**I J C T A**, 9(13) 2016, pp. 6183-6191
© International Science Press

# Enhancing Software Quality Using Fuzzy Logic and Program Slicing

**P. Dhavakumar[1] and N. P. Gopalan[2]**

**ABSTRACT**

In this paper a new program slicing metric is introduced. This metric makes use of the slicing information to measure size, complexity and coupling cohesion properties of the program. Compared with traditional code metric, slicing involves measures for program behaviours. As Reliability is one of the key parameters in estimating the quality of the software, there is a considerable amount of increase in reliability and also there is an increase in the quality of the software.

***Index Terms***: Fuzzy-Logic, Software Slicing, Software Metrics, Reliability Models

## 1. INTRODUCTION

The advent of fuzzy logic plays a vital role in estimating the reliability of the software products to an unimaginable extent, but due to the increasing demand in software industry, fuzzy logic alone can't make software industry to succeed. Engineers and researchers have provided many new models to estimate the reliability but as said earlier; as time flies the models fades. In this paper have designed a new software reliability estimation model that not only depends on fuzzy logic but also involves program slicing. The goal of this design is to achieve a minimum gain of 3-6% in reliability of the software. The question raised is whether the program slicing can be used for identifying the faults in the software or not, and if so is it a reliable or not. The answer for the above question is program slicing can be efficiently used in identifying faults process in software and it is reliable too. The main use of program slicing is predicting fault-prone in software components. The result for this process is well defined and promising [1]. Faults in the software system are the major problem which is overlapped by using this program slicing. The problems are solved by formulating hypotheses of the relationship between slice characteristics and faults in the software. Testing hypotheses on samples of software components are also done by program slicing to identify faults. Program slicing is done by java which is an object oriented programming language. Techniques of program slicing are static slicing; quash static slicing, dynamic slicing and conditional slicing. More slicing techniques are also used in the process. The main objective of program slicing is extracting parts of computer programs by tracing the program, control and dataflow related to some data item. It is applicable in the fields like debugging, program comprehension and understanding program integration, cohesion measurement, re-engineering, maintenance and testing [5]. It also includes slicing methods like forward slicing, backward slicing, syntactic slicing and semantic slicing to check errors on software.

Program slicing consists of many methods which focus on specific component of a program to check errors from which all ambiguous methods a program can be made reliable and error. The methods of slicing teachers us the statement deletion based methods for programs written in procedural languages. This also included the large scale slicing research tools. And most of them are related to software testing or debugging. The framework of statement deletion based slicing method can be extended with more powerful simplification

[1] Department of Computer Science and Engineering, Periyar Maniammai University, Thanjavur, *E-mail: dhavakumar@pmu.edu*

[2] Department of Computer Application, National Institute of Technology, Tiruchirappalli, *E-mail: npgopalan@nitt.edu*

rules. It can be computed by applying a broader range of transformation rules, including statement deletion [25].In software slicing program, slice metrics is used. There has an advantage of DSL (Domain Specific Language). The empirically collected data from system form the basis of a DSL. The analysed multiple versions of the system are sliced and it functions in three separate ways such as input, output and global variables. It is a potential slice based DSL. It will compromise for maximum expressive power and effectiveness [6]. Data Analysis and Data Collection gives the description of three categories used to slice. The categories of variable are formal ins(IP), formal outs(OP), Global Variables(GL). To integrate fault analysis with program analysis by implementing the DSL based program slicing is the major metric. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Software reliability is the key task for achieving the high reliability of any software industry. The ability of a system or component to perform its required function under stated conditions for a specified period of time [2]. Measuring the software reliability is a difficult problem because we don't have a good understanding about the nature of software. POFOD (Probability of Failure On Demand) is an important measure for safety critical system. The utilities of Fuzzy logic techniques extends up to XGMML, Web Services and the .NET framework future work is proposed to extend the system from the prototype stage and overcome the problem by making of poor scores. In evaluation of current metrics visualisation, there are many software packages available that provide visualisation of software metric. The decision support system (DSS) offer a vast array of visualisation and techniques present metric data the boundary definition is fuzzy logic. Fuzzy logic provides the ability for a machine to perceive the world as humans do by representing vague and ambiguous knowledge [7].

The ability of a system or component to perform its required functions under stated conditions for a specified period of time. The terms errors, faults and failures are often used interchange able. In software, an error is usually a programmer action or omission that results in a fault. This new model for software quality is reliable. A software reliability management program requires the establishment of a balanced set of user quality objectives and identification of informative quality objectives that will assist in achieving the user quality objectives. By increasing the requirements it is easy to produce ambiguous terminology in software metrics [24].The only way to help control defects and reduce high maintenance costs is to use refactoring. The software metrics try to qualify particular characteristic of a software systems, such as quality, maintainability or reliability. The coupling metrics provide insight into the interaction between the modules in the system, while the cyclomatic complexity gives the insight into the complexity metric of each individual module. The return on the investment in these refactoring forces can be measured in lower error rates, fewer test cases for module and increased overall understand ability and maintainability. The coupling, cohesion and cyclomatic complexity have because accepted metric in program slicing [13]. There are some types of slicing approaches in which they depend on the program and the type of program slicing namely static slicing and dynamic slicing [21]. The applications of slicing reside in debugging, regression testing, software maintenance, and reverse engineering, software quality assurance. It can be applied in many other problem areas. It performs the leading role in the advanced research topic in computer science especially in software engineering.

## 2. COMPUTINGMETHODOLOGY

First, let us characterize Software Reliability. Software unwavering quality is altogether different from equipment dependability.

**Hardware Reliability**: It is the capacity of equipment to perform its capacities for some timeframe with no disappointment. The equipment dependability is portrayed on the premise of shower tub curve.

**Software/Programming Reliability**: Software Reliability is the likelihood of disappointment free programming which works for a predefined timeframe in a predetermined situation. Programming Reliability is additionally a vital component influencing framework dependability. Programming unwavering quality is not quite the same as equipment dependability in light of the fact that it characterizes the outline flawlessness, as opposed to assembling flawlessness.

Reliability of Software is measured using its sub characteristics which are,

1) Maturity
2) Fault Tolerance
3) Recoverability
4) Reliability Compliance

Let $P_f$ be the Probability of Failure of an software and R be the reliability, then by definition R can be expressed as,

$$R = 1 - Pf \tag{1}$$

The value of R is a discrete value that lies between 0 and 1. Practical Cases have shown that removing 60%of the faults will increase the reliability of the software by 3%. The reliability of the software is measured using POFOD(Probability of Failure on Demand), ROCOF(Rate of Occurrence of Failure) and MTTF(Mean Time To Failure). The relationship between cost and reliability is directly proportional. When reliability increases the cost also increases. Figure 1 shows the relationship graph between Cost and Reliability.

## 2.1. Reliability Models

There are number of software reliability models which are used for different purposes. Some of the models are explained below compared to the static and the dynamic slice performs the leading role.

***(a) Jelinski Moranda Model**:* The Jelinski Moranda model is the earliest models in software reliability and was founded in 1972.It is the time between failure models. It assumes N software faults at the start of testing, and that assumes that the failure occur purely, at random and all the faults contribute equally to cause a failure during testing. It also assumes the fix time is negligible and the fix for each failure is perfect. The software products failure rate improves by the same amount at each fix.

***(b)*** **Littlewoods Model**: It is similar to JelinskiMoranda Model but it assumes that different faults have different size. Large sized defects tend to detected and fixed earlier. As the number of errors is driven down with the progress in the test, so this model is the average error size causing a law of diminishing return in debugging.

***(c) Goel Okumoto Imperfect Model*:**Non Homogeneous Poisson Process Model: The NHHP model was formulated in 1979. The J-M model assumes that the fix is negligible and that the fix for each failure is perfect. It assumes perfect debugging .In practice this is not always the case. In the process of fixing a defect new defects may be injected. This model proposed imperfect debugging model to overcome the limitation of the assumption. In this model the hazard function during the interval between the (i-1)st and the failure is given

$$Z(ti) = [N - p(i-1)]\,\lambda \tag{2}$$

where N is number of faults at start of testing

P is probability of imperfect debugging

$\lambda$ failure rate per fault

***(d) The Delayed S and Inflection S Models:*** With the help of defect removal process Yamada said that a testing process consists of not only a defect detection process but also defect isolation process because of

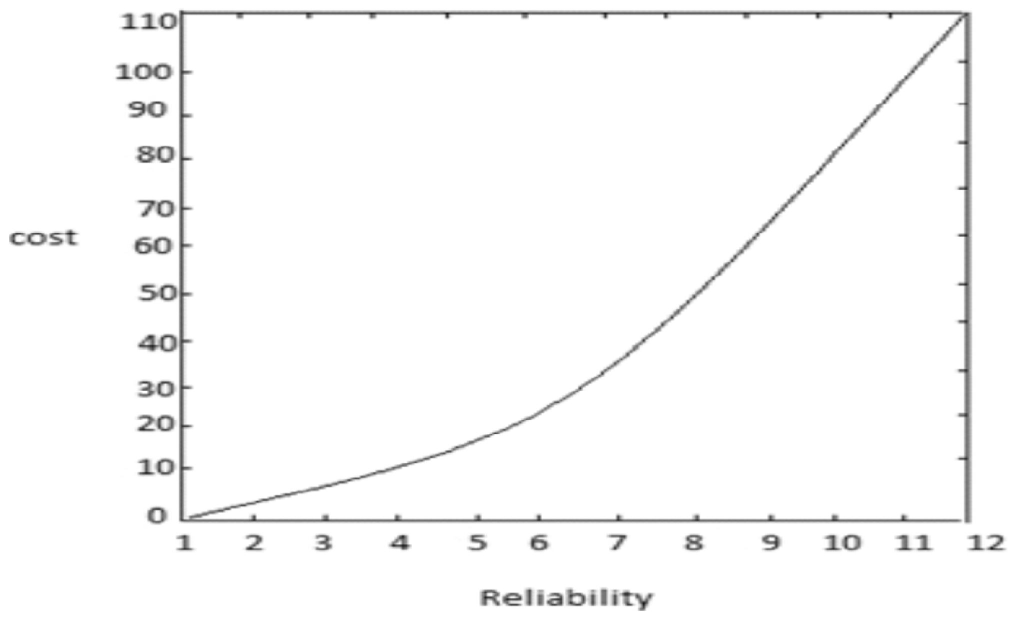

**Figure 1: Cost Vs Reliability**

the time needed for it is similar to Jelinski Moranda Model but it assumes that different faults have different size. Large sized defects tend to be detected and fixed earlier. As the number of errors is driven down with the progress in the test, so it is the average error size causing a law of diminishing return in debugging.

***(e)*** **GoelOkumoto Non Homogeneous Poisson Process Model**: The NHHP model is given in 1979.It is concerned with modelling the number of failures observed in given testing intervals. It defines the cumulative number of failure observed at time t.N (t) can be modelled as a non-homogenous Poisson process with the time dependent failure rate.

Failure analysis shows significant delay can occur between time of first failure observation and the time of reporting. They offer the delayed S-shaped reliability growth model for such a process in which the observed growth curve of cumulative number of detected defects is S-shaped. The model is based on the non-homogeneous Poison process but with a different mean value function to reflect the delay in failure reporting.

## 2.2. Fuzzy Logic

Fuzzy Logic is newly different from the traditional computational logic. In fuzzy logic the concept of partial truthfulness was introduced i lies between 0s and 1s. Fuzzy Logic is very useful to react human like thinking on the machine. Fuzzy logic involves two factors, Approximation (Granulation) and Degree (Graduation). The main process involved in using fuzzy logic to compute the metrics is fuzzication. Fuzzication is the process of translating the measured numerical values into fuzzy linguistic values. During fuzzication membership functions are applied and the degree of membership is determined. There are three types of fuzziers available, they are:

- Singleton Fuzzifiers
- Gaussian Fuzzifiers
- Trapezoidal Fuzzifiers

*Steps Involved In Fuzzy Approach*

*1)* Deciding Membership Function

2) Optimized selection of initial rule based fuzzy interference system
3) Perform the Inference
4) Calculate the % accuracy of the results

## 2.3. Program Slicing

Program slicing is a method used for abstracting from programs (i.e) Start with a subset of a program's behaviour and reduces it to a minimal form that still produces the behaviour. The reduced program is called a slice. It is helpful to find what all codes could influence a variable in a statement and is data flow analysis. To find slices the program is sliced backwards finding all possible influences on a variable. Two types of slicing techniques are available Static and Dynamic slicing. In Static slicing slices are computed statically using a dependence graph, while in case of dynamic slicing statements that affect the value of a variable for one specific input are taken. In this technique we use limited pre-processing (LP) algorithms that strikes a balance between pre-processing and slicing costs. The LP algorithm for program slicing is given in Algorithm 1.

## 2.4. Program Slicing Metrics

PS metrics indicate the size, complexity, coupling and cohesion of C programs. PS metrics are based on program slice information, which is of finer-granularity. Program slices have the additional advantage of capturing program behaviour, and hence metrics based slices are more directly related to these behaviours. We explain each of the PS metric items as follows.

The number of slices is a function that contains UC metrics, but slice count considers the global variables modified through the dereferencing of pointer variables. To achieve this, pointer analysis is used in the implementation. The number of vertices in a function program dependence graph use vertices Count. This metric is similar to Count Line Code in the UC metrics, but vertices Count are more fine-grained. For example, it treats the statement a = b () as two statements: one function call and one assignment. It additionally counts implicit vertices, such as global-formal-out vertices and edges Count. The number of dependence edges is a function program dependence graph. In contrast to the vertices Count metric, which weighs each statement evenly, edges Count is based on the control or flow dependence relationships of each statement with other statements.PS metrics indicate the size, complexity, coupling and cohesion of C programs. PS metrics are based on program slice information, which is of final granularity. Slice count isthe number of slices a function contains. To achieve this, pointer analysis is used in the implementation. Vertices count. The number of vertices in a functions program dependence graph. It treats the statement a = b() as two statements: one function call and one assignment. The number of dependence edges in a functions program dependence graph. Edges to vertices ratio is the ratio of the number of dependence edges to the number of vertices. A high Edges to vertices ratio indicates more logically complex code. Slice vertices sum is the sum of the vertices contained in each slice in a function. Max slice vertices is the number of vertices of the slice that have the maximum vertices in all the slices of a function. Global input is the number of function parameters and non-local variables used in a function. Global output. The number of non-local variables modified in a function. Direct fan out is the sum of function slices in the other module files whose output variables are directly modified in them and used in this function. Indirect fan out is the sum of function slices in the other module files whose output variables are indirectly modified in them and used in this function. Lack of cohesion metric indicates the cohesion of function slices and how much the slices in a function share the same vertices. The PS metrics listed above are computed at the function level of granularity. We have also developed a set of metrics at the file level corresponding to those at function level. The file level metrics are the sum of the function-level metrics for all the functions a file contains, except for lack Of Cohesion and max Slice Vertices, which are the average of the function-level metrics for all the functions in the file, and edges To Vertices Ratio, which is the ratio of sum of edges in a file to the sum of vertices in the file. We determine lack Of Cohesion by the overlap ratio of function slices, i.e. how much the slices in a function share the same vertices. But, since it is expensive to compute the ratio of shared vertices among a number of function slices, we calculate an approximate ratio of slice overlap by computing the ratio of the program lines each slice covers to the total number of lines in the function.

The PS metrics listed above are computed at the function level of granularity. We have also developed a set of metrics at the file level corresponding to those at function level. The file level metrics are the sum of the function-level metrics for all the functions a file contains, except for lack Of Cohesion and max Slice Vertices, which are the average of the function-level metrics for all the functions in the file, and edges To Vertices Ratio, which is the ratio of sum of edges in a file to the sum of vertices in the file. Note that, in practical computation of the program slices for functions, some normalization will be performed for complex statements. So, the actual PS metrics will take into account additional vertices and edges generated by normalization.

### 2.4. Metrics Computation Procedure

The procedure involved in computing the reliability metrics has two stages. Stage1 shows performance of program slicing using LP slicing algorithm. Stage2 involves the actual computation of the reliability metrics. The entire procedure is repeated for a set of benchmark programs and the readings are tabulated. The benchmark programs used are written in C language, the file linpack.c is used to obtain rolled source BLAS. Add -DROLL to the command lines and obtain unrolled source BLAS and add DUNROLL to the command lines. The other benchmark used is the Whetstone Double Precision Benchmark.

***Algorithm for Limited Pre-processing Algorithm***

```
Input: $\leftarrow Program(P), Variable(V), Input_D ata(I)$ Output: $\leftarrow Slices(S)$
Process: LP(P,V,I)
Begin
Perform LP
Add summary information to trace
Divide Trace(T) $\rightarrow$ Trace Blocks($B_t$)
End of Trace Block=Summary of downward exposed definitions
Perform: Backward Traversal if Definition == 'Found' then
Traverse Trace Block else
Skip to Start of Trace Block end if End
```

## 3. COMPARISON TECHNIQUE IN TERMS OF SOFTWARE RELIABILITY MODELS

The procedure is repeated twenty times by the traditional approach and the new approach. The values are recorded and the readings are tabulated as given in table. By plotting the readings we conclude that, this technique is very efficient method to enhance the quality through increased reliability. In the figure 2 source code has been converted into small piece of program using slicing techniques and figure 3 is used to compute the reliability based on the fuzzy logic.

## 4. RESULTS

The readings values of traditional methods and proposed method success rates are tabulated below. From this table the differences between the above two methods values are at peak most of the times for the proposed framework when compared with the traditional method for metrics computation.

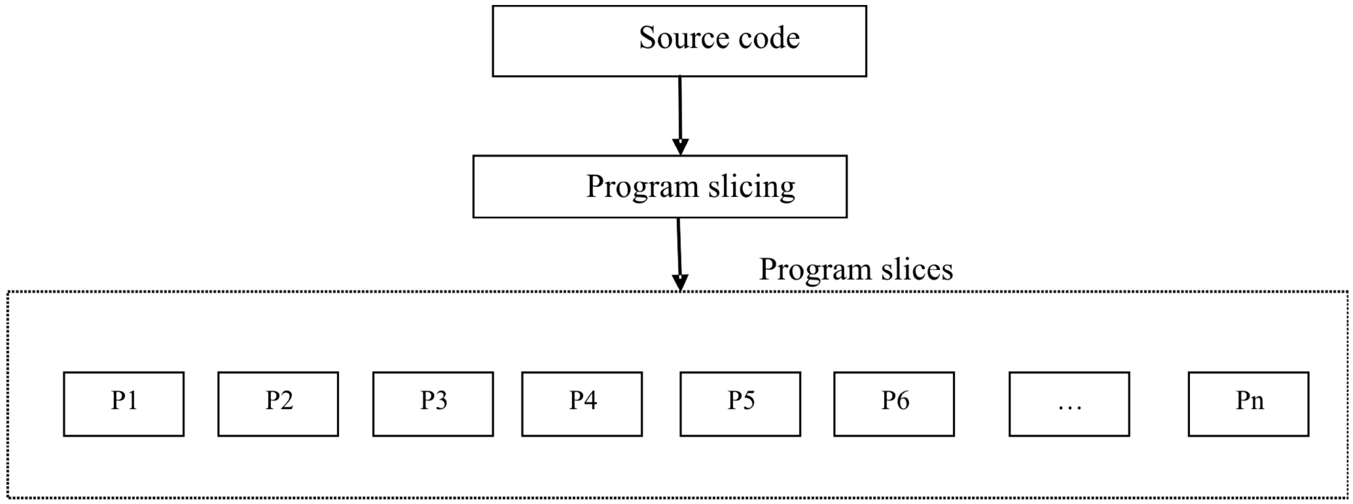

**Figure 2: Program Slicing**

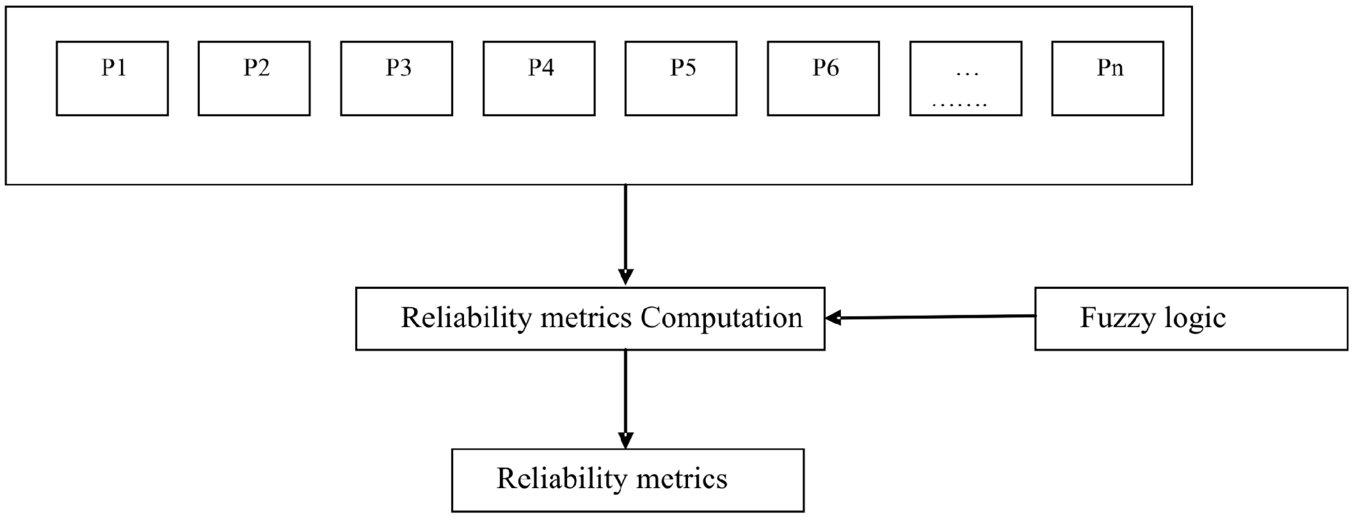

**Figure 3: Reliability Computation**

**Table of Success Rates (SR)**

| *Traditional Methods SR* | *Proposed Framework SR* | *Difference in SR* |
|---|---|---|
| 0.153 | 0.263 | 0.110 |
| 0.961 | 0.259 | -0.702 |
| 0.876 | 0.677 | -0.199 |
| 0.488 | 0.519 | 0.031 |
| 0.407 | 0.076 | -0.331 |
| 0.126 | 0.055 | -0.071 |
| 0.925 | 0.258 | -0.667 |
| 0.005 | 0.439 | 0.434 |
| 0.186 | 0.284 | 0.098 |
| 0.324 | 0.678 | 0.354 |
| 0.050 | 0.949 | 0.899 |
| 0.144 | 0.773 | 0.629 |

*contd. table*

| *Traditional Methods SR* | *Proposed Framework SR* | *Difference in SR* |
|---|---|---|
| 0.482 | 0.753 | 0.271 |
| 0.338 | 0.746 | 0.408 |
| 0.236 | 0.586 | 0.35 |
| 0.450 | 0.773 | 0.323 |
| 0.185 | 0.392 | 0.207 |
| 0.324 | 0.605 | 0.281 |
| 0.263 | 0.247 | -0.016 |
| 0.830 | 0.290 | -0.54 |
| 0.696 | 0.019 | -0.677 |
| 0.333 | 0.347 | 0.014 |
| 0.580 | 0.141 | -0.439 |
| 0.287 | 0.411 | 0.124 |

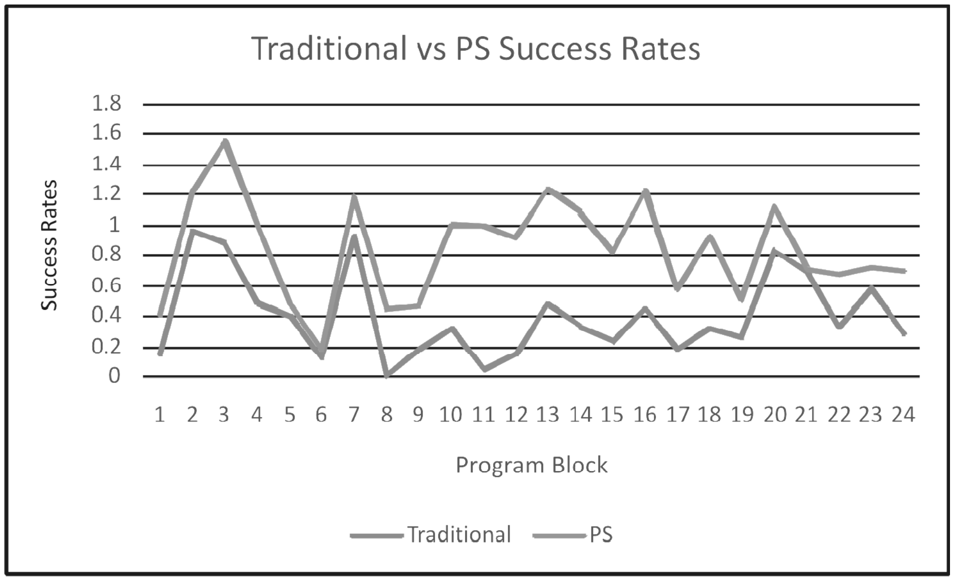

**Figure 4: Traditional vs PS Metrics**

## 5. CONCLUSION

Figure 4, the graph plots it is found that the success rate of the benchmark programs in traditional and PS metrics, the blue line indicate traditional methods success rate and red line indicate PS metric success rates. From the above graph has a high peak value on applying the proposed framework that involves program slicing and fuzzy logic. Thus we can conclude that applying program slicing to compute the metrics increases the reliability as the rate of success is directly proportional to the reliability of the software.

## REFERENCES

[1] Singh, Pradeep Kumar, Om Prakash Sangwan, Amar Pal Singh, and Amrendra Pratap. "A Framework forAssessing the Software Reusability using FuzzyLogic Approach for Aspect OrientedSoftware", International Journal of Information Technology and Computer Science, 2015

[2] Gurpreet Kaur, Kailash Bahl, "Software Reliability, Metrics, Reliability Improvement Using Agile Process", International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.

[3] Chiu, Kuei-Chen. "An improved model of software reliability growth under timedependent learning effects" , 2011 IEEE International Conference onQuality and Reliability, 2011.

[4] Wang, Shuanqi, Yumei Wu, Minyan Lu, and Haifeng Li. “Software reliability modeling based on test coverage”, The Proceedings of 2011 9th International Conference on Reliability Maintainability and Safety, 2011.

[5] N.Sasirekha, A.Edwin Robert and Dr.M.Hemalatha, “PROGRAM SLICING TECHNIQUES AND ITS APPLICATIONS”, International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3, July 2011.

[6] Steve Counsell, Tracy Hall1, David Bowes, Sue Black, “Program Slice Metrics and Their Potential Role in DSL Design”, Submitted to KISS 2009, Brisbane, 2009.

[7] Senior, Allison & Tepper, “Automated Software Quality Visualisation Using Fuzzy Logic Techniques” Communications of the IIMA, 2007 Volume 7 Issue1.

[8] Toolworks, “Maintenance, Understanding, Metrics and Documentation Tools for Ada, C, C++, Java, and FORTRAN,” 2006.

[9] Krinke, Jens. “Program Slicing”, Handbook Of Software Engineering And Knowledge Engineering Vol 3 Recent Advances, 2005.

[10] Sliwerski, T. Zimmermann, and A. Zeller, “When Do Changes Induce Fixes?” In Proceedings of Int’l Workshop on Mining Software Repositories (MSR 2005), Saint Louis, Missouri, 2005, I. H. Witten and E. Frank, Data Mining: Practical machine learning tools and techniques (Second Edition), Morgan Kaufmann, 2005.

[11] Gyimothy, R. Ferenc, and I. Siket, “Empirical Validation of ObjectOriented Metrics on Open Source Software for Fault Prediction,” Transactions on Software Engineering, vol. 31, no., pp. 897-910, 2005.

[12] M. Meyers and D. Binkley, “Slice-Based Cohesion Metrics and Software Intervention,” In Proceedings of the 11th Working Conference on ReverseEngineering (WCRE’04), Delft, the Netherlands, 2004, pp. 256-265.

[13] Dr. Ricky E. Sward, Dr. A.T. Chamillard, Dr. David A. Cook “Using Software Metrics and Program Slicing for Refactoring” The Journal of DefenceSoftware Engineering, July 2004.

[14] Li, “A Hierarchical Slice-Based Framework for ObjectOriented Coupling Measurement”, TUCS Technical Report No.415, Turku Center for Computer Science, Abo Akademi University, 2001.

[15] M. Khoshgoftaar and E. B. Allen, “Predicting the Order of Fault-prone Modules in Legacy Software,” In Proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998, pp. 344-353.

[16] J. McCabe and A. H. Watson, “Software Complexity,” Crosstalk, vol. 7, no. 12, pp. 5-9, 1994.

[17] M. Bieman and L. M. Ott, “Measuring Functional Cohesion,” IEEE Transactions on Software Engineering, vol. 20, no.8, pp. 644-657, 1994.

[18] M. Khoshgoftaar and E. B. Allen, “Ordering Fault-Prone Software Modules,” Software Quality Journal, vol. 11, no. 1, pp. 19-37, 2003.

[19] L. Ott and J. Thuss, “Slice Based Metrics for Estimating Cohesion,” In Proceedings of the First International Software Metrics Symposium, 1993.

[20] Horwitz, T. Reps, and D. Binkley, “Inter-procedural Slicing Using Dependence Graphs,” ACM Transactions on Programming Languages and Systems, vol.12, no. 1, pp. 2660, 1990.

[21] Ferrante, K. J. Ottenstein, and J. D. Warren, “The Program Dependence Graph and its Use in Optimization,” ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, pp. 319-349, 1987.

[22] M. Weiser, “Program Slicing,”IEEE Transactions on Software Engineering, vol. 10, no. 4, pp. 352-357, 1984.

[23] Kai Pan, Sunghun Kim, E. James Whitehead, Jr, “Bug Classification Using Program Slicing Metrics”, Dept. of Computer Science University of California, Santa Cruz Santa Cruz, CA 95064 USA.

[24] Sonakshi Rana, Rahul Kumar Yadav, “A Fuzzy Improved Association Mining Approach to Estimate Software Quality”

[25] Dr. Linda Rosenberg, Ted Hammer, Jack Shaw “SOFTWARE METRICS AND RELIABILITY” NASA GSFC.

[26] Andrea De Lucia, “Program Slicing: Methods and Applications”.

[27] Sue Black, Steve Counsell, Tracy Hall, Paul Wernick, “Using Program Slicing to Identify Faults in Software”