**Communications and Network, 2016, 8, 158-169**
Published Online August 2016 in SciRes. http://www.scirp.org/journal/cn
http://dx.doi.org/10.4236/cn.2016.83016

Scientific Research Publishing

# An Automated Approach for Software Fault Detection and Recovery

**Amjad A. Hudaib[1], Hussam N. Fakhouri[2]**

[1]Department of Computer Information Systems, The University of Jordan, Amman, Jordan
[2]Department of Computer Science, The University of Jordan, Amman, Jordan
Email: ahdaib@ju.edu.jo, h.fakhouri@ju.edu.jo

Received 27 February 2016; accepted 31 July 2016; published 3 August 2016



## Abstract

**Autonomic software recovery enables software to automatically detect and recover software faults. This feature makes the software to run more efficiently, actively, and reduces the maintenance time and cost. This paper proposes an automated approach for Software Fault Detection and Recovery (SFDR). The SFDR detects the cases if a fault occurs with software components such as component deletion, replacement or modification, and recovers the component to enable the software to continue its intended operation. The SFDR is analyzed and implemented in parallel as a standalone software at the design phase of the target software. The practical applicability of the proposed approach has been tested by implementing an application demonstrating the performance and effectiveness of the SFDR. The experimental results and the comparisons with other works show the effectiveness of the proposed approach.**

## Keywords

**Software Engineering, Autonomic Software Systems, Automatic Recovery, Automatic Diagnosis, Auto Restore**

## 1. Introduction

The wide range and variety of software applications make it difficult for human to monitor, control and administrate software applications and systems. Therefore, the research towards developing autonomic systems that has self-management, self-control and self-healing has increased predominantly in recent years. The development of software applications that can manage themselves, and can respond to the changes in the environment according to goals set by the system administrator has been called the Autonomic Computing Systems (ACS). ACS is inspired by the human autonomic nervous system which regulates vital body functions without the need for con-

**How to cite this paper:** Hudaib, A.A. and Fakhouri, H.N. (2016) An Automated Approach for Software Fault Detection and Recovery. *Communications and Network*, **8**, 158-169. http://dx.doi.org/10.4236/cn.2016.83016

scious human involvement [1].

Many researchers investigated software auto recovery with a goal to integrate the auto recovery mechanism inside the software [2]-[4]. However, if software itself fails to run the auto recover process, then the recovery process will not operate. This paper proposes an automated approach to solve this problem called Software Fault Detection and Recovery (SFDR). The SFDR is analyzed and implemented in parallel as a standalone software at the design phase of the target software. The SFDR detects the cases if a fault occurs with software components such as component deletion, replacement or modification, and recovers the component to enable the software to continue its intended operation. The main functions of SFDR are to automatically diagnose, monitor and recover the component faults in the target software. These autonomic functions reduce the maintenance cost and time, which plays major concerns for the software developer and for software customers.

The applicability of the SFDR has been tested by implementing an application demonstrating the performance and effectiveness of the SFDR. The experimental results and the comparisons with other works show the effectiveness of the proposed approach.

The rest of this paper is organized as follows. In Section 2, we present related work about self-healing achievements in software systems. Section 3 describes the proposed SFDR and its auto detection and recovery mechanism. Section 4 provides experimental results and comparing SFDR with other approaches. Finally, the conclusions and future work are presented in Section 5.

## 2. Related Work

Many researchers dedicated their efforts to the field of automated software recovery. Dashofy, Philip Koopman proposed a taxonomy for the problem space for self-healing systems including system completeness, design context, fault models, and system responses [2] [3]. Jiang, *et al.* presented a generic modeling framework to facilitate self-healing development system software and introduced a model-based approach which is used to categorize software failures and specify their dispositions at the model level [5]. George *et al.* was inspired by biological strategies to develop cell-based programming model that has robust and scalable features for software systems self-healing [6]. Fuad *et al.* presented a new method that use matching a fault scenario for finding self-healing actions by established fault models [7].

Michael described an approach for distributed software architecture to design a concurrent and robust self-healing component [4]. Montani *et al.* described a Case based reasoning (CBR) approach that gave capabilities of self-healing to distributed software systems, by means of real world application experimental results [8]. Park *et al.* proposed a self-healing mechanism that diagnose, heal and monitor its internal error by contextual information and self-awareness [9]. Naftaly M. worked in complex systems and mainly for heterogeneous distributed software. And he put the conditions of self-repair and self-healing for those systems [10].

Diaconescu A. worked on a system for component-based software systems to have self-optimization, and self-healing and to enable dynamic adaptation in those systems [11]. Brumley *et al.* introduced software systems self-healing architecture where programmers detect exploits, self-diagnose and self-monitor and the main cause of the vulnerability, self-recover from attacks and self-harden against future attacks [12]. Dinkel *et al.* 2008 presented a novel approach for distributed embedded systems self-healing that contain black-box application software [13]. Katti *et al.* used Gossip protocols that are inherently fault-tolerant and scalable to compare two novel failure detections and consensus algorithms [14].

Hervé Chang proposed to use Exception Handlers for Healing Component-Based Systems that heal failures activated by exceptions raised in the OTS components actually deployed in the system [15].

Angelos D. Keromytis focused on systems that effect structural changes to the software under protection, as opposed to block-level system reconfiguration provided as a first attempt to characterize self-healing software systems by surveying some of the existing work in the field [16]. Goutam S. illustrated critical points of the emergent research topic of Self-Healing Software System, described various issues on designing a self-healing software application system that relies on-the-fly error detection to repair web applications or service agent code data [17]. Edward presented a computational geometry technique and a supporting tool to tackle the problem of timely fault detection during the execution of a software application [18]. Harald and Schahram focused on the self-healing branch of the research and gave an overview of the current existing approaches and their characteristics [19]. Dabrowski *et al.* during communication failure he used architectural models to characterize how architecture, consistency-maintenance mechanism, topology, and failure-recovery strategy each contribute to self-

healing and they proposed the contribution of individual self-healing and strategies to quantify [20].

The comparisons between the different auto recovery researches according to the method used and the resulted tool or system are presented in **Table 1**.

## 3. Software Fault Detection and Recovery (SFDR) Approach

To explain the proposed approach we will start by explaining SFDR development life cycle, then present the SFDR main components, and show the of the recovery mechanism of SFDR.

### 3.1. SFDR Development and Life Cycle

SFDR is developed during the software life cycle, its model suggest the integration and developing during the software life cycle by building companion software to it (SFDR) for the targeted software that is being built. SFDR use the components of the implementation process as an input for the requirement analysis and for the SFDR development. Analyzing these components provide a wide strong knowledge base for the auto recovery of the system that leads it through the diagnosis and contain all the required information about the components of the software.

Developing SFDR aims to perform a healing process for the targeted software. Building SFDR mainly relay on the nature and structure of the released components of the targeted software. The SFDR life cycle is shown in **Figure 1** which illustrates that the SFDR knowledge will be based on the design and implementation processes. While SFDR life cycle is shown in **Figure 1**.

• Requirement Specification: Define the requirement specification of the target software.

• Analysis: Analyze the flow charts, diagrams, and components of the target software to get information about the components such as (installation folder, size of files, types, name, has, date created, manufacturer).

• Design: Design the target software and draw its diagrams such as class diagram and use case diagrams. These diagrams are used in building the SFDR.

**Table 1.** Classify different auto recovery researches according to the method used and the resulted tool or system.

| Author/s | Methodology | Tools |
|---|---|---|
| **Tom *et al.* 2009** [21] | Based on statistical information retrieved from an instrumented version of the program under analysis. | Zoltar tool that adopts a technique to localize software faults. |
| **Alessandra Gorla, 2009** [22] | Automatically locates the faults underlying the failures, derive assertions to effectively detect functional failures, and identify sequences of actions alternative to the failing sequence to bring the system back to an acceptable behavior. | Techniques to build software systems that can automatically heal such failures. |
| **Ammo Krueger, 2010** [23] | Intercepts requests and decides on a per-token basis whether a token requires automatic "healing". | Protocol-aware reverse HTTP proxy TokDoc (the token doctor) an intelligent mangling technique, which, based on the decision of previously trained anomaly detectors, replaces suspicious parts in requests by benign data the system has seen in the past. |
| **Jens Ehlers, 2011** [24] | Incorporating architectural information about the diagnosed software system, Time series analysis of operation response times is employed for anomaly localization. Comprising quality of service data, such as response times, resource utilization, and anomaly scores, OCL-based monitoring rules specify the adaptive monitoring coverage. | An approach for localizing performance anomalies in software systems employing self-adaptive monitoring, implemented as part of the Kieker monitoring and analysis framework. |
| **Boris Koldehofe, 2013** [25] | Eliminates the need for persistent checkpoints rollback-recovery by allowing for recovery from multiple simultaneous operator failures. | Event processing model to determine save oints and algorithms for their coordination in a distributed operator network. |
| **Thorat *et al.* 2015** [26] | Rapid recovery (RR) mechanism to perform an immediate link recovery at the switch level without overburdening the controller. | Self-healing SDN framework which can optimize the recovery by applying autonomic principles and analytical model for calculating the failure recovery time and the backup flow rules required for recovery. |
| **Katti *et al.* 2015** [14] | Algorithms are based on Gossip protocols and are inherently fault-tolerant and scalable. | Compares two novel failure detection and consensus algorithms. |

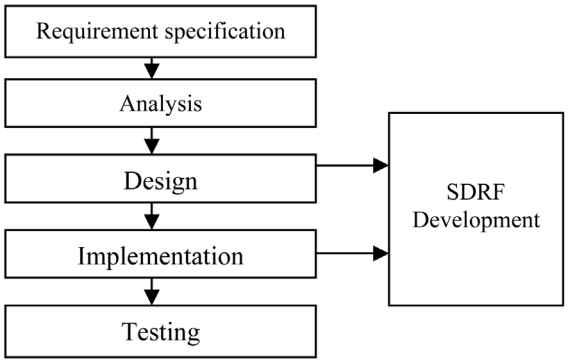


**Figure 1.** SFDR development.

• Implementation: Implement the target software. The code of the target software is used by SFDR to define the implemented components of the software and to build how to diagnose, monitor, and to recover these components.

• Testing: Test the target software.

• SFDR Development: The development process of SFDR includes the following:

-Building database: This step is done by storing all of the information about the software component (*i.e.* size of files, types, name, has, date created, manufacturer) in the database records that will be used in the diagnosis of the software.

-Testing the SFDR: This step aims to test SFDR before release to avoid any mistakes in the functionality or performance.

-Distributing SFDR: SFDR will be distributed as a separate package from the software and its function is to detect faults and to recover them.

### 3.2. SFDR Components

SFDR is developed at the time of developing the targeted software. The SFDR database and SFDR software, the way these are developed will have the functionality and capability to guide SFDR in the diagnosis and healing steps. SFDR healing mechanism starts when the user install it in the client computer that has the missing functionality i.e. software fails to run, a suspected behavior, change in the performance or the excepted results of the software. It will have the capability to diagnose, and heal the software. SFDR main components are shown in **Figure 2**.

When the user detects a change in the software he request SFDR, SFDR starts the auto detection and recovery functionality as soon as the user run it; SFDR has a build in database that has information about the software, structure and components. SFDR starts by Analyzing then comparing the components next to diagnosis then Healing and finally Reporting and providing notification.

•Analyzing: as soon as SFDR run, it will start the analyzing process; it gathers information about the installed software and prepares it for the next step. The analysis of the software components includes gathering information about the files such as name, date of creation, date of last modified, hash of the file, and size.

•Comparing: the comparison step aims to compare the information that we have got from the analysis to the data stored in the database.

•Diagnosis: here it decides whether the system needs to be healed or it is in good health. In this step a solution to the system will be required and suggested if the system is infected or in defect state. SFDR provides Diagnosis for the software in an automated manner and for many cases such as software failure to run. SFDR corrects the fault by reversing the software component to its original state and this heals the software. According to previous information defined by the preprocess phase in the system, SFDR can diagnose the following cases: Deletion of component that cause the system to fail to run, change of component by external factor either human or non-human factor, original component replace, or addition of external component to the software folder.

•Connect to server: after determining the component that is needed to be compensated or replaced. SFDR establishes a connection to the server to get original component.

•Download and install components: SFDR will download and install it to make sure that the software original

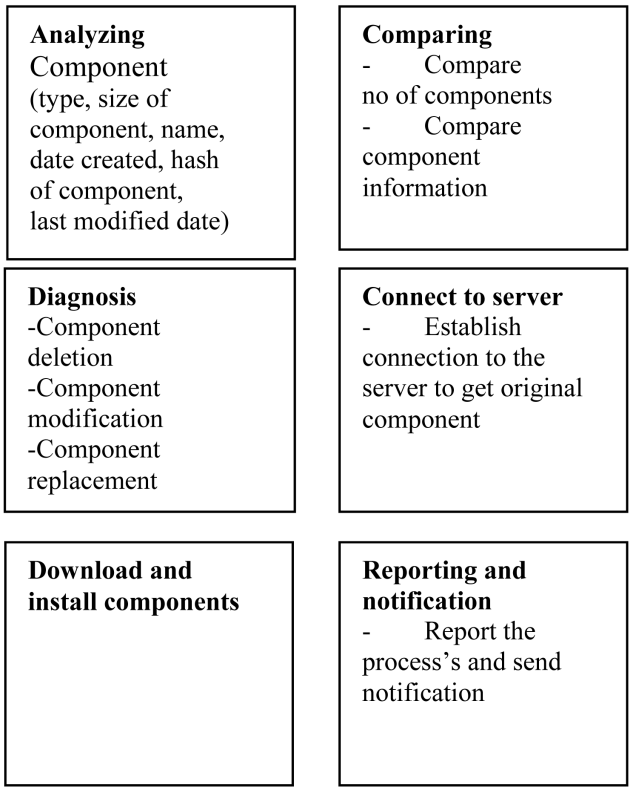


**Figure 2.** SFDR components.

components with no modification or defect, using different procedures for different cases that the software may face *i.e.* replacing infected component, compensating a deleted component.

•Reporting and notification: The unexpected activities and the healing activities will be reported to the central database of the software and a notification will be sent to the user and the manufacturer. Analyzing the reports sent by SFDR will help to determine the solution and distributing it through SFDR makes it to exhibit the Prevention feature of software reengineering process. For example, if the reason that caused the software fault is illegal access and deleting of a component. Then analyzing the reasons and discovering the cause will make the software owner take several procedures to prevent further cause. This feature makes the software more easily corrected, adapted, and enhanced and because of this the report is sent to the software manufacturer then they will take the analysis results into consideration in designing better software that avoids any leak that the report have shown. Then the software engineers will consider the results in their future software design.

### 3.3. The SFDR Auto Detection and Recovery Mechanism

When the user detects a defect or any type of error in the application software installed on his computer, he download the SFDR that is specified for the software that he wants to fix. As soon as he run SFDR it starts to work by first Counting the number of components of the software after that SFDR analyze the software component to get component information: type, size of component, name, date created , hash of component , last modified date.

**Figure 3** shows the flow chart of SFDR auto detection and recovery Mechanism. SFDR automatically modifies the application to detect and recover the fault. The changes that have happen will be reported to the software developer. If the error happened often then the recorded information will guide the software developer for the reason by analyzing the stored information or the affected component. If the reason is from the software itself then it will be recorded as new detected defect in the software then the developer will contact all other installed software to run SFDR so that it correct similar defect and perform the suitable replacement to enhance the software.

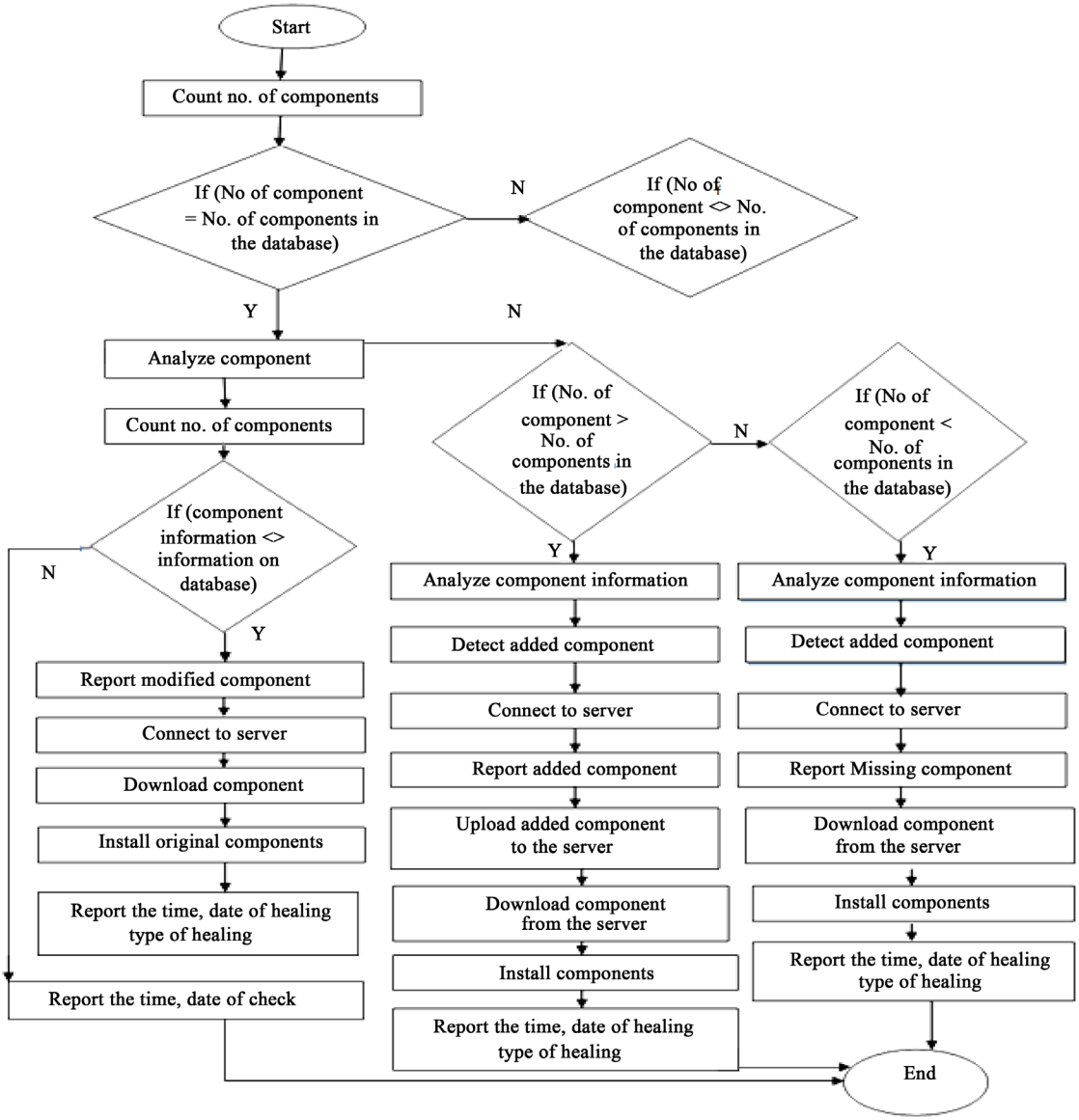


**Figure 3.** SFDR auto detection and recovery mechanism.

## 4. Evaluation of the Proposed Approach

To evaluate SFDR effectiveness the following cases are used:

I. The normal case; the normal case is defined by the case that no change has been made to the software component after installation.

II. The deletion case. Where one component/s has/have been deleted from the software installed folder.

III. The replacement case; where a component has been replaced by another one with the same name and extension.

IV. The modified case; in this case a file has been modified manually.

### 4.1. Evaluation Based the Ability of SFDR to Heal Different Cases

#### 4.1.1. Case I: Normal: No Change Has Been Done to the Software

SFDR compares the software component with the information that it has on the database. In case of no changes has been found, no action will be taken, the set of procedures performed by the SFDR are shown in **Figure 4**.

1. Decrement the number of software components for the targeted software
2. If (number of the examined software components = number of software components stored in the database of the system)
3. Loop
4. Read and analyze components info (type, size of component, name, date created , hash of component , last modified date)
5. if (size of <> original size or date created <> original date created or hash of component <> original hash or last modified date <> last modified date or name of <> original name )
6. do healing process
7. Else
8. generate healthy software report
9. end if
10. End loop
11. End if

**Figure 4.** High level SFDR response algorithm for normal case scenario.

#### 4.1.2. Case II: Deletion of Component

SFDR detected the deleted component and compensate it. A set of procedures performed by the SFDR are shown in **Figure 5**.

#### 4.1.3. Case III: Replacement with Similar Component

For testing this case we have replaced a component that has similar file name and similar extension found in one of the software installed folder. SFDR responded to this case by deleting the affected component and restoring the original component. The set of procedures performed by the SFDR are shown in **Figure 6**.

#### 4.1.4. Case IV: Modifying a Component

SFDR responds to this case by deleting the modified component and restoring the original component. The set of steps performed by the SFDR are shown in **Figure 7**.

### 4.2. Average Recovery Time Measurement

To measure the performance of the SFDR; we will calculate the Average Recovery Time (ART) for SDFR when a component fault occurs, taking into account the software size and the defect component size. Equation 1 is used to calculate the ART.

$$\mathrm{ART}(\mathrm{SFDR}) = \mathrm{SFDR\_DIT} + \mathrm{DET} + \mathrm{RT} \tag{1}$$

where SFDR_DIT is SFDR download and installation time; DET is SFDR Detection Time; RT is Recovery Time which include (component replacement time).

To compare this performance, we calculate the average recovery time for software re-installation ART(SW Re_inst) when a component fault occurs, taking into account the software size and the defect component size. Equation 2 is used to calculate the ART.

$$\mathrm{ART}(\mathrm{SW\ Re\_inst}) = \mathrm{UT} + \mathrm{SDT} + \mathrm{SW\_inst} \tag{2}$$

where UT is the software Un-installation time; SDT is software download time; SW_inst software install time.

To compare the SFDR performance with software reinstallation, the experiments has been performed in the following technical environment. The internet download speed of 2.2 MB and upload speed of 1.46 using ping of 50 ms measured using [27], Windows 7 operating system, RAM of 4 G, Processor Intel core™ i3 -311M , 2.4 GHz.

1. Decrement the number of software components for the targeted software
2. If (number of the examined software components < number of software components stored in the database of the system)
3. Analyze components (type, size of component, name, date created, hash of component, last modified date)
4. Determine the missing component
5. Store information about the missing component in database
6. Download missing component
7. Install the missing component on its correct folder.
8. Send report to the server
9. Display report on to the user
10. End if

**Figure 5.** High level SFDR response algorithm for deletion case scenario.

1. Decrement the number of software components for the targeted software
2. If (number of the examined software components = number of software components stored in the database of the system)
3. Loop
4. Analyze components (type, size of component, name, date created , hash of component, last modified date)
12. if (size of <> original size or date created <> original date created or hash of component <> original hash or last modified date <> last modified date or name of <> original name )
5. Determine the replaced component
6. Store information (name, type, size, has) about the replaced component in database.
7. Store the replaced component in different directory
8. Download original component
9. Install the original component on its correct location
10. Send report to the server
11. Upload the replaced component to the server
12. Display report
13. End if
14. End loop
15. End if

**Figure 6.** High level SFDR response algorithm for replacement with similar component case scenario.

We tested several cases to make defect in different component size in the system to measure the average recovery time for SFDR,

**Table 2** shows the experimental results of the SFDR and the software re-installation time. The experimental results show that the average recovery time for SFDR is lower than the average recovery time for the re-install times because the recovery process involve replacing the defect component which size is always lower than the whole software. The comparison results are represented in **Figure 8**.

### 4.3. Comparison of SFDR with Other Approaches

**Table 3** shows the comparison between SFDR and other approaches (Microsoft Windows System Restore,

1. Decrement the number of software components for the targeted software
2. If (number of the examined software components = number of software components stored in the database of the system)
3. Loop
4. Read and analyze components info (type, size of component, name, date created, hash of component , last modified date)
5. if (size of <> original size or date created <> original date created or hash of component <> original hash or last modified date <> last modified date or name of <> original name )
6. detect the modified component
7. Store component information (name, type, size, has) about the replaced component in Db
8. Store the replaced component in different directory
9. Download original component
10. Install the original component on its correct location
11. Send report to the server
12. Upload the replaced component to the server
13. Display report on to the user
14. End if
15. End loop
16. end if

**Figure 7.** High level SFDR response algorithm for modifying a component case scenario.

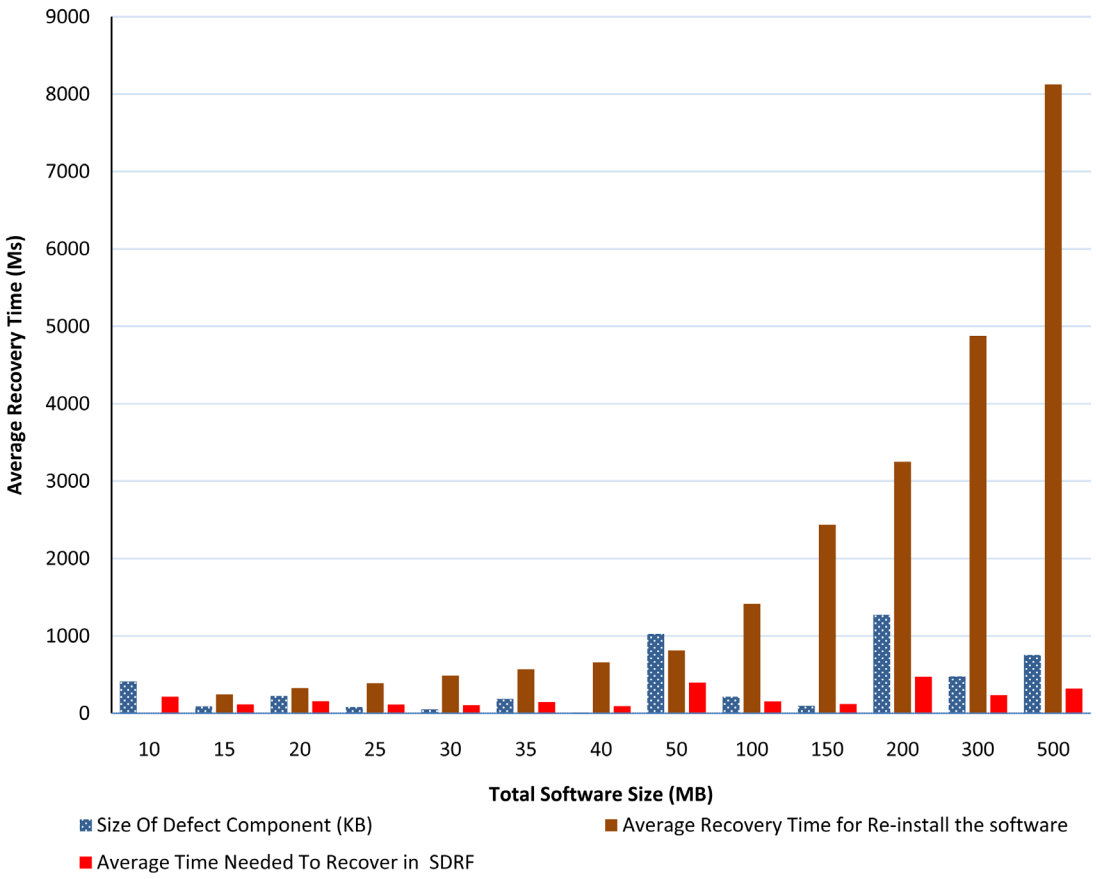


**Figure 8.** Comparing average recovery time for SFDR VS software reinstalling method using different software and different defect component size.

**Table 2.** Comparing average recovery time for SDRF VS average recovery time for reinstalling the software according to software size and the defect component size.

| Total Software Size (MB) | Size Of Defect Component (KB) | Average Recovery Time for Re-install the software (MS) | Average Recovery Time in SFDR (MS) |
|---|---|---|---|
| 10 | 412 | 162.5 | 213.6 |
| 15 | 91 | 243.75 | 117.3 |
| 20 | 225 | 325 | 157.5 |
| 25 | 291 | 406 | 177 |
| 30 | 617 | 487 | 275 |
| 35 | 188 | 568.75 | 146.4 |
| 40 | 566 | 633 | 259.8 |
| 45 | 171 | 698.75 | 141.3 |
| 50 | 588 | 820.6 | 266.4 |
| 100 | 463 | 1722.5 | 133.8 |
| 300 | 478 | 4875 | 233.4 |
| 500 | 756 | 8125 | 316.8 |

**Table 3.** Comparison between MS windows system restore, ASSURE, exception handlers and SFDR.

| Approach/methodology/system Feature | MS Windows (System Restore. (Ed Bott, 2009) | ASURE (Stelios Sidiroglou, 2009) | Exception Handlers for Healing (Herve Chang, 2013) | The SFDR Approach |
|---|---|---|---|---|
| **Recover error resulting from deleting software component** | Yes | No | No | Yes |
| **Recover and Replaced component that has same functionality** | No | No | Yes | Yes |
| **Fault recovery** | No | Yes | No | Yes |
| **Generate reports of the default diagnosis and the recovery process** | Yes | No | No | Yes |
| **Store the affected component for future analysis** | No | No | No | Yes |
| **Approach of repairing** | System Restore | Dynamically patches the running production application to self-checkpoint at the rescue point | Heal fault s activated by exceptions raised in the OTS components actually deployed in the system | Automatically including Comparing, analyzing, diagnosing and recovery to return the software to its original status of the manufacturer |
| **State of the recovery** | To a specified restore point | To a specified rescue point. | Original state of the manufacturer | To the manufacturer state either the original or with updates |
| **Knowing the structure of the software and its component** | Yes | Yes | Ye s | Yes |
| **Recovery time** | After release | After release | After release | After release |
| **Building time** | During the development of the software | During the development of the software | During the development of the software | During the development of the software |

ASURE, and Exception Handlers) based on set of features such as Recover error resulting from deleting software component, Recover and Replaced component that has same functionality, Fault recovery, Generate reports of the diagnosis of the problem and the healing process, Store the affected component for future analysis, Approach of repairing, State of the recovery, Knowing the structure of the software and its component, Recovery time, and the building time.

## 5. Conclusions and Future Work

This paper presented Automated Approach for Software Fault Detection and Recovery (SFDR). SFDR has the required information about the software that is needed to diagnose and recover the software from fault. A set of cases have been developed to test SFDR. The results of the experiments with different cases illustrated the ability and efficiency of SFDR to diagnose and recover software even with cases of software fails to run.

For the future work, the proposed approach can be enhanced by applying data mining techniques to make SFDR more efficient in fault recovery.

## References


[1] Huebscher, C. and McCann, A. (2008) A Survey of Autonomic Computing Degrees, Models, and Applications. *ACM Computer Survey*, **40**, Article No. 7. http://dx.doi.org/10.1145/1380584.1380585

[2] Dashofy, M., André, H. and Richard, T. (2002) Towards Architecture-Based Self-Healing Systems. WOSS'02, Charleston. http://dx.doi.org/10.1145/582128.582133

[3] Hudaib, A., Al-Zaghoul, F., Saadeh, M. and Saadeh, H. (2015) ADTEM-Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics. *Journal of Software Engineering and Applications*, **8**, 201.

[4] Michael, S. (2005) Self-Healing Component in Robust Software Architecture for Concurrent and Distributed Systems. *Science of Computer Programming*, **57**, 27-44. http://dx.doi.org/10.1016/j.scico.2004.10.003

[5] Jiang, M., Zhang, J., Raymer, D. and Strassner, J. (2007) A Modeling Framework for Self-Healing Software Systems. *Lecture Notes in Computer Science*, **7003**, 61-68.

[6] Selvin, G., David, E. and Lance, D. (2002) A Biologically Inspired Programming Model for Self-Healing Systems. *WOSS*'02 *Proceedings of the First Workshop on Self-Healing Systems*, Charleston, November 2002, 102-104. http://dx.doi.org/10.1145/582128.582149

[7] Fuad, M., Deb, D. and Baek, J. (2011) Self-Healing by Means of Runtime Execution Profiling. *Proceedings of* 14*th International Conference on Computer and Information Technology* (*ICCIT* 2011), Dhaka, 22-24 December 2011, 202-207. http://dx.doi.org/10.1109/iccitechn.2011.6164784

[8] Montani, S. and Cosimo, A. (2008) Achieving Self-Healing in Service Delivery Software Systems by Means of Case-Based Reasoning. *Applied Intelligence*, **28**, 139-152. http://dx.doi.org/10.1007/s10489-007-0047-1

[9] Park, J., Youn, H., Lee, J. and Lee, E. (2009) Automatic Generation Techniques of a Resource Monitor Based on Deployment Diagram. *ICHIT*'09 *Proceedings of the* 2009 *International Conference on Hybrid Information Technology*, New York, 2009, 189-192.

[10] Naftaly, M. (2003) On Conditions for Self-Healing in Distributed Software Systems. *Proceedings of the Autonomic Computing Workshop*, 25 June 2003, 86-92. http://dx.doi.org/10.1109/ACW.2003.1210208

[11] Diaconescu, A. (2003) A Framework for Using Component Redundancy for Self-Adapting and Self-Optimizing Component-Based Enterprise Systems. *Proceeding OOPSLA*'03 *Companion of the* 18*th annual ACM SIGPLAN Conference on Object-Oriented Programming*, *Systems*, *Languages*, *and Applications* (*OOPSLA*'03), New York, 2003, 390-391.

[12] Brumley, D., Newsome, J. and Song, D. (2007) Sting: An End-to-End Self-Healing System for Defending against Internet Worms Book. In: Christodorescu, M., Jha, S., Maughn, D., Song, D. and Wang, C., Eds., *Malware Detection and Defense*, *Advances in Information Security*, Vol. 27, Springer, United States, 147-170.

[13] Dinkel, M. (2008) A Novel IT-Architecture for Self-Management in Distributed Embedded Systems. PhD Thesis, TU Munich.

[14] Katti, A., Fatta, G. and Naughton, T. (2015) Scalable and Fault Tolerant Failure Detection and Consensus. *EuroMPI*'15 *Proceedings of the* 22*nd European MPI Users*' *Group Meeting*, Bordeaux, 21-23 September 2015, Article No. 13. http://dx.doi.org/10.1145/2802658.2802660

[15] Hervé, C., Leonardo, M. and Mauro, P. (2013) Exception Handlers for Healing Component-Based Systems. *ACM Transactions on Software Engineering and Methodology* (*TOSEM*), **22**, 1-40.

[16] Angelos, K. (2007) Characterizing Self-Healing Software Systems, Computer Network Security of the Series Commu-

nications. *Computer and Information Science*, **1**, 22-33.

[17] Goutam, S. (2007) Software—Implemented Self-Healing System. *CLEI Electronic Journal*, **10**, 5.

[18] Edward, S., Kevin, L., Maxim, S., Chris, R. and Spiros, M. (2010) On the Use of Computational Geometry to Detect Software Faults at Runtime. *Proceedings of the 7th International Conference on Autonomic Computing*, Washington DC, 7-11 June 2010, 109-118.

[19] Harald, P. and Schahram, D. (2011) A Survey on Self-Healing Systems: Approaches and Systems. *Computing*, **91**, 43-73.

[20] Dabrowski, C. and Mills, K. (2002) Understanding Self-Healing in Service-Discovery Systems. *WOSS*'02 *Proceedings of the First Workshop on Self-Healing Systems*, Charleston, 18-19 November 2002, 15-20.

[21] Tom, J., Arjan, R.A. and van Gemund, J.C. (2009) Zoltar: A Spectrum-Based Fault Localization Tool. SINTER'09, Amsterdam.

[22] Alessandra, G., Mauro, P. and Jochen, W. (2009) Achieving Cost-Effective SOFTWARE Reliability through Self-Healing. *Computing and Informatics*, **29**, 93-115.

[23] Ammo, K., Christian, G., Konrad, R. and Pavel, L. (2010) TokDoc: A Self-Healing Web Application Firewall. SAC'10, Sierre. http://dx.doi.org/10.1145/1774088.1774480

[24] Ehlers, J., André, H., Jan, W. and Wilhelm, H. (2011) Self-Adaptive Software System Monitoring for Performance Anomaly Localization. ICAC'11, Karlsruhe.

[25] Boris, K., Ruben, M., Umakishore, R. and Marco, R. (2013) Recovery without Checkpoints in Distributed Event Processing Systems. DEBS'13, Arlington. http://dx.doi.org/10.1145/2488222.2488259

[26] Thorat, P., Raza, S.M., Nguyen, D.T., Hyunseung, G., Choo, I. and Kim, D.S. (2015) Optimized Self-Healing Framework for Software Defined Networks. *IMCOM*'15 *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, Article No. 7. http://dx.doi.org/10.1145/2701126.2701235

[27] http://www.speedtest.net/