Greener Journal of Science, Engineering and Technological Research
ISSN: 2276-7835
Vol. 15(1), pp. 1-9, 2026

https://gjournals.org/GJSETR
DOI: https://doi.org/10.15580/gjsetr.2026.1.062626099

Autonomic Self-Healing System - A Review

Ugwuanyi, Fidelis Onyebu (Ph.D)

Department of computer Science, Federal University of Education, Pankshin, Plateau State.

ABSTRACT

The increasing complexity of computer systems results in systems which are prone to errors and disruptions, creating major problems for the user. In order to cope with the rapidly growing complexity of integrating and managing today's computer-based systems, autonomic computing was introduced. Autonomic computing is a computer environment that can detect and adjust its system automatically to manage itself (self- heal) without the assistance of any human interaction. In this paper autonomic computing system was introduced, the properties and architecture, what self- healing means, why self-healing, fault model for self- healing and then the future of self- healing.

ARTICLE'S INFO

Article No.: *062626099*
Type: *Research*

Full Text: *PDF, PHP, HTML, EPUB, MP3*
DOI: *10.15580/gjsetr.2026.1.062626099*

Accepted: *26/06/2026*
Published: *29/06/2026*

Corresponding Author

Ugwuanyi, Fidelis Onyebu, Ph D

E-mail: *fidelisugwuanyi@fuep.edu ng, fugwuanyi2005@gmail.com*

Phone No: *+2348067730730, +2348050577220*

Keywords: *Autonomic, Computing, Self-Healing, System*

1.0 INTRODUCTION

Self–Healing System is an autonomic computing model named after, and patterned on the body's autonomic nervous system. An autonomic computing system would control the functioning of computer applications and systems without input from the user, in the same way that the autonomic nervous system regulates body system without conscious input from the individual (Strassner and Kephart, 2006)

A self- healing system is one that has the ability to discover, diagnose, and repair (or at least mitigate) disruptions to the services that it delivers. For large scale systems, many different types of faults may exist and their differing natures often require disparate, tailored approaches to deter, let alone fix them (Jiang, Zhang Raymer and Strassner, 2007). Hence, for large scale systems, a self- healing system should also be able to use multiple types of detection, diagnosis and repair mechanisms.

Autonomic systems extend the above notion of self – healing to include capabilities to adapt to changes in the environment, for example to maintain its performance, or availability of resources (Jiang etal 2007).

Autonomic computing can extend and adjust its system automatically to heal itself without the assistance of any human interaction. Figure 1 below displays the typical procedures implanted in the various IT organizations. The IT industry need a system that would force the users need and allow users to focus more on completing their work task and less on troubleshooting their computer system. According to (Laster, and Olatunji, 2007), autonomic computing is conceived to lessen the spiraling demands for skilled IT resources, reduce complexity, and to drive computing into a new era that may better exploit its potential to support higher order thinking and decision making. Implementing an autonomic system according to (Laster and Olatunji, 2007) will help companies eliminate the increasing costs of restoring hardware, and software failures. Consequently, autonomic computing will effectively prevent downtimes and system failures. In addition to less downtimes and system failures, the production rate for computer environments controlled by autonomic system will increase dramatically.

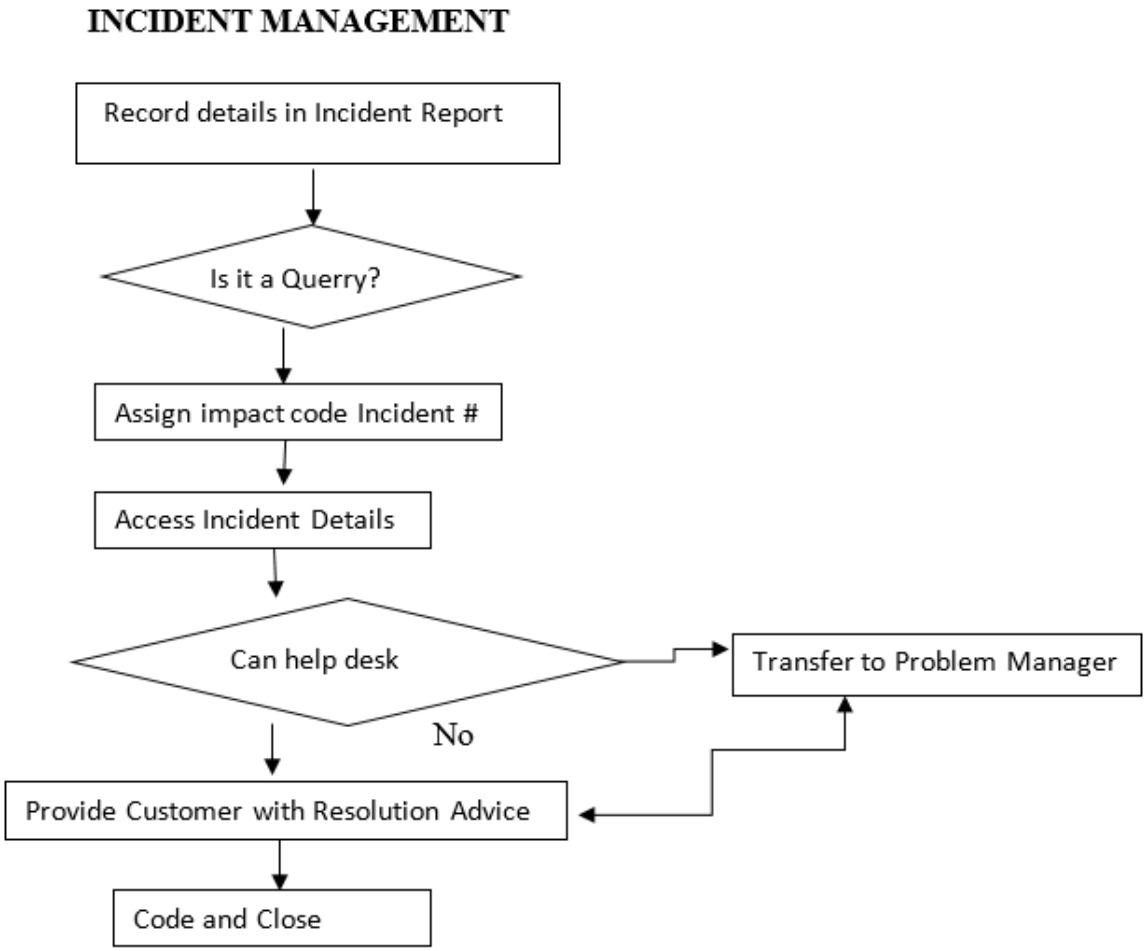

Fig 1A

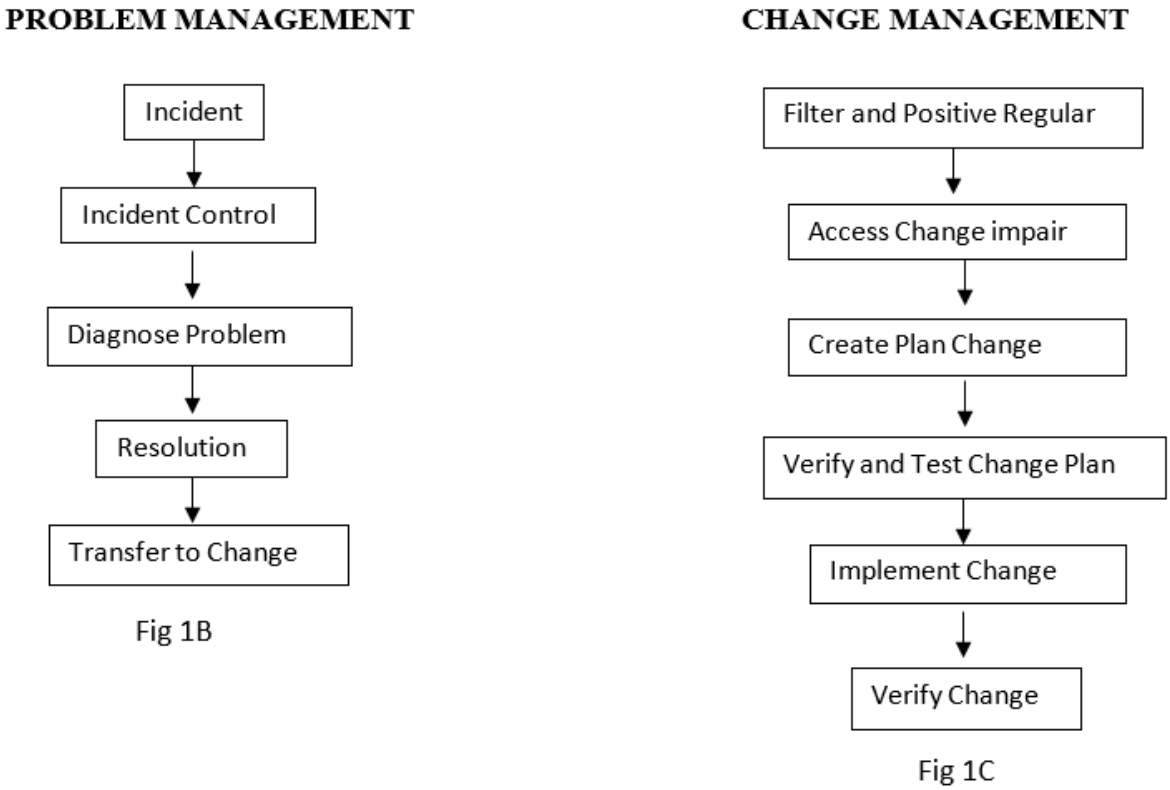

Figure 1A, 1B and 1C: Typical IT process of solving a problem.
Source: IBM white paper (2003)

1.2 Related Literature

Self-healing is an essential component of every computing system. It is an integral part of devices running in autonomic pervasive environment. It is widely researched topic in the field of distributed computing, grid computing, and autonomic computing. In each of these areas many schemes are proposed that attack this problem from various stand point. Researchers have worked on several policies like architecture-based system (Garlan and Schmert, 2002; Dashofy, Hoek, and Taylor 2002), infrastructure-based approach (Appavoo, Hui, Wisniewski, Sylva, Krieger, and Souls 2002) for long.

2.0 METHODOLOGY

Self-healing framework leverages a diverse set of methodologies to autonomously detect and recover from faults. This section is organized as follows: section 2.1is on the autonomic computing characteristics, while section 2.2 is on properties of autonomic computing.

2.1 Autonomic Computing Characteristics

The IBM white paper (2003) listed the following as the eight key characteristics of an autonomic computing.

- To be autonomic, a computing system needs to "know itself"- and comprise components that also possess a system identity.
- An autonomic computing system must configure and reconfigure itself under varying and unpredictable conditions.
- An autonomic computing system never settles for the status quo- it always looks for ways to optimize its workings.
- An autonomic computing system must perform something akin to healing. It must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction.
- A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection.
- An autonomic computing system cannot exist in a thematic (protected from outside influence) environment.
- Perhaps most critical for the user, an autonomic computing system needed will anticipate the optimized resources needed while keeping its complexing hidden.

2.2 Properties of Autonomic Computing

In addition to possessing the eight key characteristics (Paul, 2011) is of the opinion that an autonomic self-healing system must possess at heal one of the four fundamentals' elements of the following self- managing properties.

- Self- healing: System discover, diagnoses and reacts to disruptions (Paul, 2001). For a system to be self- healing, it must be able to recover from a failed component by first detecting and isolating the failed components taking it off line, fixing or isolating the failed component and reintroducing the fixed or replacement component into service without any apparent application disruption (Laster and Olatunji, 2007). Systems will need to predict problems and take actions to prevent the failure from having an impact on applications. The self-healing objectives must be able to keep enterprise applications up and available at all times. Developers of system components according to (Garnek and Corbi, 2003) need to focus on maximizing the reliability and availability design of each hardware and software product toward continuous availability.
- Self-optimizing: Systems monitor and tune resources (Paul, 2001), the concept of self-optimization is when a computer environment manages resource allocations and workloads to meet the end user needs without any or minimal human interference (Laster and Olatunji, 2007).
- Self- Configuring: Systems adapt automatically to dynamically changing environments (Paul, 2001). The idea of configuration is for of hardware and software systems to have the ability to define themselves at any given moment. Having the desired capabilities, allow new components to be dynamically added without any or minimal human interference (Laster and Olatunji,2007)
- Self- Protecting: Systems anticipate, detect identify, and protect themselves from attacks from anywhere (Paul, 2001). The ability to detect unauthorized access, eliminates intrusions, reports such activities for each occurrence, and secure backup capabilities as the original source manager system (Laster and Olatunji, 2007).

3.0 DESIGN OF THE SELF-HEALING

3.1 Autonomic Computing Architecture

According to (Kephart and Chess, 2003), the building block of an autonomic computing system is called an autonomic element. An autonomic element is an individual system constituent that contains resources and delivers services of human and other autonomic elements. Autonomic elements learn from past experiences by managing their internal behavior and their relationship with other autonomic elements in accordance with guidelines that humans or other elements have established to manufacture and execute action plans. The general structure of an autonomic element is depicted in the figure 2 below. The formation of an autonomic elements consists of an autonomic manager and managed elements (Laster and Olatunji,2007). According to architectural blueprint for autonomic computing, a managed element is what the autonomic manager is controlling and an autonomic manager is components that implements a particular control loop.

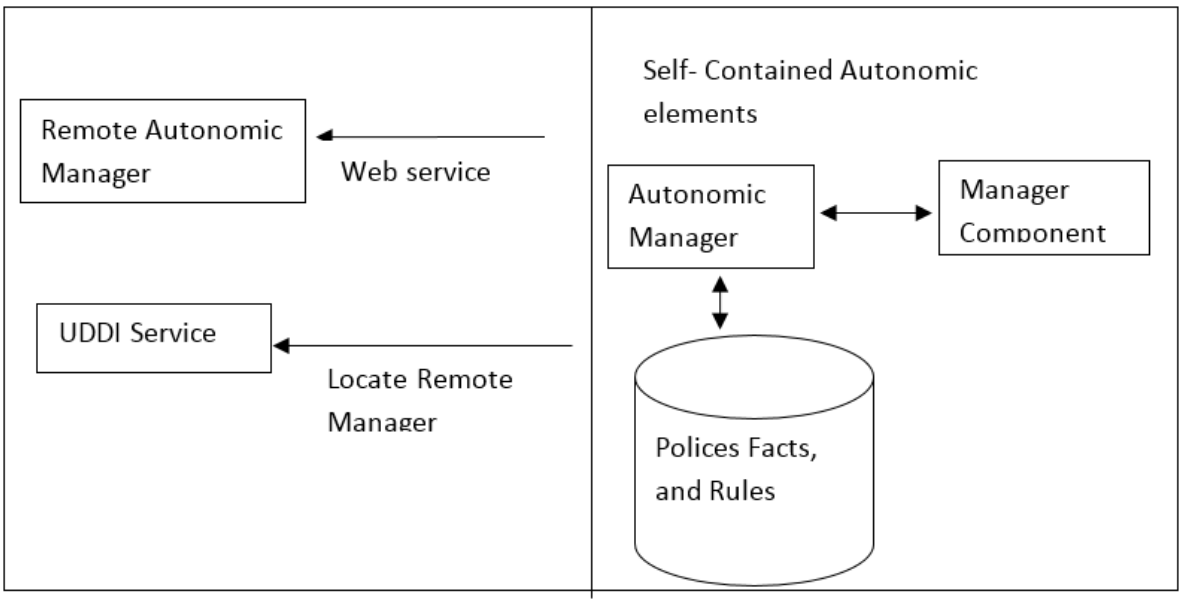

Fig 2: Architecture of an autonomic element

Source: Bantz, Bisdikian, Challener, Karidis, Mastrianni, Mohindra,Sheaand Vanover (2003).

3.2 Autonomic Manager

An autonomic Manger is a component that implements the control loop (Laster and Olatunji,2007). The control loops can be executed through numerous permutations of management tools and products, or distributed by a resource provider in the four sections that share the knowledge (IBM,2003). They are as follows:

- The monitor part provides the mechanism that collects, aggregate, filter, manager, and report details (metrics and topologies) collected from an element (IBM, 2003).
- The analyze part provides the mechanism to correlate and model complex situations (time series forecasting and queuing models, for example). These mechanisms allow the autonomic manager to learn more about the IT environment and help predict future situations (IBM, 2003).
- The plan part provides the mechanism to structure the action needed to achieve goals and objectives. The planning, mechanism uses policy information to guide its work (IBM, 2003).
- The execute part provides the mechanism that controls the execution of a plan with considerations for on- the- fly updates (IBM, 2003).

3.3 Managed Elements

A controlled system component is known as the managed element where they can be either a single resource (server or monitor) or a set of resources (Cluster or pool of servers). The managed element is controlled through it senses and effectors (IBM, 2003).

- The sensors provide mechanisms to collect information about the state transition of an element. To implement the sensors, either use a set or “get” operations to retrieve information about the current state, or a set of management events (Unsolicited, asynchronous messages or notifications) that flow when the state of the element changes in a significant way (IBM, 2003).
- The effectors are mechanism that change the state (configuration) of an element. In other words, the effectors are a collection of “set” commands or application programming interfaces, (APIs) that change the configuration of the managed resources in some important way.

In order for autonomic computing environments to communicate, collaborate, and use management tools, the computing environment must organize their control loop into either autonomic manager or managed element. The manageability interface accessible to an autonomic manager is created due to the combination of sensors and effectors.

3.4 Fault Model Characteristics

According to (koopman, 2003), the following are typical fault model characteristics that seem relevant.

- Fault duration: Faults can be permanent, intermittent (a fault that appear occasionally) or transient (due to an environmental condition that appear only occasionally). Since it is widely believed that transient and intermittent faults out-number permanent faults, it is important to state the fault duration assumption of a self-healing approach to understand what situations it addresses (Koopman 2003; Laster and Olatunji, 2007).

- Fault manifestation: Intuitively, not all faults are as severe as others. Beyond that, components themselves can be designed to exhibit specific characteristics when they encounter faults that can make system-level self-healing computer. A common approach is to design components that are fail-fast, fail-silent. However, other systems must tolerate Byzantine faults which are considered “arbitrary” faults. (It is worth nothing that Byzantine faults exclude systematic software defects that occur in all modes of a system, so the meaning of “arbitrary” is only with respect to an assumption of fault independence) Beyond the severity of the fault manifestation, there is the serenity of how it affects the system in the absence of a self-healing response. Some faults cause immediate system crashes. But many faults cause less catastrophic consequences, such as system slow-down due to excessive (PU loads, thrashing due to memory hierarchy overloads, resource leakage, file system overflow and so on (Koopman, 2003; Laster and Olatunji, 2007).

- Fault Source: Assumptions about the source of fault can affect self-healing strategies. For example, faults can occur due to implementation detects, requirement defects, operational mistakes and so on. Changes in operating environment can cause a previously risking system to stop working, as can the onset of a malicious attack while software is essentially deterministic, there are situations in which it can be assured that a random or “wear-out-“model for failures is useful, suggesting techniques such as periodic rebooting an a self-healing mechanism. Finally, some self-withstand hardware failures such as loss of memory or CPU capacity, and not soft-ware failures (Koopman, 2003; Laster and Olatunji 2007).

- Granularity: The granularity failure is the size of the components that is compromised by that fault. (The related notion of the size of a fault

containment regain is a key design parameter in fault tolerant computers.) A fault can cause the failure of a software module (causing an exception), a task, an entire CPU's computation set, or an entire computing site. Different self-healing mechanisms are probably appropriate depending on the granularity of the failures and hence the granularity of recovery actions (Koopman, 2003; Laster and Olatunji, 2007).

- Fault Profile expectation: Beyond the source of the fault is the profile of fault occurrences that is expected. Faults considered for self-healing might be only expected faults (Such as defined exceptions or historically observed faults). Faults considered likely based on design analysis, or faults, that are un-expected. Additionally, faults might be random and independent, might be correlated in space or time, or might even be intentional due to malicious intents. (Koopman 2003; Lasterand Olatunji, 2007

4.0 DISCUSSION

4.1 What is self- healing?

Self- healing denotes the system ability to examine, find, diagnose, and react to system malfunctions (IBM, 2003). Self – healing components or applications must be able to observe system failures, evaluating constraints imposed by the outside, and to apply appropriate corrections. In order to automatically discover system behavior, autonomic system must have knowledge about own behaviour, then they must have a knowledge in order to determine if the actual behaviour is consistent and expected in relation of the environment. (Laster and Olatunji, 2007).

In new contexts or in different scenarios, new system behaviour can be observed and the knowledge module must evolve with the environment (Davide, 2004)

Self- healing systems basically endures a process in order to maintain satisfactory quality of service of the principal system during routine in the presence of any fault (IBM, 2003). The first cycle is called monitoring cycle. During the monitoring cycle, the systems monitor will inspect the computer environment for any improper conduct. After the monitor's inspections are completed, it will send the data gathered and current observations to the next stage. The second phase of the cycle is called error dictation and diagnosis; if the diagnosis reports that there is no fault in the system, then it will loop back to the monitor for more observations. If there is an error detected by the monitor, the error detection cycle will report it to the next stage of the cycle. The third stage, of the cycle is known as analysis and selection of a repair operation. At this stage, the fault is analyzed and a method of repairing is determined at this part of the cycle. After the report is passed into the final phase of the cycle called execute repair and operation (self-repair). Any repairs that are needed are completed at this phase in the cycle. Once, the faulty areas are self-repaired; the cycle is a closed loop, the process of self-healing environment is a continuous as depicted in the figure below (Laster and Olatunji, 2007).

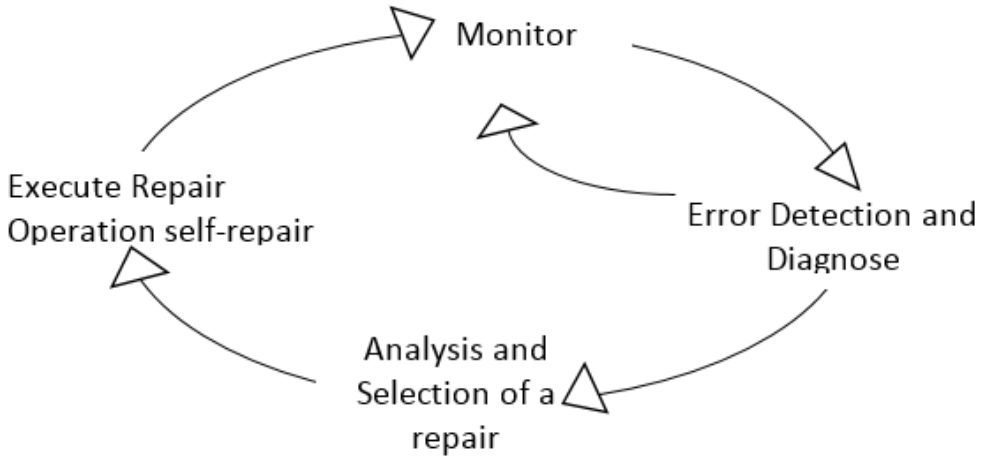

Fig. 3: Self-healing system Process
Source: IBM: 2003

Self-healing environments have comparable objectives to the common area of dependable computer environments (Laster and Olatunji, 2007). One of the fundamental tenets of dependable computing is that a fault hypothesis (often called "fault model) must be specified for any fault tolerant system (koopman, 2003). The fault hypothesis answers the question of what faults the system is to tolerate. Similarly, to dependable computer system, self-heading systems must have a fault model in terms of what faults they are expected to self-heal. Without a fault model, there is no way to assess whether a system actually can heal itself in situation of interest.

4.2 Why Self-Healing Systems

Despite considerable work in fault tolerance and reliability, software remains notoriously buggy and crash-prone. The current approach according to (Locasto, Stavrou, Cratu and Keromytis, 2007) to ensuring the security and availability of software consists of a mix of different techniques as follows:

- Proactive techniques: This seeks to make the code as dependable as possible through a combination of safe languages (eg Java), libraries and computers code analysis tools and formed methods, and development methodologies (locasto et al, 2007).

- Debugging techniques: This aim to make post-fault analysis and recovery as easy as possible for the programmer that is responsible for producing a fix (locasto et al, 2007)

- Runtime Potation techniques. This try to detect the fault using some type of fault isolation such as stack Guard and format Guard which addresses specific types of faults or security vulnerabilities. (locasto et al, 2007).

- Containment techniques: This seeks to minimize the scope of a successful exploit by isolating the process from the rest of the system, e.g. through use of virtual machine monitors such as VMware or Xen, system, call sandboxes such as systrace or operating system constructs such as Unix *chroot ()*, Free BSD's jail facility and so on (locasto et al, 2007). These approaches offer a poor tradeoff between assurance, reliability in the face of faults, and performance impact of protection mechanism. In particular, software availability has emerged as concern of equal importance and integrity (Watson, 2001).

The need for techniques that address the issues of recovering execution in the presence of faults is reflected by recent emergence of a few novel research ideas (Stelions, locasto, Boyd and Keromytis, 2005). For example, error virtualization operates under the assumption that there exists a mapping between the set errors that are explicitly handled by the program's code. Thus, a failure that would cause the program to crash is translated into a "return with an error code" from the function in which fault occurred (or from one of its ancestors in the stack): These techniques, despite their novelty in dealing with this pressing issue, have met much controversy, personally due to lack of guarantee in terms of altering perform semantics, that can be produced. Making the ordinance of faults will swings carry this stigma since it forces programs down unexpected execution paths? However, the basic premise of masking failures to permit continued program execution is promising.

Stalious et al. (2005) generally believes that a new class of reactive protection mechanism need to be added to the above list. Some techniques that can be classified as reactive include intrusion prevention system (IPS) and automatically generated content signature blockers (Watson 2001). Most such systems have focused on network-based prevention, augmenting the functionality of firewalls. However, a number of friends make the use of such packet inspection technologies unlikely to work with in the future. According to locasto et al, (2001) they are as follows:

- Due to the increasing line speeds and the more computation – intensive protocols that a firewall must support (such as IPsec), firewalls tend to become congestion points. This gap between processing and networking speeds is likely to increase at least for the foreseeable future, while computers (and hence firewalls) are getting faster, the amount of data that must be passed through the firewalls has been and likely will continue to out-pace Moore's law (Locasto et al, 2001).
- The complexity of existing and future protocols makes packet inspection an expensive proposition, especially in the context of increasing the speeds. Furthermore, a number of protocols are inherently difficult to process in the network because of lack of knowledge that is readily available at the endpoints (Locasto et al 2001).
- End – to –end encryption, especially of the on-demand, opportunistic type effectively prevents inspection0based systems from looking inside packets, or even at packet heeders (Locasto et al 2001).
- Worms and other malware have started using polymorphism or metamorphism as cloaking techniques. The effect of these is to increase the analysis requirements, in terms of processing cycles, beyond the budget available to rooters or firewalls. (Locasto; et al; 2001).

4.3 The Future of Self-Healing Systems

Given the embryonic state of the research in self-healing systems, it should come as no surprise that there are significant gaps in our knowledge and the understanding of such systems capabilities and limitations. In order words, this is an extremely fertile area for further research. Rather than describe in detail specific research topics, (Keromytis, Parekh, Gross, Kaiser, Misra, Nieh, Rubentein and stoifo, 2013) outlined three general research thrusts: Fault detection, fault recovery/mitigation, and assurance.

- Fault Detection: One of the constraining factors on the effectiveness of self-healing systems is their ability to detect faults. Thus, ways, to improve fault detection at lower memory and computation cost will always be of importance. One interesting direction of research in this topic concerns the use of hardware features to improve fault detection and mitigation (Keromytis 2013).

- Fault Recovery/Mitigation: Most systems have depended on snapshot/recovery, often combined with input filtering. The three notable exceptions are error virtualization, failure oblivious xxx and data structure repair. The former two techniques can be characterized as "fault masking", while the last uses caring to correct possibly corrupt data values. The technical success of select healing system will depend, on their ability to effectively mitigate or recover from defected faults while ensuring system availability. This, research on additional fault recovery (masking/mitigation techniques is of key importance, especially when dealing with faults at different higher levels (Keromytes et al 2013)

- Assurance: To gain acceptance in the real world, self-healing systems and techniques must provide reasonable assurance that they will not cause damage in the course of healing an application, and that they cannot be exploited by an adversary to attack an otherwise a secure system. This is perhaps the biggest challenge faced by automatic defense system in general. Although, to a large extent, acceptance is dictated by market perceptions and factors largely outside the technical realm, there are certain measures that can make self-healing systems more palatable to system managers and operators.

5.0 CONCLUSION

For decades, the advancement of technology and science has mirrored the increase of complexity in many computer environments.

However, there is an unbalance of growth between the advancement of science and technology with that of complexity. As the scale and complexity of these systems and applications from, their development, configuration and management challenges are beginning to break up current paradigm, overwhelms the capabilities of existing tools and methodologies, and rapidly reminder the system and applications, brittle, unmanage system and insecure. These findings have researchers and scientist in an uproar. As a result, researchers were faced with the test of finding an alternative approach to complexity.

In 2001, IBM developed a mean of overcoming the unbalance of growth between complexity and technology. IBM's solution for reducing the total cost of ownership and coping with the rapidly group complexity of integrating and managing today's computer – based systems was called autonomic computing. Autonomic computing is a new computing environment that can detect and adjust its system automatically to heal itself (self- healing) without the assistance of any human interaction.

Autonomic computers that heal themselves basically endure a process in order to maintain satisfactory quality of service of the principal system during routine in the presence of any fault. The process is a closed loop cycle that consists of monitoring error detection or diagnoses, selection of repair operation and execute repair operation cycle. Each self- healing system process has a fault model that defines what fault it is expected to have.

REFERENCES

Appavoo, K.J; Hui, M.S; Wisniewski, D.D.; Sylva, D.D; Krieger, O and Souls, C.A.N. (2002). "An Infrastructure, for Multiprocessor runt-time adaptation", *Proceedings of the first Workshop on Self-healing System.* Charleston, South Carolina Pp.3-8.

Bantz, D. F., Biskidain, C., Challener,D, Karidis J. P, Mastriani, S., Mohindra, A,,Shea D. G., and Vanover, M.(2003)"Autonomic personal computing, *IBM System Journal* Vol 1 No 42

Davide, T (2004) "Research perspective in self- healing systems", *ERCIM News* No. 58.

Dashofy, E.M.; Hoek, A.V.D. and Taylor, R.N. (2002). "Towards architecture-based self-healing systems", *Proceedings of the first workshop on self-healing systems.* Charleston, South Carolina, Pp. 21 – 26.

Garlan, D. and Schmerl, B. (2002) "Model-based adaption for Self-healing Systems"; *Proceedings of the first workshop on Self-healing systems*. Charleston, South Carolins, Nov. 18 – 19, Pp 27 – 32.

Ganek, AG.; Corbi, T. A. (2003) "The dawning of the autonomic computing era, *IBM system Journal* Vol 42 No 1.

IBM (2003) Whitepaper, an architectural blue print for autonomic

Jiang, M., Zhang J., Raymer, D and Strassner J, (2007)' A modeling framework for self- healing software systems*. academia .edu*

Kephart, J. O. and Chess D M (2003) 'The vision of autonomic computing'*: In IEEE computer,* Vol. 36 No 1, PP 41- 50 DOI: 1109, MC. 2003. 1160055.

Keromytics, A.D Parekh, J., Gross, P. N., Kaiser G., Misoa, V., Nieh, J., Rubenstein, D., and Stolfo S. (2003) Aholistic approach to service survivability, *In Proceedings of the AGM survivable and self-Regenerative Systems Workshop.*

Koopman, P. C. (2003) "Elements of the self- healing system problem space", ICSE, WADS.

Laster, S.S. and Olatunji, A. O. (2007), Autonomic computing: towards a self – healing system. *Processing of the Spring 2007 American Society for Engineering Education* Illinois- Indiana section Conference

Locasto, M.E,Stavrou, A, Cretu, G. E., and Keromytis, A. D, (2007) From STEM to SEAD: Speculative Execution for Automated Defense*. In Proceedings of the USENIX Annual Technical Conference*.

Paul, H. (2001) Automatic computing: *IBM's Perspective on the state of information*

technology" IBM research. http: www. Research gsm. Com/ autonomic)

Stelios, S. Locasto, M. E. Boyd, W.S and Keromytis, A.D (2005), Building a reactive immune system for software services. *In proceedings of the UNSEN IX Annual Technical Conference,* PP 149 – 161

Strassner, J and Kephart, J.O. (2006)" Autonomic system and network: theory and practice", *Network Operations and Management Symposium* (NOMS

Watson, R N. M. (2001) Trusted BSD: Adding trusted operating system features to FreeBSD. *In Proceedings of USENIX Annual Technical Conference,* Freemix Track. PP 20- 40.

Cite this Article: Ugwuanyi, FO (2026). Autonomic Self-Healing System - A Review. *Greener Journal of Science, Engineering and Technological Research*, 15(1): 1-9, https://doi.org/10.15580/gjsetr.2026.1.062626099.