM) and Bayesian Belief Network (BBN) is presented. By integrating the functions of HSM construction, source code structure and logical path identification, sensitive analysis and reliability estimation into the platform, it is expected to provide a comprehensive assist in the design, development, test and V&V activities of safety-critical digital I&C software.

Keywords: software reliability; safety-critical software; digital I&C system

1 Introduction

Instrumentation and Control (I&C) system is the central system of a nuclear power plant and is crucial to the safe operation. With the wider applications of digital I&C systems in the nuclear power plants, the quality of a digital I&C system software, especially those executing safety functions, is getting more attention.

Hardware products, especially electronic products, are consisted of basic elements. The same type of basic elements has similar functions and failure modes. It is possible to explore the defects of a hardware product through various experiments. A hardware failure occurs randomly after repair of defects. Unlike hardware products, software is consisted of codes. The functions and failure modes of software codes are quite different. The design and development defects will remain in the software, be triggered under a certain condition, and will cause the software behaviors inevitably rather than randomly deviate from the expected specifications. Due to the complexity of software as an intelligent product, test and the reliability analysis methods of hardware, such as Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA), cannot be applied to fully identify the software defects to prevent the occurrence of software failures.

Received date: February 11, 2015
(Revised date: March 9, 2015)

Software reliability analysis is applicable to any phase of software engineering. The results of reliability analysis can be used for (1) evaluating whether the reliability requirements are satisfied; (2) providing valuable feedback to designers for improving software quality; and (3) assisting in test scenario planning. Software reliability analysis has two major areas, that is, quantitative evaluation to estimating the software reliability using the failure data obtained from software tests and operations, and qualitative analysis for identifying the key factors which may influence the reliability of software. Both the quantitative and qualitative analysis have to construct reliability models.

In general, software reliability models can be classified into white-box models and black-box models. The major difference is the white-box models like Flow Network Model (FNM) [1] reflect the structure of software, while the black-box models like Software Reliability Growth Model (SRGM) and Bayesian Belief Network [2-3] will not take the software structure into account. In this paper, we only focus on white-box model technology.

The software development life cycle (SDLC) is consisted of different phases including problem definition, feasibility analysis, general description, system design, coding, debugging and test, acceptance, operation, updating and out-of-service.

The requirements for reliability analysis and available documents and data in each phase of SDLC are quite different. Software development is an abstract-concrete, part-whole, gradual process. In order to well satisfy the needs of reliability analysis in different phases of SDLC, this paper presents a Hierarchical Structure Modeling (HSM) method. The basic idea of HSM is to model the structure of software at different layers of abstraction and enable an easy and step-by-step model extension.

2 Concepts of hierarchical structure modeling

A HSM model describes the structure and data flow of a software system using abstract-concrete and part-whole conceptions. As shown in Fig.1, a HSM model of software consists of 4 common elements, that is, unit, module, relation and structure.

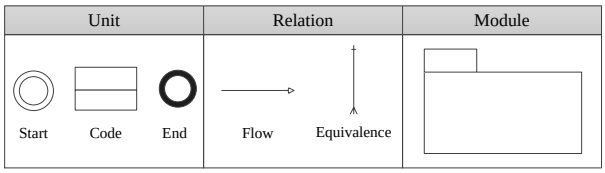


Fig. 1 Graphical expressions of HSM elements.

1. Unit: is the most basic element of HSM and has the following three types.

Start: is the start point of a HSM module. A Start unit may just have a semantics meaning that each software module will be executed from somewhere, but it may also indicate data inputs from other module(s).

Code: is an abstraction of a statement, expression, command and function.

End: is the end of a software module.

2. Module: is a group of interrelated Unit for realizing a certain function. The conceptions of Unit and Module in HSM have relative meanings. On one hand, a Unit at one level of abstraction may be extended to a Module at a lower level of abstraction, and one the other hand a Module can be treated to be a Unit at a higher level of abstraction.

3. Relation: describes the execution order or equivalence relationship between HSM Units and Modules.

Flow Relation: connects two Units for describing that the execution order is from one to another along the arrow direction.

Equivalence Relation: connects a Unit to a Module for indicating that the Unit is an equivalent code of the Module.

4. Structure: describes the reliability logical relation between HSM Units.

5. Serial Structure: Taking a group of Units as a system, if any failure of Unit will result in the system failure, or in other words, the system will be successful if all Units work successfully; these Units are organized as a serial structure.

6. Parallel Structure: Taking a group of Units as a system, if at least one Unit successfully works then the system will be successful, or in other words, the system will fail if all Units lose their functions; these Units are organized as a serial structure. In C Programming Language, a Parallel Structure is realized by “if-else” and “Switch” commands.

7. Compound Structure: is a mixing structure of Serial and Parallel Structure.

An example of HSM model is shown in Fig.2. By utilizing HSM technology, it is possible for modeling the software structure in any phase of SDLC even in the early stages of system design and less of detailed system design situations.

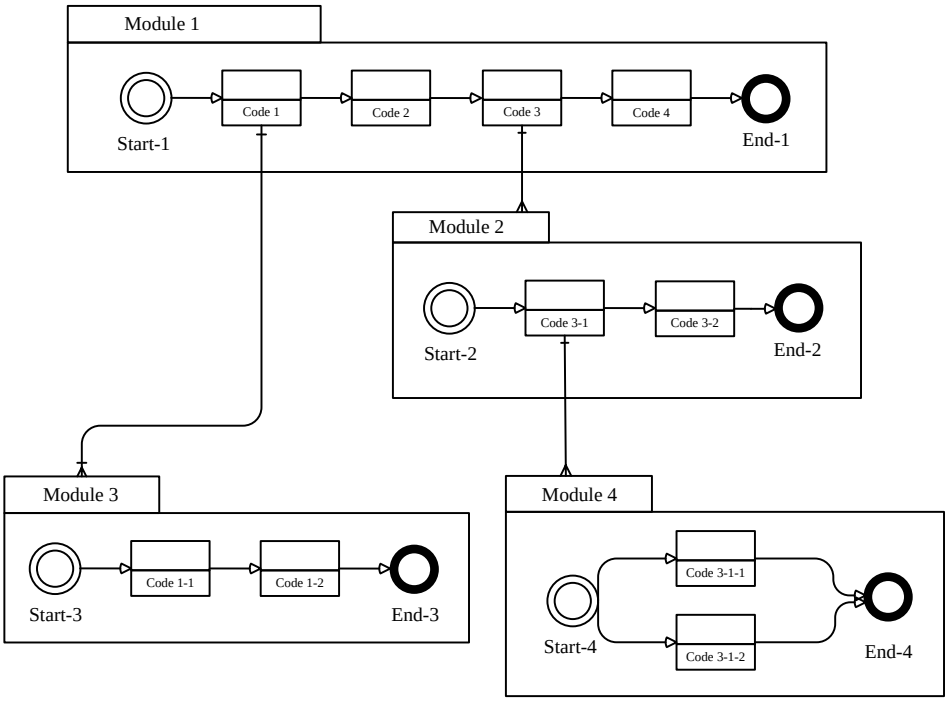


Fig. 2 An example of HSM model of software.

3 Software reliability estimation based on HSM

3.1 Reliability estimation for a single code

In this paper, the FNM method proposed by the literature [1] is applied to estimate the reliability of software.

Let T_{C_i} and h_i be the average execution time and the total number of executions of a single line of code C_i .Let T_i be the total execution time of C_i . Let p_i denote the probability of C_i being executed once and the test result meets the expectations. Let q_i denote the probability of C_i being executed once and the test result fails to meet the expectations. Clearly, the one-time test scenario follows a Bernoulli distribution. Set the prior failure probability of C_i to

$$q_i = 10^{-m_i} \tag{1}$$

where m_i is the failure metrics of C_i which is a positive real number and can be estimated according to the past performance and complexity of C_i , or developer experience. The more complexity, longer execution time and lower technology maturity of C_i , the higher failure probability of C_i , i.e., the smaller m_i . For example, code C_i and C_j were developed by the same technical personnel and have a similar complexity. If C_i has been tested while C_j has not been tested yet, the failure probability of C_j should be higher than that of C_i . Then the failure probability of C_j can be set to

$$q_j = 10^{\frac{-m_i}{2}} \tag{2}$$

If C_i is executed exactly once and the test result meets the expectations, the following relationship holds:

$$r_i = p_i = 1 - q_i \tag{3}$$

where r_i is the reliability of C_i .

If C_i is executed more than once in a software test and all test results meet the expectations, considering the failure probability of C_i should be lower than its prior failure probability, then the failure probability and reliability of C_i can be updated as

$$q_i = 10^{-h_i m_i} \tag{4}$$

$$r_i = 1 - q_i = 1 - 10^{-h_i m_i} \tag{5}$$

If C_i is executed exactly h_i times and only the h_i th test fails to meet the expectation, according to the requirements for the safety-critical software, the

errors in C_i must be corrected. After correction, the failure probability of C_i should be within the interval $(10^{-m_i}, 10^{-h_i m_i})$. The failure probability of C_i can be therefore set to

$$q_i = 10^{-\frac{3h_i}{4}m_i} \tag{6}$$

The failure probability of the corrected C_i can be also set using formula (1) if it is treated conservatively as a new code.

3.2 Reliability estimation for a serial configuration

For a serial structure S_{C_i} composed of $C_i\ (i = 1,2,...,n)$, it can be equivalent to a Code S_C. If the serial structure S_{C_i} is executed exactly h_i times and all test results meet the expectations, then the reliability of S_{C_i} is

$$r_{S_C} = \prod_{i=1}^{n} r_i \tag{7}$$

Because all Codes in S_{C_i} are all executed h_i times, the execution time of the equivalent Code S_C is:

$$h_{S_C} = h_i \tag{8}$$

The total execution time of the equivalent Code S_C is:

$$T_S = \sum_{i=1}^{n} T_{C_i} h_i \tag{9}$$

The average execution time of the equivalent Code S_C is:

$$T_{S_C} = \sum_{i=1}^{n} T_{C_i} \tag{10}$$

3.3 Reliability estimation for a parallel structure

For a parallel structure P_{C_i} composed of $C_i\ (i = 1,2,...,n)$, it can be equivalent to a Code P_C. The total number of executions of the parallel configuration P_{C_i} is:

$$h_p = \sum_{i=1}^{n} h_i \tag{11}$$

The total number of executions of the equivalent Code P_C is:

$$h_{P_C} = \sum_{i=1}^{n} h_i \tag{12}$$

The total execution time of the equivalent Code P_C is:

$$T_P = \sum_{i=1}^{n} T_{C_i} h_i \tag{13}$$

If consider the number of executions reflect the importance of a Code in P_{C_i}, then the average execution time of the equivalent Code P_C is:

$$T'_{P_C} = \frac{1}{h_P} \sum_{i=1}^{n} T_{C_i} h_i \tag{14}$$

In this case, the reliability of the parallel structure P_{C_i} is:

$$r'_{p_C} = \sum_{i=1}^{n} \frac{T_{C_i} h_i}{T'_{P_C}} r_i \tag{15}$$

From formula (15) it can be concluded that the more number of executions and more execution time of a Code, the more contribution to the reliability of the parallel structure.

Otherwise, if consider all Codes in a parallel structure have the same importance, then the average execution time of P_C is:

$$T''_{P_C} = \frac{1}{n} \sum_{i=1}^{n} T_{C_i} \tag{16}$$

In this case, the reliability of the parallel P_C structure is:

$$r''_{p_C} = \sum_{i=1}^{n} \frac{T_{C_i}}{T''_{P_C}} r_i \tag{17}$$

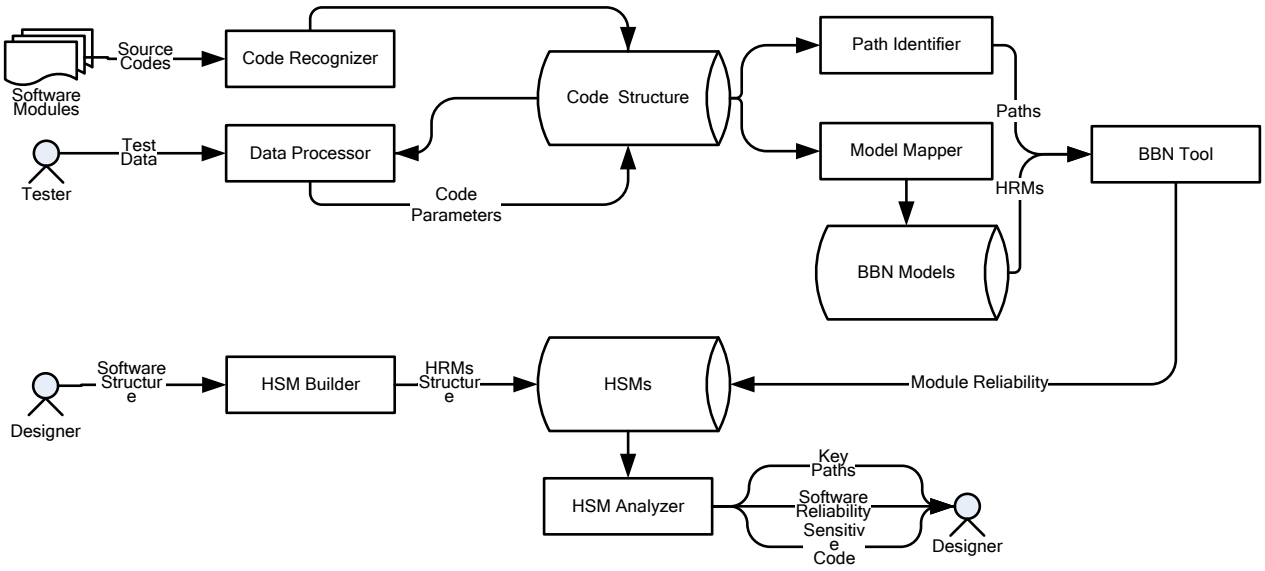


Fig. 3 Structure of software reliability analysis platform.

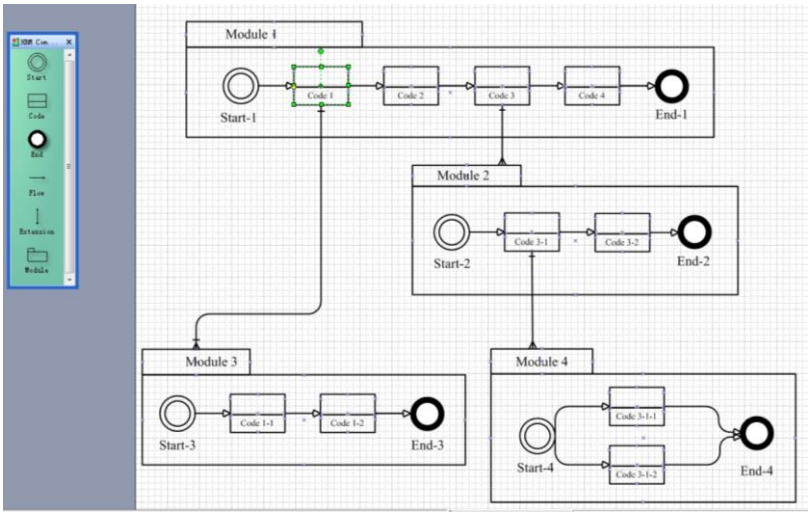


Fig. 4 Graphical interface of HSM builder.

3.4 Reliability estimation for a compound structure

As shown in Fig. 2, by repeating the equivalent process of serial and parallel structure, a compound structure will be finally equivalent to a serial structure and formula (7) will be applicable to estimate the reliability of a compound structure and even the reliability of the whole software.

4 Design of software reliability analysis platform

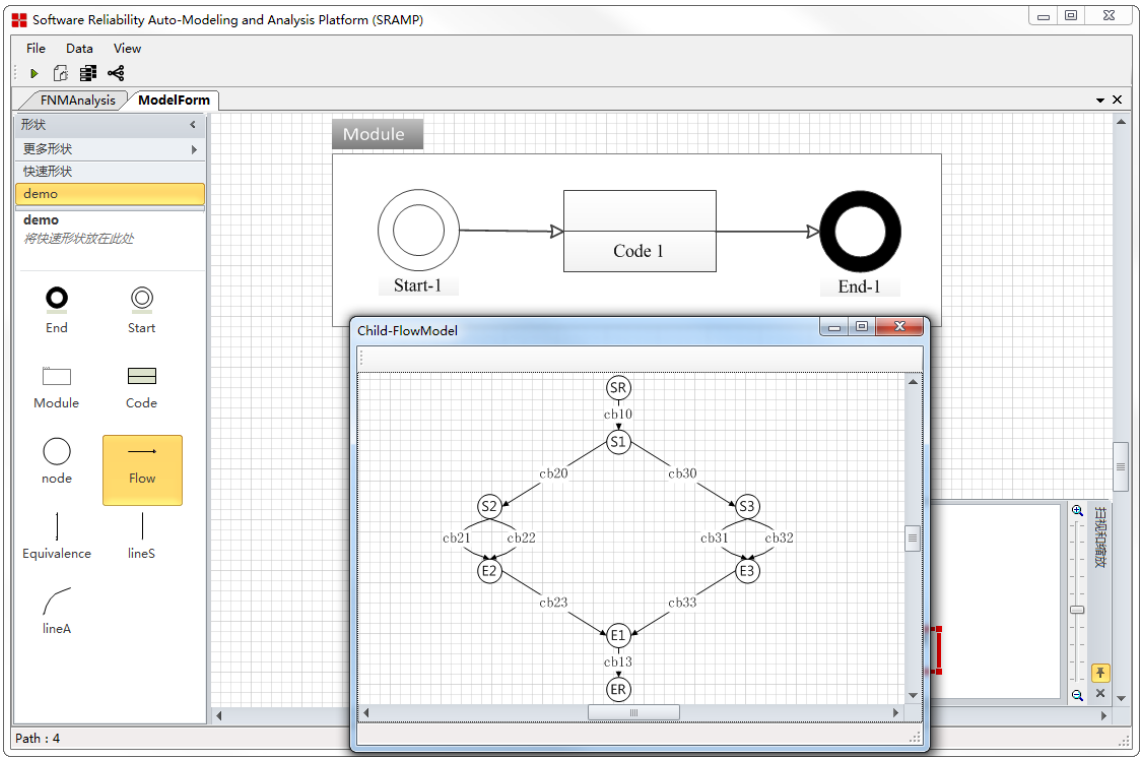
A platform for software reliability analysis has been designed and developed for helping technical personnel to construct hierarchical structure of software, identify the code and key paths, and evaluate the software reliability. The structure of the platform is shown in Fig. 3. All functional modules except the HSM Builder are developed with C# language in Windows XP.

4.1 HSM builder

As shown in Fig.4, HSM Builder is a graphical modeling tool for constructing HSM models. HSM Builder was developed with Microsoft Office Visio 2000 in Windows XP. The symbols of HSM elements are saved as a template enabling Drag-and-Drop operations for fast constructing HSM models. The parameters for reliability assessment including failure metrics, number of executions, execution time and execution result are defined in the property of each element. The HSM models can be exported to HSMs Database in XML format.

4.2 Code recognizer

The function of Code Recognizer is to automatically identify the structure including the serial and parallel structures of the source code of a Module in a HSM models by matching the key words such as “if”, “else-if”, “else”, “Switch” and “Case”. The code identification results are saved in the Code Structure Database and can be presented to users graphically in a tree or a figure.



Fig, 5 Code identification and graphical display.

4.3 Data processor

Data Processor provides a graphical interface for inputting test results into the platform. The test personnel first select a Code unit from the Code Structure Database, and then inputs failure metrics, the number of executions and execution time of the Code. Data Processor will accordingly estimate the reliability of the Code using the reliability estimation method for a single code. The estimation results will be used to update the relevant data in the Code Structure Database.

4.4 Path identifier

Path Identifier can identify all possible code execution paths according source code structure.

4.5 BBN mapper

The function of BBN Mapper is to map the code structures into a Bayesian Belief Network (BBN) model for qualitative and quantitative reliability analysis of a HSM Module. The procedures are as follows:

Step 1: map each Code Unit in the HSM Module into a leaf node of BBN model.

Step 2: find a pure parallel structure P_{C_i} .

Step 3: map P_{C_i} into a sub-model of BBN.

Step 4: repeat step 2-3 until all pure parallel structures are processed.

Step 5: find a pure serial structure S_{C_i} .

Step 6: map S_{C_i} into a sub-model of BBN.

Step 7: repeat step 5-6 until all pure serial structures are processed.

Step 8: repeat step 2-7 until the Module is equivalent to a Unit.

Taking the HSM Module shown in Fig. 6 as an example, C3/C4 and C7/C8 are pure parallel structures.

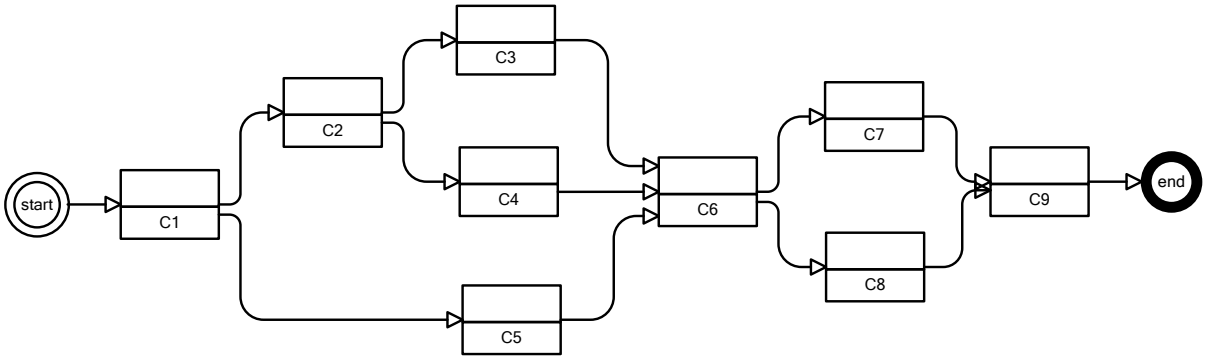


Fig. 6 A HSM module with a compound configuration.

The general sub-model BBN for a pure parallel or serial structures is shown in Fig.7, where C_i denotes the i th Code of P_{C_i}. The parallel logic of P_{C_i} can be expressed using a conditional probability table. Table 1 gives an example of a conditional probability table of the parallel structure C3/C4.

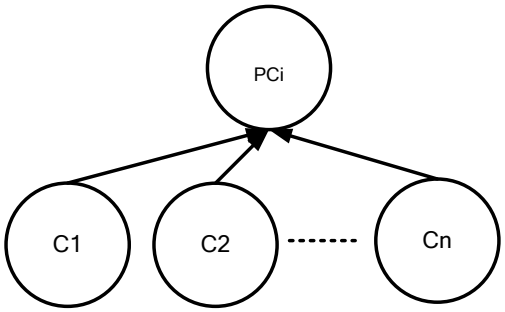


Fig. 7 A general sub-model of BBN for a pure serial or parallel structure.

Table 1 Conditional probability setting for a pure parallel structure

C3	Success		Failure	
C4	Success	Failure	Success	Failure
P3-4= Success	1	$\frac{h_{C_3}T_{C_3}}{T_P}$	$\frac{h_{C_4}T_{C_4}}{T_P}$	0
P3-4= Failure	0	1- $\frac{h_{C_3}T_{C_3}}{T_P}$	1- $\frac{h_{C_4}T_{C_4}}{T_P}$	1

Note: P3-4 is the equivalent code of the pure parallel structure $P_{C_3C_4}$

The estimation result of a pure parallel structure by BBN is the same as Formula (15) presents. The proof is given as follows, where P_C denotes the equivalent Code of P_{C_i}.

$$\begin{aligned} r_{P_C} &= r_{C_3}r_{C_4} + r_{C_3}(1-r_{C_4})\frac{h_3T_3}{T_P} + r_{C_4}(1-r_{C_3})\frac{h_4T_4}{T_P} + 0\times(1-r_{C_3})(1-r_{C_4}) \\ &= r_{C_3}r_{C_4} + \frac{h_3T_3}{T_P}r_{C_3} + \frac{h_4T_4}{T_P}r_{C_4} - \frac{h_3T_3+h_4T_4}{T_P}r_{C_3}r_{C_4} = \frac{h_3T_3}{T_P}r_{C_3} + \frac{h_4T_4}{T_P}r_{C_4} \end{aligned} \quad (18)$$

The general sub-model of BBN can be also applied for mapping a pure serial structure into a sub-model of BBN. However, the conditional probability has to be set according to Table 2.

Table 2 Conditional probability setting for a pure serial structure

C2	Success		Failure	
P3-4	Success	Failure	Success	Failure
S2-3-4= Success	1	0	0	0
S2-3-4= Failure	0	1	1	1

Note: S2-3-4 is the equivalent code of the pure serial structure $S_{C_2C_3C_4}$

By performing the mapping procedures, the final BBN model for Fig.6 is shown in Fig. 8.

4.6 BBN Tool

BBN Tool is to analyze a BBN based software structure model for the following purposes.

(1) Software reliability estimation: given the reliability of each leaf node (HSM Code) is known, the software reliability can be estimated automatically without manually processing each HSM module into a pure serial structure.

(2) Sensitivity analysis of code: BBN Tool can show the changes of software reliability with the reliability of a code or a module in the form of a figure. The technical personnel can therefore know which code

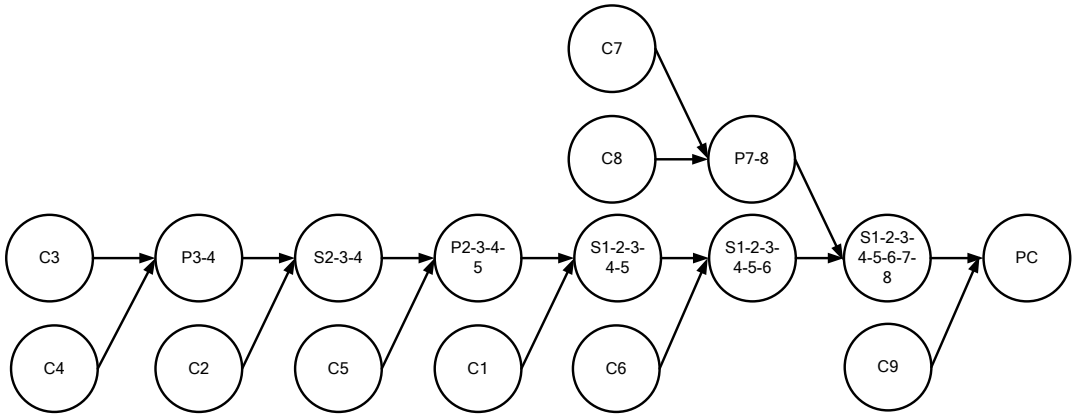


Fig. 8 An example of mapping a HSM module into a BBN model.

or module should be paid more attention to check and test.

(3) Key test path identification: provide a "what-if" analysis that given the results of the next software test meet with expectations, how much the contribution of each path to the growth of software reliability should be. The technical personnel can therefore consider and prepare a better scenario for the next software test.

4.7 HSM analyzer

After the estimation of the reliability of all HSM codes and modules have been done, the HSM Analyzer can estimate the reliability of the whole software by utilizing the hierarchical structure of HSM.

5 Case study

A PID controller code, a segment of digital feed water control system software, was selected for case study to verify the functions of the Software Reliability Analysis Platform. The PID Controller code consists of 186 lines programmed by FORTRAN language. The HSM model of the PID controller code is shown in Fig. 9.

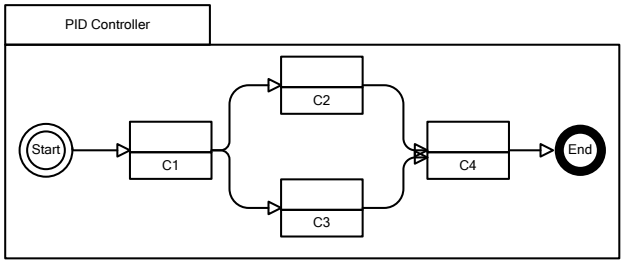


Fig. 9 HSM model of a PID controller software code.

(1) Code and path identifications

The code identification result is shown in Fig.10 where the parts corresponding to the modules of the HSM model in Fig.9 are marked. There are totally 115,212 execution paths in this PID controller code.

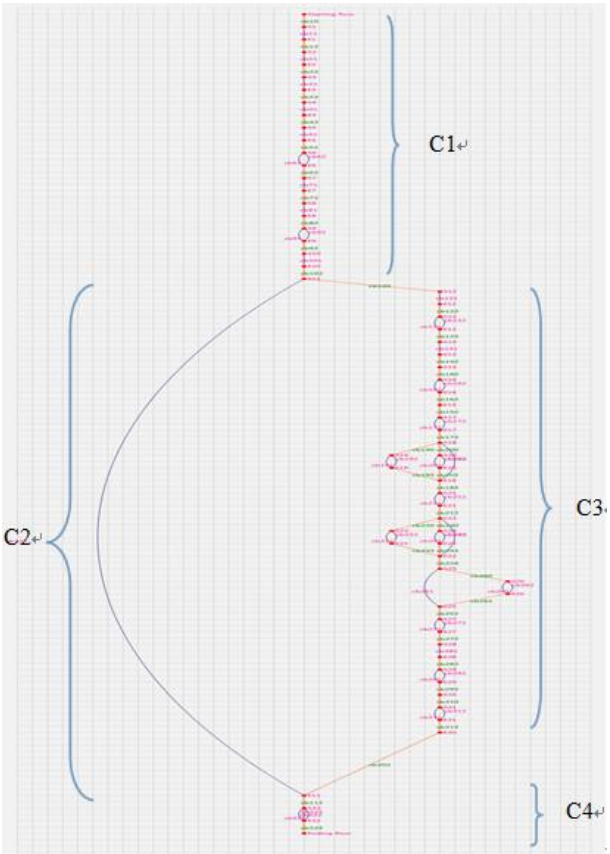


Fig.10 Code structure of PID controller.

(2) Reliability estimation

First, the failure metrics (m_i) of the Unit in Fig. 9 and all lines in Fig.10 is assumed to be 1, *i.e.*, the prior reliability of each Unit in Fig.9 and each line in Fig.10 is 0.9. Then we executed the PID Controller

code 5 times using different test scenarios. Meanwhile in the parallel structure C2/C3, only C2 module was covered and all test results met the expectations. Using the Software Reliability Analysis Platform, the software reliability of PID controller is 0.9989976.

(3) Test scenarios evaluation

Given the 6th test will also meet the expectations, the contribution of all execution paths to the growth of the software reliability of PID controller were evaluated. Table 3 presents the top 20 execution paths which may contribute to a higher relative magnitude of software reliability of PID controller. In Table 3, the No. 1-5 execution paths will cover C2 Module, while others will cover C3 Module. From Table 3, it can be concluded that successfully testing the same code and module will definitely contribute to the growth of software reliability, however the estimated reliability may not so creditable. Note that the No.6-20 execution paths will lead to negative reliability growths.

The reason is that we assumed that the prior reliability of C3 module is 0.9, even after a successful execution of one path covering C3 the reliability of the codes in C3 module will reach to 0.99, but due to C3 consists of too many serial codes, the reliability of C3 will be lower than what we pre-estimated.

(4) Evaluation of failure metrics (m_i)

To evaluate the influence of setting different value of failure metrics (m_i) to each code, we estimated the software reliability of PID controller by changing the failure metrics values from 0.5, 1, 1.5, 2 to 3, respectively. The test scenarios for evaluations are same and all test results met expectations. Figure 11 presents the comparison of different failure metrics values contributing to reliability changes of PID controller with the number of tests. It can be seen that a lower failure metrics value will result in lower software reliability estimation at the beginning stage of software test. However, with the number of tests increasing, there is no significant difference between the estimated software reliability using different failure metric values, which reveals that after each single line of code reaching a certain reliability level, the structure of software becomes the determining factor of software reliability estimation.

Table 3 Top 20 execution paths of PID controller for the 6th software test

No.	Execution Paths	Software Reliability of PID Controller	Relative Magnitude
1	2	0.999999970499998	0.001002330497241
2	3	0.999956399815828	0.000958759813071
3	4	0.999729584980548	0.000731944977791
4	10264	0.999729584980548	0.000731944977791
5	19864	0.999459272569428	0.000461632566671
6	67868	0.997956455516570	-0.001041184486187
7	106272	0.997956455516570	-0.001041184486187
8	29448	0.997999939057999	-0.000997700944758
9	29462	0.997999939057999	-0.000997700944758
10	29463	0.997999939057999	-0.000997700944758
11	29464	0.997999939057999	-0.000997700944758
12	29468	0.997999939057999	-0.000997700944758
13	29472	0.997999939057999	-0.000997700944758
14	29496	0.997999939057999	-0.000997700944758
15	29592	0.997999939057999	-0.000997700944758

16	29624	0.997999939057999	-0.000997700944758
17	29784	0.997999939057999	-0.000997700944758
18	30104	0.997999939057999	-0.000997700944758
19	30232	0.997999939057999	-0.000997700944758
20	34264	0.997999939057999	-0.000997700944758

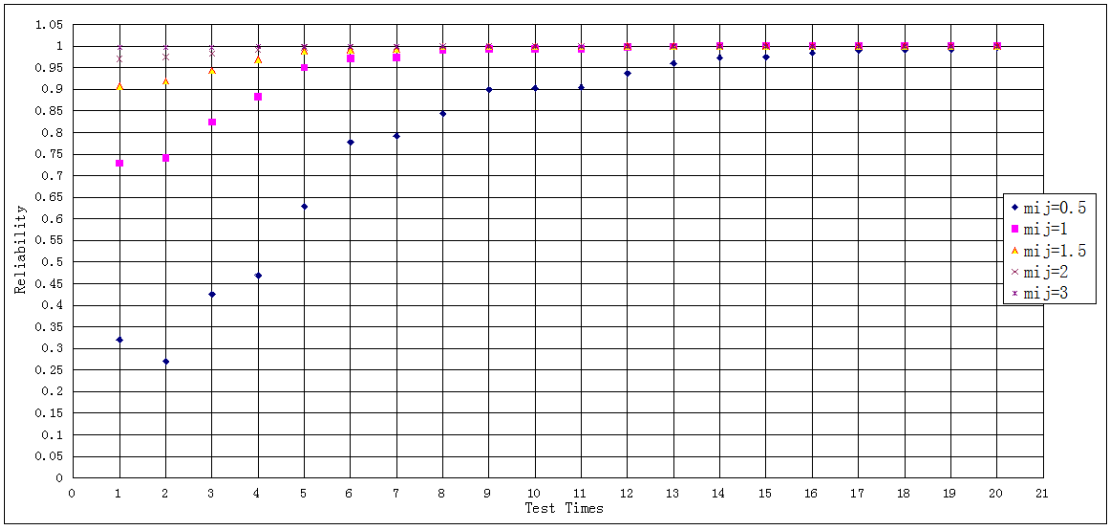


Fig. 11 Influence of failure metrics to software reliability estimation.

6 Conclusions

In this paper, a Hierarchical Structure Modeling (HSM) of software is presented, which can be applied for software reliability analysis even in the early stage of software development life cycle. With the process of the software design and development, the HSM model can be easily refined and extended. Based on HSM technology, the Flow Network Model (FNM) approach is applied to estimate the reliability of a single line of code, serial and parallel structures. This paper also presents a solution for automatically estimating a compound structure by mapping HSM into a Bayesian Belief Network (BBN) model. A platform for software reliability analysis developed by authors is introduced, which integrates the functions of code and execution paths identification, sensitivity analysis and software reliability estimation. The platform will be helpful to the technical personnel in the software design, development, test and acceptance tasks. With the help of the proposed software reliability analysis platform, the software designers and developers can clearly understand about the requirements of software reliability design and test.

References

[1] YANG Y.G., and SYDNOR R.: “Reliability Estimation for A Digital Instrument and Control System”, Nuclear Engineering and Technology, 44(4), pp.405-414(2012)

[2] KIM M.C.: “Reliability Analysis of Digital I&C Systems in KAERI”. International Journal of Nuclear Safety and Simulation, 3(4), pp. 276-280(2012)

[3] HOLMBERG J.E.: “Software Reliability Analysis in Probabilistic Risk Analysis”, International Journal of Nuclear Safety and Simulation, 3(4), pp. 281-291(2012)

[4] YANG M., and SONG M.C.: “Study on Quantitative Software Reliability for Digital Control System”, Nuclear Power Engineering, 35, pp. 54-58(2014)