Volume 3, Issue 8, August 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Acquiring CHOP Properties in Generic Software Development Life Cycle

Raghavendra Kr.Dwivedi, Awadhesh Srivastava
Department of IT, KIET Ghaziabad
India

Abstract— Software engineering is a process of development and evolution of complex and critical system and application software. In this era software systems require more effort to enhance automated features in their development phases by which such systems can be timely developed and produce better application. In this paper we focus on to enhance automated features like self-Configuration, self-Healing, self-Optimization, and self-Protection in the generic software development phases for the progressive refinement and enhancement of high-level requirement and system specifications. The objective of injecting Autonomic properties is to build a self-managing application software system. In this paper we proposed an automated software engineering life cycle and try to integrate autonomic feature in conventional life cycle to develop self managing software application.

Keywords— Configuration, Healing, Optimization, Protection, SDLC, etc.

I. INTRODUCTION

Autonomic Computing is a concept that brings together many fields of computing with the purpose of creating computing system that are reflective and self adoptive. It is a collection and integration of the technologies into the IT application with the abilities to manage it and dynamically adopt to change in accordance with business policies and objectives. The Autonomic Computing Initiative aims to provide the foundation for autonomic system. It is inspired by the autonomic nervous system of the human body. This nervous system controls important bodily functions (i.e. heart rates and blood pressure) without any conscious intervention [1]. The autonomic research activities in software systems can broadly be categorized into four areas: monitoring of components, interpretation of monitored data, creation of a repair plan (i.e. an adaptation of the system), and execution of a repair plan. Based on this, we choose to group the approaches to autonomic computing systems orthogonally into two categories: intelligent multi-agent systems and architecture design-based autonomic systems [2]. However, the two approaches have common concepts it is sometimes difficult to place a research project in one particular category [6].

Software Engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build. It is a "systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of a software by applying engineering to the software". IEEE Computer society's software engineering body of knowledge defines software engineering as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. For developing any software a proper process is followed and that process consists of many development phases like: requirements, design, implementation, verification and maintenance. This process makes a cycle and this cycle is known as software development life cycle. Automatic and collaborative systems are both important areas of automated software engineering, as are computational models of human software engineering activities [7]. Knowledge representations and artificial intelligence techniques applicable in this field are of particular interest, as are formal techniques that support or provide theoretical foundations.

II. AUTOMATED SOFTWARE ENGINEERING

Automatic and collaborative systems are both important areas of automated software engineering, as are computational models of human software engineering activities [7]. Knowledge representations and artificial intelligence techniques applicable in this field are of particular interest, as are formal techniques that support or provide theoretical foundations.
Software engineering is concerned with the development and evolution of large and complex software-intensive systems. It covers theories, methods and tools for the specification, architecture, design, testing, and maintenance of software systems. Today's software systems are significantly large, complex and critical, that only through the use of automated approaches can such systems be developed and evolve in an economic and timely manner [3].
Automated software engineering applies computation to software engineering activities. The goal is to partially or fully automate these activities, thereby significantly increasing both quality and productivity [4]. This includes the study of techniques for constructing, understanding, adapting and modeling both software artifacts and processes.
Automated software engineering approaches have been applied in many areas of software engineering. These include requirements definition, specification, architecture, design and synthesis, implementation, modeling, testing and quality assurance, verification and validation, maintenance and evolution, configuration management, deployment, reengineering, reuse and visualization [4]. Automated software engineering techniques have also been used in a wide

range of domains and application areas including industrial software, embedded and real-time systems, aerospace, automotive and medical systems, Web-based systems and computer games.

Articles in the Requirements group address tool support for the scenario-based discovery of requirements, animation and validation of business transactions, or the use of natural language processing to improve the quality of requirements. Autonomic computing capabilities simply making applications that can take care of themselves. Applications would automatically diagnose their own problems and fix them, reasoning out how to protect themselves from future bugs. The applications would become self-aware, self-protecting, self-healing, self-optimizing, and self-configuring. [1] **Self-configuration:** an autonomic computing system configures itself according to high-level goals, i.e. by specifying what is desired, not necessarily how to accomplish it. This can mean being able to install itself based on the needs of a given platform and the user. **Self-optimization** an autonomic computing system optimizes its use of resources. It may decide to initiate a change to the system proactively (as opposed to reactive behavior) in an attempt to improve performance. **Self-healing** an autonomic computing system detects and diagnoses problems. What kinds of problems are detected can be interpreted broadly: they can be as low level as a bit-error in a memory chip (hardware failure) or as high-level as an erroneous entry in a directory service (software problem) [3]. If possible, it should attempt to fix the problem, for example by switching to a redundant component or by downloading and installing software updates. Fault-tolerance is an important aspect of self-healing. Typically, an autonomic system is said to be reactive to failures or early signs of a possible failure. **Self-protection:** an autonomic system protects itself from malicious attacks but also from end users who inadvertently make software changes, e.g. by deleting an important file [1]. The system autonomously tunes itself to achieve security, privacy and data protection. Self-management requires that a system monitor its components (internal knowledge) and its environment (external knowledge), so that it can adapt to changes that may occur, which may be known changes or unexpected changes where a certain amount of artificial intelligence may be required [3]. Self-configuration enables to deploy large applications requiring solely a few simple actions. Intelligent mechanisms cope with deployment complexity. These mechanisms can be based on a set of rules or a semantic description of applications, depending on the used approach. Self-healing and Self-protection make systems more secure and less fragile to occurring physical or software faults. Finally, self-optimization creates more proactive systems that are always searching for opportunities to enhance their performance, for instance by balancing the processing loads dynamically.

III. HISTORY AND WORK DONE SO FAR

The goal of autonomic computing is to build a self-managing system that addresses these challenges using high levels guidance. It can enable a new generation of knowledge-based, Data information driven, computationally intensive and context aware application. This concept was first proposed by IBM in 2001.IBM defined five evolutionary level for its deployment [6].

IBM DB2 universal database has several autonomic properties like rapid DB2 deployment via optimized configuration tooling and dynamic adjustment for system security and management. WINDOW XP/7 incorporates self healing technology. When an application crashes, the user can shut it down systematically, thereby preventing the entire system from freezing or hanging. This Operating System also offers to report program errors to the Microsoft Support System team.WINDOW XP/7 looks out for updates and automatically downloads these when available. Plug-and-play is another element of autonomic computing. Plug-in a new device to PC and the system will automatically detect it (MS-Office 2007 include a Repair feature).Intel Itanium 2 processor has built-in Autonomic Features. It allows the system to continue executing transactions as it recovers from several errors conditions. Several articles deal with Program Understanding and Architecture and present research in software maintenance support via reengineering, automatic support for software reuse and support for designing and validating architectural specification.

A larger group of articles addresses Software Testing, covering aspects such as automatic generation of test cases, testability analysis, or test-suite planning and derivation using the Unified Modeling Language (UML).The articles on Software Verification deal with model checking and the verification of groupware protocols. Another group of articles addresses Aspects and Language Technology, with topics including aspect-oriented techniques for software adaptation for pervasive computing, use of language technology for the analysis and transformation of large software systems, or the identification of cross-cutting concerns in embedded C code [6]. The articles on Configuration Management and Deployment include aspects such as tool environments for software maintenance and adaptation, automation of software release and deployment, or frameworks and associated services for deploying and maintaining networks of collaborative, distributed environments [8]. Finally, several articles demonstrate the use of Models in software engineering; transformation and verification of models, model transformation languages, and model-driven development of pervasive systems [9].

IV. AUTONOMIC MODEL

A conventional software life cycle contains system requirements, system design, coding, testing and implementation phases. Typically software processes order these phases in sequentially or spiral manner. Most software engineering experts expressed their opinion that an autonomy software cannot be developed using conventional life cycle.

In this paper we proposed an automated software engineering life cycle and try to integrate autonomic feature in conventional life cycle to develop self managing software application.Fig-1 illustrates the inheritance process where CHOP properties are being inherited in each phase of life cycle. An autonomic element's life cycle begins requirement analysis and end with implementation or maintenance phase while configuration, optimization, upgrading, monitoring, problem determination, and recovery has been comprehensive in CHOP phase.

Requirement Analysis: A graphical representation of the phases begins with Requirement Analysis phase which contains the all autonomic prerequisite to make our application more self proficient. Users will provide the qualitative as Well as quantitative high level, reusable and domain specific information for the business logic which may be inspired from fuzzy requirements to self configure in order to meet business requirements.

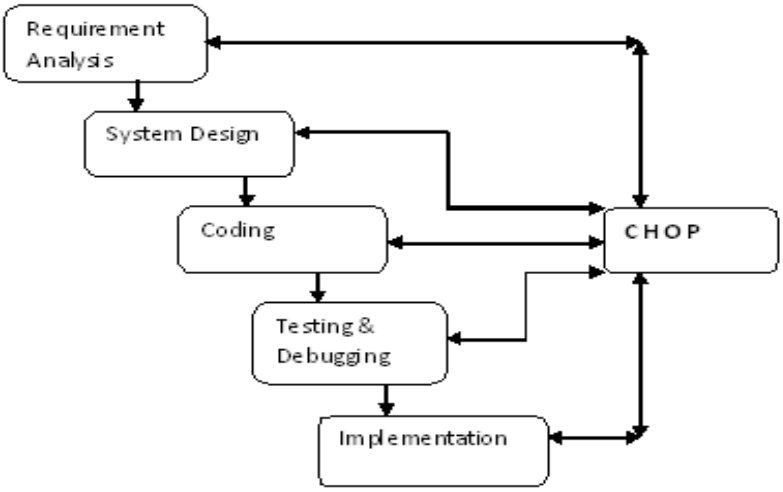

Fig-1

System Design: An autonomic design required a multi-tiered Architecture , where components across the tiers are open to self-configuration. Design must support the space of possible proactive behaviours. To make such designs possible, we need concepts for characterizing large spaces of alternative behaviours/configurations.

Coding: The autonomic software coding has self-management characteristics that make product line instances more resilient to context changes and some aspects of dynamic application. 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 [18].

Testing and Debugging: Testing autonomic elements [11] 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. Testing networked applications that require coordinated interactions among several autonomic elements will be even more difficult. Major objective of testing process to check and validate the fault tolerance of the application.

Characteristic & Phase	Requirement Analysis	System Design	Coding	Testing	Implementation
Configuration					
Healing					
Optimization					
Protection					

Fig-2

Implementation and Deployment phase: Installing and configuring autonomic elements will come in the last phase of life cycle Fig-2 shows the partially and fully participation of CHOP feature in each phases. Design and Testing phases required fully participation and represented by dark shadow while rest phases required partial efforts to implement respective autonomic features.

V. FUTURISTIC APPLICATION ASPECT

Autonomic computing can be implemented in various areas like electricity, railway management, air traffic control systems, Multimedia application etc. The directory service to discover suppliers or brokers that may provide information or services it needs to complete its initial configuration. It can also use the service to seek out potential customers or brokers to which it can delegate the task of finding customers. Current electricity transmission systems employ manual management of interconnected subsystems that supply electricity to the users. This manual management is inefficient and error prone. To remove these shortcomings, autonomic capability can be integrated to these systems. The sensor could monitor the environment factors such as temperature, humidity, wind speed, month, etc and autonomously provide the necessary power. Introduction of high-speed trains has posed the need for a faster management of the whole system. Manual management is quite slow as compared to management through computers. Also the level of imprecision in these real time systems is highly dynamic. Train schedules are also dynamic. Autonomic computing can provide this speed of decision making. This technique of using technology to manage technology has real benefits to these real time systems. The system will configure itself upon introduction of new trains and new tracks. This will also decrease collision probability or any slowdown due to failure at any point.

Self-management capability is the key to successful multiprocessing where each processor is running their own instance of operating system. With autonomic capability the throughput, resource utilization, performance of the system will increase. Centralized architecture of current traffic control systems will soon become brittle and unmanageable as the traffic volume and number of devices used as a sensor to provide situation of traffic at a time increases. Thus, there is a need of a decentralized system that consists of autonomic devices capable of monitoring the environment and communicates with other adjacent autonomic devices so as to make decisions for the complete traffic management. Self-configuration nature can be built as the capability to identify new node and take it into working use. If a node fails others around must notify and take charge of its working. A multimedia application able to adapt the quality of data transmitted in response to a change in the communication bandwidth. Application will take input from networking service and then dynamically adapt to the frame rate with a resolution that synchronizes with the bit rate of the connection. Travel-Guide application can be developed which is capable to provide location-dependent service in response to a change in the user's location. GPS system can be used to track a user and provide this input to an autonomic element which can then search for available services in that locality in its database.

VI. CHALANGES IN AUTONOMIC COMPUTING

There are various technological challenges in developing autonomic elements. Among a few important ones are: Autonomic elements must be designed with a focus on interaction with other autonomic elements so that they could communicate easily [17]. For this there is a requirement of a standard interface for communication, to which all autonomic elements adhere to. AC can provide exchange of information (data or control message) between them. Architecture of autonomic elements must be designed with multiple threads each for a closed loop [17][18]. Human experts may need some form of rule based learning as authoring a large number of rules is quite difficult. Models and methods for learning need extracting correlations between low-level system measurements and high-level service level objectives based on statistical regression/modelling. Information about events must be stored in a standard log file format so that every autonomic element can use it [18].This would cope with the difficulty in handing heterogeneity of individual elements. Self-adjustment of system must lead system to an optimal configuration .This must be done continually as soon as machine starts learning. Architecture of an autonomic system must support the intercommunication and interaction among autonomic elements. Human administrator must be provided with an expressive interface so as to able to monitor, visualize and control autonomic systems.

VII. CONCLUSION

Automated software engineering is known for the computation of software engineering activities while motive is to automate these activities, to increase both quality and productivity. The aim of acquiring CHOP properties is to build a self-managing application software system that addresses these challenges using high levels guidance. In this paper we proposed an automated software engineering life cycle and try to integrate autonomic feature in conventional life cycle to develop self managing software application. After studying various articles and research papers we have concluded that autonomic computing can be implemented in various areas like electricity, railway management, air traffic control systems, Multimedia application etc.

REFERENCES

1. J. Kephart and D. Chess, "The Vision of Autonomic Computing," IEEE Computer 36(1): 41-50 (2003).
2. M. Parashar, S. Hariri, "Autonomic Computing: An Overview," UPP 2004, Mont Saint-Michel, France, Editors: J.-P. Ban, Â¢tre et al. LNCS, Springer Verlag, Vol. 3566, pp. 247-259, 2005. Abbas N., Andersson J., and Lowe W. 2010. Autonomic Software Product Lines (ASPL). In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (ECSA '10), Carlos E. Cuesta (Ed.). ACM, New York, NY, USA, pp: 324-331.
3. Arnautovic E., Kaindl H., Falb J. 2008. High-Level Modeling of Software Management Interactions and Tasks for Autonomic Computing. In Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS '08). IEEE Computer Society, Washington, DC, USA, pp: 212-218.

4. Castelli G., Mamei M., and Zambonelli F.. 2008. Engineering Contextual Knowledge for Autonomic Pervasive Services. Information Software Technology.Vol. 50, No.1-2 (January 2008), pp: 36-50.
5. Lightstone S., Seven Software Engineering Principles for Autonomic Computing Development. Innovations in Systems and Software Engineering, Vol. 3, No. 1, pp:71-74, March 2007.
6. Lin P., MacArthur A., and Leaney J.. 2005. Defining Autonomic Computing: A Software Engineering Perspective. In Proceedings of the 2005 Australian Conference on Software Engineering (ASWEC '05). IEEE Computer Society, Washington, DC, USA, pp: 88-97.
7. Nami M.R. and Bertels K., 2007. A Survey of Autonomic Computing Systems. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems* (ICAS '07). IEEE Computer Society, Washington, DC, USA, 26-.
8. Quitadamo R. and Zambonelli F., Autonomic Communication Services: A New Challenge for Software Agents, Autonomous Agents and Multi-Agent Systems Volume 17, Number 3, pp : 457-475.
9. Rubio-Loyola J., Astorga A., Serrat J., Chai W. K., Mamatas L., Galis A., Clayman S.,Cheniour A., Lefèvre L., Mornard O., Fischer A., Paler A., Meer H. de. Platforms and Software Systems for an Autonomic Internet. In Proceedings of GLOBECOM'2010, pp:1-6.
10. Salehie M. and Tahvildari L.. 2005. Autonomic Computing: Emerging Trends and Open Problems. *SIGSOFT Software Engineering Notes* Vol.30, Issue. 4, (May 2005), pp: 1-7.
11. Sadjadi S. M., McKinley P. K., and Cheng B.H. C.. 2005. Transparent Shaping of Existing Software to Support Pervasive and Autonomic Computing. SIGSOFT Software Engineering Notes, Vol. 30, No. 4 (May 2005), pp: 1-7.
12. Smith D., Morris E., and Carney D.. 2005. Interoperability Issues Affecting Autonomic Computing. *SIGSOFT Software Engineering Notes,* Vol.30, No. 4 (May 2005), pp: 1-3.
13. Vassev E. and Hinchey M. 2009. ASSL: A Software Engineering Approach to Autonomic Computing. Computer, Vol. 42, No.6 (June 2009), pp: 90-93.
14. Zhang H., Wang H., and Liu H.. 2008. Research on Autonomic-Trusted Software Evaluation. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 05 (CSSE '08), Vol. 5. IEEE Computer Society, Washington, DC, USA, pp:1180-1183.
15. S. R. White, J. R. Hanson, I. Whalley, D. M. Chess, J. O. Kephart, "An Architectural Approach to Autonomic Computing," Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04), IEEE Computer Society Press, pp 2-9, May 2004.
16. Smith MF Software Prototyping: Adoption, Practice and Management. McGraw- Hill, London(1991).
17. Dr. Arun Sharma, Sandeep Chauhan and Dr. P S GroverAutonomic computing :Paradigm Shift for Software Development CSI communications September 2011www.csi-india.org

