Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 8 (2013) pp. 167-176

A Survivability Architecture for Object-Oriented Software Systems

Aborisade, Dada O.[1*], Sodiya Adesina S.[2] and Ikuomola Aderonke J[3]

[1]Department of Computer Science
Federal University of Agriculture
Abeokuta, (FUNAAB) Nigeria.
aborisadeda@funaab.edu.ng

[2]Department of Computer Science
Federal University of Agriculture
Abeokuta, (FUNAAB) Nigeria.
sinaronke@yahoo.co.uk

[3]Department of Computer Science
Federal University of Agriculture
Abeokuta, (FUNAAB) Nigeria.
deronikng@yahoo.com

Abstract: The design of software systems has become an important research area of interest because of the role software plays in all facets of human lives. In spite of existing software development techniques and approaches, software failure is still a common occurrence, especially in safety-critical environments. In this paper, a survivability architecture based on threshold scheme and code rejuvenation for object-oriented software system is proposed, with intention to solve the problem of software degradation commonly caused by steady growth in classes and methods contained in object oriented software. A threshold value (T_V) is set to the final degradation point for the software system with reference to the behaviour of methods and classes it contain. Probability density function theory was used to derive values for critical region denoted by (C_V) and also to determine whether the probability of threshold value (T_V) falls within the critical region or not. A mechanism was integrated into the architecture to monitor and determine the steady growth of methods and classes such that when threshold value $T_{V)}$ <= critical value (C_V) set within the critical region, the methods and classes that constitute the software system are re-initiated in self-healing process. The architecture is tested using two software programs developed to implement treemap algorithms using nine (9) attributes. Experimental observations show that the number of classes in both standard and rejuvenated-enriched program remain at the count of 50 after seven (7) days of running the programs. It was also observed that while the number methods grow from 250 to 2500 in standard program it remain at the count of 250 in rejuvenated-enriched program. Therefore, the proposed architecture is observed to be capable of preventing code degradation and failure while the software without the proposed architecture embedded could not prevent degradation and failure as a result of classes and methods growth.

Keywords: Survivability, Object-oriented systems, Threshold Scheme, Software rejuvenation, and Code rejuvenation

***Corresponding author:** aborisadeda@funaab.edu.ng

I. Introduction

The trends in the World today point to the fact that human activities depend, and would continue to rely on software systems for their day to day operational activities. These software systems are also relied upon by individuals and corporate bodies. As the importance and complexity of software systems increase, software practitioners and researchers have not failed to advocate for a more systematic and effective development methods for software systems [1]. A number of these software systems are designed using object-oriented techniques. Object-oriented Software design approach follows a design approach whereby the software is composed of chunks of codes that are referred to as classes; a basic unit of an object- oriented software system. The concept of a class presents information and describes behaviour that manage the information. The unavoidable reliance on software systems prompted the need for an effective survivability model in Object-oriented software design, especially for mission-critical applications. Survivability is defined as the ability of a system to continually provide essential services to support the system mission even in the presence of malicious attacks or system failures [17]. In this paper, we describe survivability as the capability of every unit of a software system component (i.e.

class) to continually play certain roles expected to contribute to the overall smooth running of the software systems in a timely manner in the face of system failure. When compared with the definition by [17], the term system used in broad sense refers to software system that supports application functional requirements of an operational environment. The term mission refers to specific role a software system is to play in an application working environment. The term timely manner refers to the need of a software system to be available and capable of rendering its expected services within a time space that must not be noticeable during failure. The terms attack, failure, and accident refer to all potentially damaging events. Software system attacks are potentially damaging events perpetuated by an intelligent adversary on any or all of its components. Software attack could include probes and denials of service. As software systems are deployed in mission-critical environment, the need for its survivability is very important for mission-critical systems. As the reliance on software systems increases, its survivability picture becomes more and more complicated [3]. Failures in software system are potentially damaging events caused by deficiencies in the system or an external element on which the system depends such as software design errors, hardware degradation, human errors or corrupted data [4]. Software aging has also been identified as the cause of software failure. Software aging effects are the practical consequence of errors caused by software fault activations. They work by gradually leading the system's erroneous state towards a failure occurrence [2]. Accidents on the other hand describe a broad range of randomly occurring and potentially damaging events such as natural disasters. Accidents are often externally generated events (e.g. outside the system) and failures are typically internally generated events [4]. One important consideration for the success of survivability in any software system is application-level rejuvenation technique. Fine-grain techniques such as application-level rejuvenation strategies are better approach to mitigate the aging effects in software systems [2]. This research paper presents an architecture for ensuring survivability of object oriented software systems usually occasioned by software degradation owing to classes and method growth using code rejuvenation techniques.

Ellison *et al* opined that a system is considered to be a survivable system, when it is capable of exhibiting the following attributes and key properties.

- Capability to maintain essential properties.
- Capability to deliver essential services must be sustained even if a significant portion of the system is incapacitated.
- Capability to identify essential services, and the essential properties that support them, within an operational system.
- Capability to fulfill its mission in a timely manner.

And to maintain their capabilities to deliver essential services [4] stated the following four key properties of a survivable system:

(i) Resistance to attack.

(ii) Recognition of attacks and the extent of damage.

(iii) Recovery of full and essential services after attack and

(iv) Adaptation and evolution to reduce effectiveness of future attacks [4].

In our present information world, individuals and Organizations depend on software systems either directly or indirectly in all facets of their daily lives and operations. Software systems are either directly used to solve human complex problems or embedded into gadgets that modern users carry. Therefore, a new survivability strategy for software systems is needed to make software systems reliable and dependable especially in a safety-critical environment. A number of research efforts have been carried out to prevent software failure. These include topics like object-oriented approach to software design in distributed computing systems and method to measure the survivability of object-oriented software in design phase. To the best of our knowledge however, no research has been reported on survivability architecture for Object-Oriented Software systems. Hence, the need for an architectural framework proposed in this paper. Common causes of object oriented software failure apart from other external factors mentioned in this paper include continuous growth in methods and classes, corrupted data, undetected missing method calls [9], and software degradation. Our proposed survivability architecture for object-oriented software systems design is targeted at solving the problem of software degradation occasioned by increased growth in classes and methods. This is achieved through threshold scheme and software rejuvenation technique. The remaining part of this paper is organized as follows; Section II discusses reviews of related literature. Section III discusses the proposed survivability architecture, Section IV discusses results and implementation while Section V concludes the work.

II. Related Work

A. A number of related research works reviewed for the purpose of this work include the followings;
Ellison *et al* described survivability as an approach to ensure that a system that must operate in an unbounded network is robust in the presence of attack and will survive attacks that result in network intrusions. They included as part of their discussions of survivability concepts such as integrated engineering framework, survivability practice, the specification of survivability requirements, and strategies for achieving survivability, but did not discuss anything about survivability of object oriented system. [15] introduced a survivability framework for distributed systems through the use of virtualization technology and software rejuvenation methodology. They presented a recovery model and evaluated the steady-state system availability and survivability based on Markovian analysis through SHAPE tools. However, their software rejuvenation effort does not prevent removal of faults but rather prevented faults from being responsible for the whole system failure. [6]

introduced an entropy-based approach for assessing object-oriented software maintainability and degradation. They reported that Object oriented software degradation may be assessed by measuring the increase in the number of "links", or coupling, within an abstraction model and between abstraction models of the software. They also found that software degradation may also be measured using cyclomatic complexity, since it has been shown to be highly correlated with fault proneness of OO classes. They took the approach of defining software decay in terms of Shannon entropy and McCabecyclomatic complexity using industry-established complexity threshold criteria. [7] presented a method to measure the survivability of object-oriented software in design phase. They characterized Object oriented software components Petri net which combines the features of State chart and Object Diagram. A fuzzy number was introduced to this net to represent uncertain elements that might affect the survivability. Survival Possibility Theory was then used to produce survivability measure function for each component. A survivability measure index is defined for the system. They could only prove that this index is monotonic. [8] proposed a framework and analyzed existing software update systems with their framework. They examined the ability of the framework to communicate information securely in the event of a key compromise to be weak or non-existent. They also identified core security principles that allow software update systems to survive key compromise. Using these ideas, they designed and implemented a software update framework that increases resilience to key compromise. [13] explored the quality of design of software components using object oriented paradigm. They used a number of object oriented metrics proposed in the literature for measuring the design attributes such as inheritance, coupling, polymorphism etc. These metrics were used to analyze various features of software component. They however, did not relate it to survivability of object oriented systems. [10] proposed a survivability evaluation model and analysis performance of Wireless Sensor Networks. They presented the model by representing the states of Wireless Sensor Networks under attack. The survivability of WSN is expressed as a continuous time Markov Chain to describe the status of real WSN's in the face of Dos attack. They proposed a threshold condition to trigger the transition between the states of the Wireless Sensor Network (WSN). They further presented a survivability model and evaluation of WSN under key compromise. Their study however did not include key revocation scheme, single node software rejuvenation and reconfiguration scheme for mission critical operations, based on self- healing concept.

B. Entropy-based Measure of OO Software Degradation

Hector *et al* reported that measure of Object Oriented software shows that software systems could be degraded overtime. Industrial-based risk complexity threshold criteria called McCabeCyclomatic Complexity (CC) indicate that a class comprises simple methods (e.g. simple class constructors, simple class destructors, and other simple procedures)[6]. The behaviour of methods, classes, constructors, or destructors as related to the degradation of software were defined. Similarly, NASA SATC (Software Assurance Technology Center) WMC (Weighted Methods per Class) risk thresholds were also stated [12]. The application of these thresholds in industry has gone a long way in helping software developers and managers to make good judgments about the condition of their software, and has also given a good insight into developing our own survivability model as explained in the following section [6].

C. Survivability of Software Systems

[2] proposed evaluation of six rejuvenation strategies categorized in terms of granularity. Their results show that the overhead impact of the rejuvenation techniques is related to their granularity. They reported that techniques such as application-level rejuvenation strategies are better as a first tentative approach to mitigate the software aging effects. They claimed that if the first strategy fails, then a rejuvenation technique from the next higher level of granularity could be used. They reported a very important result related to the virtualization technology which says that virtualization was the major reason for high increase in memory fragmentation, which commonly results in aging-based failures in old software systems. This assertion is important in modern day technology in which virtualization is a core technology of the cloud-computing. [2] also presented guidelines for the use of the appropriate scenario for each rejuvenation techniques. In [5] a set of extended metrics were proposed to account for information gained from a user's point of view regarding the intensity of the observed failures. To estimate the software reliability through testing, an extended adaptive testing strategy, namely Modified Adaptive Testing (MAT) was proposed. The use of test history information allows the resulting test process to be adaptive in the selection of tests under limited test budget. Simulations and experiments on real-life programs were conducted to evaluate the effectiveness of MAT. Results showed that the reliability estimates obtained using MAT are closer to the real reliability than those obtained using random testing and lead to lower variance than the techniques used for comparison, which means MAT can be applied to help testers and reliability engineers better understand the reliability of their programs. They concluded that the proposed approach can enhance the software reliability estimation testing by guiding the test case selection process by providing more descriptive and accurate results. [16] defined a new multidimensional measure of OSS (Open Source Software) project survivability and followed the traditional method of new measure evaluation to validate the new measure. They defined Project viability in terms of vigor, resilience, and organization in mathematical and natural language. They also formulated VI (Viability Index) to capture the dimensions of project viability and provide a tangible measure of survivability. Their empirical validation confirms that project viability is consistent with reality and that vigor, resilience, and organization are three different attributes of the project viability, each contributing to the survivability of OSS projects over their lifecycle. In [14], a language called Stitch for representing repair strategies within the context of an architecture-based self-adaptation framework was presented. Stitch supports a clear representation of repair decision trees together with the ability to state objectives and allow a self-adaptive system to select a strategy that has optimal utility in a given context, even in the presence of potential timing delays and outcome uncertainty. [11] described the survivability of the network in recent years. The work

highlighted technical measures for improving survivability in open environment, in which single functional mechanism (safety or reliability) is not sufficient to ensure a system. They opined that it is necessary to introduce research ideas on collaborative mechanism of survivability that will improve information sharing and self-repair ability. They believed that by optimizing the survivability mechanisms, it can improve the reliability and survivability of the whole network service system. They also claimed that the research advancements on network system's survivability is relatively slow when compared with growth of network .

III. Proposed Survivability Architecture

A. The survivability architecture proposed in this paper is based on software rejuvenation technique as depicted in Figure 1.

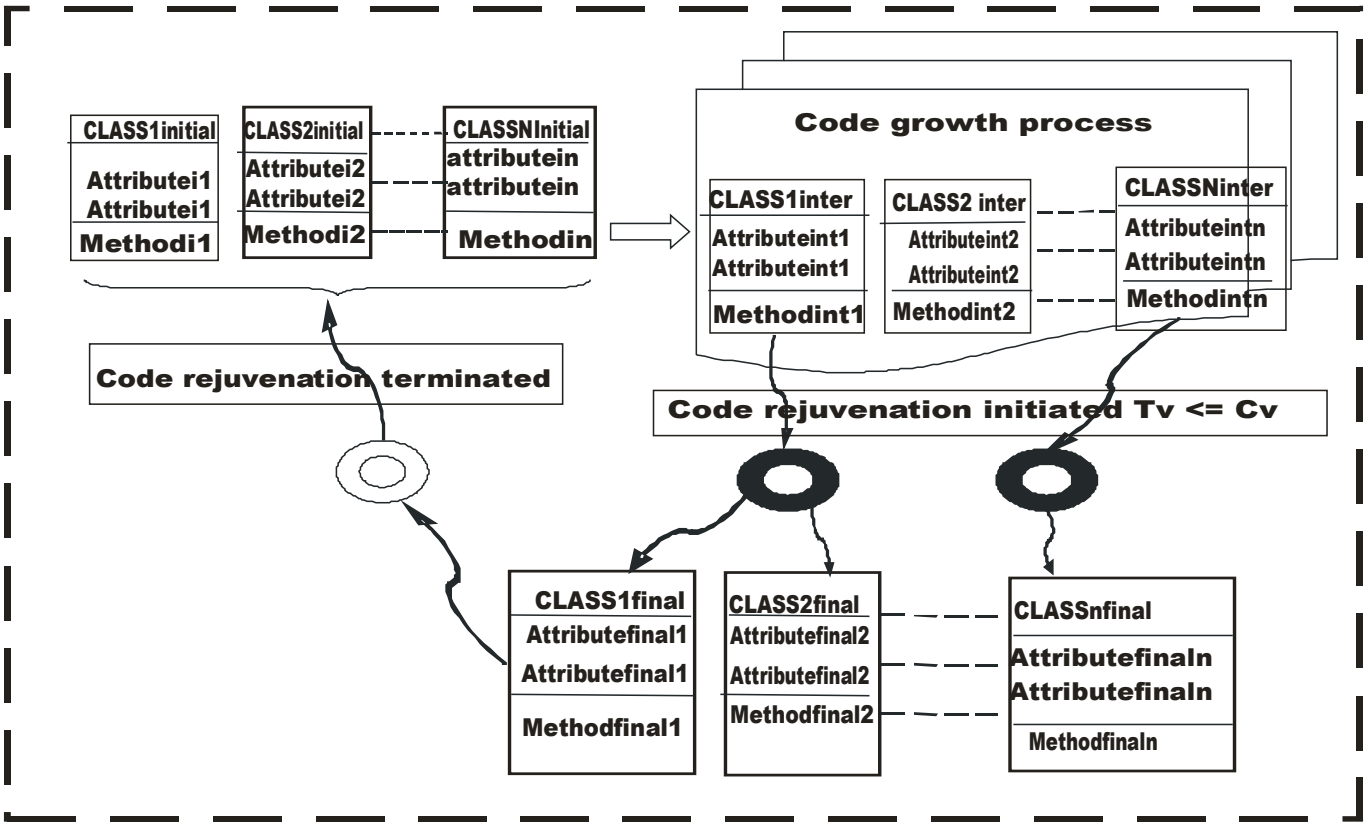

Figure 1: Proposed Survivability Architecture

In the proposed survivability architecture, an object oriented software system is visualized as a system composed of internal structures like classes, attributes and methods (operations). These classes are represented as CLASS1 to CLASSN for every class state. These are represented as CLASS1initial, CLASS2initial, and CLASSNinitial (for classes in initial state).CLASS1inter, CLASS2inter, and CLASSNinter (for classes in intermeadiate state) and CLASS1final, CLASS2final, and CLASSnfinal (for classes in final state). The initial state of classes refers to the state a software system is rendering the services for which it is developed without any indication of degradation. The intermediate state is captured in the architectural diagram as the state when the internal structures of the software system like classes and methods start to show indications of growth (getting degraded). In this design, a threshold value (T_V) is set to the final degradation point for the software system using McCabeCC threshold metrics idea with regards to the behaviour of methods and classes. The initial degradation point (T_v) is the point when the growth of classes and methods constituting a software system begins. The final degradation point (T_v) is the point when the value of method growth is just next to the critical region. The initial and final degradation value (T_v) using average transition probabilities. A critical region is the range of values for method growth that is above the final degradation point. The critical region values denoted by (C_V) is also determined using the probability density function theory to determine the probability of threshold value T_V falling within the region or not. A mechanism is integrated into the architecture to monitor and determine the continuous

growth of methods and classes such that when threshold value T_V <= critical value C_V set within the critical region, the methods and classes that constitute the software system are re-initiated in self-healing process. Code rejuvenation is initiated at this time when the classes state change from the code growth process to the final state. Code rejuvenation is terminated when the classes states move from the final state to the initial state.

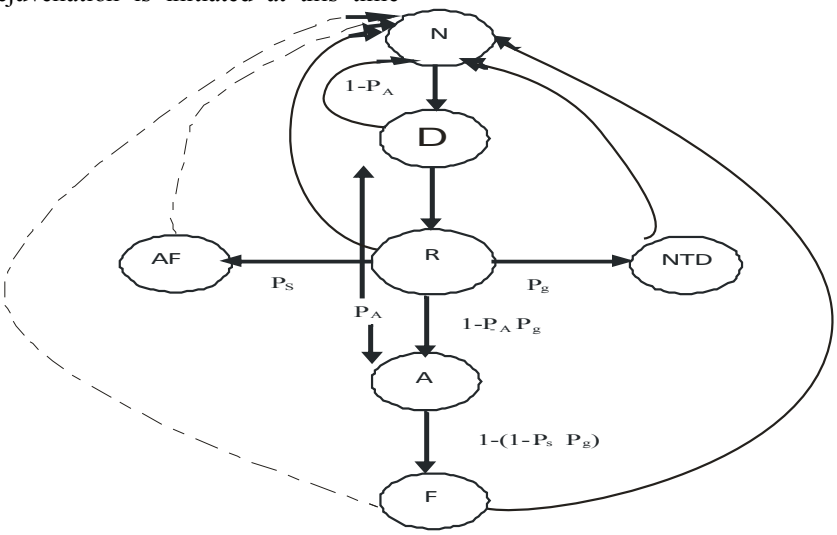

Figure 2: Transition State Model for the Proposed Architecture

B. Software Rejuvenation Transition State Model

Figure 2 is the rejuvenation transition state model that further describes the rejuvenation technique in the proposed architecture. In our proposed software system survivability technique, the software system is said to be in **Normal state N** when all the classes and methods in the system are rendering their services and have not indicated any sign of growth. The system is in **Degradation state (D)** when the classes and methods start to grow i.e **N-D**. The system enters the **Attack A state** when the attack or failure is not noticed or properly checked at the **repair (R) state.** The system enters the **Not Too Degraded (NTD)** if repair state is not capable of preventing the attack but is only able to allow the system supporting important services. Similarly, if repair state could not stop the system from failing but is capable of protecting the confidentiality and integrity of the data therein by stopping the system from rendering any service, it is said to be in **Almost Fail (AF) state.** If all failure prevention and recovery from degradation state to Attack fail then the system is said to have entered a **Failed state (F).** The **repair state (R)** is the most important state in the proposed architecture because it is the first state after degradation state where the software system is rejuvenated. It is also the state where the threshold value and critical values are determined and compared with respect to the value of classes and methods. The determination of threshold and critical values are based on the sojourn times in each state with their transition probabilities as described below;

M_n: Mean time for the system to resist entering degradation state D.

M_v: Mean time for the system to resist entering the failure state from degradation state D.

M_A: Mean time taken by the system to detect a failure and initiate the repair or rejuvenation.

P_A: Probability of the software system entering fail state when it is already in a degradation state.

Pg: Probability that the system will resist failure by graceful Not Too Degraded (NTD).

Ps: Probability that the system would respond to Almost Fail manner.

Therefore, the figure above indicate that N-D: $1\text{-}P_A$, N-A:P_A, A-F:$1\text{-}(1\text{-}P_s\text{-}P_g)$, R-A:$1\text{-}P_s\text{-}P_g$, R-AF:$P_s$, and R-NTD:$P_g$. The minimum mean time t_1 and maximum mean time t_2 are determined from the values of M_n, M_v, *and* M_A. The average transition probabilities are also determined. Probability density function is then applied to determine the probability of threshold value T_V falling within a particular region also to decide the critical region for the system. In probability density function a

random variable X has a density *f*, where *f* is non-negative function, if

$$P[t_1 \leq X \leq t_2] = \int_{t1}^{t2} f(x)\,dx \quad \ldots\ldots\ldots\ldots\ldots (1)$$

where T_v is taken as the random variable X and where one can think of $f(x)\,dx$ as being the probability of X falling within the infinitesimal interval $[x, x + dx]$. In the system, software rejuvenation is initiated when the value T_v is found to be just less or equal to any lowest value of measure within the critical region.

C. Availability Model for the Proposed Architecture

Let δ be the time rate for the software system failures and φ be the time rate for software system repairs that are exponentially distributed. Also let x be the time rate for its functional environment failures and y be the time rate for the functional environment repairs be exponentially distributed. The system functional environment failure either recovers fully with coverage probability p or not with coverage probability (1-p). When this happens, the system is said to be functioning in a degraded mode. The assumption that the system and its functional environment failures and repairs are exponentially distributed, the availability model for the proposed architecture is CTMC (Continuous Time Markov Chain) with the transition diagram indicated in Figure 3. In the Figure, *i= {1,2,3,..,n}* represents the number of classes in the software system that are still providing normal service while state $State_i$ corresponds to the state of functional environment failure in the system with *i* normal classes. The software system continues to render its services when all the classes are able to either call or being called. Therefore, states $i \in \{1,2,3,\ldots\ldots, n\}$ are normal states. If otherwise, it fails. Then states $State_i$ $i \in \{1,2,3,\ldots\ldots,n\}$ are failed states. The steady-state availability of the software system represented by P_A which is obtained as the sum of the steady-state of the classes with threshold values less than the critical value set (i.e $T_v <= C_v$), and is defined as;

$$P_A = \sum_{k=1}^{n} \pi k \quad \ldots\ldots\ldots\ldots\ldots (2)$$

where πk is the steady-state probability of CTMC being in state k and implemented in the system code.

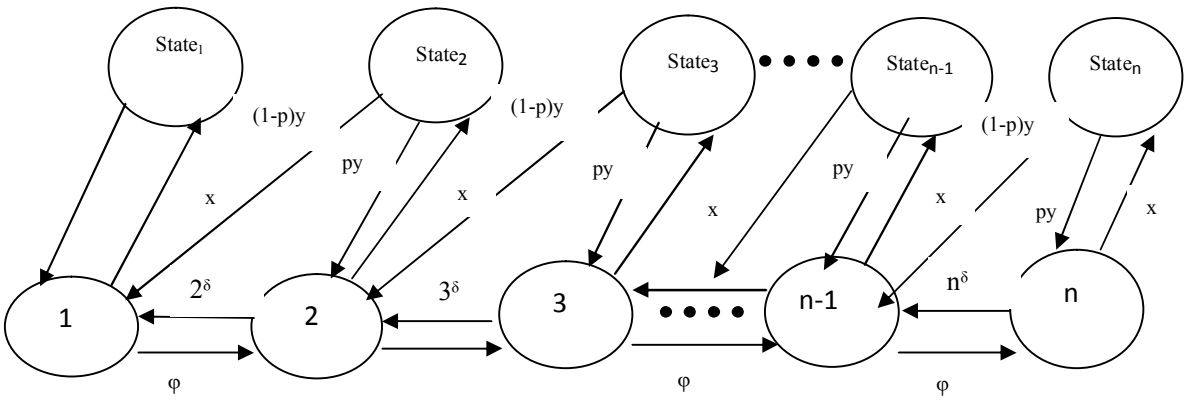

Figure 3: CTMC-based Model for the Proposed Architecture

IV. Implementation and Results

Two programs are written to implement treemap algorithms in Java for testing the efficiency of the proposed architecture. The two treemap programs are made to have the same initial number of methods and classes. One of the treemap programs (called the standard program) is allowed to run and grow the classes and methods in it without incorporating the proposed mode elements (threshold value and code rejuvenation condition) into it. The second treemap program (called the rejuvenated–enriched program) has the proposed architecture like threshold value and code rejuvenation technique incorporated into it. The two software programs are tested on the same computer with same processor speed and RAM capacity for 7 days. The internal structures (attributes) are observed depicted in Tables I and II.

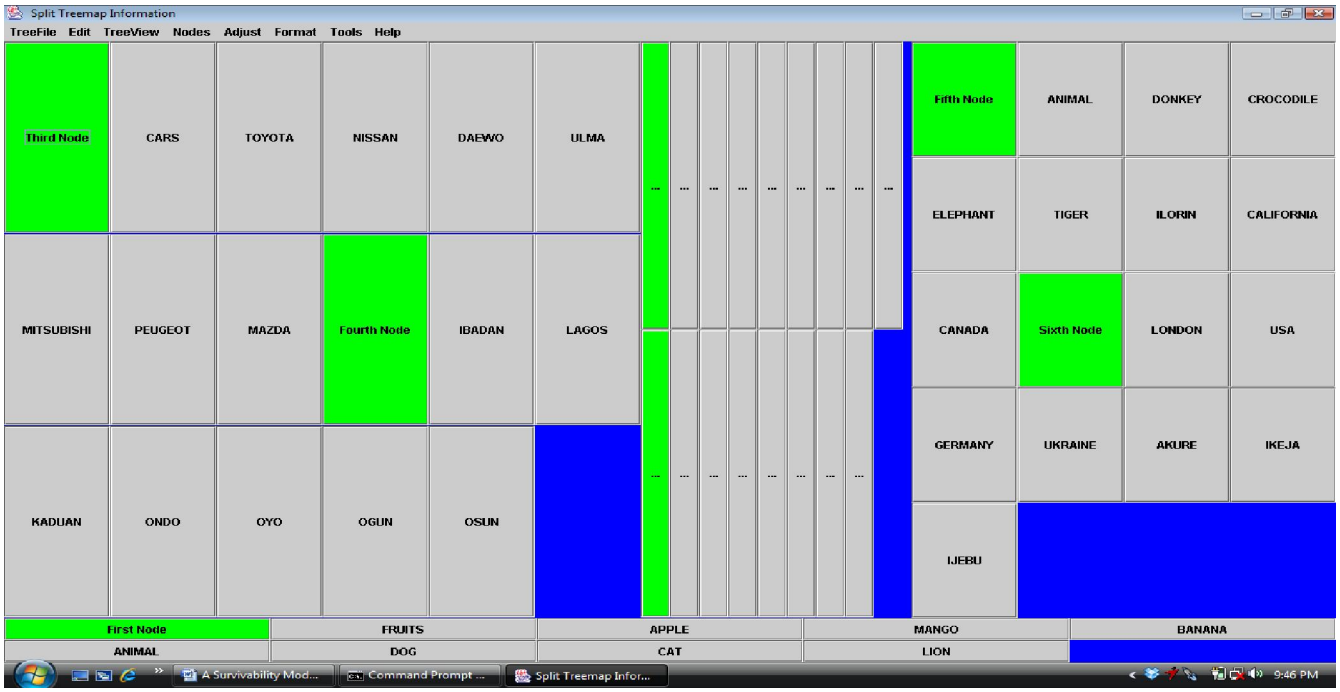

Figure 4: Snapshot for rejuvenated–enriched program

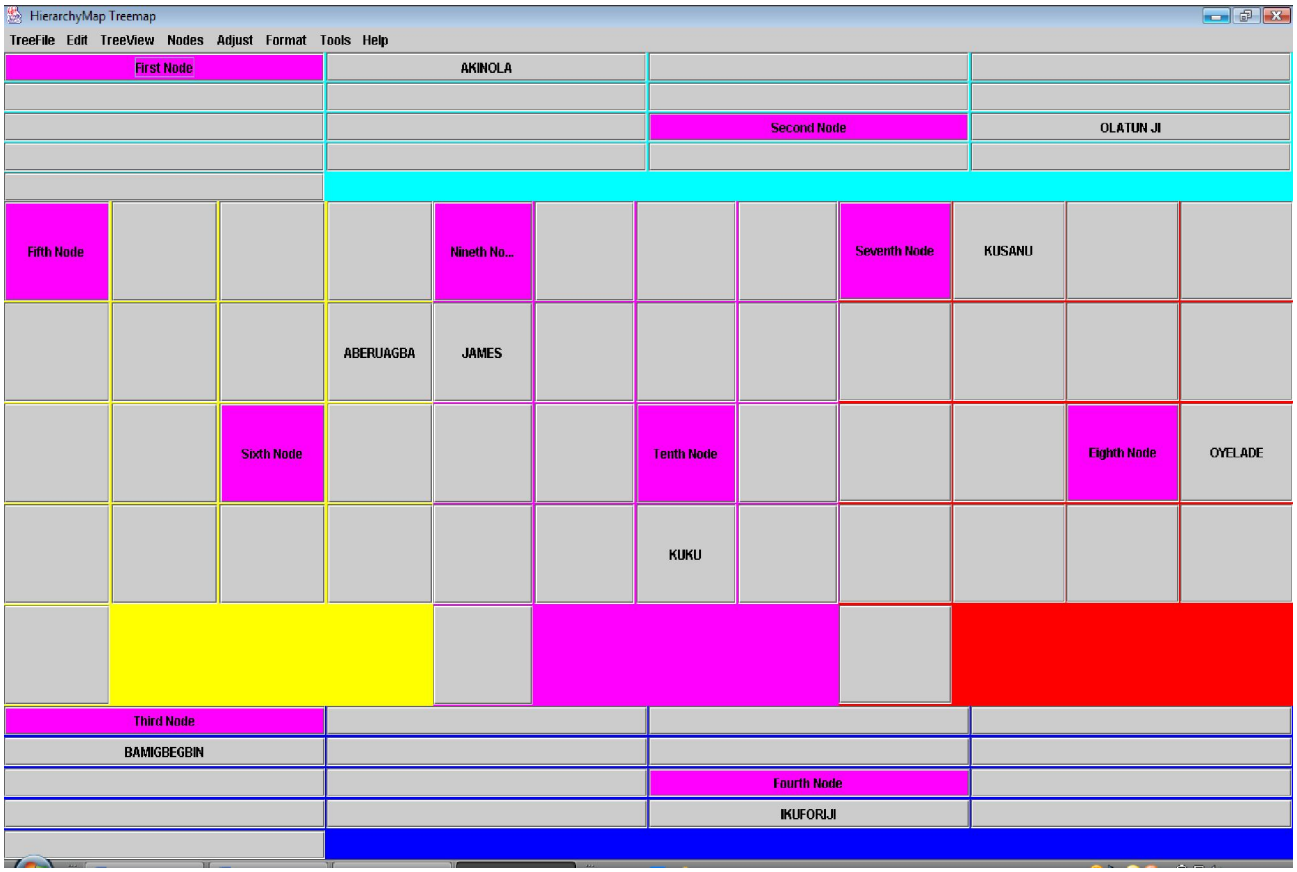

Figure 5: Snapshot for standard program

Table I: Program Attribute values at the beginning of run time

Attributes	Standard Program	Rejuvenated-Enriched Program
Blank Lines	10	10
Code Lines	1200	1200
Comment Lines	20	20
Executable statements	800	800
Declarative Statements	20	20
Number of Classes	50	50
Number of Methods	250	250
Error reported	75	25
Number of corrections made	20	20

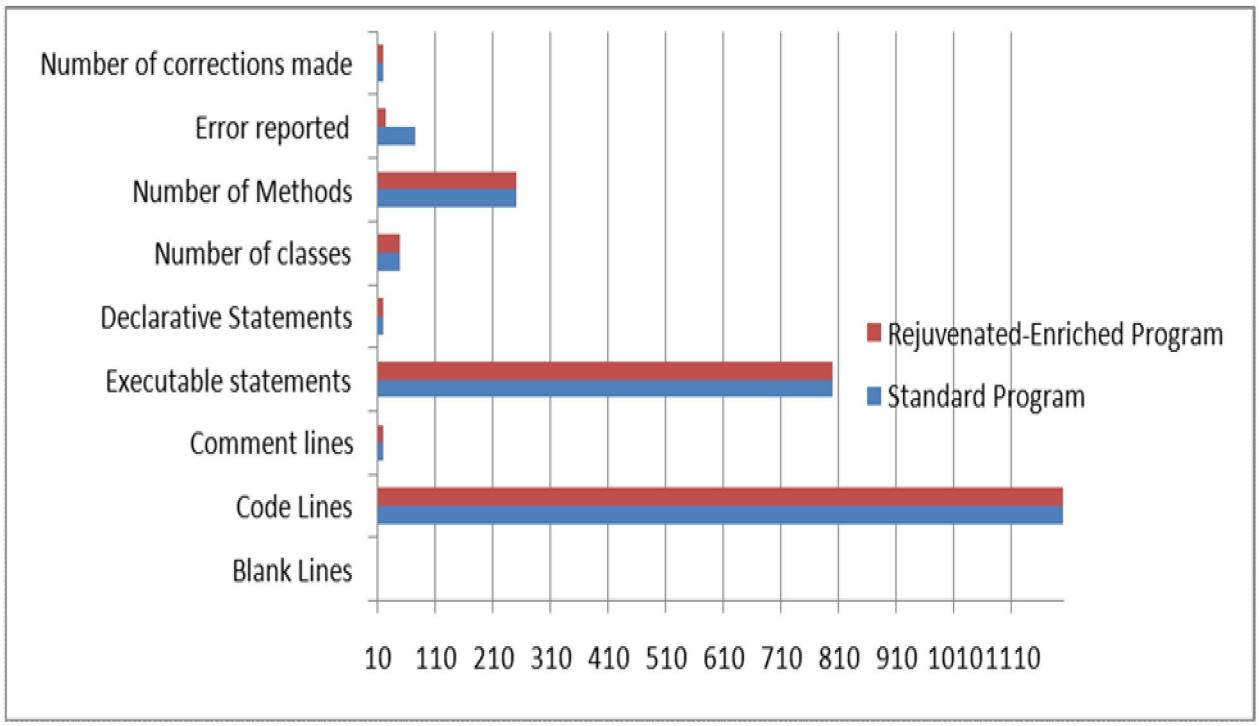

Figure 6: Analysis for the behaviour of Standard Program and Rejuvenated–Enriched program at the beginning of run time.

Table II: Program Attribute values at the end of a Week (7 days) of run time.

Attributes	Standard Program	Rejuvenated-Enriched Program

Blank Lines	10	10
Code Lines	1200	1200
Comment Lines	20	20
Executable statements	800	800
Declarative Statements	20	20
Number of Classes	50	50
Number of Methods	2500	250
Error reported	55	25
Number of corrections made	200	20

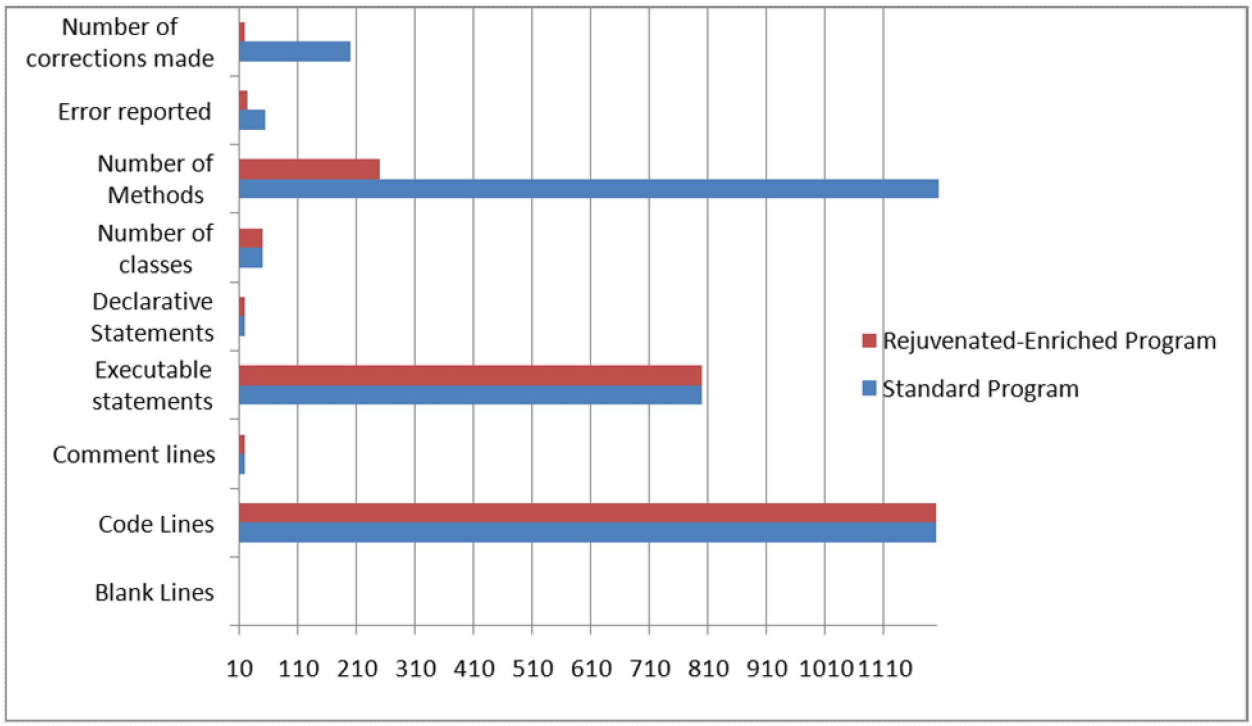

Figure 7: Analysis for the behaviour of Standard Program and Rejuvenated Program at the end of 7 days run time.

The analysis in Figure 6 and 7 show that incorporation of threshold and rejuvenation technique in software system development can help ensure software survivability, thereby preventing software system failure.

V. Conclusion and Future Work

This paper presented a survivability architecture for object-oriented software system that is targeted at solving the problem of software degradation and failure usually occasioned by increased growth in classes and methods. The model is based on the idea of code rejuvenation technique that is triggered by the attainment of a threshold value. It is tested on two programs developed to implement treemap algorithms and found to be effective. Our future research effort in this area would be geared towards implementing this model on a more standard object oriented software systems. It is also part of our future research endeavour to investigate and propose survivability for general software system.

References

[1] S. Ali and F. Robert .:"A Systematic Review of Software Robustness", *Journal of Information and Software Technology*. Vol 55, Issue 1. Elsevier . Page 1–17. (2013).

[2] J. Alonso, R. Matias, E. Vicente, A. Maria and K. S. Trivedi."A Comparative Experimental study of Software Rejuvenation Overhead", *An International*

Journal of Performance Evaluation Vol. 70 Elsevier B.V. Page 231-250. © 2013.
[3] A. K. Charles, A. K Kevin, and S. P. Joon." Surviving in Cyberspace: A Game Theoretic Approach", *Journal of Communications, Vol. 7, No. 6, June, 2012.* Academy Publisher. Pages 436-450. © 2012.
[4] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Lonstaff, N. R. Mead. "Survivability: Protecting Your Critical Systems". Published by *Internet Computing, IEEE. Vol 3, Issue 6.* Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213-3890. Pages 55-63.(1999).
[5] H. Hai, J. Chang-Hai, C. W. Kai-Yuan, W. Eric, P. M. Aditya." Enhancing Software Reliability Estimates using Modified Adaptive Testing". *Journal of Information and Software Technology Vol 55*. Elsevier . Pages 288-300. © 2012.
[6] M. O. Hector, H. E. Letha, and C. Glenn. "An entropy-based approach to assessing Object-oriented software maintainability and degradation − A method and case study," In *Proceedings of Software Engineering Research and Practice*, 2006, Pages 442-452. (2006).
[7] H. Jueliang, D. Zuohua, L. Jing, Y. Ling. "Measuring the Survivability of Object-Oriented Software". TASE '09 *Proceedings of the 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering.* Pages 329-330. IEEE Computer Society Washington, DC, USA ©2009.
[8] S. Justin, M. Nick, and C. Justin."Survivable Key Compromise in Software Update Systems", *In Proceedings of the 17th ACM conference on Computer and Communications Security.* Pages 61-72. 2010.
[9] M. Martin, B. Marcel, and M. Mira. "Detecting Missing Method Calls in Object-Oriented Software". *Proceedings of the 24th European Conference on Object Oriented Programming (ECOOP'2010).* Pages2-25. Springer-Verlag Berlin, Heidelberg ©2010.

[10] S. Parvin, K. H. Farookh, S. P. Jong, and S. K. Dong. "A survivability model in Wireless Sensor Network". *An International Journal of Computer and Mathematics with Application (2012)*, Elsevier Ltd. *Vol 64.* December, 2012. Pages 3666–3682. (2012).
[11] W. Qingliang, Z. LiFang, J. Zheng-Tao, H. Yunbing. "Progress and Research of Network System Survivability Scheme with Cooperative Information Management". *Journal of Networks, Vol. 7, No. 10, October, 2012.* Page 1609-1615. (2012).
[12] L. H. Rosenberg. "Applying and Interpreting Object Oriented Metrics," *In the Proceedings of Software Technology Conference, Utah, April, 1998*.
[13] J. S. V. S. Sastry, K. V. Ramesh, and M. Padmaja. **"**Measuring Object-Oriented Systems Based on the Experimental Analysis of the Complexity Metrics".*International Journal of Engineering Science and Technology (IJEST).* ISSN : 0975-5462 Vol. 3 No. 5 May 2011. Pages 3726-3731. (2011).
[14] C. Shang-Wen and G. David. "Stitch: A language for Architecture-Based Self-Adaptation". *The Journal of Systems and Software Vol. 85 (2012).* © 2012 Elsevier Inc. Page 2860– 2875. (2012).
[15] T. Thandar, P. Manish, C. Sung-Do, and S. P. Jong. "A recovery Model for Survivable Distributed Systems through the use of Virtualization". Fourth *International Conference on Neworked Computing and Advanced Information Management. Vol. 1,* pp.79-84. IEEE Computer Society (2008).
[16] R. Uzma and J. T. Marietta"Defining and Evaluating a Measure of Open Source Project Survivability". *IEEE Transactions on Software Engineering, Vol. 38, No. 1.* Published by the IEEE Computer Society. Pages 163-174. (2012).
[17] Z. Yanjum (2010). "Survivable RFID Systems: Issues, Challenges and Techniques". *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Review. Volume 40, No 4,* July, 2010.Page(s):406 - 418.2010.

Author Biographies:

Aborisade .D O. Olaniyi (Bsc, Msc) is a PhD student in the Department of Computer Science, Federal University of Agriculture Abeokuta, Nigeria. He has research interest in Cloud Database Security, Digital Forensics and Human Computer Interaction (HCI).
Sodiya A.S. (Bsc, Msc, PhD) is presently a Reader in the Department of Computer Science, Federal University of Agriculture, Abeokuta, Nigeria. His research interests include Information Security, Artificial Intelligence and Software Engineering. He has published many papers in both Local and International Journals.
Ikuomola A. J. (Bsc, Msc, PhD) is presently a Lecturer in the Department of Computer Science, Federal University of Agriculture Abeokuta, Nigeria. Her research interest include Information and Network Security, Cloud Computing, Artificial Intelligence and HCI. She has published in both Local, International Journals and refereed Conferences.