A Survey of Self Healing of Software Faults: Recent Advances and Research Challenges

Ajay Bandi
Mississippi State University
ab1370@msstate.edu

Edward B. Allen
Mississippi State University
edward.allen@computer.org

Tomasz Haupt
Mississippi State University
haupt@cavs.msstate.edu

Abstract

Autonomic computing is a biologically inspired computing paradigm for managing large and complex software systems. The four main aspects of autonomic computing are: self-configuration, self-healing, self-optimization, and self-protection. This paper presents a survey to develop a taxonomy of self-healing of software faults, to identify various techniques to diagnose and recover from software faults, and to note some research challenges in self-healing of software systems. Our survey aims to facilitate a fault based approach to self-healing systems. We present a comprehensive tabular overview of software faults, detection methods, and recovery techniques.

Keywords: *autonomic computing, self-healing, software faults, literature*

1. Introduction

The ever-increasing dependence on cyberinfrastructure to perform tasks related to almost all aspects of human activity has resulted in the development of unprecedentedly complex software systems that operate in large-scale, distributed, heterogeneous environments. In addition to the inherent complexity of the business logic, there is an increasingly significant requisite for these software-based systems to manage resource variability, ever-changing user needs, and system faults. Designing systems with the capability to reliably perform specified tasks while compensating for such demanding and fluctuating parameters has thus become a grand challenge for software engineering.

The endeavor to answer this challenge originated the concept of self-healing systems (and autonomic computing in general, which involves also self-configuration, self-optimization, and self-protection). Self-healing can be defined as the property of a system that enables it to first perceive that is not operating correctly, then to make adjustments to restore its normal operation. More specifically, a self-healing system recognizes when it is incapable of producing results according to Quality of Service (QoS) (whether explicit or implicit) specifications, due to unavailability of resources, software faults, degradation of performance, etc., and consequently makes necessary adjustments to restore its capability to perform its tasks. Both aspects of self-healing, detection of anomalies and actual healing, are expected to be performed autonomously, i.e., without explicit user intervention. Instead, the behavior of the system is defined by a set of high-level policies.

The field of self-healing is a new one, and judging by the volume of the scholarly publications on this subject, it has already attracted considerable attention in computer science communities. The aim of this paper it to analyze the current progress made in this field. We mention our goals in Section 3.1

2. Related work

What are the research categories in existing self-healing of software system surveys? Several researchers surveyed self healing systems in different perspectives. Ghosh et al. [13] described the process of self-healing systems and classified different phases in self-healing software systems. They described a self-healing process usually consists of three steps: 1) maintenance of “health” 2) detection of system faults and 3) recovery from system faults. This survey categorized different techniques for maintaining, detecting and recovery of the systems. In addition, they mentioned a few application areas of self-healing strategies such as grid computing, software agent based computing, reflective middleware, clustering etc. Keromytis [15] discussed the self-healing of software failures especially using structural changes to the software under protection. He mentioned several techniques to self-heal software faults such as memory updates, data structure repair etc.

Psaier and Dustdar [18] concentrated on identifying principles by explaining the concepts and evolution of self-healing systems. Furthermore, they discussed in detail the approaches and applications of

self-healing systems. Some of the approaches mentioned in their survey are: separation of concerns, intrusive versus non-intrusive, closed versus open, detecting and reporting suspicious behavior, diagnosing and policy selection, and recovery techniques. Likewise the authors discussed self-healing application areas in embedded systems, operating systems, discovery systems, and architecture based systems. However, our categories are based on software faults which is different from others work.

3. Research methodology

A literature survey is a procedure to identify, synthesize and depict the published research based on research questions. This is useful to identify research opportunities from the existing literature. We followed these steps to carry out this survey. 1) Formulate goals/research questions. 2) Search and select articles. 3) Extract data. 4) Synthesize data. 5) Report the survey results.

3.1. Formulate goals/research questions

The main goal is to answer the question, "*What are the recent advances and current research challenges in the automatic detection and correction of software faults?"* We also interested in how software faults are categorized. Our categories are different from the related work. This high level question is subdivided into three other research questions.

1. What are different types of software faults identified in recent literature?
2. What techniques are used to detect software failures automatically in recent literature?
3. What techniques are used to recover from failures automatically in recent literature?

3.2. Search and selection of articles

We exhaustively searched articles from the following databases using the keyword search strings similar to (self-healing OR self healing) AND software AND (survey OR review): ACM Digital Library, Computer Science Index, Google Scholar, IEEE Xplore, Proquest (dissertations and theses), Scopus (Elsevier and Springer), and Web of Science.

Inclusion criteria: We selected survey/review articles on self-healing software systems. The latest survey paper we found had surveyed all the papers until 2008 [18]. So we considered papers published in 2009-2011 plus the earlier survey papers. We included papers that focus especially on self-healing of software faults (not hardware or communication faults).

Exclusion criteria: We excluded papers that are not related to our research questions, papers focused on self-healing on biology topics, electrical power grids, and other faults. Also, studies not in the English language and invited talks are excluded. Most of the excluded papers were eliminated after reading the abstract, introduction and conclusions.

3.3. Data extraction

We use a data extraction form to collect these data items: Bibliographic information (author, title, source and year of publication), type of article (journal/conference), goals of the paper, type of software fault the author is trying to heal, and techniques for automatically detecting and correcting software faults. The first author extracted data from all the selected papers and the other authors reviewed the extracted data. This data from the selected papers was synthesized to answer each research question in Section 3.4

3.4. Report the survey results

Question 1. What are different types of software faults identified in recent literature?

The self-healing of software faults taxonomy was created by grouping similar types of software faults into a single category from the observed literature. Errors that related to similar characteristics are grouped as a category. These software faults are identified mainly in distributed systems, web services, component-based, service-oriented systems, web servers, and application servers. We categorized errors in off-the shelf components as a separate group and it consists of source code errors and other integration problems. Failures in servers may be caused intrusion or attack of malicious software and other software faults in the servers.

Process errors consist of corrupted data structures, un-handled exceptions, memory errors and other process errors. Race conditions and deadlocks are grouped as concurrency errors. Changes in the environmental or missing components, unanticipated changes in the architectures and exception handling in service-based processes management system, which could be viewed as change in functionality or architecture of workflow, are set in the degraded functionality category. Degraded performance consists of response time and QoS degradation. These categories are explained in next paragraphs. Our taxonomy of software faults in self-healing systems is shown in Figure 1.

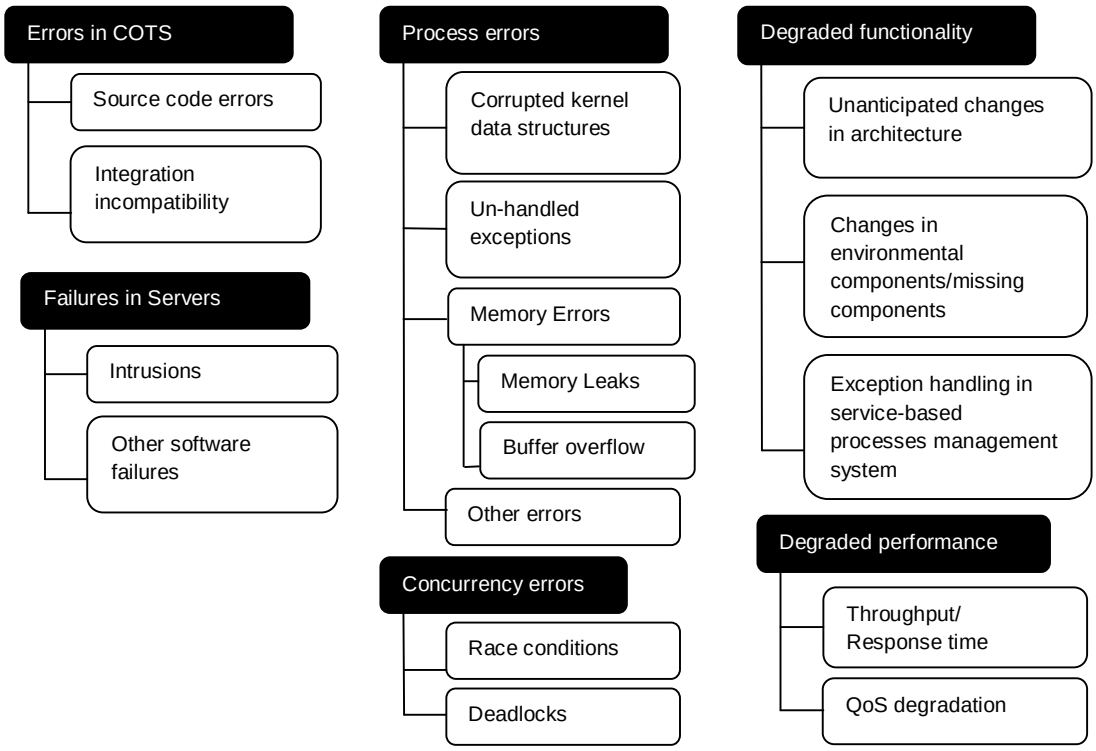

Figure 1. Taxonomy of software faults in self-healing systems.

Question 2. What techniques are used to detect software failures automatically in recent literature?

Question 3. What techniques are used to recover from failures automatically in recent literature?

We combined the research questions 2 and 3 in this subsection to answer them easily. Based on our findings, we categorized the software faults into six different categories. Table 1 shows the category, type of software fault and automatic detection and correction techniques.

Errors in Commercial-Off-The-Shelf (COTS) software: Developers use third-party vendor software modules/components called COTS. The source code or documentation for these modules is typically not available. Therefore, self-healing of COTS is difficult. Several exceptions may arise when these complex components integrate with software systems. Chang et al. [5] proposed injection of healing connectors into system to recover from in-field errors in COTS. Gama and Donsez [11] presented an autonomic approach of execution of COTS outside the main application, in a sort of sandbox without disturbing the trusted components. Bisadi and Sharifi [3] used a cellular adaptation method to self-heal components when the source code is not available to manipulate. Errors in the components are detected by monitoring the messages passed through connectors when the components exchange data between each other. By reconfiguration of the component or by using appropriate recovery policy from a repository, the system can recover. Silva et al. [21] develop a virtualization layer to recover data from off-the-shelf application servers.

Failures in servers: Intrusions or attacks in the application servers can be detected by monitoring and checking for any violation of security policy. Once the intrusion is encountered, proactive recovery techniques are used to recover the application server from intrusion [19]. Sidiroglou et al. [20] developed the ASSURE system which introduces rescue points that recover software from unknown faults. Rescue points are locations in existing application code for handling a given set of anticipated faults from a larger class of unanticipated faults. Chen et al. [6] developed SHelp which enables the system to survive software faults in virtual machines. They use error virtualization and weighted rescue points to circumvent the fault paths. Error virtualization is the method used to force heuristic-based error return in a function.

Process errors: Errors in kernel data structures are detected by comparing the run queue and task list and these data structures are recovered by repair plans [22]. Gaudin et al. [12] proposed an approach to self-heal unhandled exceptions at run time. This is based on the supervisory control approach. The supervisor is embedded in the application through instrumentation and controls the execution of the program consulting with the Finite State Machine (FSM), which is built from the method calls of the Java application. The supervisor monitors for un-handled exceptions consulting with the FSM. The faulty sequence of method calls is derived by code instrumentation.

Bond and McKinely [4] introduces leak pruning to self-heal out-of-memory errors. These errors are caused by double frees, dangling pointers, and buffer

overflows. Such errors are detected when the memory consumed is greater than a threshold value. Memory recovery is obtained by pruning the selected references. Dai et al. [8] proposed a new approach called "consequence based self-healing" of memory corruption by using multivariate decision diagrams, neural networks and fuzzy logic. The consequences and prescriptions should be predefined based on the goals. The two types of consequences identified by Dai et al. are context consequence (memory consumption) and content consequence (state variables in the program). A consequence based approach diagnoses problems not only based on the consequence contents but also on the severity level of the consequence. Some of the recovery techniques are reclaiming the leaked memory or restarting the whole system from the latest check point depending on the severity level. Andrzejak [2] advocates discovering the process errors by monitoring online data and performance modeling of the process and is recovered by replicating the process.

Concurrency errors: Concurrency related bugs are identified by failures in interactions between threads. Recovery is achieved by additional synchronizing or influencing scheduling [14]. Wang et al. [23] prevented deadlocks by control logic using code instrumentation.

Degraded functionality: Martinez and Dobson [16] developed a decentralized component-based Java framework for complex heterogeneous distributed systems called "Functionality Recomposition for Self-Healing" (FReSH). The components can be methods, web services, and procedures. Detection of operation disruptions can be identified by lack of feedback between components to the manager component. Recovery takes place by recomposing any missing functionality by dynamically identifying, reusing and self-assembling software components. Andrade and de Araújo Macêdo [1] discussed self-healing of unanticipated architectural changes in distributed heterogeneous environments. This paper describes a non-intrusive component-based approach to detect changes by monitoring data gathered by environment devices and to recover by redeploying changes with an updated configuration. Friedrich et al. [10] identified another approach to exception handling in service based processes management system, which could be viewed as change in functionality or architecture of workflow. This can be detected by model-based diagnosis. A model based approach to repair planning is used to recover the original process.

Degraded performance: Yu et al. [7] talk about the self-healing of composite web services by using performance predictions based on a semi-Markov model. An error in a composite web service is detected by monitoring the performance of the web service and can be recovered by reselecting in execution. Moo-Mena et al. [17] present Quality of Service (QoS) parameters and a statistical model approach to self-healing web services in heterogeneous networks. An error is detected when the QoS parameters are degraded.

4. Research challenges

In addition to challenges mentioned in other surveys [13, 15, 18], we identify the following challenges. Self-healing system architecture incorporates the self-healing features during the design phase of software development life cycle to avoid the software faults that may occur in later phases. Estwick [9] addressed design of a self healing software architecture using business rules but with constraints on the architectural style. This work deals with single changes occurring within the architecture. An open opportunity is to design self-healing software architectures to accommodate more than one change occur simultaneously within the architecture.

Assurance of self-healing systems is another research opportunity in self-healing systems. In terms of biology, curing one disease may cause another disease if the diagnosis or the treatment of disease is incorrect. In self-healing systems, the system along with detection and correction of faults, should assure that when an error is detected it is in fact real and the suggested recovery technique heals the error without causing another error. Hrubá et al. [14] used bounded model checking once the data race error was found.

5. Conclusions

Our taxonomy of software faults in self-healing systems shows promise. We observed in most of the categories, software faults were detected by monitoring and evaluation of data or messages. Replication, repair plans, rescue points, recompose, and reselecting techniques are widely used for recovery of systems. Finally, we discussed research opportunities in self-healing architectures and self-healing systems assurance.

Table1. Detection and recovery of faults in self-healing software.

Category	Type of fault	Citation	Detection	Recovery
Errors in COTS	Source code errors	[3]	— Messages passed through connectors are monitored	— Recovery policy from policy repository or reconfiguration of the components
	Integration Incompatibility	[5]	— When exceptions raised by COTS components	— Injection of healing connectors into the system
Failures in servers	Intrusions	[19]	— Monitoring and checking for any violation of security policy	— Proactive recovery
	Other software failures	[6,20]	—	— Rescue points and error virtualization
Process errors	Corrupted kernel data structures	[22]	— Compare the run queue and task list	— Repair plans
	Un-handled exceptions	[12]	— Supervisor monitors for un-handled exceptions consulting with FSM.	— The faulty sequence of the method calls is derived by code instrumentation.
	Memory leaks	[4]	— Monitoring heap threshold	— Poison selected references and reclaim memory
	Buffer overflow	[8]	— Monitor for consequence (buffer overflow)	— Reclaim the leaked memory or restart the system
	Other process errors	[2]	— Monitor online data and performance modeling of the process	— Process replication
Concurrency errors	Race conditions	[14]	— Dynamic analysis (failure in response)	— Additional synchronizing/ Influencing scheduling
	Deadlocks	[23]	—	— Deadlocks can be prevented by control logic using code instrumentation
Degraded functionality	Unanticipated architectural changes	[1]	— Periodic evaluation of data gathered by the environment devices	— Redeploy changes with updated configuration
	Changes in environmental components /missing components	[16]	— Probing (feedback)	— Dynamically recompose components for functionality
	Exception handling in service-based process management system	[10]	— Model-based diagnosis	— Execution of model - based repair plans
Degraded performance	Throughput/response time	[7]	— Inspecting the response time	—
	QoS degradation	[17]	— Monitor data of QoS parameters	— Reselecting service

Acknowledgements

We would like to thank the Center for Advanced Vehicular Systems (CAVS) at Mississippi State University for their support. Also, thanks to Mr. Bradley D. Brazzeal for help using EndNote.

References

[1] S. S. Andrade and R. J. de Araújo Macêdo, "A Non-Intrusive Component-Based Approach for Deploying Unanticipated Self-Management Behaviour," *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009)*. May 2009, pp. 152–161, IEEE.

[2] A. Andrzejak, "Generic Self-Healing via Rejuvenation: Challenges, Status Quo, and Solutions," *Proceedings: 2010 Fourth IEEE International Conference on Self- Adaptive and Self-Organizing Systems Workshop*. Sept. 2010, pp. 239–242.

[3] M. Bisadi and M. Sharifi, "Component-Based Self-Healing via Cellular Adaptation," *Proceedings: The Fifth International Conference on Autonomic and Autonomous Systems ICAS 2009*, Valencia, Spain, Apr. 2009, pp. 75–81, IEEE Computer Society.

[4] M. D. Bond and K. S. McKinley, "Leak pruning," *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2009, pp. 277–288, ACM.

[5] H. Chang, L. Mariani, and M. Pezz´e, "In-field healing of integration problems with COTS components," *Proceedings: 2009 31st International Conference on Software Engineering*. May 2009, pp. 166–176, IEEE.

[6] G. Chen, H. Jin, D. Zou, B. B. Zhou, W. Qiang, and G. Hu, "SHelp: Automatic Self-Healing for Multiple Application Instances in a Virtual Machine Environment," *Proceedings: 2010 IEEE International Conference on Cluster Computing*, Heraklion, Crete, Greece, Sept. 2010, pp. 97–106.

[7] Y. Dai, L. Yang, and B. Zhang, "QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction," *Journal of Computer Science and Technology*, vol. 24, no. 2, Mar. 2009, pp. 250–261.

[8] Y. S. Dai, Y. P. Xiang, Y. F. Li, L. D. Xing, and G. W. Zhang, "Consequence Oriented Self-Healing and Autonomous Diagnosis for Highly Reliable Systems and Software," *IEEE Transactions on Reliability*, vol. 60, no. 2, June 2011, pp. 369–380.

[9] A. C. Estwick, A Business Rules Approach to Self-Healing Software Architecture, doctoral dissertation, The George Washington University, Jan. 2011.

[10] G. Friedrich, M. Fugini, E. Muss, B. Pernici, and G. Tagni, "Exception Handling for Repair in Service-Based Processes," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, March-April 2010, pp. 198–215.

[11] K. Gama and D. Donsez, "A Self-healing Component Sandbox for Untrustworthy Third Party Code Execution," *Component-Based Software Engineering*, vol. 6092 of LNCS, Springer Berlin, 2010, pp. 130–149.

[12] B. Gaudin, E. I. Vassev, P. Nixon, and M. Hinchey, "A Control Theory Based Approach for Self-Healing of Unhandled Runtime Exceptions," *Proceedings of the 8th ACM International Conference on Autonomic Computing*, Karlsruhe, Germany, 2011, pp. 217–220.

[13] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, "Self-Healing Systems—Survey and Synthesis," *Decision Support Systems*, vol. 42, no. 4, 2007, pp. 2164–2185.

[14] V. Hrubá, B. Křena, and T. Vojnar, "Self-healing Assurance Based on Bounded Model Checking," *Computer Aided Systems Theory—EUROCAST 2009*, vol. 5717 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 295–303.

[15] A. D. Keromytis, "Characterizing Self-Healing Software Systems," *Proceedings of the 4th International Conference on Mathematical Methods, Models and Architecture for Computer Networks Security*, St. Petersburg, Russia, Sept. 2007.

[16] J. Martinez and S. Dobson, "Functionality Recomposition for Self-healing," - *Proceedings of the 4th International Conference on Software and Data Technologies*. July 2009, pp. 159–164.

[17] F. Moo-Mena, J. Garcilazo-Ortiz, L. Basto-Díaz, F. Curi- Quintal, S. Medina-Peralta, and F. Alonzo-Canul, "A Diagnosis Module Based on Statistic and QoS Techniques for Self-healing Architectures Supporting WS based Applications," *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC '09*. Oct. 2009, pp. 163–169.

[18] H. Psaier and S. Dustdar, "A Survey on Self- Healing Systems—Approaches and Systems," *Computing*, vol. 87, no. 1, 2010, pp. 43–73.

[19] M. Qiang, Z. Rui-peng, and Y. Xiao-hui, "Design and Implementation of an Intrusion-Tolerant Self-Healing Application Server," *Proceedings: 2010 International Conference on Communications and Intelligence Information Security (ICCIIS)*, Oct. 2010, pp. 92–95.

[20] S. Sidiroglou, O. Laadan, C. Perez, N. Viennot, J. Nieh, and A. D. Keromytis, "ASSURE: Automatic Software Self-healing Using Rescue Points," *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2009, pp. 37–48, ACM.

[21] L. M. Silva, J. Alonso, and J. Torres, "Using Virtualization to Improve Software Rejuvenation," *IEEE Transactions on Computers*, vol. 58, no. 11, Nov. 2009, pp. 1525–1538.

[22] L. Sun, D. K. Nilsson, T. Katori, and T. Nakajima, Online Self-Healing Support for Embedded Systems," *Proceedings of the 12th IEEE International Symposium on Object/Component /Service-Oriented Real-Time Distributed Computing*, Mar. 2009, pp. 283–287.

[23] Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke, "The Theory of Deadlock Avoidance via Discrete Control," *Proceedings of the 36th Annual ACMSIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2009, pp. 252–263, ACM