A New SDLC Framework with Autonomic Computing Elements

Kunal Sahadeva
Krishna Institute of Engineering and Technology

Sanjeev Kumar Yadav
Krishna Institute of Engineering and Technology

Arun Sharma
Krishna Institute of Engineering and Technology

ABSTRACT

The autonomic complex nervous system of human has ability to manage itself. It adapts changes in its environment and manages by self-healing, self-protection and self-optimization. This gave a robust idea to manage current complex engineering systems and reduce big efforts in maintenance in terms of cost and time of technically skilled human resource. This paper is providing a concept of developing an autonomic manager simultaneously with the developments of its particular system or software, via injecting the autonomic elements into conventional software development life cycle (SDLC) and hence it will become secure SDLC or autonomic SDLC framework.

Keywords

Autonomic computing, self-healing, self-protection, self-optimization, self-configure, secure SDLC & autonomic SDLC

1. INTRODUCTION

Software maintenance requires good amount of skilled human effort and cost of development. Sometimes it is catastrophic for industry when they need to deploy their experts in old projects to maintain. Jones Capers, 2006 has shown that how maintenance budget is going to increase from 9% to 77% (1950-2025) [17]. With rapid growth of technology complexity of systems have been evolved so now there is a special need of systems those can manage themselves. Motivated by this idea we studied concept of autonomic computing and found if we add autonomic elements during software development phase to make software with ability of managing themselves will reduce a very big amount in maintenance phase and will keep free skilled human resources to develop new systems/technologies. Autonomic computing is a computer environment that can detect and adjust its system automatically to heal itself without the assistance of any human interaction. Autonomic Computing is inspired by the robust functioning of human autonomic complex nervous system having the ability to manage itself and dynamically adapt to changes in environment [4, 6]. Autonomic computing systems are analogous to human nervous system having the property of self-adjustment, self-managing mechanism, self-configuration, self-healing, self-optimization and self-protection. In this thesis we have discussed important theories and technologies to develop Autonomic Computing Models and we used them to develop new software development life cycle with autonomic manager so this manager will be able to sense problem in advance so it can take appropriate action to avoid problem. If a problem or error happened to be notice during run of software autonomic manager would be able to manage and optimize system during run time too. To illustrate our concept of injecting autonomic element in SDLC and impact on maintenance we made a case study of library management system and it is found in long term developing autonomic manager is more cost effective than maintenance and problem caused during failure or malfunction of software**.**

2. AUTONOMIC COMPUTING

Autonomic computing is the study of developing the systems or software, which can monitor environmental changes itself, analyze the symptoms of problem itself, plan the remedial actions, execute change-plan automatically, and reconfigure itself according to the changing demands by the business put upon it. There are four different characteristics of an autonomic computing system or software [1, 2, 3, and 4].

- Self-Configure
- Self-Healing
- Self-Optimization
- Self-Protecting

Definitions:
Self-Configuration: This characteristic refers to the adapting mechanism of an autonomic software component, which dynamically adapts with any changes arising in its environments, by using policies set by the IT professionals.

Self-Healing*:* This refer to the mechanism of fixing the malfunctions had occurred in autonomic software , for that it discover the causes, diagnoses and take corrective actions to resolve the problem, by using policy based corrective plans and actions set by technologists.

Self-Optimization: This characteristic refers to reducing the workload of a system (autonomic) by means of tuning system’s resources, like for e.g., reallocating resources against the dynamic changing workload, for that it would monitor the dynamic workload and tune essential parameters of a autonomic software or systems.

Self-Protection*:* It means prevents software or a system (autonomic) from malicious attacks or from a danger of crashing or uncertain shutting down, for that it will anticipate, identify proactively the symptoms of any threats and generate early warnings for users to protect their systems, software or hardware.

3. LITERATURE WORK

IBM company in 2001, discovered the term known as an autonomic computing; in the paper “Autonomic computing system” was compared with robust “Autonomic computing nervous system” of Human Body and pointed out four properties to achieve self-achievement, that is, Self configuration, Self-optimization, Self- Healing, and self-protection[1, 2, 3].

In 2003, IBM presented a reference model based on “two intelligent control loops (local and global)” which accumulates data from internal and external environments and the system known as “MAPE-K” [2, 3]. This model consisted of autonomic elements (AEs) like sensors, effectors, planner and executor, which is responsible of managing the "Managed Resources" and this complete model, is called as autonomic manager (AM). The Control loop with sensors and effectors together with Monitor, Analysis, Plan, Execute, Knowledge components makes the autonomic element to be self-manage. The managed resources can be operating systems, wired or wireless network, CPU, database, servers, routers, application modules, Web service or virtual machine and so on.

Autonomic manager consists of monitoring component, analysis component, planning component, and execution component and knowledge repository. The monitoring component provides the ability of self-awareness and detects the external environment. The analysis component then carries out autonomic decision-making and decides the adaptive goal of system; planning and execution components achieve the adaptive function when the system state departures from the expected goal. The operation of four components is supported by the knowledge repository [2, 7].

In June 2005 edition, IBM presented a white paper and highlighted the high-level architectural blue print of autonomic systems. Autonomic computing is mean to design the software, or systems which cans shift the burden of managing the complexity and maintenance from human resources to the technologies [2, 3, 4]. Here, it has discussed that how the work of autonomic manager of monitoring, analyzing, planning and executing makes an intelligent control loop, which could be abbreviated as MAPE. This control loops be injected to the run-time environment of managed resources, and can control them, see in the Figure 1.

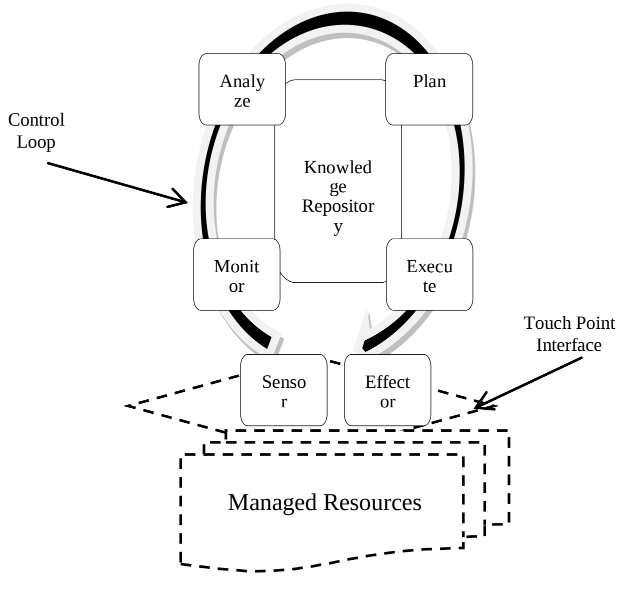
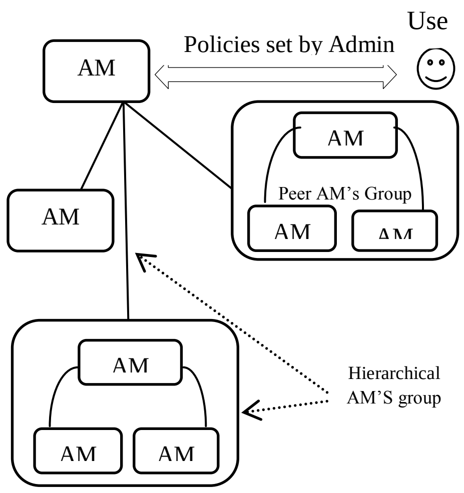


Figure 1: Autonomic Manager Architecture Centerpiece, motivated by [2, 3, and 11]

Moreover, this blueprint gives the framework how interconnection of many distinct Autonomic Managers (AMs) can be joined in hierarchy and work together for a whole complex system, see in Figure 2. Here, in this figure various autonomic managers at each level communicating with each other; moreover they are consuming and providing services to each other.



AM: Autonomic Managers
Figure 2: Hierarchical interactions amongst AM(s) [3, 11]

Mark S. Merkow and Lakshmikanth Raghava in the article [5] pointed out the securities measures for the development of software, according to authors software security play an important role everywhere because if a software have flaw then it may affects those systems on which they are running, and in turn it will cause danger to the physical world because system and software works for physical world. Maintenance and security could become expensive if they are handling and fixing after the development of software. Hence the maintaining expenses and vulnerabilities to securities of software can be save if the securities would add proactively in all the phases of software development life cycle (SDLC) and it cheap the software development.

The article [8] tell us about the misconception with the application's developments, and that incorrect practices are in auditing the problems related to security and maintenance, because security and maintenance usually concerns and fixing after the deployment of software in its operational environment, which is really a time consuming and dramatically expensive task, so to prevent software's from these issues, developer need to change their usual software's development practices, they should integrate security-requirements in early phases of software development life cycle (SDLC) and the outcome of these practices will impact on application's development, i.e., their development can be obtained at lower costs, moreover it will become more secured and robust.

Autonomic manager is useful for web services also which is itself is the source of providing services to autonomic manger itself, in paper [13], authors proposed a self-healing model for web services by using autonomic computing, This framework is consisted of autonomic manager, data logger, adaptive content wrapper, underlying services. Data logger is using to calculate the response time of request made by client and send the information with request- specific information to the autonomic manager, where autonomic manager track the current state of the server, check the distinct response time at different resolution, and that resolution at which request is to be handled, one filter, written for every available services, and depending upon the resolution filter called Adaptive content wrapper modify a request.

An accord programming framework proposed in [14], describing the platform for designing the self-managed application inspired by autonomic concept, authors of this paper, tried to use adaptability of autonomic applications with the existing programming systems, for example, existing programming systems like, Grid programming systems, distributed programming models like object oriented, service oriented, and component oriented etc. Analogy to the objects oriented programming system, autonomic elements will encapsulate rules, constraints, and mechanisms of autonomic properties, and dynamically they will interact with each other.

According to accord programming framework, autonomic elements are defined by three operational ports listed below [14] and shown in Figure 3:

a) Control Port
b) Functional Port
c) Operational Port

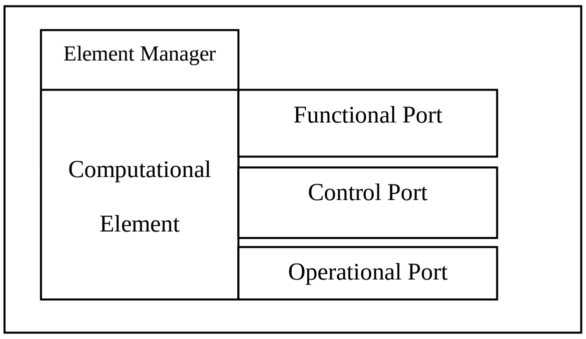


Figure 3: An Autonomic Element in Accord Framework referred from [14]

Functional Port: it is a set of all functional behaviors that are requires by elements and provided by elements.

Control Port: defines the set of tuples, tuples (sensors/ actuator, constraints) that is tuple sensor/actuator is a set of sensors and executers that are utilize by elements for getting details about managed resources and applying change-plan on managed resources [14].

Operational Port: defines the SET of interfaces between the sensors and managed resources and interface between effectors and manageable resources, these interfaces are also called as Touchpad or touch point interfaces, through touch points, autonomic manager can formulate rules, dynamically inject policies and change plan, transfer actions via effectors and retrieve details of managed resources through sensors, and one manager can communicate with another Autonomic managers in hierarchy [14].

4. PROPOSED FRAMEWORK

Influence by the thought of adding security in software's from beginning of its development processes not after its production, and the benefits of incorporating autonomic computing in them , we are describing an idea for developing applications with their own autonomic managers. For that purpose, here the table is designed, see Table1.

If autonomic computing will involve with traditional software development life cycle (SDLC), then applications develops will be robust, highly secure and have ability of self-managing itself, moreover, due to such abilities, an application will available at lower cost. And thus maintenance cost of an application will reduced incredibly. Then this way conventional SDLC will become Secure SDLC or alternatively Autonomic-SDLC.

The idea is to collect its autonomic manager requirements for the application and documented these non-functional requirements (NFR- SRS), and later merge them with the requirements of software's SRS, similarly do the same in its designing phase, process it to the coding phase, and then in testing phase, test, validate and verified the autonomic manager on its application, whether it is really fit for its software or not. The difference will come between the traditional SDLC and Autonomic SDLC are in the requirement and in the designing phase mostly, because the documentation for developing AM will depends on the type of a application, on which AM will perform its operations, and in designing phase the decision making algorithms have to be translated and tested differently from normal type of algorithms for a software, the general software development life cycle (SDLC), see in Figure 4.

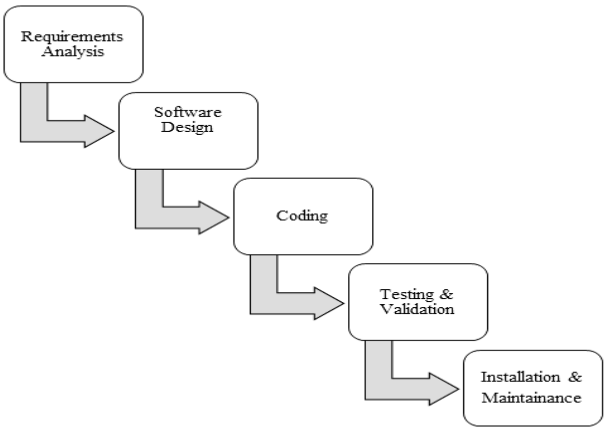


Figure 4: Conventional Software Development Life Cycle referred from [10]

Following are factors and practices have represented below, which could responsible for leaving negative impacts on the maintenance of software [15, 16, and 17].

- Defect tracking software has(-24%) negative impacts on software maintenance
- No maintenance specialists also leaving (-18%) negative impacts on it.
- Management inexperienced has (-15 %) negative impacts on maintenance phase.
- No Online defect reporting tools also leaving (-12 %) negative impacts.
- No user satisfaction measurements have (-4 %) negative impressions on maintenance process.
- Manual change control methods also have (-27%) negative impacts on maintenance of software.

Developer and technologist must follows the utilization of autonomic manager, as a part of the conventional SDLC, it is possible that initial costs of developing software and systems will increase initially little more than the conventional costs of developing software due to addition of autonomic manager, but the running costs or maintenance costs of software at will guaranteed decreased due to autonomic manager as a part of a software.

Table1: Importance of autonomic computing elements in conventional SDLC phases: - Table 1 is showing, how security measures are taken care by autonomic computing elements in SDLC inherently or implicitly by injecting autonomic computing elements and proposed new SDLC frame work with autonomic manager [5, 8, 9]. Conventional SDLC phases are [10]:

[1] Requirement phase
[2] Designing phase
[3] Coding phase
[4] Testing (Verification Validation) phase
[5] Maintenance phase

Table 1: Autonomic SLDC

1) REQUIREMENT ANALYSIS PHASE		
Conventional SDLC's Phases INPUTS	***Autonomic Elements and security measure in SDLC***	***Importance of autonomic elements in SDLC phases***
Key Inputs[5]	**Key Deliverables**	
▪Defines Needed Information, ▪Behavior ▪Functions ▪Industry Requirements ▪Interfaces ▪Organization Requirements ▪Performance ▪Privacy Requirements	▪Compliance Goals ▪Industry/Organizational Standards ▪Lessons ▪Map securities ▪Measurements Etc. ▪Privacy Requirements ▪Security Requirements Engineering ▪Technical Requirements ▪Threat Modeling	Need to include requirements of *self-Protection and self-healing* components of Autonomic Computing. Need to document the following in this Phase. Important security related design goals. Prioritized security & privacy requirements.

2)SOFTWARE DESIGNING PHASE		
Conventional SDLC's Phases INPUTS	**Autonomic Elements and security measure in SDLC**	**Importance of autonomic elements in SDLC phases**
Key Inputs	**Key Deliverables**	
▪Inputs From Previous Phase ▪Algorithmic Details ▪Data Flow Diagrams ▪Data Structures ▪Design Principles ▪Interface Representations ▪Software Architecture ▪Technical &Non-Technical Security Control Requirements	▪ Architecture & Design Patterns ▪ Architecture & Design Review ▪ Measurements ▪ Security Design Review ▪ Security Test Planning ▪ Threat Modeling ▪ Threat Modeling	Need Self- optimizations (Tune parameters),Self-Protection(prevent from attacks on Algorithms, Data's and Data structure) Programming an autonomic element will mean extending Web services or grid services with programming. Tools and techniques that aid in managing relationships with other autonomic elements. Because autonomic elements both consume and provide services, representing needs and preferences will be just as important as representing capabilities.

3)IMPLEMENTATION AND INTEGRATION PHASE		
Conventional SDLC's Phases INPUTS	***Autonomic Elements and security measure in SDLC***	***Importance of autonomic elements in SDLC phases***
Key Inputs	**Key Deliverables**	
▪ Inputs from previous phases ▪ Database ▪ Secure coding standards ▪ Secure configuration standards ▪ Source code ▪ Testing ▪ User documentation	▪ Architecture patterns ▪ Architecture review ▪ Design Review ▪ Design Patterns ▪ Measurements ▪ Security Test Planning ▪ Static Analysis ▪ Threat Modeling ▪ Peer Review	Self-configuration configure themselves automatically, High-level policies (what is desired, not how) Need tools to build elements that can establish, monitor, and enforce agreements. Developers will need tools that help them acquire and represent policies—high-level specifications of goals and constraints typically represented as rules or utility functions—and map them on to lower-level actions.

4)TESTING(VERIFICATION AND VALIDATION) PHASE		
Conventional SDLC's Phases INPUTS	***Autonomic Elements and security measure in SDLC***	***Importance of autonomic elements in SDLC phases***
Key Inputs	**Key Deliverables**	
▪ Inputs from previous phases ▪ Documentation ▪ Requirement ▪ Software Deployed In Test Environment	▪ Attack Patterns ▪ Automated Testing ▪ Regression Testing ▪ Stress Testing ▪ Measurements ▪ Prioritized list of vulnerabilities from automated and manual dynamic analysis ▪ Security test cases document ▪ Third Party Assessments ▪ Threat Model Updates	Testing autonomic elements and verifying that they behave correctly will be particularly challenging in large-scale systems because it will be harder to anticipate their environment, especially when it extends across multiple administrative domains or enterprises. It might be possible to test newly deployed autonomic elements by having them perform alongside more established and trusted elements with similar functionality.

5)INSTALLATION PHASE		
Conventional SDLC's Phases INPUTS	***Autonomic Elements and security measure in SDLC***	***Importance of autonomic elements in SDLC phases***
Key Inputs	**Key Deliverables**	
▪ Installation is all of the activities that make a software system available for use	▪ Deployment ▪ Incident Management ▪ Measurements ▪ Operational Security ▪ Patch Management ▪ Threat Model Updates	Security Monitoring and Response plan .i.e., Self-healing (Analyze information from log files and monitors). Installing and configuring autonomic elements will most likely entail a bootstrapping process that begins when the element registers itself in a directory service by publishing its capabilities and contact information. The element might also use the directory service to discover suppliers or brokers that may provide information or services it needs to complete its initial configuration.

6)MAINTENANCE PHASE		
Conventional SDLC's Phases INPUTS	***Autonomic Elements and security measure in SDLC***	***Importance of autonomic elements in SDLC phases***
Key Inputs	**Key Deliverables**	
Is the modification of a software product after delivery to correct faults, to improve performance or other attributes	▪ Adaptive – dealing with changes and adapting in the software environment ▪ Perfective – accommodating with new or changed user requirements which concern functional enhancements to the software ▪ Corrective – dealing with errors found and fixing it ▪ Preventive – concerns activities aiming on increasing ▪ Software maintainability and prevent problems in the future	**Self-configuration** Configure. themselves automatically High-level policies (what is desired, not how) **Self-optimization** Hundreds of tunable parameters. Continually seeks ways to improve their operation. **Self-healing:** Analyze information from log files and monitors. **Self-protection:** From Malicious attacks, cascading failures.

The autonomic computing software development life cycle or AUTO-SDLC has showed in the Figure 5. The left hand side of the cycle is showing the conventional SDLC's phases, and at the mid portion of figure, showing development of interfaces (Touch point) between autonomic manager and software's resources, and at the right hand side of the figure, representing development of autonomic manager along with all phases of conventional SDLC.

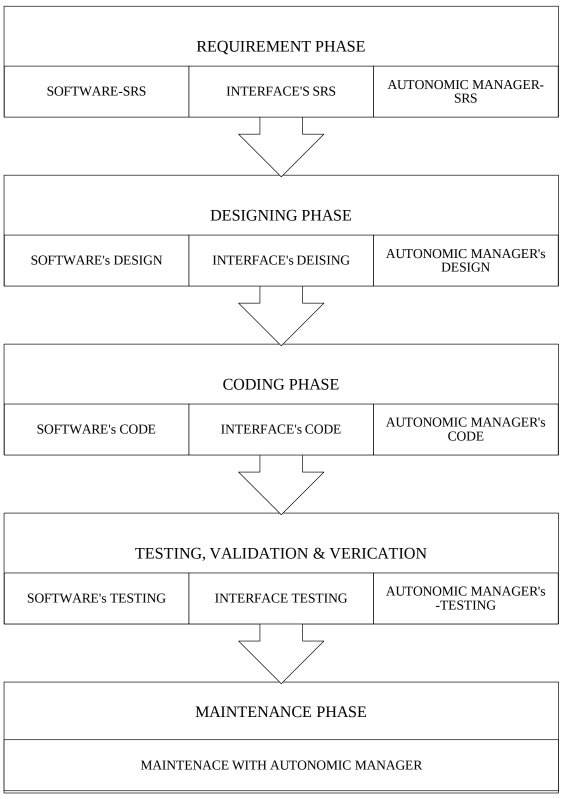


Figure 5: AUTONOMIC-SDLC

5. SUMMARY

We discussed about several literature work related to maintenance problems and their running costs. Then we introduced about a Autonomic computing technology, an emerging in IT world, that is, how autonomic computing, inspired from autonomic intelligent nervous system of human’s body, which runs all essential functions of a body without our conscious efforts. Same way, information technologies like, Internet, Computers, software, operating systems etc. also required such an intelligent controller, which can save technologist and user’s time and money for maintaining their software and systems. Ours aim is to suggest a path for autonomic application development. For example, if securities are mapped in all phases of conventional-SDLC early, plus an application must develop with its own Autonomic Manager, then technologist can get framework for developing autonomic applications, in this paper one Table1 has composed, which map the autonomic computing elements with conventional SDLC and their importance, and later on we showed that autonomic-SDLC in the Figure 5 known as AUTO-SDLC. Auto-SDLC framework could provide a direction to technologists or developers to develop autonomic applications means to say, applications built together with their own autonomic managers. For example, development of operating system with its own autonomic manager, business application with its own autonomic managers etc., and to do this, it must follows all types of autonomic computing strategies and algorithms to develop the Autonomic computing equipped systems and software in future. And finally for systems to be autonomic system at all level that is, from hardware level to software it requires challenging modifications. In case of software level, for example, for operating systems, an active code of software may be replaced while if any hardware is detected to replace it can be re-change dynamically. Some efforts have also needed to be focused in the direction of installing of autonomic middleware’s programming systems at runtime.

6. CONCLUSION

We analyzed present and future scenarios of problems in maintenance of software and found a need to establish a new paradigm of software development using autonomic computing concepts.

Autonomic computing systems are analogous to human nervous system having the property of self-adjustment, self-managing mechanism, self-configuration, self-healing, self-optimization and self-protection. It will reduce the natural intelligence i.e. human being involvement.

We proposed a new framework of SDLC with collaborative development of autonomic manager. Autonomic manager will also be developing in different phases of software life cycle starting with special emphasis from requirement elicitation to testing and implementation phase of SDLC. Impact may be measurable on the initial cost of software development and it will be higher than traditional development but during the whole life cycle of software the overall cost will decrease as it would have cheaper maintenance and running cost along with great user satisfaction and will keep free skilled human resources to develop new systems/technologies.

Therefore for a lifetime usage it will be more efficient, cheaper and system will run smoothly.

7. FUTURE WORK

Proactive Fault Management Systems- System which can sense the fault in advance i.e. in an operating system there various performance parameters and combination of these parameters has a certain impact in failure/degraded performance. Then based on study and prediction of maximum probable error a system can take proactive steps to avoid fault(s).

Evolutionary Data Structures- This is well known fact that during software development a particular data structure is decided to implement based on suitability to performance of algorithm but in most of scenario where input size is uncertain and input pattern is too, therefore based on study of distribution of uncertainty of inputs automated selection and evolution of appropriate data structures.

Dynamic Query Optimization- In database systems where queries are designed statically and many time they did not perform efficiently in dynamic environment so making system to optimize queries dynamically will make a shift towards autonomic computing.

Failure Recovery Systems- Systems got failed due to missing of some critical parts or failures of those parts in that case if our system can replace with healthier part so it can be recover from failure

8. REFERENCES

[1] Horn, P. 2001. Autonomic computing: IBM's Perspective on the State of Information Technology.

[2] Dr. Guy, M. Lohman. 2003. SMART (Self Managing and Resource Tuning). IBM Research.

[3] IBM Corporation. 2005 Third Edition. White Paper: An architectural blueprint for autonomic computing.

[4] Wong, S.F., and Ives, B. 2003. ISRC Future Technology Topic Brief Autonomic Computing. Available at www.bauer.uh.edu/uhisrc/FTB/Autonomic/AutonComp.pdf.

[5] Mark, S. Merkpw and Raghhavan, L. 2010. Software Security for Developers. Available at the online link http://www.csoonline.com/article/618463/software-security-for-developers.

[6] Zhao, Z., Gao, C., Duan, F. 2009. A Survey on Autonomic Computing Research, Second Asia-Pacific Conference on Computational Intelligence and Industrial Applications. Vol. 2. pp. 288-291.

[7] Jeffrey, O. Kephart., David, M. Chess. 2003. IBM Corporation Thomas J. Warson Research center. The Vision of Autonomic Computing. Available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf".

[8] McAfee service datasheet. Intel Corporation. Secure software development life cycle (SSDLC). Available at http://www.mcafee.com/us/services/strategic-consulting/program-development/application-and-software-development-lifecycle.aspx.

[9] Sharma, A., Chauhan, S. and Grover, P. S. 2011. Autonomic computing: Paradigm Shift for Software Development. CSI Communications.

[10] Roger S**.** Pressman. 2005. Software Engineering: A Practitioner's Approach. 5th Edition. Tata McGraw Hill.

[11] Tiwari, V., Milenkovic, M. 2006. Standard for Autonomic Computing. Intel Technology Journal at http://www.intel.com/technology/itj/2006/v10i4/3-standards/1-abstract.htm. Volume 10. Issue 04. Published November 9, 2006.

[12] Manish, P., Hariri, S. 2006. Autonomic Computing Concepts, Infrastructure and Applications.

[13] Naccache, H., Gannod, G. C. 2007. A self-healing for web services, IEEE Conference on 9-13 July 2007. pp. 398-345.

[14] Hua, L., Parashar, M. 2006. Accord: A Programming Framework for Autonomic Applications. IEEE transaction. Journal and magazines. Vol. 36. pp. 341-352.

[15] William, R. Estimating Software Costs. Available at the link "cs.iupui.edu/~mroberts /n361/sdarticle1.pdf".

[16] Jussi Koskinen. 2010. Software Maintenance Costs. Department of Computer Science and Information Systems. University of Jyvaskyla. Available at the online link http://users.jyu.fi/~koskinen/smcosts.htm.

[17] Reifer, D., Judy, J., et al. 2010. Total Cost of Software Maintenance Workshop. Approved for public release, review by AMRDEC Public Affair Office, FN4344. At http://"csse.usc.edu/csse/../Software%20maintenance%"20Workshop.pdf".