Evaluate reliability, availability, serviceability of computing system using fault injection and analysis tool

Harjeet Singh Chhabra
PhD(Pursuing), M.S , B.E (CSE)
Information Technology Department
Jodhpur Institute of Engineering and Technology
Jodhpur, India
harjeetsingh.chhabra@jietjodhpur.com

Shubha Agarwal
M-Tech(SE), MCA ,BSc(Computer Science)
Information Technology Department
Jodhpur Institute of Engineering and Technology
Jodhpur, India
shubha.agarwal@jietjodhpur.com

***Abstract*— Measuring a system's performance is the well-understood approach for evaluating and comparing computing systems. Besides being fast there are a number of other demands placed on computing systems. Software metrics and performance-oriented benchmarks are not the most suitable way to evaluate systems, given the extra-functional demands concerning reliability, availability and serviceability placed on them. What is required is an evaluation methodology that allows us to estimate predicate faults in the given system by the use of fault injection tool. This paper suggest new approach for developing an evaluation methodology that can be used to compare systems based on fault-injection tools that can be used to study the failure behavior of computing systems and analysis tool that can be used to analyze the system on the basis of LOC, complexity, etc. .**

Keywords- RAS (reliability, availability and serviceability), Fault-injection tool JACA, Analysis tool CCCC.

I Introduction

Measuring a system's performance is the well-understood approach for evaluating and comparing computing systems. Besides being fast there are a number of other demands placed on computing systems. Computing systems that meet additional non-functional requirements such as reliability, high availability and serviceability has increased research efforts into enhancing the reliability, availability and serviceability (RAS) capabilities of systems as well as next-generation self-managing, self-configuring, self-healing, self-optimizing and self-protecting systems [1].

A common desired characteristic of these systems is that they collect, analyze and act on information about on operation and changes to their environment while meeting their functional requirements. Whereas instrument systems collect, analyze and act on behavioral and environmental data potentially impact the performance of a system by diverting processing cycles away from meeting functional requirements, these diverted cycles are used by mechanisms concerned with improving the RAS capabilities of the system. To reason about tradeoffs between RAS-enhancing mechanisms or to evaluate these mechanisms and their impact something need other than software metrics. Whereas software metric are suitable for studying the feasibility of having RAS-enhancing mechanisms activated, i.e., to demonstrate that the system provides "acceptable" performance with these mechanisms.

Performance measures do not allow us to analyze the expected or actual impact of individual or combined mechanisms on the system's operation. Software metric limits the scope and depth of analysis that can be performed on systems possessing RAS-enhancing mechanisms. Analysis the RAS capabilities of the systems address three evaluation-related factors. First, identifying practical fault-injection tools that can be used to study the failure behavior of computing systems and exercise any mechanisms the system has available for resolving problems. Second, identifying analysis tool that can be used to analyze the system on the basis of LOC, complexity. Third, developing an evaluation methodology that can be used to compare systems based on the above two factors.

II. SOFTWARE METRICS AND FAULT MEASUREMENT

Jaca is a tool for injection of faults and it use software engineering concepts. Jaca had been developed in Java language program and currently have thousands and thousands lines of code. In the

last versions, a graphic interface to facilitate the relationship between Jaca and users. Besides, as Jaca is used in science research, there are needs to implement new requirements. The need for software to evolve as its usage and operational goals change has added the non-functional requirement of adaptation to the list of facilities expected in systems. Example system-adaptations include the ability to support reconfigurations, repairs, self-diagnostics or user-directed evaluations driven by fault-injection [2].

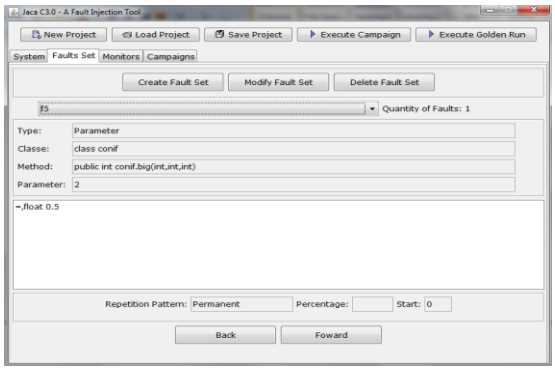


Figure 1: Jaca Tool frontend

Measurements of source code referred to as 'software metrics', or more precisely 'software product metrics' (as the term 'software metrics` also covers measurements of the software process, which are called 'software process metrics'). There is a reasonable consensus among modern opinion leaders in the software engineering field that

measurement of some kind is probably a Good Thing, although there is less consensus on what is worth measuring and what the measurements mean[3].

Software metrics tool [4] as a program which implements a set of software metrics definitions. It allows to assess a software system according to the metrics by extracting the required entities from the software and providing the corresponding metrics values.

CCCC is a tool[4] for the analysis of source code in various languages (primarily C++), which generates a report in HTML format on various measurements of the code processed. Although the tool was originally implemented to process C++ and ANSI C, the present version is also able to process Java source files, and support has been present in earlier versions for Ada95. The name CCCC stands for 'C and C++ Code Counter'.

Measurements of source code of this kind are generally referred to as "software metrics", or more precisely software product metrics[6]. There is a reasonable consensus among modern opinion leaders in the software engineering field that measurement of some kind is probably a Good Thing, although there are fewer consensuses on what is worth measuring and what the measurements mean.

Table 1: Report generated by CCCC tool of source code

Metric	Tag	Overall	Per Module
Number of modules	NOM	3	
Lines of Code	LOC	46	15.333
McCabe's Cyclomatic Number	MVG	6	2.000
Lines of Comment	COM	12	4.000
LOC/COM	L_C	3.833	
MVG/COM	M_C	0.500	

III. EVALUATING RAS CAPABILITIES

Evaluating and comparing the Reliability, Availability and Serviceability (RAS) capabilities of systems requires reasoning about aspects of the system's operation that may be difficult to capture or quantify using software metrics alone. An additional consideration for evaluating the RAS capabilities of systems is that the notions of "good" and "better" are dependent on the environmental constraints governing the system's operation. For example, service level agreements (SLAs), policies, and internally/externally visible service level objectives including but not limited to: uptime guarantees, meeting production targets, reducing production delays, improving problem-resolution and service-restoration activities, etc. Whereas there are aspects of the environmental constraints that can be evaluated using software metrics, such as response time guarantees in SLAs, these metrics are insufficient for evaluating other constraints.

In below example analysis numerical parameter values shown in Table to describe a specific failure and complexity scenario used to evaluate the efficacy of applications.

Table 2: Output generated by 3 source codes after applied on JACA and CCCC tool

A. Reliability Measures

	App 1	App 2
LOC	213	46
M_C complexity (e-v+2)	(high) 1.48	(very low) 0.5
No. of Faults	9	3
effort (Person-Month)	0.562	0.1
Availability (nano sec)	12	0.01

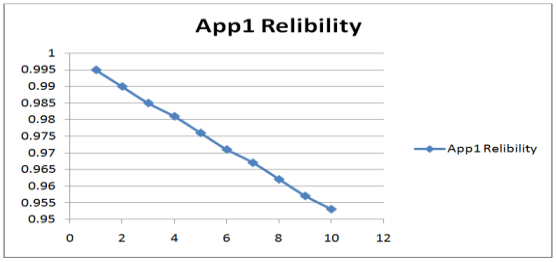
Reliability measures emphasize the occurrence of undesirable events in the system. There are a number of forms and metrics that can be used to express the reliability of a system including:

Reliability Functions – the probability that an incident of sufficient severity has not yet occurred since the beginning of a time interval of interest.

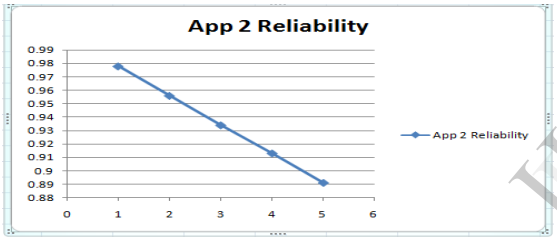
Line of code (LOC) – the line of code of an application.

Frequency of Faults –number of faults that occur in application at run time.

$$\text{Reliability} = 1 - \frac{\text{no.of faults}}{\text{LOC}} \qquad (1)$$



Graph 1: Graph between reliability and fault injected in source code 1 on x and y axis respectively



Graph 2: Graph between reliability and fault injected in source code 2 on x and y axis respectively

By both graphs, it is depicted that as faults increased, reliability is decreased. So the formula to calculate reliability is proved correctly.

B. *Availability Measures*

Availability measures capture the proportion of total time in which a system is in an operational condition. To discuss system availability using the RAS model, identify faults and effort which direct affect to the availability of an application[5].

There are three forms and metrics that can be used to express the availability of a system :

Effort (Person-Month) – how much effort has to be applied to make the system available.

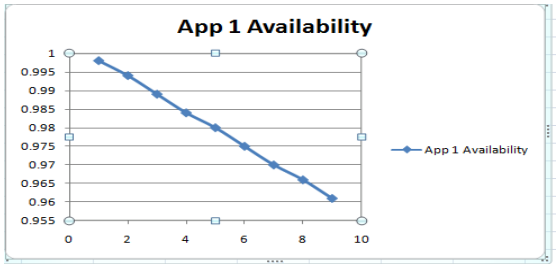
Number of faults– number of faults injected in run time by fault-injection tool.

Line of Code (LOC) – total line of code of an application counted by Analysis tool.

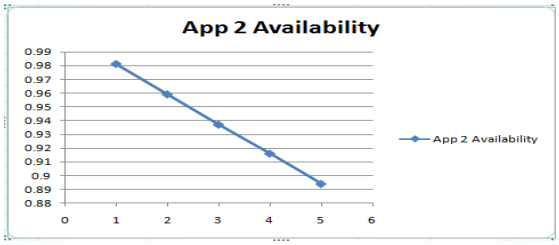
$$\text{Availability} = 1 - \frac{(\text{no.of faults} + \text{effort})}{\text{LOC}} \qquad (2)$$

$$\text{Effort} = a_b \ (\text{KLOC})^{b_b} \qquad (3)$$

Where KLOC means kilo line of code of project



Graph 3: Graph between availability and fault injected in source code 1 on x and y axis respectively



Graph 4: Graph between availability and fault injected in source code 2 on x and y axis respectively

By both graphs, it is depicted that as faults increased, availability is decreased. So the formula to calculate availability is proved correctly.

C. *Serviceability Measures*

Serviceability measures capture the impacts of system failures and/or remediation activities and programming complexity. Serviceability defines that how much the system is repairable or can say it is maintainable.

For evaluation purposes, when discuss the serviceability of a system, specifically interested in qualifying the fault injected and McCabe's cyclomatic complexity in the ratio of line of code of project.

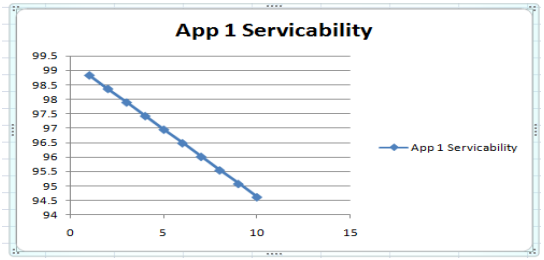
Three measure criteria are used to evaluate the serviceability of system or project or an application, they are

LOC- line of code of an application in which fault is injected.

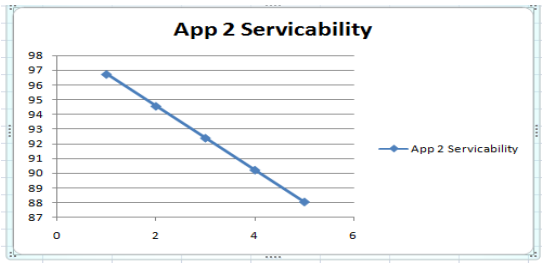
No. of faults – no. of faults injected by Fault-injected tool by changes in parameters of processes or return value type.

McCabe's cyclomatic complexity (M_C) – calculated through Analysis tool "CCCC" that define the complexity of data flow of project. Complexity is evaluated by the equation "e-v+2" , in which 'e' defines the total edges 'v' defines the total vertices of the graph that represent the system[7].

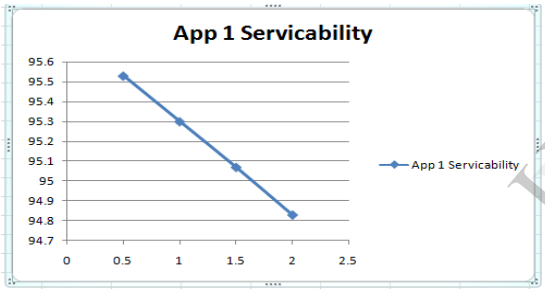
$$\text{Serviceability} = \frac{\text{LOC} - \text{no.of faults} - \text{M\_C}}{\text{LOC}} * 100 \quad (4)$$



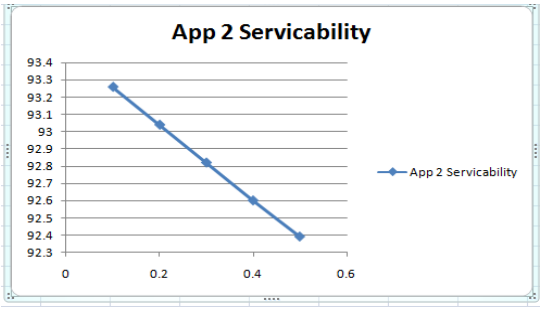
Graph 5: Graph between serviceability and fault injected in source code 1 on x and y axis respectively



Graph 6: Graph between serviceability and fault injected in source code 2 on x and y axis respectively



Graph 7: Graph between serviceability and complexity in source code 1 on x and y axis respectively



Graph 8: Graph between serviceability and complexity in source code 2 on x and y axis respectively

By both graphs, it is depicted that faults as well as complexity of system is increased, Serviceability is decreased. So the formula to calculate Serviceability is proved correctly.

VI. CONCLUSION

As the future work above demonstrates, this paper enables the beginning of new research areas, especially in the areas of realizing systems capable of runtime adaptations and improving fault-injection tools and environments used for RAS evaluations. Further, this paper presents a framework for developing RAS benchmarks for systems that combines practical tools with rigorous analytical techniques. Ultimately, I hope the work done here bridges the gap between practical and analytical approaches for studying and understanding the failure behavior of systems and reasoning about mechanisms that improve the reliability, availability and serviceability of current and next-generation systems.

REFERENCES

[1] Rean Griffith and Gail Kaiser. Manipulating managed execution runtimes to support self-healing systems. In DEAS'05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software,, New York, NY, USA, 2005. ACM Press.

[2] Rean Griffith and Gail Kaiser. A Runtime Adaptation Framework for Native C and Bytecode Applications. In 3rd International Conference on Autonomic Computing, 2006.

[3] Mei-Chen Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. Computer, 30(4), 1997.

[4]sourceforge . http://sourceforge.net/projects/cccc.

[5] Ji Zhu et al. R-Cubed: Rate, Robustness and Recovery An Availability Benchmark Framework. Technical Report SMLI TR-2002-109, Sun Microsystems, 2002.

[6] S.R.Chidamber and C. F. Kemerer. A MetricsSuite for Object-Oriented Design. IEEE Transactions on Software Engineering, 1994.

[7] B.Henderson-Sellers. Object-oriented metrics: measures of complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.