CHAPTER 7. ESTIMATION OF SOFTWARE RELIABILITY

7.1 Introduction

This chapter explains the step by step procedure involved in calculating software reliability for the safety systems of NPP. A comprehensive procedure to estimate software reliability for a given software program coded in “C” language is detailed. The safety systems of NPP are embedded systems, implemented in “C” language. The software metrics required for each function is obtained by running the “Static Analyser” tool developed as a part of this research or by any Computer Aided Software Engineering tools. It is also assumed that the software has undergone vigorous V&V procedure developed as part of this research, which is detailed in chapter 5. It is also assumed that the subsequent version improves reliability i.e. changes fixes the bug or new functionality added without introducing error in the existing working code. In view of this the reliability growth model is assumed.

It is from the experience of the software deployed in NPP and Heavy water plants, a set of parameters influencing the software reliability is derived. The reliability estimation is based on the following parameters:

a) Lines of code
b) Number of functions
c) Cyclomatic Complexity
d) Local and Global Variables
e) Nesting Levels
f) Number of times functions invoked
g) Number of distinct operators
h) Dynamic Memory usage
7.2 Reliability Estimation

This procedure is evolved with the data collected on the successful operation of software systems from operating nuclear power plants and other Atomic Energy establishments like Heavy Water Plants over the last two decades. This procedure involves basically five steps. The detailed explanation of these steps is as follows:

**Step 1: List out the “functions” used in the software**

The “C” language basically deals with only the “functions” as its basic sub-units of the program. The first step for calculating reliability starts with finding out the “functions” for the complete project. The complete project includes Source files, Library files and Header files. For the software deployed in safety systems the complete source code is available including library and header files. All the necessary library functions are developed instead of using object files so that it is amenable for Verification and Validation at source code level.

List out the names of all the functions used in the complete software project including the function “main ()”

**Step 2: Measure the parameters and tabulate for all the functions**
The parameters pertaining to each function of the complete software is measured using the “Static Analyser”. It is basically the software metrics pertaining to each of the functions are evaluated and tabulated. These parameters directly influence the reliability of the software. In terms of severity of parameter’s influence on reliability, it is categorized as different levels with “Level 0” as the highest severity of levels are tabulated in Table 7.1. The next sequence is to measure the following parameters and fill up the Table 7.2.

The parameters identified are

a) Lines of Code

Less the number of lines of code, it will be easier to understand, test and maintain. On the other hand, when the code is lengthy and complex it is prone to error. So the number of code lines should be less to improve the reliability. This forms the basis for the base failure rate of the function.

b) Cyclomatic Complexity (CC)

This attribute measures the complexity of each function in terms of number of independent paths, i.e., number of branches. Less the number of branches, better is the reliability. It is applied to individual functions within a program. The number of “test cases” in the Basis Path Testing strategy is proportional to the cyclomatic complexity of the program. This falls in the category of Level 0, which means that its contribution is significant.

c) Local Variables

Usage of more variables results in occupying more memory and particularly the stack in case of usage of “static” variables under Local Variables. Usage of local variables is preferred compared to Global Variables because of vulnerability in affecting and becomes concern in “Impact Analysis” on top of being occupying memory throughout the life of the
program. This falls in the category of Level 3, which means that its contribution is minimum in affecting the reliability.

d) Global Variables

It is always better to minimise global variables but still some values are required to be of scope covering the complete project. Usage of local variables is preferred compared to Global Variables because of vulnerability in affecting and becoming concern in “Impact Analysis” on top of being occupying memory throughout the life of the program. This falls in the category of Level 2, which means that its contribution is moderate and affects reliability to some extent.

e) Number of times invoked / used

This parameter is to give weight-age as per the usage of the functions. Less the frequency of usage less will be the combination of inputs that result in better reliability. This falls in the category of Level 3, which means that its contribution is minimal in affecting the reliability.

f) Level of nesting

The depth of “condition branches” or the “loops” is the measure for each function. When the depth is less, the reliability is better, number of test cases is less and hence, the testability and coverage are good. This falls in the category of Level 1 which means that its contribution is large and affects reliability to large extent.

g) Dynamic Memory usage (Number of times)

Dynamic memory allocation also known as heap-based memory allocation is the allocation of memory storage for use during the run-time. It can also be seen as a way of distributing ownership of limited memory resources among many pieces of data and code. Hence, it is vulnerable for memory overflow resulting in failure. When the number of occurrences is
more it is more vulnerable for failure resulting in decrease in reliability. This falls in the
category of Level 1 which means that its contribution is large and affects reliability to a
large extent.

h) Number of distinct operators

The complexity of arithmetic is directly related to the number of distinct operators used in
the algorithms. As the number of operators increases the reliability decreases. This falls in
the category of Level 3, which means that its contribution is minimal in affecting the
reliability.

The Π factors are derived depending on their “Category of Level” assigned to the
parameters. Table 7.1. Shows the mapping of level to Π factor equation. In the next step
using this table Π factors are evaluated and used in calculation for its contribution to
reliability.

Table 7.1 Mapping of Π factors to “Category of Levels”

<table>
<thead>
<tr>
<th>Severity Levels</th>
<th>Π factor equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0 x Parameter</td>
<td>Significant</td>
</tr>
<tr>
<td>1</td>
<td>1 + (1.0 x Parameter)</td>
<td>Large</td>
</tr>
<tr>
<td>2</td>
<td>1 + (0.1 x Parameter)</td>
<td>Moderate</td>
</tr>
<tr>
<td>3</td>
<td>1+(0.01 x parameter)</td>
<td>Minimum</td>
</tr>
<tr>
<td>4</td>
<td>1/(1+(0.01 * parameter))</td>
<td>Small improvement</td>
</tr>
<tr>
<td>5</td>
<td>1/(1+(0.1 x parameter))</td>
<td>Large improvement</td>
</tr>
</tbody>
</table>

Table 7.2 Template for Quality Parameters of Function
Step 3: Apply the equation to calculate failure per combination for all the functions

The equation for failure of a function per input combination is defined as $\Phi_f$

The $\Phi_f$ is calculated as,$\Phi_f = \Phi_b \Pi_C \Pi_{LV} \Pi_{GV} \Pi_{TU} \Pi_N \Pi_D \Pi_O$  
…………………………..(7.1)

where

$\Phi_b$ Base failure / Combination  
= Number of lines $\times 10^{-8}$ For Non Nuclear Safety  
= Number of lines $\times 10^{-9}$ For Safety Related systems  
= Number of lines $\times 10^{-10}$ For Safety Critical systems  

$\Pi_C$ Cyclomatic Complexity Factor  
= 1.0 x Cyclomatic Complexity  

$\Pi_{LV}$ Local Variable Factor  
= 1 + (0.01 x Number of Local Variable)  

$\Pi_{GV}$ Global Variable Factor  
= 1 + (0.1 x Number of Global Variable)  

$\Pi_{TU}$ No. of Times Used Factor  
= 1 + (0.01 x Number of Times used)  

$\Pi_N$ Nesting Factor  
= 1 + (1.0 x Nesting Level)  

$\Pi_D$ Dynamic Memory Factor  
= 1 + (1.0 x Dynamic Memory usage)  

$\Pi_O$ Operators Factor  
= 1+(0.01 x number of operators used)
The weights for each factor is arrived from analyzing around fifty software modules of fast reactor and with the methodology applied in the organization. If necessary these factors can be tuned as per the organization methodology and its perception. The Failure / Combination $\Phi_f$ of all the functions shall be calculated and tabulated as in the Table 7.3.

### Table 7.3 List of Failures for functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Failure / Combination $\Phi_f \times 10^{-10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func 1</td>
<td></td>
</tr>
<tr>
<td>Func 2</td>
<td></td>
</tr>
<tr>
<td>Func 3</td>
<td></td>
</tr>
<tr>
<td>Func 4</td>
<td></td>
</tr>
</tbody>
</table>

#### Step 4: Apply the equation to calculate failure per combination for software

Now for the complete software the failure is calculated by arriving at subtotal and then by multiplying the common factor affecting the complete software to achieve the total failure / input combination. The sub-total of failure / input combination is calculated using the Equation 7.2 by summing up all the functions failures since all the functions have to operate to achieve the required functionality of the software.

$$\Phi_{\text{SubTotal(ST)}} = \sum_{i=1}^{N} \Phi_{fi} \quad \text{...................................................} (7.2)$$

Once the sub-total is calculated the total failure / input combination for the complete software is calculated using Equation 7.3 by normalizing with the common quality factors pertaining to the project.
\( \Phi_{TOTAL} = \Phi_{ST} \Pi_{Ver} \Pi_{CC} \Pi_{STD} \Pi_{int} \) ...........................................(7.3)

Where

\( \Pi_{Ver} \)  Version Number factor  
\( = 1 / (1+(0.1 \times \text{number of versions})) \)

\( \Pi_{CC} \)  Comment to Code Factor  
\( = 1 / (1+(0.1 \times \text{percentage of comment to code})) \)

\( \Pi_{CC} \)  Conformance to Standards Factor  
\( = 1+(10 - (10 \times (0.01 \times \text{percentage of conformance}))) \)

\( \Pi_{int} \)  Number of Interrupts used factor  
\( = 1 + 10 \times \text{Number of interrupts} \)

Once the sub total is calculated, the factors which are affecting / increasing the reliability are considered to arrive the total failure / input combinations. The global factors are

a) Version Number : As the version number increases, the new releases covers the “bug” fixing and covers the requirements overlooked. In essence the new releases increase the reliability to certain extent.

b) Comment to Code Ratio : The Comments in the code is like document embedded in the code. This helps in large extend to understand the algorithm and flow of the software.

c) Conformance to Standards: The degree of conformance to standards (like MISRA) greatly enhances reliability because we will be avoiding error prone and complex way of expression. It also facilitates good practices, which can trap invalid combinations (like “default” in Case statement).

d) Number of Interrupts: As the number of interrupts used in the software increases, the software becomes complex. It becomes difficult to do the testing since the simulation of asynchronous interrupts during the normal courses of action and also simultaneous
triggering of multiple interrupts. Hence the reliability tends to decreases as the number of interrupts increases.

**Step 5: Estimation of reliability**

The reliability over the number of input combinations is given by

\[ R(c) = e^{-\Phi \ast COMBINATIONS} \] \hspace{1cm} (7.4)

**7.3 Typical Estimation for Safety Critical System**

As a case study, the software developed for a fast reactor, which does the supervision of sodium temperature at inlet and outlet of fuel subassemblies and initiating shutdown of the nuclear plant on finding abnormalities is taken up. This software falls in the category of Safety Critical System. The step by step procedure for estimation of the software reliability is shown next

**Step 1: List out the functions used in the software**

The Safety Critical System software has the following functions

- **main**: Main module
- **scan**: Scan Module
- **median**: To Calculate Median
- **ptsr**: Pre Treatment Sub Routine
- **mvsr**: Mean Value Sub Routine
- **mgsr**: Mean Gradient Subroutine
- **spcs**: Supervision of Core Subassembly
- **init**: Initialization
Step 2: Measure the parameters and tabulate them for all the functions

The parameters are calculated for each functions and filled in the Table 7.3. Then $\Phi_f$ is calculated using Eq. 7.1. and filled in Table 7.4 as explained in step 3.

Table 7.4 Software Failure Calculation for function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Lines Of code</th>
<th>CC</th>
<th>Local Var</th>
<th>Global Var</th>
<th>Times used</th>
<th>Nesting</th>
<th>Dyn. mem</th>
<th>ope. rats</th>
<th>Failure / combination $\Phi_f \times 10^{-10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>26</td>
<td>2</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>342.0354</td>
</tr>
<tr>
<td>Scan</td>
<td>240</td>
<td>36</td>
<td>19</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>68537.1456</td>
</tr>
<tr>
<td>Median</td>
<td>55</td>
<td>11</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>2259.2229</td>
</tr>
<tr>
<td>Ptsr</td>
<td>35</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>588.2466</td>
</tr>
<tr>
<td>Mvsr</td>
<td>87</td>
<td>12</td>
<td>8</td>
<td>17</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>6579.8586</td>
</tr>
<tr>
<td>Mgsr</td>
<td>63</td>
<td>7</td>
<td>4</td>
<td>20</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>13002.4440</td>
</tr>
<tr>
<td>Spcs</td>
<td>78</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>7665.9242</td>
</tr>
<tr>
<td>Main</td>
<td>98</td>
<td>8</td>
<td>6</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>10575.8150</td>
</tr>
</tbody>
</table>

CC – Cyclomatic Complexity  Sub Total 108550.9623 x $10^{-10}$

Step 3: Apply the equation to calculate failure / combination for all the function

The equation for failure of a function per input combination is computed as $\Phi_f$

$\Phi_f = \Phi_b \Pi_C \Pi_{LV} \Pi_{GV} \Pi_{TU} \Pi_{N} \Pi_{D} \Pi_{O}$

For the “Init” function

$\Phi_b = 26 \times 10^{-10}$  $\Pi_C = 1.0 \times 2$  $\Pi_{LV} = 1+(0.01 \times 1)$

$\Pi_{GV} = 1+(0.1 \times 21)$  $\Pi_{TU} = 1+(0.01 \times 1)$  $\Pi_{N} = 1+(1.0 \times 1)$
\[ \Pi_D = 1 + (1.0 \times 0) \quad \Pi_O = 1 + (0.01 \times 4) \]

\( \Phi_f \) of “Init” function is 342.0354 \( \times 10^{-10} \) failures / combination

In the same way, \( \Phi_f \) is calculated for all the functions and entered on to the Table 7.4.

**Step 4: Apply the equation to calculate failure / combination for software**

Now for the complete software the failure is calculated by arriving at subtotal and then by multiplying the common factor affecting the complete software to achieve the total failure / input combination as

\[
N \\
\Phi_{\text{SubTotal(ST)}} = \sum_{i=1}^{N} \Phi_{fi} = 108550.9623 \times 10^{-10} \text{ failures / combination}
\]

Once the sub-total is calculated the total failure / input combination for the complete software is calculated using Eq. 7.3 by normalizing with the common quality factors.

\[
\Phi_{\text{TOTAL}} = \Phi_{ST} \Pi_{\text{Ver}} \Pi_{\text{CC}} \Pi_{\text{STD}} \Pi_{\text{int}}
\]

The Safety Critical System software has the data as follows:

- Number of Versions = 39
- Comment to Code Ratio = 63%
- Conforming to Standards = Fully, 100%
- No. of Interrupts = Nil

\[
\Phi_{\text{TOTAL}} = 108550.9623 \times 10^{-10} \times [1.0/(1+(0.1\times39))] \times [1.0/(1+(0.1\times63))] \times [1+(10-10\times(0.01\times100))] \times [1+(10 \times 0)]
\]

\[
\Phi_{\text{TOTAL}} = 108550.9623 \times 10^{-10} \times [0.204] \times [0.136] \times [1.0] \times [1]
\]

\[
=3034.6928 \times 10^{-10} \text{ failures / combination}
\]
Step 5: Estimation of reliability

The reliability over the number of input combinations is given by Eq. 7.4

\[ R(c) = e^{-(\Phi \cdot \text{COMBINATIONS})} \]

The Table 7.5 lists the estimated reliability based on the number of input combinations for the \( \Phi_{\text{TOTAL}} \) of 3034.6928 failures / combination with the variation of criticality levels as Safety Critical, Safety Related and Non-nuclear safety. The reliability of the sample system for 5000 combinations becomes \( R(5000) = 0.9984838041 \).

Table 7.5 Estimated reliability based on the number of input combinations

<table>
<thead>
<tr>
<th>Reliability with input Combinations</th>
<th>Safety Critical (SC1)</th>
<th>Safety Related (SR)</th>
<th>Non Nuclear Safety (NNS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(100)</td>
<td>( e^{-0.003034 \times 100} )</td>
<td>( e^{-0.00003034 \times 100} )</td>
<td>( e^{-0.00003034 \times 100} )</td>
</tr>
<tr>
<td></td>
<td>= 0.9999696535</td>
<td>= 0.9996965767</td>
<td>= 0.9969699072</td>
</tr>
<tr>
<td>R(1000)</td>
<td>( e^{-0.003034 \times 1000} )</td>
<td>( e^{-0.00003034 \times 1000} )</td>
<td>( e^{-0.00003034 \times 1000} )</td>
</tr>
<tr>
<td></td>
<td>= 0.9996965767</td>
<td>= 0.9969699072</td>
<td>= 0.970115638</td>
</tr>
<tr>
<td>R(5000)</td>
<td>( e^{-0.003034 \times 5000} )</td>
<td>( e^{-0.00003034 \times 5000} )</td>
<td>( e^{-0.00003034 \times 5000} )</td>
</tr>
<tr>
<td></td>
<td>0.9984838041</td>
<td>= 0.9849410729</td>
<td>= 0.859246015</td>
</tr>
<tr>
<td>R(10000)</td>
<td>( e^{-0.003034 \times 10000} )</td>
<td>( e^{-0.00003034 \times 10000} )</td>
<td>( e^{-0.00003034 \times 10000} )</td>
</tr>
<tr>
<td></td>
<td>0.9969699072</td>
<td>= 0.97010891721</td>
<td>= 0.738303715</td>
</tr>
</tbody>
</table>

The typical safety critical software deployed in the reactor is taken for reliability estimation. This software basically does the core supervision, initiates alarm if the value exceeds the Alarm limits. It also gives command to shutdown whenever the parameter is approaching to design limits. These functions help to keep the core integrity and safe operation of the plant. The version history of the software is tabulated in Table 7.6 and 7.7.
<table>
<thead>
<tr>
<th>Version No. &amp; Date</th>
<th>obs</th>
<th>Description of important observations(obs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.5.2005 Ver 1.0</td>
<td>30</td>
<td>Comments were less - 23%, Three Functions Cyclomatic Complexity (CC) &gt; 15, One function Nesting is greater than 5, Initialisation of some variables missing, Multiple Returns in functions, Usage of Library, Power Function is wrong i.e $10^{1.9} = 79.4$ instead of 10, Compute was wrong, cr_count ++ - No Reset keeps on incrementing</td>
</tr>
<tr>
<td>7.1.2006 Ver 4.0</td>
<td>20</td>
<td>Comments 21%, CC &gt; 15 : 5 functions, Nesting &gt; 3 : 2 functions, unused variables, Multiple Returns, Library Used, Float equality check</td>
</tr>
<tr>
<td>17.1.2006 Ver 4.1</td>
<td>15</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 2 functions, unused code, Library Used, typecasting</td>
</tr>
<tr>
<td>2.2.2006 Ver 4.2</td>
<td>10</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 fn, Library Used, Hard-coding of numbers</td>
</tr>
<tr>
<td>14.2.2006 Ver 4.21</td>
<td>5</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function, Library Used</td>
</tr>
</tbody>
</table>
Table 7.7 Software Version history of SCS – Part II

<table>
<thead>
<tr>
<th>Version No. &amp; Date</th>
<th>obs</th>
<th>Description of important observations(obs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.3.2007 Ver 4.21</td>
<td>5</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function</td>
</tr>
<tr>
<td>13.1.2009 Ver 4.3</td>
<td>5</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function, Unused Macros &amp; Variables</td>
</tr>
<tr>
<td>30.1.2009 Ver 4.4</td>
<td>3</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function, This version is temporary</td>
</tr>
<tr>
<td>2.9.2009 Ver 4.3.1</td>
<td>3</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function</td>
</tr>
<tr>
<td>13.12.2011 Ver 4.3.1</td>
<td>3</td>
<td>Comments 46%, CC &gt; 15 : 3 functions, Nesting&gt;3 : 1 function</td>
</tr>
<tr>
<td>10.2.2012 Ver 4.4</td>
<td>2</td>
<td>CC &gt; 15 : 2 functions, Nesting &gt;2 : 1 function, Ref. Tolerance count is high</td>
</tr>
<tr>
<td>15.1.2013 Ver 4.4.1</td>
<td>2</td>
<td>CC &gt; 15 : 1 function, Nesting&gt;2 : 1 function, This version is temporary</td>
</tr>
<tr>
<td>17.10.2013 Ver 4.5</td>
<td>2</td>
<td>CC &gt; 15 : 1 function, Nesting&gt;2 : 1 function, Separate PDSR for sphere Pac Fuel Sub Assembly</td>
</tr>
</tbody>
</table>

7.4 Inferences

A comprehensive procedure to estimate software reliability for a given software program coded in “C” language has been established. The data required for each function is obtained by running the “Static Analyser” tool developed as a part of this research.