Deploying Self-Organizing-Healing Techniques for Software Development of Iterative Linear Solver.

Okon S. C.[1] and Asagba P. O.[2]

[1]Department Of Computer Science, Akwa Ibom State University, Ikot Akpaden, Mkpat Enin L. G. A. Akwa Ibom State, Nigeria.

[2]Department Of Computer Science, University Of Portharcourt, Choba, Rivers State, Nigeria.

Abstract:

Self-Organization and Self-Healing are fundamental survival/evolutionary technique in natural complex Systems. In this work we present a formal approach to the specification and design of Software that can apply the technique of Self-Organization and Self-Healing to survive unforeseen circumstances. We achieved this by engineering a System that can autonomously reconfigure, reorganized its states to overcome faults/errors thus continuing normal, gracefully degrading or enhanced performance at execution time.Our specification shows how we apply structured system analysis and design methodology, neural network and descriptive model as methodologies to engineer Software whose constituent parts are designed and developed as rule players as is obtainable in autonomous natural complex system. Our Prototype Software is in the area of solving systems of linear equations iteratively. This work can easily be adopted for other Software Projects by making all modules participate in the Software Architecture as rule players.

Keywords: Self-Organization, Self-Healing, Software Component Capacity, Rule Player

1. Introduction

This paper presents the methodology used, the analysis and design of the proposed system. We will also present the algorithms for our software prototype, to demonstrate self-organizing-healing principles. We identify with the fact that, it is becoming increasingly important for software to have a built-in capability to adapt at runtime in varying resources, system errors and changing environmental parameters. Our research has shown that various ongoing work exist on modeling Self-organizing [5, 6, 7, 14, 15, 19, 21], self-healing [25, 36] based-hardware and/or information communication technology systems. We employ a combined therapy for software architecture based on both self-organizing and self-healing software systems. Finally a self-organizing and self-healing mechanism of the software framework is designed and a prototype model developed and analyzed by applying a formal systematic software architecture specification and analysis methodology in order to establish that our system has satisfied the system's time constraint requirements and improve the system's availability and reliability.

2. Methodology

There are several methodologies applicable in the analysis and design of a generic software system. We have decided to use; Structured System Analysis and Design Method (SSADM), Neural Network and Descriptive Models as our methodologies. Here we use models to represent the structure and internal dynamics of individual components, or the deterministic interactions between components. Our model provides leverage on the difficult and important problem of connecting local, autonomous behaviours of individual component to the global, emergent properties of the system. Figure 1 shows the families of models for self-organizing system. We have used Descriptive model which have assisted us in the modularization of both mechanism and components of the system.The transformation from a descriptive, validation and analytical model to a self-organizing application is a forward engineering process while that of self-organizing application to the models is reverse engineering process.

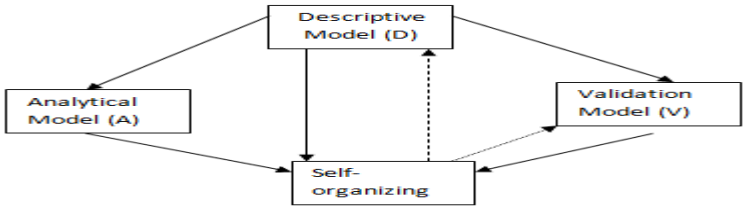


Figure 1: Families of models for self-organizing system

Under modularization, we design the components of our system in modules which are self-similar entities thus defining our software organizational structure. In order to allow modular and reusable modeling that minimizes the impact on the underlying architecture, we proposed a framework based on an organizational approach. We have adopted the Role-Interaction-Organization (RIO) model to represent organizations or systems which enables our module to acquire mechanism for dynamic role playing. We have leaned on this model since it enables formal specification.
Neural network principles are used to establish the interaction between the modules.

2.1 Module Capacity

Large scale systems are expected to organize and cooperate in open environments [34].To satisfy their needs and goals, agents in this case modules often have to collaborate. Thus a module has to be able to estimate the competences of its future partners to identify the most appropriate collaborator. We have introduced the notion of capacity to deal with this issue. The capacity allows us to represent the competences of a module or a set of modules. A capacity is a description of a know-how/service. This description contains at least a name identifying the capacity and the set of its input and output variables which may have default values. The *requires* field defines the constraints that should be verified to guarantee the expected behaviour of the capacity. Then the *ensures* field describe what properties the capacity guarantees, if requires is satisfied. Finally, we add a textual description to informally describe the behaviour of the capacity.

2.1.1 Design of Module Capacity for SOR

Figure 2 defines a modules capacity for SOR Module. This capacity definition applies for the other four modules namely; CGIM, IRM, GSIM, and JAM. Each of them derived their operational competence from the required parameter and their ranking. Table 1 shows the Ranking and Condition for choice module. The ranking is based on their convergence rate.

Name : SuccesiveOverRelaxation
Input : The number of equations and unknowns n; the entries a_{ij} of matrix A; the entries b_i of b; the entries XO_i of XO; omega; tolerance; maximum number of iterations.
Output : The approximate solution $x_i, \ldots, x_n$ or a message that number of iterations was exceeded.
Requires : Requires the parameter Omega
Ensures : That the solution vector $x_i, \ldots, x_n$ is within the tolerance level TOL

Figure 2: The SuccesiveOverRelaxation capacity

However, from the system point of view, we can categorize the systems capacities into three subcategories:

Atomic: The capacity is already present in one of the members of the modules.In this case, the head has to simply request the member possessing the required capacity to perform it.

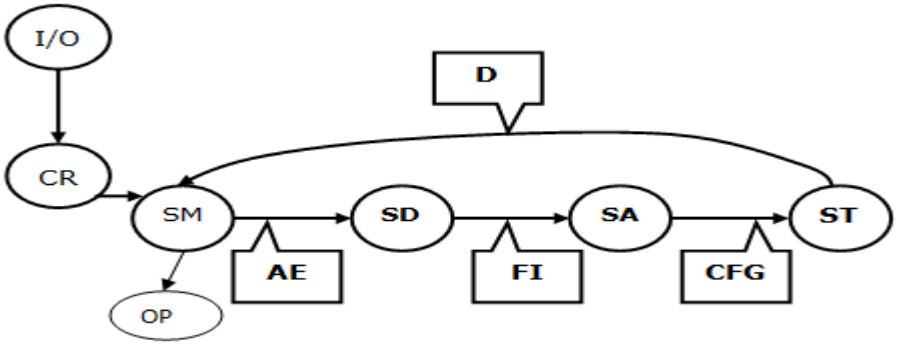
Liaised: The capacity is obtained from a subset of the member's capacities.

Emergent: The capacity is not present as an atomic capacity nor obtained as composition of them, but the capacity emerges from the interactions of the members.

The capacity is atomic for a system if one of the members provides the capacity, but it does not have any implications on how this member obtains this capacity. This taxonomy of capacity is only relative to the system point of view.

3. Analysis Of Self-Organizing-Healing System

Here we used a descriptive model to analysis the operation of a self-organizing-healing system Figure 3 shows input (I/O) being supplied to the system, then a choice (Choose Role Player) is made out of the components present to carry out execution, then the system monitors (Self Monitor) itself for indications of anomalous behavior (Anomalous Event). When such is detected, the system enters a self-diagnosis mode (Self Diagnosis), that is aimed at identifying the fault (Fault Identification) and extract as much information as possible with respect to its cause, symptoms, and impact on the system. Once these are identified, the system tries to adapt (Self Adaptation) itself by generating candidate fixes (Candidate Fix Generation), which are tested to find the best target state (Deployment). Output is published if there is no error.



Key

I/O – Input/Output CR – Choose Role player
SM - Self Monitor AE – Anomalous Event
SD – Self Diagnosis FI – Fault Identification
SA – Self Adaptation CFG– Candidate Fix Generation
ST – Self Testing D – Deployment OP - Output

Figure 3: Descriptive model of Self-organizing-healing System.

4. Design

This section presents the design showing the system architecture with components interconnections.

4.1 Design of the New System

Research has shown that almost 80% of software fault is due to design flaws, while almost 20% is cause by data flaws [9]. Our solution to design flaws is to construct our software in a modular form, where each module has the capacity of solving the same problem, thus implicitly applying design diversity. Some of the techniques used here to solve system of linear equations rearrange the matrix A (say into diagonal matrices). This rearrangement of the data sets provides for a solution to an envisaged data flaws by applying data diversity. Our software prototype is dissected into two main operational modes namely; Self-organizing mode as shown in Figure 4 and self-healing mode as shown in Figures 5, 6, 7, 8 were bio-inspired computing as applied in hardware engineering are now being projected into software reliability engineering. Self-organization refers to a property by which complex systems spontaneously generate organized structures, patterns or behaviors from random initial conditions. It is the process of macroscopic outcomes emerging from local interactions of components. In our software prototype we developed SOR, CGIM, IRM, GSIM, and JAM as module and enable them to interact in order to achieve the system goal. Here the system receives input (the required parameter) from the input file, then autonomously choose the module with the capacity to solve the problem. Self-healing mode enables our prototype to automatically detects, diagnoses and repairs localized software problems. Hence it is able to perceive that it does not operate correctly and, without human intervention, makes the necessary adjustments to restore itself to normalcy (leaning on the ranking of the modules). Our software prototype is based on five method of solving systems of linear equations iteratively. It demonstrates self-organization and self-healing principles at run time and is highly reliable and robust.

4.2 Components of the New System

Our new system is a programmable Self-Organizing-Healing Systems which has answer the following questions. Can software re-arrange its part and evolve towards better performance? Can a swarm of software agents self-organize, self-healed and collectively innovate in problem solving tasks?

Figure 4 shows an overview of Self-organized Software Modules in interaction. The system behaviour is based on the assumption that the role player has the ability and capacity to Solve Linear Equations Iteratively. As long as the implementation honors the constraints established by the capacity, the module is authorized to play the role. In our Prototype, five implementations are present. The SOR owns an implementation based on the Successive over relaxation algorithm, JAM obtains its capacity through Jacobian method, GSIM obtains its capacity from Gauss Seidel technique, CGIM obtains it capacity through conjugate gradient iterative technique while IRM obtains its capacity through iterative refinement method.

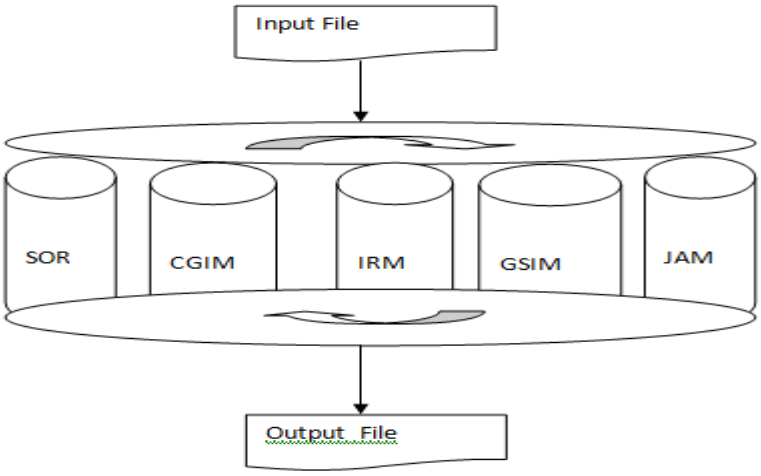


Figure 4: Overview of Self-organized Software Modules in interaction (role interaction basis).

4.3 Schematic Diagrams of the Proposed System.

Here, we will present our proposed system in pictorial forms. Figure 5 shows a descriptive model of our Self-organizing system, demonstrating a network of five modules namely; SOR, CGIM, IRM, GSI, and JAM. Figure 6 shows a neural network representation, were each module is a node at the hidden level in the network and can processes information using a connectionist approach. Like other artificial neural network our system changes its structure base on external or internal information flow through the network. In Table 1, the required parameter is shown. This is the external signal, which triggers state change. Figure 5 forms our descriptive model that enable us to illustrate the network of simple processing element (modules) that exhibit complex global behavior determined by connections between modules and certain parameters (required parameter). Our algorithms are designed to alter the strength (capability) of the connections in the network to produce the desired signal flow through the system. Our attraction in using neural network is that; given a set of systems of linear equations to be solved, our system can use a set of observations to find the solution optimally; this simply means that our system can learn in a supervised manner. Our System based on artificial neural network can self-organize by creating its own organization or representation of the information it receives. It is also capable of graceful degradation in the face of fault or partial destruction, thus retaining the system capability by reconfiguring its state were another module will render the needed services as shown in figures 7, 8, 9 and 10.

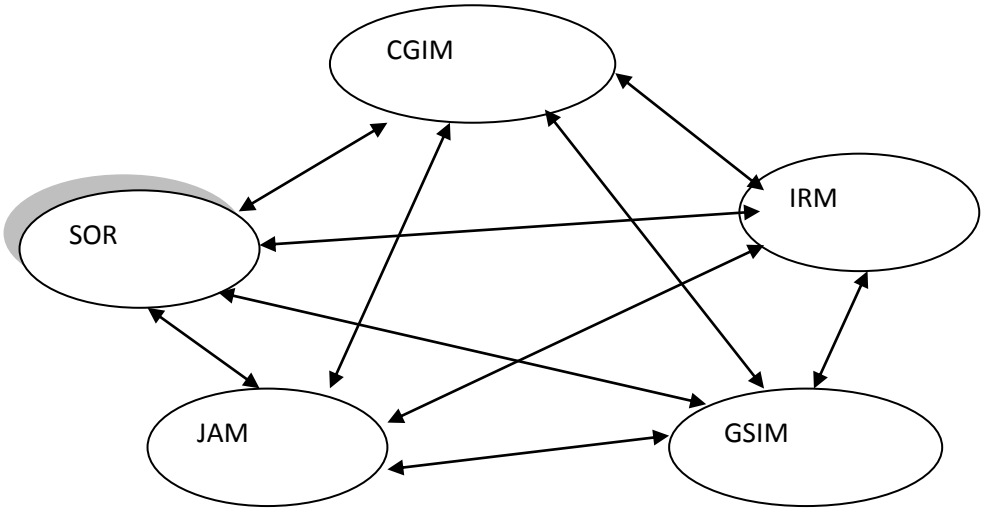
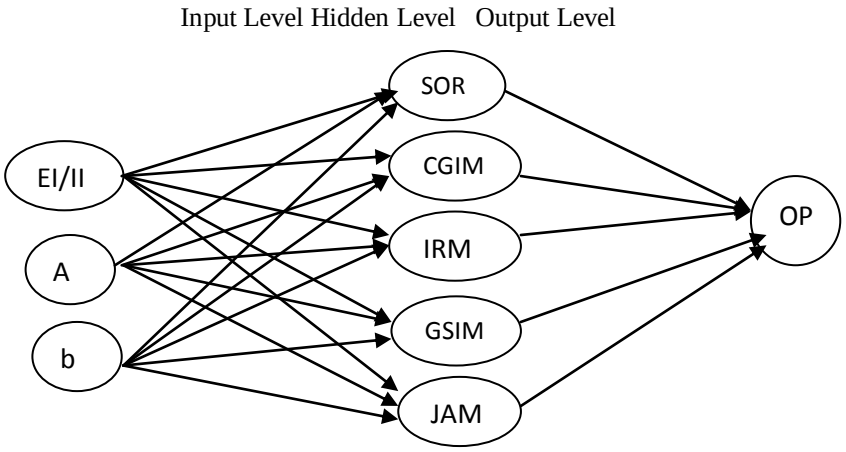


Figure 5: A descriptive model of our Self-organizing system

Table 1: Ranking and Condition for choice module

RANK	MODULE	REQUIRES PARAMETER
1	SOR	Omega
2	CGIM	Pre-condition inverse matrix
3	IRM	'A' the coefficient of X being ill-conditioned
4	GSIM	Few iterations needed
5	JAM	More iteration needed



Key:EI/II-External Input (Required Parameter)/
Internal Input (Rank), A-Matrix A, b – Vector b,
OP- Output (Mode, Scheme, Solution Vector, etc).

Figure 6: Neural Network representation of new system

Figure 7 shows self-healing mode with Recovery block Scheme were SOR was used while executing under self-organization. Figure 8 is for self-healing mode if IRM was used while executing under self-organization. The system continues to change its state/structure based on internal or external input. Consequently, the network structure continues to change until the desired result is found, being one of the benefits of using neural networks. Thus at the failure of self-organizing mode the four other modules are used as alternative modules until result is found.

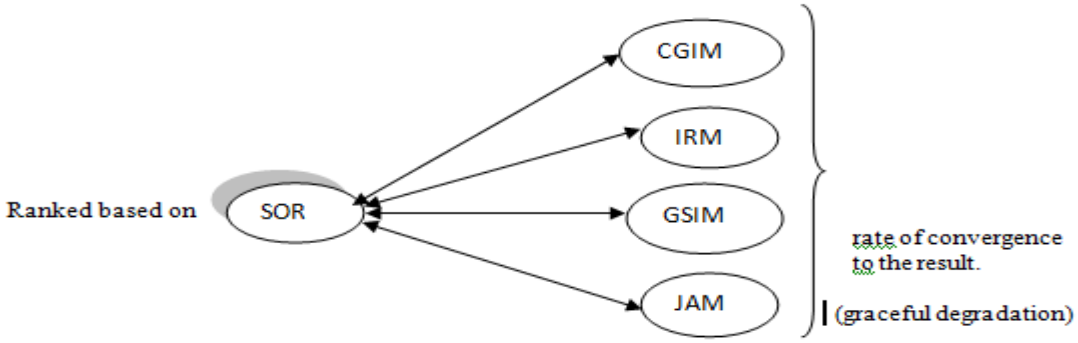


Figure 7: Recovery block as SOR fails in Self-Organization mode

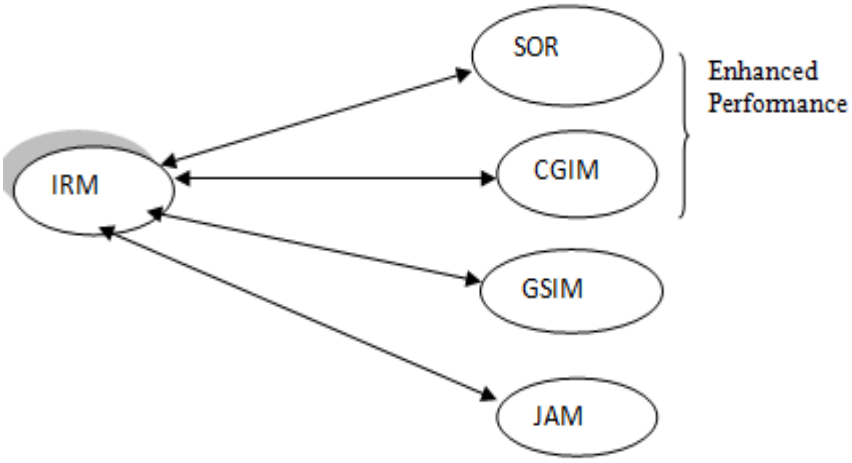
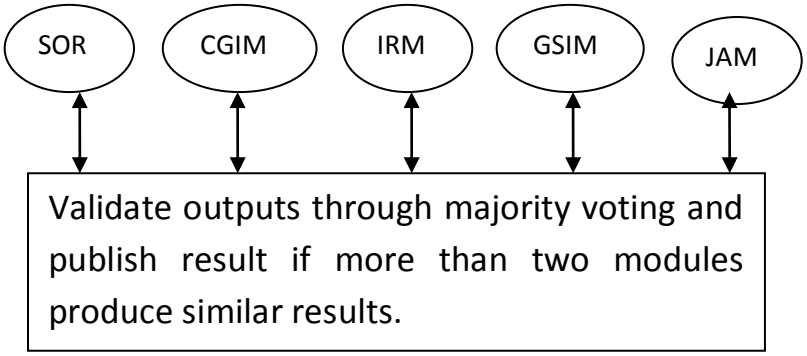


Figure 8: Recovery block as IRM fails at Self-Organization mode.

If the recovery block scheme fails to produce acceptable result, the self-healing mode enters into the N-version scheme as shown in Figure 9. Here the system carry out a majority vote on all the results from all the module, if more than two modules have the same/similar result the result is adopted and publish as the answer.



Para venture, N-version scheme of the self-healing mode fails to produce a result, the system state change to Consensus Recovery block. Here the importance of the algorithm use in testing the result for acceptability is reduced as like other algorithms it may have its own fault. The result of the N-version may not agree due to round-off errors and or hardware constraints (e.g. word length). This may lead to rejection of multiple correct results by the N-version scheme. Here some of the results submitted by the N-version are deem to be correct hence a modified recovery block is entered were a variant of the acceptance test that does not test for correctness but test whether any of the results falls within an acceptable region is used and if that happen the result is announced otherwise a system failure is announces as shown in Figure 10.

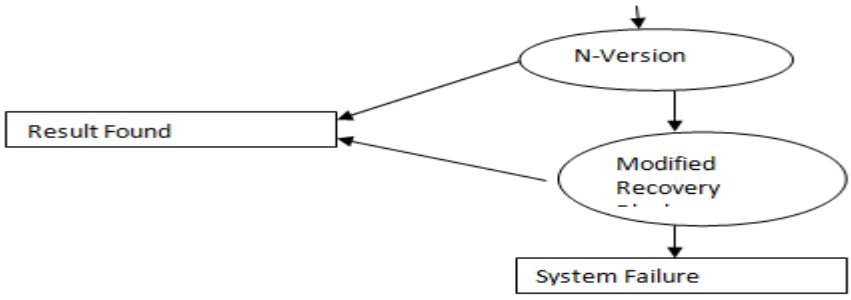
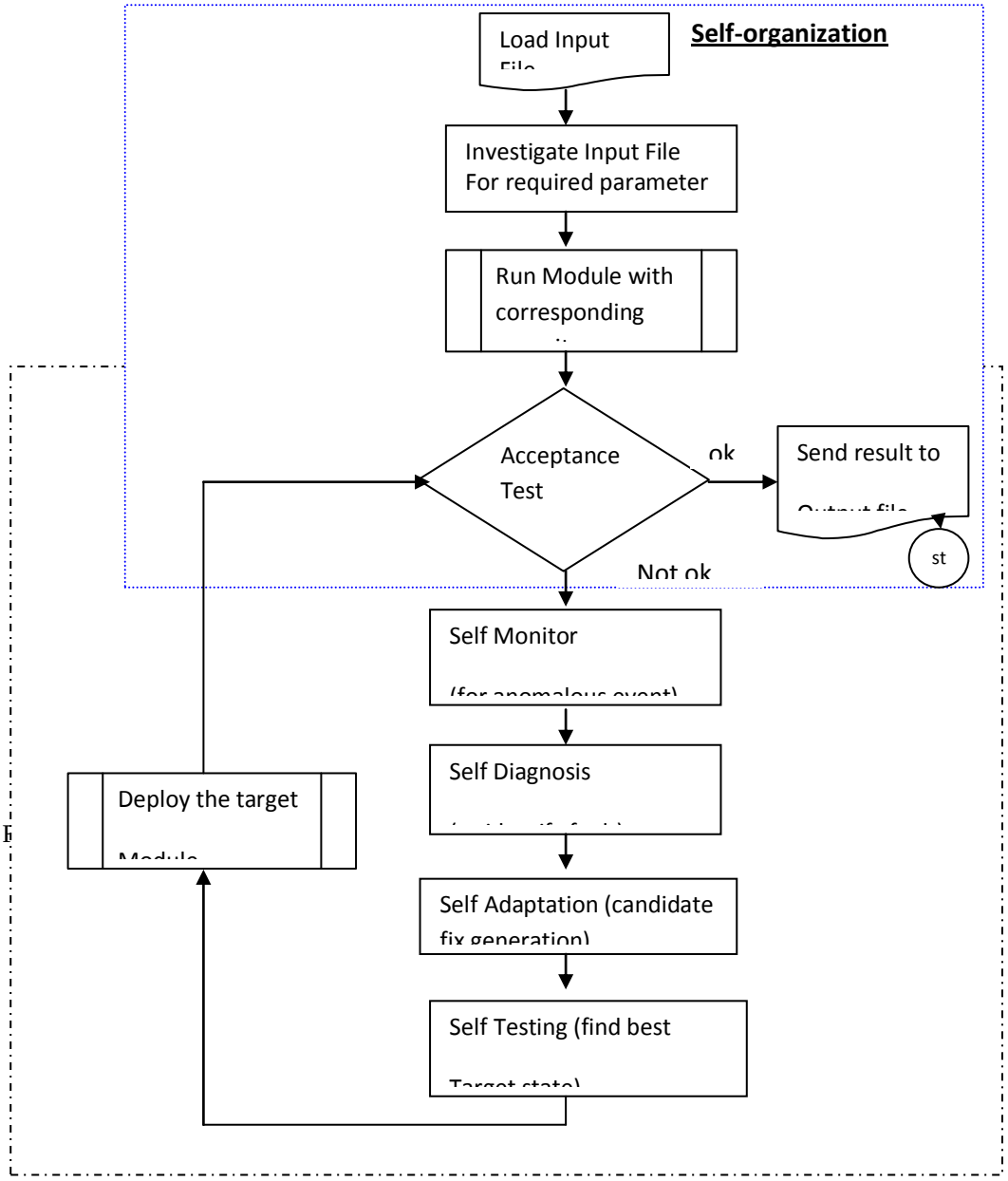


Figure 10: Consensus Recover Block

4.4 System Flowchart of the new System

The figure 11 shows the System Flowchart of the new system highlighting the self-organization and self-healing mode.



4.5 Algorithm for the Self-Organizing-Healing System

In this section we present the algorithm of the Self-organizing-healing software system. Step 1-5 falls under self-organization while step 6-11 falls under self-healing. In steps 6, 7, 8, 9 and 10 the system state is reconfigure and an appropriate scheme is loaded for execution.

1. Load Input File.
2. Search Input File for Required Parameter.
3. Execute Module with corresponding Capacity
4. Submit result for Acceptance Test
5. If result satisfies the Acceptance Test, publish the result and Stop
6. Perform Self Monitor.
7. Perform Self Diagnosis.
8. Perform Self Adaptation
9. Perform Self Test

10. Deploy Target Module.
11. Goto Step 4.

4.6 Design Specification

The System is design to receive input from an Input file, the input include the matrix A, the vector b, the initial approximations and the required parameter as shown in Figure 12. Figure 13 shows the specification of the output file.

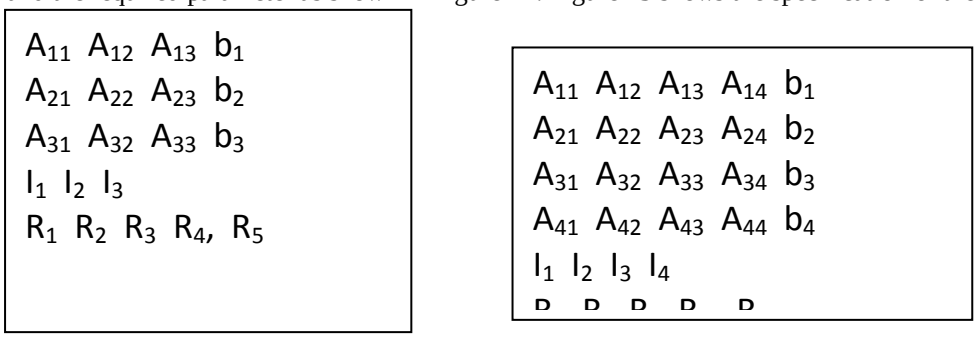


Figure 12: Input File Format, where A_{ij} are elements of Matrix A, b_i are elements of scalar b, I_i are initial approximations and R_i are the required parameters

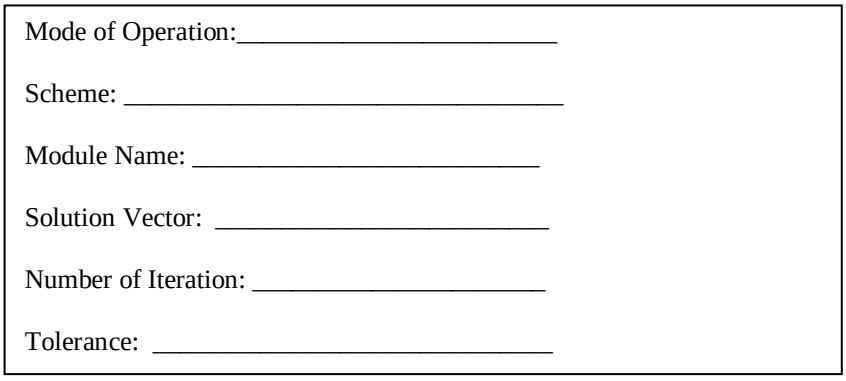


Figure 13: Output File Format.

5. Conclusion

Our paper presents self-organization and self-healing as useful techniques in engineering a safety software system. These ensure software reliability and availability of the resulting software product. Natural Complex Systems have shown resilience in terms of reliability by employing self-organization and self-healing of itself and or its components; it is therefore recommendable for Software Engineers to embrace these techniques in their attempt at developing reliable and available software systems especially in safety critical application.

References

[1] Bar-Yam, Y. and Minai, A. A. (2006), "Complex Engineered Systems: Science Meets Technology", Springer.
[2] Beal, J. and Bachrach, J.(2006), Infrastructure for engineered emergence on sensor/actuator networks, IEEE Intelligent System. 21 (2), pp. 10–19.
[3] Bloch, I. (1995); Information Combination Operators for Data Fusion: A Comparative Review with classification. IEEE Transaction on Systems, Man and Cyber metrics.
[4] Buttazzo G. C. (2005); Hard Real Time Computing System: Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers, Norwel, Ma, USA.
[5] Camazine S., Deneubourg J., and Bonabeau (2001); Self-organization in Biological Systems, Princeton University Press.
[6] Camazine, Deneubourg, Franks, Sneyd, Theraulaz, Bonabeau,(2003). "Self-Organization in Biological Systems", Princeton University Press. ISBN 0-691-11624-5 --ISBN 0-691-01211-3.
[7] Christian P. (2005); Self-organization in Communication Networks: Principles and Design Paradigms, IEEE Communication Magazine.
[8] Denney, R. (2005), "Succeeding with Use Case: Working smart to deliver Quality. Addison- Wesley Professional Publishing.
[9] Dubey A. (2006); Towards a Verifiable Real-time, Autonomic, Fault Mitigation Framework For Large Scale Real-time System. Innovations in Systems and Software Engineering, N ashville, TN 37203, USA

[10] Doursat, R.(2006), "The growing canvas of biological development: multiscale pattern generation on an expanding lattice of gene regulatory networks", InterJournal: Complex Systems 1809.

[11] Doursat, R. (2008), "Programmable architectures that are complex and self-organized: from morphogenesis to engineering, in 11th Int'l Conf. Simulat. Synth. Living Syst.(ALIFE XI), pp. 181–188, MIT Press.

[12] Doursat, R. and Ulieru, M. (2008), Emergent engineering for the management of complex situations, 2nd Int'l Conf. Autonom. Comput. Commun. Syst. (Autonomics).

[13] Doursat, R. (2009), "Facilitating evolutionary innovation by developmental modularity and variability. Generative Dev. Syst. Workshop *(GDS 2009)*, 18th Genet. Evol. Comput. Conf. (GECCO).

[14] Doursat, R. (2010), "Morphogenetic Engineering weds Bio Self-organization into Human Designed Systems", Complex Systems Institute, Paris Ile-de-France, CREA, Ecole Polytechnique, Paris, France.

[15] Doursat, R., Sayama, H., and Michel,O. (2010), "Morphogenetic Engineering: Toward Programmable Complex Systems", NECSI Studies on Complexity, Springer.

[16] Doursat, R. (2011), "The myriads of Alife: importing complex systems and self- organization into engineering", Proc. 3rd IEEE Symp. Artificial Life (IEEE-ALIFE), pp.xii–xix.

[17] Emmanuel Adam and Ren´e Mandiau.(2005). A hierarchical and by role multi-agent organization: Application to the information retrieval. In ISSADS, pages 291–300, 2005.

[18] Endy, D. (2005), "Foundations for engineering biology", Nature 438, pp. 449–45.

[19] Eric Matson and Scott A. DeLoach.(2005). Autonomous organization-based adaptive information systems. In IEEE International Conference on Knowledge Intensive Multiagent Systems (KIMAS '05), Waltham, MA, April 2005.

[20] Eric Matson and Scott A. DeLoach.(2005). Formal transition in agent organizations. In IEEE International Conference on Knowledge Intensive Multiagent Systems (KIMAS '05), Waltham, MA, April 2005.

[21] Falko D. (2007), "Self-organization in Sensor and Actor Network", Wiley & Sons.

[22] Ghosh, D., Sharman, R., Rao, H.R., and Upadhyaya, S. (2007), "Self-healing systems- Survey and synthesis", Decision Support Systems, 42(4):2164–2185,2007.

[23] Kennedy, J. and Eberhart,R. C. (1995), "Particle swarm optimization", Proc. IEEE International Conference on Neural Networks, pp. 1942–1948.

[24] Mikhail P. (2008), "Advances in Applied Self-organizing Systems", Springer.

[25] Modafferi, S., Mussi, E., and Pernici, Barbara (2006), "A Self-Healing plug-in for Ws-BPEL engines, Middleware for Service Oriented Computing (MW4SOC)", Workshop of the 7th International Middleware Conference Nov. 2006, Melbourne, Australia

[26] Musa, J. D. (2005). Software reliability engineering: more reliability software faster and cheaper, 2nd Edition, AuthorHouse.

[27] Myrna E. (2006), "Self-organizing Natural Intelligence: Issues of Knowing, Meaning, and Complexity, Springer-Verlag.

[28] Nabuco, O., Halima, R. Ben, Didra, K., Fugini, M. G., Modafferi, S., and Mussi, E.,(2008),"Model-based QoS-enabled self-healing Web Services", EN-FINES'08 DEXA Workshop, Turin.

[29] Okon S. C. (2006); A Fail-safe Strategy for Scientific/Engineering Project (A Tool for Sustainable Development). Journal of Sciences and Technology Research Vol. 5 No. 2,pp. 6-9.

[30] Okon S. C. (2006); Microprocessor Devices (Computer) like Man; A Self-organizing System. Journal of Research in Physical Sciences Vol. 2 No. 4 pp. 1-7.

[31] Ricard V. and Jordi B. (2006), "Self-organization in Complex Ecosystems", Princeton University Press.

[32] atnam A., Kai G. , and Alice A. (2007); A Framework for Intelligent Sensor Validation, Sensor Fusion, and Supervisory Control of Automated Vehicles in IVHS. Intelligent System Research Group, Department of Mechanical Engineering, UC Berkeley.

[33] Scott K. J. A.and Engstrom D. A. (2006), "The Complementary Nature", MIT Press, Cambridge, MA.

[34] Sebastian Rodriguez, Vincent Hilaire, and Abder Koukam(2005). Fomal specification of Holonic multi-agent system framework. In Intelligent Agents in Computing Systems, International Conference on Computational Science (3), number 3516 in LNCS, pages719–726, Atlanta, USA.

[35] Takahashi, M. and Kamayachi, Y.(1995), ``An Emprical study of a Model for Program Error Prediction,'' Proc. Int. Conference on Software Engineering, Aug. pp. 330-336.

[36] The WS-Diamond Team (2008), "Self-healing Web Services in the WS-DIAMOND project". Proc. E-Challenges Conference, October 2008, The Hague.

[37] Ulieru, M. and Doursat, R. (2011), "Emergent engineering: a radical paradigm shift",International Journal on Autonomous and Adaptive Communication System. (IJAACS) 4(1), pp. 39–60.

[38] Wasson, C. S. (2006), "System analysis, design and development", John Wiley & Sons. ISBN 0471393339.