Analysis of Current Trends in Software Aging: A Literature Survey

Tajmilur Rahman[1], Joshua Nwokeji[1] & Tejas Veeraganti Manjunath[1]

[1] Gannon University, Erie, PA, United States

Correspondence: Tajmilur Rahman, Department of Computer and Information Science, Gannon University, Erie, PA, 16541, USA. E-mail: rahman007@gannon.edu

Received: July 4, 2022 Accepted: August 17, 2022 Online Published: August 22, 2022

doi:10.5539/cis.v15n4p19 URL: https://doi.org/10.5539/cis.v15n4p19

Abstract

Software aging and architecture degradation are important areas in software quality assurance. Existing research in these areas has developed mitigation strategies for software aging, other researchers have analyzed strategies for identifying software aging. Regarding architectural degradation, current studies have designed techniques for reducing degradation. However, there appears to be a paucity of studies on the causes of software aging and architectural degradation. Insight into the causes of software aging and architectural degradation can provide a critical perspective and further strengthen the research endeavors on prevention techniques. Using recursive literature review (RLR) and Bootstrapping techniques, this research identifies and analyzes the causes of software aging and architecture degradation in software systems. We found that besides many other causes, architectural degradation is one of the key reasons that cause software aging and acts as a barrier to the sustainability of software architecture.

Keywords: survey, software aging, architectural degradation, architectural drift

1. Introduction

Software failure is an undesirable quality of any software, especially from the end-users perspective, it threatens the reliability, availability, and ability of software to achieve the intended business objectives (Saraswat, 2008). Software aging is a phenomenon that contributes to software failures (Torquato, 2018). Software aging is defined as a deterioration in the state of a software resulting from the gradual accumulation of errors, naturally occurring aging-related bugs (ARBs) e.g., data corruption, memory bloating, computational errors, and resource depletion are the major causes of software aging (Castelli, 2001) (Cotroneo & Iannillo, 2020a).

In order to enhance reliability and availability, software engineering scholars have studied and proffered solutions to the many other factors that contribute to software failure. Some examples of these factors are bugs, hardware errors, and human errors (Yakhchi, 2015). However, because of the organic and subtle depletion that characterizes as ARBs, software degradation can be difficult to identify during testing (Cotroneo & Iannillo, 2020).

Indeed there has been an increasingly scholarly effort in the study of software aging, this is easily seen in the number of available peer-review publications. However, current efforts focus on approaches for predicting and detecting software aging. For instance, some authors (Liu & Meng, 2019) (Liu & Tan, 2019) have applied artificial intelligence and machine learning models to predict software aging. A growing number of studies focus on understanding the impact of software aging on the performance, availability, and reliability of computing systems. Some examples include the use of edge computing to analyze the impact of software aging on cyber-physical systems (Andrade & Machida, 2019) and the use of stress testing to investigate how software aging impacts the availability and reliability of Android applications (Cotroneo & Iannillo, 2020). Other scholarly efforts on software aging include the proposal for SWARE (Torquato, 2018). The main objective of SWARE is to provide an experimentation framework for investigating software aging symptoms and the effectiveness of rejuvenation' as a technique for preventing software aging.

While existing scholarly work in software aging makes important contributions to knowledge, there is a need to provide insight into the trends in the research and practice of software aging. Such insight would be very helpful in articulating and defining future research direction, and in addition, provides critical perspectives that could inform practice. It is possible that new computing technologies such as machine learning and artificial intelligence which are now integrated into software development may have changed the causes and other trends in software aging. Therefore, a continuous review of literature is necessary to understand how and if the trends in software aging have changed, the factors that are causing such change in trends, and how the new trends can be leveraged to improve research and practice.

In this paper, we use bootstrapping and recursive literature review methodologies to articulate critical issues and trends in software aging that are often neglected in existing research. We reviewed a total of 236 papers that met our search criteria to identify: current problems and challenges in software aging and causes of software aging. More so, we report on the citation trends, type, and rank of publication venue as well as the publication year. To meet the aim of our study, we set up the following research questions:

- **RQ1** - What are the publication trends in software aging in terms of rank, type, and year of publication?
- **RQ2** - What factors contribute to software aging and architectural degradation in software?
- **RQ3** - Is there a relationship between architecture degradation and software aging?
- **RQ4** - What are the problems or challenges in the area of software aging?

2. Study Significance

Insight into the critical issues covered in these research questions would be imperative to the research and practice of software aging for many reasons. First, understanding the factors that contribute to software aging would help design effective solutions for software aging. Moreover, such understanding could inform the development of preemptive strategies for software aging at the pre-implementation phase of SDLC i.e., during planning and requirements analysis. This would introduce a critical shift in the current approach to solving software aging problems i.e., solutions are designed post-implementation and deployment. This approach is reactive, costly, and inefficient. Secondly, it has been hypothesized that software architecture degradation contributes to software aging (De Silva & Balasubramaniam, 2012) (Heinkens & Graaf), however, the relationship between software aging and architecture degradation has not been adequately studied in literature and are often neglected. Our study fills this knowledge gap by providing insight into if and how software architecture degradation contributes to software aging.

The rest of the paper is outlined as follows. Section 3 gives a background of the study, Section 4 gives an overview of the related works in the domain of our study, Section 5 describes the methodology that we apply for this study, and Section 6 discusses the current literature from different aspects. Section 7 describes what we have observed and answering the research questions defined by us in Section 1 followed by a conclusion and future work in Section 8.

3. Background

David Parnas was the first to identify the phenomenon of Software Aging and published a paper proposing some best practices to prevent aging or at least increase the life span of a software system (Parnas, 1994). He stated – “Programs, like people, get old. We can’t prevent aging, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable.”

The phenomenon of software aging has grown tremendously in both academic and industrial areas. Its causes and effects have been a subject of investigation, detection, and research. A few of the primary reasons why Software aging is caused are memory bloating, memory leakage (Cassidy, 2002) (Cotroneo, 2016) (Cotroneo & Iannillo, 2020), as well as data corruption (Vaidyanathan & Gross, 2003) (Cotroneo, 2010), and unreleased file locks (Garg, 1998) (Vaidyanathan & Trivedi, 2005).

The biggest cause of software failure is unplanned system outage as mentioned by Castelli et al. (Castelli, 2001). Hardware failure does not cause a significant impact on the development of software failures. A number of factors such as numerical error accumulation, data corruption, and unlimited resource consumption can play an enormous role in the increasing failure rate and it causes software failure. The software can never be fully deemed bug-free. Aging arises due to the complexity in the software that is never free of errors. The development of software tends to be managed by the need to meet the deadlines of the release rather than to ensure that the software is reliable. Designing software that can be immune to aging is difficult (Castelli, 2001). The rate of aging can differ from software to software.

Android OS Architecture consists of a stack of software components that can be divided into Linux kernel (ex. display driver, binder driver .etc), Android Libraries (ex. software manager, media framework .etc), Android Runtime (ex. Dalvik Virtual Machine .etc), Application Framework (high-level services such as memory utilization and consumption, and garbage collection .etc), and Application or Task (stock application or user-installed applications). According to the paper “Software Aging Analysis of the Android Mobile OS” by Domenico Cotroneo et al (Cotroneo & Iannillo, 2020). Mobile devices utilizing the Android OS are significantly complex, having rich and heavily customizable features. Thus, they are prone to software reliability and performance issues (Cotroneo, 2016). The devices are degraded in responsiveness over a period of time and eventually fail. This degradation in the device is categorized as Software aging in Android mobile OS.

The authors considered different metrics such as memory, storage utilization, garbage collection, and tasks performed by the application, that can be correlated with one key metric, “Launch Time”. Launch time is the time between the user interaction to the start of an application and the appearance of the user interface of the application. An increasing trend in launch time was exhibited by Android applications that were triggered by experiments that showcased software aging effects. The type of workload processed on the device contributes to the amount of software aging observed. An increase in memory consumption causing memory degradation was directly proportional to the degradation of the launch time for specific processes of the android system. Similarly, garbage collection time and the launch time, which is the time spent on the Android Runtime to reclaim heap memory from Android processes, are highly correlated with each other.

According to task analysis, certain components like the activity manager in the Android OS frequently caused a trend in terms of virtual memory utilization and CPU, which showed behavioral deviations in the presence of software aging. Android processes that are part of the Android AOSP (Android Open Source Project) codebase, are used by a variety of devices and vendors. The focus of vendor customization is on device drivers and apps, with very minor changes to the underlying AOSP operations. As a result, software aging-related problems that are identical are likely to impact devices from a variety of vendors. Vendor customizations may, however, cause additional software aging problems. From the above observations and considering a transitive relationship, we can identify that architecture also plays a role in software aging.

The authors Erik Whiting and Sharon Andrews of the paper "Drift and Erosion in Software Architecture: Summary and Prevention Strategies" talk about how architectural degradation is caused due to system complexity increasing gradually over time and changes in the requirement set. Architectural degradation can be classified into two types: architectural drift and architectural erosion (Whiting & Andrews, 2020). Architectural drift is the gap between the planned and current architecture of the software system that does not violate the intended architecture. On the other hand, software erosion is similar to software drift but violates the intended design. They say that both forms of these degradations threaten the integrity of the system's architecture, hence the project and team as a whole. Software that has undergone architectural erosion or decay may be of less quality, more complex, and more difficult to maintain. As the software application receives oncoming changes, it becomes increasingly difficult to comprehend the software architecture that was originally intended (Whiting & Andrews, 2020).

Gurp et al. discussed in the paper "Design erosion: problems and causes" how erosion affects a software system in the evolution of its design (Gurp & Bosch, 2002). They found that despite being cautious about the design, software erosion was caused over a period of time, and the only solution was redesigning it from scratch. They have also shown how design decisions and requirements changes can affect the software system, which in turn also changes the architecture, resulting in erosion.

According to Gurp et al. architectural decisions that address architecturally significant requirements are referred to as design decisions. The type of application, the distribution of the system, the architectural styles to be employed, and how the architecture should be documented and evaluated are all architectural design decisions. The non-functional aspects of a system are influenced and impacted by architectural decisions. Each architectural decision refers to a specific, architecturally significant design problem that has several alternative solutions. An architectural decision documents the conclusion of a conscious, typically collaborative option selection process and provides a design explanation for the decision. Gurp and Bosch considered an example of a simulator of a bank machine that replicates the functionality of an Automated Teller Machine (ATM).

In the first version, a simple work of the system with the basic requirements of an ATM was considered. It was a compact version but with low maintainability, design, and flexibility, it was not a very ideal system. Later on, four updates were made to the initial version to address these issues. To achieve this, they changed the program structure by adding new classes, moving around blocks of code, and other optimization techniques. The design of version five had the same functionality as the first version, but it was much more complex and heavy. The newly added code structure improved the flexibility but compromised the maintainability of the code by making it harder to understand for the developers. The final version's features are the outcome of the design decisions made in the previous versions, such as adding GUI etc. The options were no longer considered ideal for version five due to the changes in the requirements. As a result, version five was not the best design decision for the requirements that they took into account. However, building an ideal system would require discarding much of the code that they had already written in previous versions.

From this, they conclude that even an optimal design strategy during the design phase does not deliver an optimal design. The reason is that changes in the requirements will occur in future evolution cycles which may cause the design decisions taken earlier to be less optimal. This proves that architectural drift and architectural erosion are parts of software development, no matter how meticulously the software is developed (Gurp & Bosch, 2002).

From the above studies, we understand that the causes of software aging are related to the design decisions taken over a period of time, which in turn causes architectural degradation. The resulting effects of both software aging and architecture degradation are homogeneous. Therefore, according to existing studies, architectural degradation is one of the key factors of software aging.

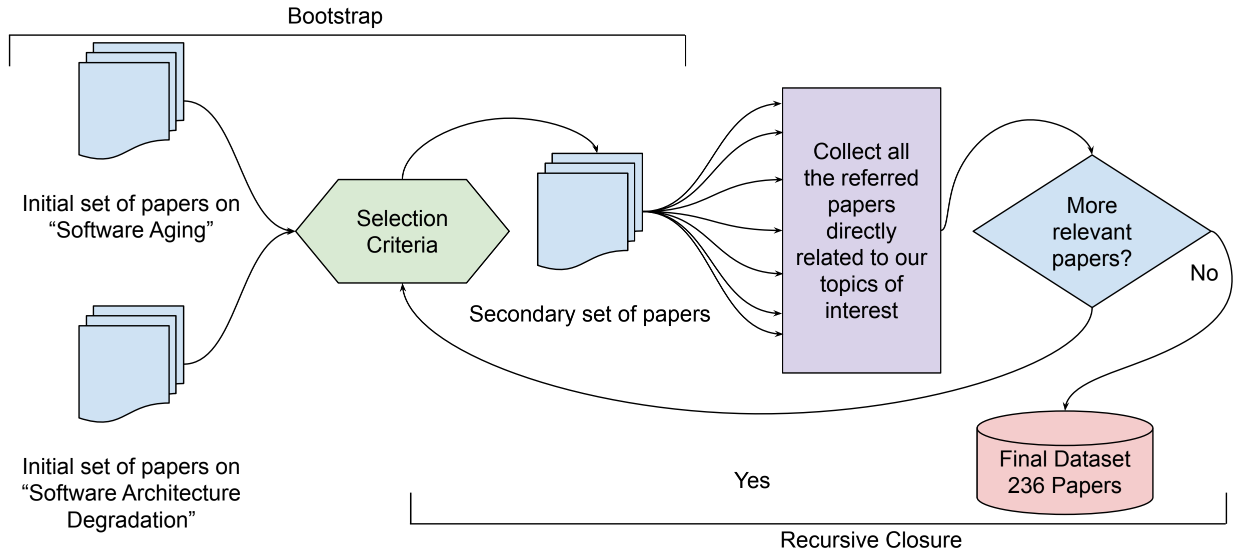

Figure 1. Steps of our analysis

4. Related Works

Baabad et al. (Baabad, 2020) did a systematic literature review (SLR) on the existing studies on the architectural erosion of the Open Source Software (OSS) projects. They collected the research papers from eight major online databases (ACM, Springer, ScienceDirect, Taylor, IEEE Explorer, Scopus, Web of Science, and Wiley) and ended up with a total of 74 papers for their study. They found that rapid software evolution, frequent changes, and the lack of developers' awareness are the most common reasons behind the architecture degradation for OSS projects. The repeated architectural erosion shortens the lifetime of the system affecting the architecture entirely which results in redesigning the architecture of the system from scratch (rejuvenation) (Li and Long, 2011). This phenomenon is also known as software aging (Parnas, 1994).

Another similar study has been performed by Pietrantuono (Pietrantuono & Russo, 2020). They reviewed the effort conducted by the software aging and rejuvenation (SAR) community in the cloud domain. From a total of 105 existing research papers from three major digital libraries, they characterized the existing literature according to four dimensions: the publication trends, the aging analysis methods, and metrics, the rejuvenation solutions, and the validation approach. Compared to their study, we analyze the existing studies into "citation trend", "year of publication", "publication venue", "reasons behind software aging", and "solutions proposed".

Contronneo et al. (Cotroneo, 2014) did a survey on software aging and rejuvenation. They discussed the concepts such as aging-related failures, aging effects, aging bugs, and aging indicators in short. Similar to our approach they also followed bootstrap and recursive closure. Controneo classified the software aging and rejuvenation papers into four dimensions: type of system, aging effects, aging indicators, and rejuvenation techniques. They discussed different approaches that were conducted by researchers. For example, model-based analysis, measurement-based analysis, and hybrid analysis. In model-based analysis, they mostly took theoretical scenarios rather than real case scenarios. They make assumptions about a system that may not apply to all real-world scenarios. In measurement-based analysis, they collect system data such as memory space allocations and swap memory during the run-time or while starting the system. They compare the current measurement values with the previous or initial values and analyze if there is a degradation in the system which is caused by aging. They also discuss the threshold-based approach in which the system is run or ignored until a certain amount of fault or problems occurs. The hybrid analysis is a combination of model-based analysis and measurement-based analysis. Although the study approach is similar, the objective of our study is different where our survey focuses on what are the primary causes identified for software aging and what are the common solutions proposed by the existing literature.

5. Methodology of Our Analysis

This study aims to synthesize and present a critical perspective on the current research in the field of Software Aging and Architectural Degradation. We followed the methodology applied by Cotroneo et al. (Cotroneo, 2014) in a similar study on a survey of software aging and rejuvenation. Cotroneo et al. explored the fundamental ideas of Software Aging and Rejuvenation (SAR). Figure 1 shows the steps of our methodology at a glance. They discuss how software aging impacts a system and the necessity of rejuvenation.

They briefly examine aging-related failures, aging impacts, aging bugs, and aging indications. The primary goal of the paper is to conduct a literature analysis on SAR. Cotroneo applied two approaches to gather the information that they named "Bootstrap" and "Recursive Closure". Bootstrap comprises of gathering papers using a keyword-based search, and recursive closure is to collect papers from the references that are related to the topic they are interested in for their study. However, they selected only the relevant papers from the top most conference and journal publications. A thorough review of Software Aging and Rejuvenation studies is provided by them and the articles are divided into four categories based on the "Method of Analysis", "Type of the System", "Aging Effects" and "Aging indicators" approaches.

5.1 Bootstrap

Table 1 represents the number of papers we collect from the bootstrap method. "Keyword" column lists all the keywords that we used to search for the initial set of papers. The column "Default Search Result" shows the original search result using the corresponding keyword in the first column. The column "Result After Filtering" shows the number of papers after applying the inclusion (IC) and exclusion (EC) criteria.

Table 1. Table of Keyword-based search results

Keyword	Source	Default Search Result	Result After Filtering
"Software Aging"	IEEE Xplore	109	98
	ACM Digital Library	5	5
	Wiley Online Library	6	5
"Software Aging" and "Rejuvenation"	IEEE Xplore	20	10
	ACM Digital Library	23	20
	Wiley Online Library	2	2
"Architectural Degradation"	IEEE Xplore	1	1
	ACM Digital Library	2	2
	Wiley Online Library	0	0
"Architectural Drift"	IEEE Xplore	0	0
	ACM Digital Library	1	1
	Wiley Online Library	1	1
"Architectural Erosion"	IEEE Xplore	2	2
	ACM Digital Library	0	0
	Wiley Online Library	0	0
"Architecture Degradation"	IEEE Xplore	2	2
	ACM Digital Library	2	2
	Wiley Online Library	0	0
"Architecture Drift"	IEEE Xplore	0	0
	ACM Digital Library	0	0
	Wiley Online Library	0	0
"Architecture Erosion"	IEEE Xplore	7	6
	ACM Digital Library	8	7
	Wiley Online Library	1	1

Search result for different keywords in different databases of publications. The default search result includes papers that matched with the keyword but actually not inline with our the context of this study.

The bootstrap phase uses a keyword-based search for the initial set of papers. Keyword-based search is the process of finding and researching specific search terms in search engines to look up a database of a collection of papers, journals, and conferences. First, we collect the initial set of papers based on keyword-based search as shown in Figure 1. We use the following keywords that are closely related to our research goals and questions to find papers related to the field or subject and assess the impact of the papers:

- "Software Aging"

- "Software Aging" and "Rejuvenation"
- "Architectural Degradation"
- "Architectural Drift"
- "Architectural Erosion"
- "Architecture Degradation"
- "Architecture Drift"
- "Architecture Erosion"

For our search with the keywords, we are interested in the high volume, and highly popular publication databases since they have a rich collection of research papers, journals, and articles.

While performing the keyword-based search it is also important to understand the context of where the keyword is being used. For example, while searching for the term "Software Aging", the search engine may find results that include the words "Software" and "Aging" in different parts of the abstract in the paper. However, these papers may not discuss the concept of software aging. Hence, it is crucial that we find papers that are apt for our study.

With the help of advanced search, we skim through multiple databases that help us connect various different ideas across domains through prominent publications and citations. We performed our search in three different databases, IEEE Xplore, ACM Digital Library, and Wiley Online Library. IEEE Xplore is by far the most comprehensive academic database in the field of engineering and computer science. It is possible to search for not just journal articles, but also conference papers, standards, and books that have met rigorous quality with billions of cited references. The ACM Digital Library provides a complete collection of publications and articles in the fields of computing and information technology. Wiley Online Library focuses on academic publications and instructional materials and has one of the largest collections of online journals, research resources, and books on a large variety of subjects.

Each keyword is subject to an advanced keyword-based search in all three of the research databases i.e IEEE Xplore, ACM Digital Library, and Wiley.

To the default set of papers that were produced by the databases after the search, several inclusion criteria (IC) and exclusion criteria (EC) were applied to filter the papers.

The final selection of papers is then collected when these conditions were fulfilled for the next step which is recursive closure. For example, when the keyword "Software Aging" was searched in all three databases, IEEE Xplore generated 109 papers, ACM Digital Library generated 5 papers and Wiley generated 6 papers, bringing the total initial set of papers generated for the keyword Software Aging to 120. After applying the inclusion criteria (IC) and exclusion criteria (EC), IEEE Xplore generated 98 papers, ACM Digital Library generated 5 papers and Wiley generated 5 papers for the final set of papers bringing the total of the final set of papers generated from the keyword "Software Aging" to 108.

A total of 192 papers have been obtained in the bootstrap phase, and after applying the inclusion and exclusion criteria the initial set of papers was 165.

5.2 Recursive Closure

The recursive closure phase is based on gathering further papers from the references of the initial set of papers obtained during the bootstrap phase. 1 We examine these papers and scrutinize them to see whether they are relevant to our field of research. All relevant referenced papers are added to the gathered collection of papers once each paper's references are reviewed. A search is conducted to see whether the references contain any further relevant publications. If there are, the papers are subjected to the selection criteria and the procedure is restarted until no more relevant articles are found. The final selection of papers contains all of the gathered publications and brought the total number of papers to 236 papers.

Following are the inclusion criteria (IC) and exclusion criteria (EC) using which the papers were filtered:

- IC1: Papers discussing studies and research on "Software Aging", "Software Aging and Rejuvenation", "Architectural Degradation", and "Architectural Drift and Erosion".
- IC2: Papers that have only been published in conferences and journals are considered.
- IC3: Papers that are indirectly related to the topics and are relevant to our study.
- EC1: Language in which the paper was published. In our survey, we only considered the papers that were written in English.
- EC2: Dissertations or thesis have been excluded.

- EC3: The previous versions of a paper are excluded and only the most recent versions are included.
- EC4: Studies that are incomplete.
- EC5: Papers that are not discussing “Software Aging”, “Software Aging and Rejuvenation”, “Architectural Degradation”, and “Architectural Drift and Erosion” in the domain of “Software Engineering”.

After evaluating the final set of papers and gathering a substantial number of papers in both domains, we start accounting for various qualities such as venue of publication, year of publication, and the number of cites.

6. Our Findings

6.1 Year of Publication

The phenomenon of Software Aging was first conceptualized in the year 1994, and the first paper related to this subject was published at the International Conference on Software Engineering (ICSE). Since then, there has been exponential growth concerning research in the field of Software Aging and Rejuvenation. Even though Software Aging and Rejuvenation is not a new topic, it has been constantly gaining importance throughout the years. There have been several conferences and workshops held to publish and discuss topics related to Software Aging and Rejuvenation.

Figure 2 represents the timeline of the number of publications, we can observe that there are multiple inconsistencies in the number of publications each year.

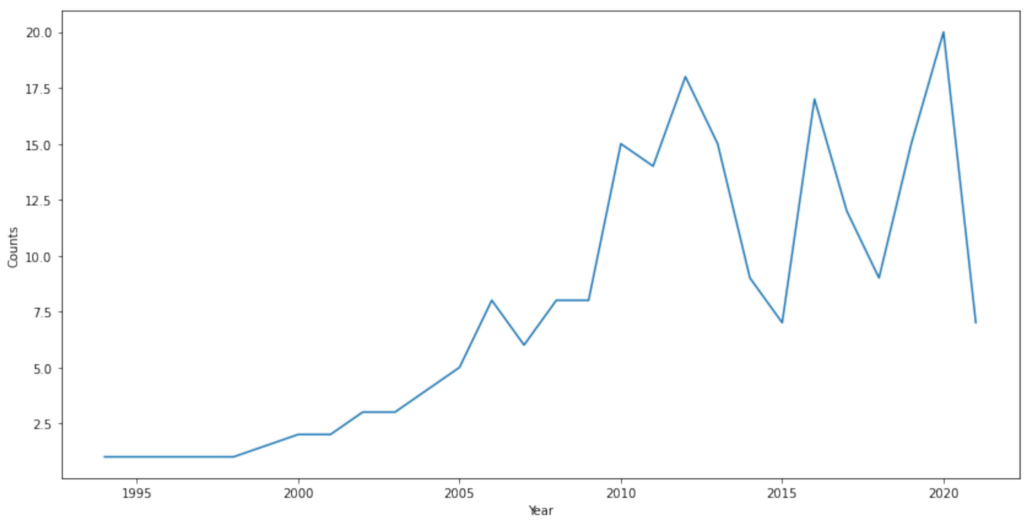

Figure 2. Timeline of the number of publications in the field of Software Aging

A few of the prominent years in which there were a high number of publications are 2020, 2012, 2016, 2019, and 2013 respectively, and the prominent locations of those publications are the IEEE International Conference on Software Reliability Engineering Workshops (ISSREW), the International Symposium on Software Reliability Engineering (ISSRE), the IEEE International Workshop on Software Aging and Rejuvenation (WoSAR), and Software Quality Journal. In the years 2014, 2015, and 2018 most of the prominent venues did not conduct a conference. For example in 2015, only 7 papers were published with the IEEE International Conference on Software Reliability Engineering Workshops (ISSREW) publishing 2 papers. This caused a sudden drop in the number of papers published during that year.

Figure 3 shows the timeline of the number of publications in Architectural Degradation.

Architectural Degradation is an important area to be researched but it is quite neglected. Even though Architectural Degradation has been around for a very long time, the number of papers published in this area is inadequate when compared to Software Aging. There is very little prominence given to Architectural Degradation compared to Software Aging even though they have common effects on the same software.

As we see in Figure 3 only a few papers published on this subject before 2003 with the first paper published in the year 1998. The year 2020 had the highest number of papers published related to Architectural Degradation with 5 papers published from different venues. The most number of papers published before that was in the year 2003, where 4 papers

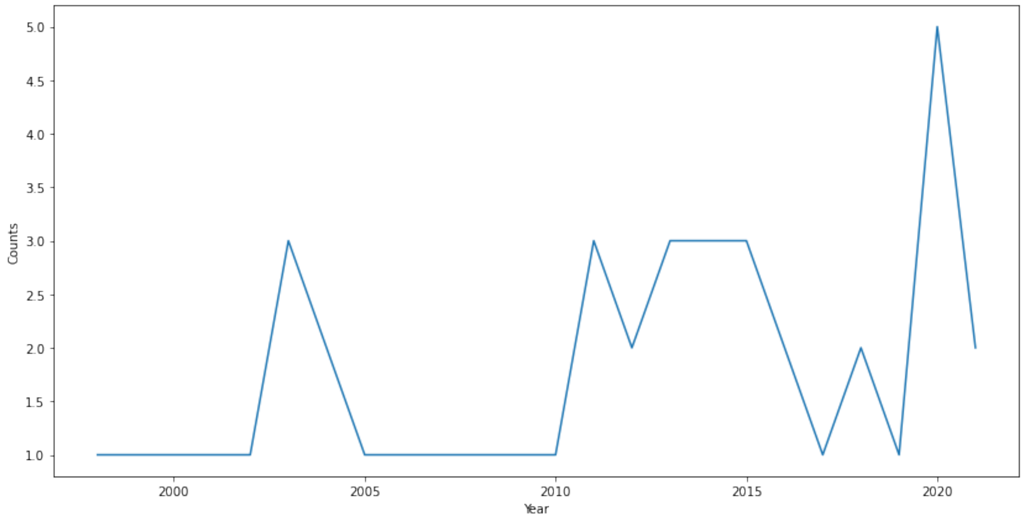

Figure 3. Timeline of the number of publications in the field of Architectural Degradation

that are related to Architectural Degradation were published. Apart from 2011, 2013-2015 there were only around one or two papers published related to this subject. However, there are several textbooks written and published on this subject.

6.2 Number of Citations.

6.2.1 Most Cited Paper in the Domain of Software Aging

Table 2 lists the papers that explicitly did study on software aging. The DOI is also associated to the paper titles.

Table 2. Software Aging papers with DIO and Title

Title of the paper	DOI
Software aging	https://doi.org/10.1109/ICSE.1994.296790
Proactive management of software aging	https://doi.org/10.1147/rd.452.0311
A methodology for detection and estimation of software aging	https://doi.org/10.1109/ISSRE.1998.730892
Analysis of software aging in a web server	https://doi.org/10.1109/TR.2006.879609
The fundamentals of software aging	https://doi.org/10.1109/ISSREW.2008.5355512
An approach for estimation of software aging in a web server	https://doi.org/10.1109/ISESE.2002.1166929
Modeling and analysis of software aging and rejuvenation	https://doi.org/10.1109/SIMSYM.2000.844925
A survey of software aging and rejuvenation studies	https://doi.org/10.1145/2539117
A workload-based analysis of software aging, and rejuvenation	https://doi.org/10.1109/TR.2005.853442
Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers	https://doi.org/10.1109/DSN.2002.1028933

Table containing the research papers along with their doi, that discuss about software aging.

Figure 4 shows the most cited papers on Software Aging with DOIs on the Y axis instead of titles (due to space limitation) so that we care relate to the Table 2. The paper titled as “Software Aging” by Parna et al. (Parna, 1994) is the most cited paper in the history.

Parnas, in the paper “Software Aging”, states that the two main causes of software aging are the failures caused due to the owner’s unable to modify a product to meet its changing needs and also the result of changes that are made due to ignorant coding by developers.

Software aging cannot be prevented, but it can be slowed down (Parnas, 1994). The prominent method to slow down software aging is by documenting the code well and making notes of the major and minor changes as well as documenting which function or module can perform what functionality. This helps the maintainers of the code to make changes easily without creating further problems. It also prevents deterioration of the aging problem by adding new bugs without understanding the previous code that can break the product as a whole. Adding a new feature to the existing software should be done with background knowledge of the program. Software aging causes financial problems to the company if

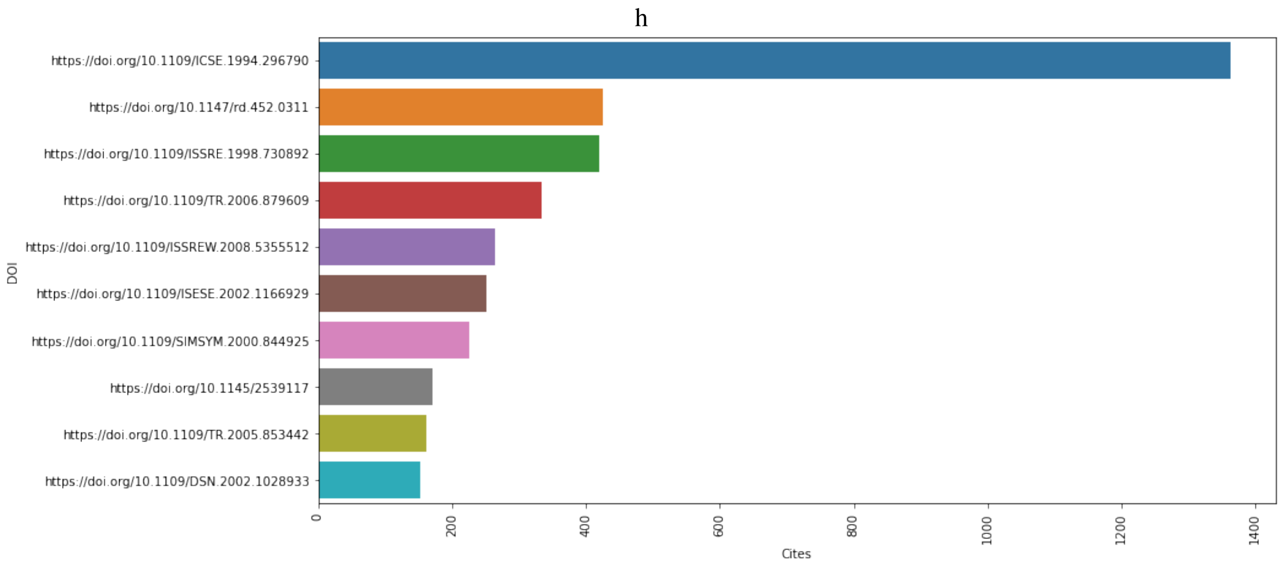

Figure 4. Number of citations with DOI for Software Aging paper

the software is not able to meet the current standards or the requirements of the customer.

Parna et al. (Parnas, 1994) also states that we must design for change which is a concept where the program is tolerant to the new changes. The main code is separated from the code that can be changed or is susceptible to change. It also states that the programmer developing the program should be the one writing the documentation rather than a non-technical person. Finally, it states that the company should be ready for software aging well before the actual product is released, that is the company has to be financially and physically ready to face the old age of the program by taking precautions and making assumptions about the future.

Castelli et al. (Castelli, 2001) had the second most cited paper in our database. The paper states that a system-wide blackout is a result of multiple software failures. The main contributors to software failures are the exhaustion of system resources, data corruption, numerical error accumulation, storage space fragmentation, bloating and leaking of memory, unterminated threads, and unreleased file locks. These failures cause performance degradation over a period of time. This phenomenon of degradation is Software Aging. As the software grows over a period of time, the failure rate increases due to the increase of the factors mentioned earlier. Another factor is that the code becomes extremely complex over a period of time and contains bugs or errors that are settled down like a deep stain, then fighting against Software Aging becomes extremely complex.

Castelli et al. created a framework that detects, predicts, and manages rejuvenation techniques such as re-developing the entire system or restarting the middleware. They implemented the framework on the xSeries of IBM servers. The xSeries Software Rejuvenation Agent (SRA) was developed to track and examine the consumable resource and estimate the time that these resources would be exhausted and alert the organization to prepare for rejuvenation. This resulted in a significant improvement of the cluster system availability and reduced the downtime cost.

As we see in Figure 4, Garg et al.'s paper "A methodology for detection and estimation of software aging" (Garg et al., 1998) was the third most cited paper in our database. In this paper, the authors claim Software Aging to be a phenomenon of failure or end-user crash which is caused by the accumulation of errors during the execution of the software. They also claim that software aging causes performance degradation. The factors affecting are similar to the previously discussed paper such as memory bloating and leaking, data corruption, storage space fragmentation, etc. The authors proposed an SNMP-based distributed monitoring tool that was built to collect data on the usage of the operating system and also the system activity on a UNIX Workstation. They proposed a new metric "Estimated time to exhaustion" which was used to detect and validate the existence of aging. They used techniques like slope estimation technique and statistical techniques for the detection and estimation of aging. They found that the metric was inversely co-related with aging that is as the value of the metric was high the effect and presence of aging were low. On the contrary, if the value of the metric was low then the presence and effects of aging were high. The metric also helped in comparing the effect of aging with respect to different system resources and the identification of the most prominent resource to be monitored and managed to prevent aging.

6.2.2 Most Cited paper in the Domain of Architectural Degradation

Number of research works have been done on architectural degradation as well. Table 3 lists the papers that explicitly studied architectural degradation. The table lists the paper titles along with their DOIs.

Table 3. Architectural Degradation papers with DOI and Title

Title of the paper	DOI
Controlling software architecture erosion: A survey	https://doi.org/10.1016/j.jss.2011.07.036
Recommending refactorings to reverse software architecture erosion	https://doi.org/10.1109/CSMR.2012.40
Assessing architectural drift in commercial software development: a case study	https://doi.org/10.1002/spe.999
Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery.	http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8661
Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance	https://doi.org/10.1109/ICSM.2011.6080779
Using architectural properties to model and measure graceful degradation	https://doi.org/10.1007/3-540-45177-3_12
Blending and reusing rules for architectural degradation prevention	https://doi.org/10.1145/2577080.2577087
Stop the software architecture erosion: building better software systems	https://doi.org/10.1145/1869542.1869563
Complementing model-driven development for the detection of software architecture erosion	https://doi.org/10.1109/MiSE.2013.6595292
Lightweight prevention of architectural erosion	https://doi.org/10.1109/IWPSE.2003.1231211

Table containing the research papers that discuss about the architectural degradation along with their doi.

Figure 5 shows the number of citations of the papers in Architectural Degradation. The research paper by Lakshitha de

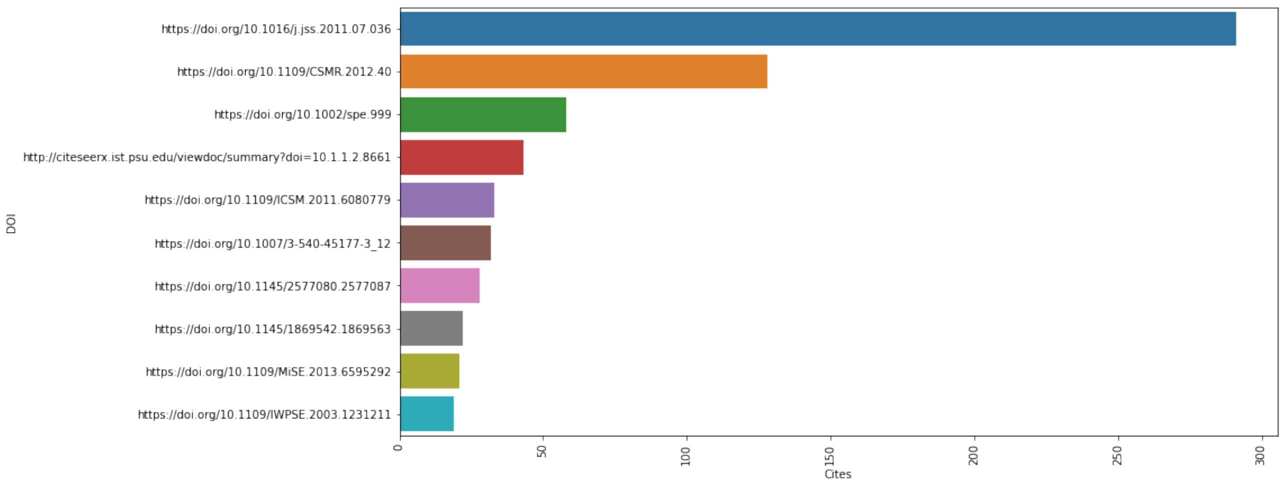

Figure 5. Number of citations with DOI for Architectural Degradation paper

Silva and Dharini Balasubramaniam is the most cited paper titled as "Controlling software architecture erosion: A survey" (De Silva & Balasubramaniam, 2012). In this paper, authors state that the most important properties and design elements of a software system are captured by software architecture. The modification to the system increases over a period of time and it violates the architectural principles which in turn degrades the system and its performance. Resulting in shortening the lifetime of the system. To combat and address these issues, the paper discusses the different strategies along with their advantages and disadvantages that can be used to detect and restore the eroded architecture of the system. They classify and combine different strategies to either minimize, repair, or prevent architectural erosion.

The second most cited paper titled "Recommending Refactorings to Reverse Software Architecture Erosion" authored by R Terra, M T Valente, K Czarnecki, R S. Bigonha, states that Architectural Erosion is a gap between the design architecture compared to the implemented architecture of the source code. Due to this the software architecture degrades the quality of the system and decreases maintainability, evolvability, extensibility, and reusability. Over a period of time,

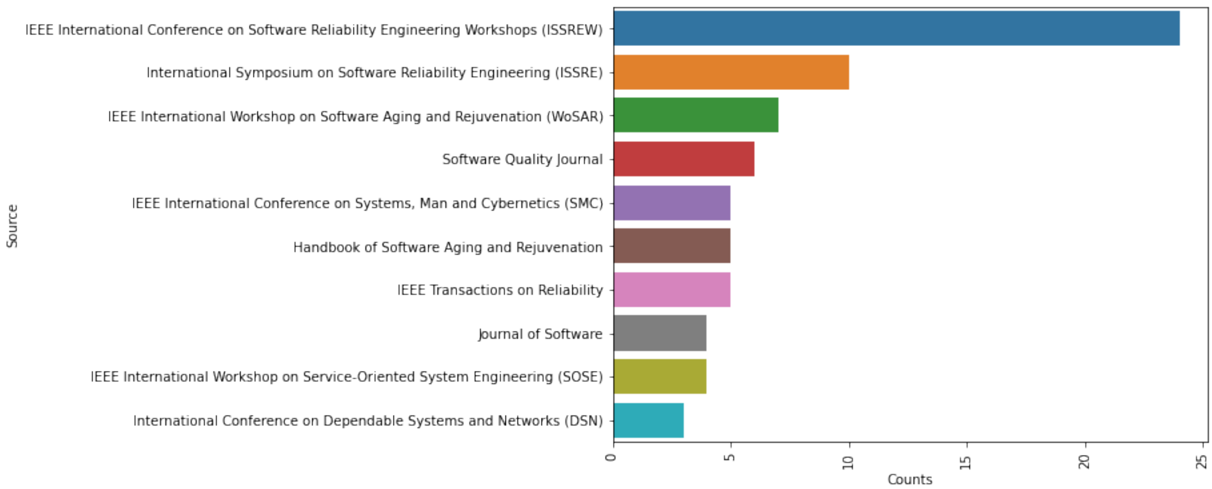

Figure 6. Top 10 publication venues for Software Aging with count

these gaps increase transforming the software architecture into an unmanageable state (Terra, 2012). To counteract this, they proposed a tool that can locate the point of erosion in a software system that is detecting the source code violations from the implemented architecture to the proposed architecture. Also providing the solution to resolve the violations that are caused by the deviation from the proposed architecture. In the paper, they have considered an example of a web-based eCommerce system called “MyWebMarket”. They created an architecture to resemble a real-world small-scale project in which they used architecture erosion and used their approach to detect it. They considered four constraints that caused violations to the architecture. After applying their tool and correcting with the help of the recommendations, in three constraints all the violations were rectified whereas in the remaining one constraint 60% of the violations were rectified. This shows that their tool was very effective in reducing architectural erosion.

One of the most cited papers is “Assessing Architectural Drift in Commercial Software Development: A Case Study” by Jacek Rosik, Andrew Le Gear, Jim Buckley, and Muhammad Ali Babar. In this paper, they have considered a two-year case study of software that was developed from scratch at IBM Dublin Software Lab. They say that software architecture is one of the most important attributes of system design. The main intent of the paper is to find Architectural Drift during software development and to evaluate if the detection leads to inconsistency removal. They have used an approach known as Reflexion Modeling which is used to find the architectural drift. Reflexion Modelling is an architectural recovery method that is used to extract the implemented architecture and can also visualize the differences in the planned architecture. They say that architectural drift occurred during the redevelopment of the system even when the developers had a clear vision of how the system had to be implemented at the architectural level. The inconsistencies were due to the oversight of the original design architecture, dependence on legacy code, and misplacing or duplication of methods or constants. Even though inconsistency identification was possible, the inconsistencies were not removed. There were many reasons for this. One of the reasons was that there would be a ripple effect i.e while removing a minor inconsistency the whole project could stop working (Rosik, 2011).

6.3 Discussion Based on Publication Venue

In Figure 6, we see that out of many different publication venues, some of the prominent ones are the IEEE International Conference on Software Reliability Engineering Workshops (ISSREW) with over twenty-four papers published. The International Symposium on Software Reliability Engineering (ISSRE) with around ten papers published, the IEEE International Workshop on Software Aging and Rejuvenation (WoSAR) with about seven paper published, and Software Quality Journal with about six papers published where most of the work related to the field of Software Aging is published or curated.

The year 2020, had the highest number of papers related to Software Aging and Rejuvenation published. From the year 2013, different venues apart from the ISSREW, ISSRE, WoSAR, or Software quality journals also started publishing papers related to software aging and by the year 2020 many other different venues like the International Journal of Electrical and Computer Engineering (IJECE), the International Conference on Computer and Applications (ICCA), the IEEE International Conference on Systems, Man and Cybernetics (SMC), Journal of Information Technology Research (JITR), etc. also published papers related to Software Aging.

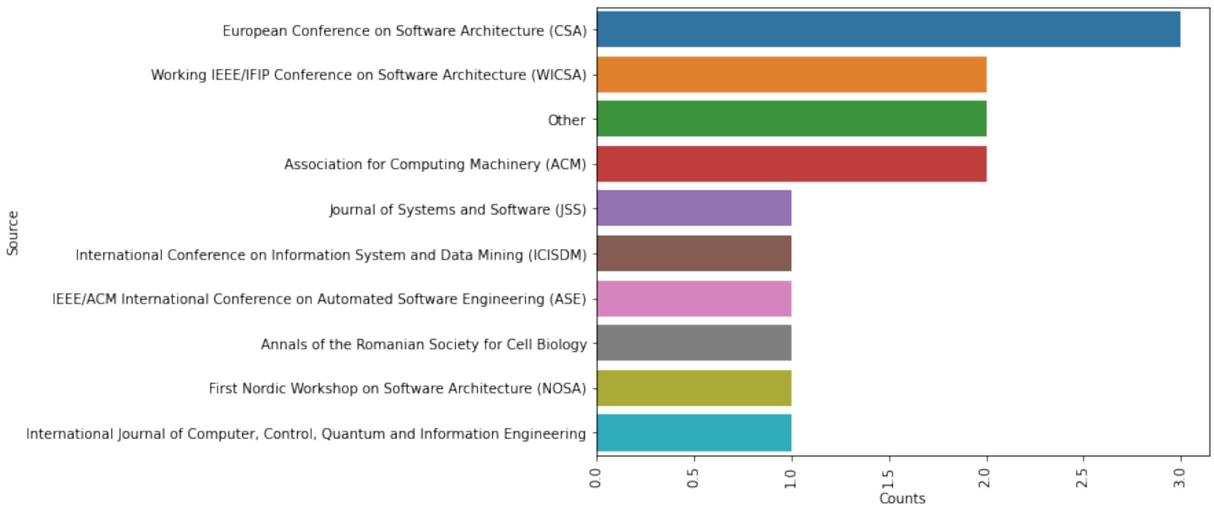

Figure 7. Top 10 publication venues for Architectural Degradation with count

The International Symposium on Software Reliability Engineering (IEEE ISSRE) was first held in 1990 in Washington DC. It is an academic conference that covers the reliability engineering of software. The symposium runs usually for 4 days with strong industry participation and has also been held in over 18 cities outside The USA. The main properties of interest are reliability, safety, security, availability, and quality of the software. The symposium later started a multi-track program where it integrated workshops and tutorials like IEEE International Conference on Software Reliability Engineering Workshops (ISSREW). These workshops provide further added opportunities for exchanging information and collaborating. The main aim of these workshops is to discuss software challenges and research development in the early stages.

The IEEE International Workshop on Software Aging and Rejuvenation (WoSAR) is an international workshop that is usually co-located with The International Symposium on Software Reliability Engineering. The workshop provides a forum for practitioners and researchers to discuss recent advances or discoveries in the field of software aging and rejuvenation and to present their work on the practical and theoretical characteristics of software aging and its mitigation using software rejuvenation. Software Quality Journal is a journal published by Springer New York. First published in 1992, this journal has seen a rising impact since 2011. The main aim of the Software Quality Journal is to promote awareness of the crucial role of quality management in the effective construction of the software systems developed, used, and maintained by organizations in pursuit of their business objectives. It also provides a forum for the exchange of experience and information on quality management and the methods, tools, and products used to measure and achieve it. Finally, it provides a vehicle for the publication of academic papers related to all aspects of software quality.

In figure 7 we can see some of the most eminent venues of publication are European Conference on Software Architecture (CSA) with about three papers published, Working IEEE/IFIP Conference on Software Architecture (WICSA) with about two papers published, Association for Computing Machinery (ACM) with around two papers published, and Journal of Systems and Software (JSS) with only one paper published. There were also two papers published from other sources such as textbooks, articles, and handbooks.

7. Discussion and Answering to the Research Questions

In the previous section, we discussed the top-cited papers and the research that has been conducted in the domain of software aging and architectural degradation. We identify that the problems causing those phenomena are similar and tend to have homogeneous consequences. For instance, many researchers (Castelli et al., 2001) believe that multiple factors cause different types of software failures and some of the main contributors are data corruption, storage space fragmentation, exhaustion of system resources, unterminated threads, numerical error accumulation, bloating and leaking of memory, and unreleased file locks. These failures can cause performance and system degradation failure gradually over a period of time. This type of degradation is termed software aging.

Similarly, as part of the evolution of the software, it has to adapt to newer technologies and updated requirements while updating and modifying the software to the current needs. During the modifications, the design principles and the system architecture are ignored which leads to a gap between the conceptual and the concrete architecture. For instance, Rahman et al. found that the conceptual and the concrete architecture are fairly different in the Google Chrome web browser

(Rahman et al. 2019). This gap can grow over time and become unmanageable, complex, and expensive to maintain. This type of degradation is known as architectural degradation.

Software aging and architecture degradation complement each other. One of the main reasons that a system declines due to software aging is because changes are made to it without sufficient documentation and comprehension of the code. A system that is prone to architectural deterioration is also prone to software aging eventually. As mentioned earlier, during the lifetime of a software, it undergoes several changes and updates at several intervals. These changes are done to allow the software to adapt to new requirements and for the software to provide functionalities with the advancement of technologies and the latest customer demands. However, the changes to the software do not always undergo proper planning and are done ignorantly without accounting for repercussions that the software might have in the future.

During a minor modification or update in the software, changes are not made to the existing architecture to incorporate them. Assuming that these changes can be documented during the next phase or release, the documentation will be either forgotten or ignored. As a result, gaps are then created between the intended architecture and the existing architecture. These minor changes which are unaccounted for might cause errors or failures. Whenever a developer is trying to make new changes, they might find it difficult to understand the current software system, and will be inefficient as the implementation is different from the documentation. It is also difficult to understand by just debugging the code as it is very complex and may contain a large number of files or dependencies. Without a proper understanding of the system, the modification may add more errors and failures causing a system-wide disruption. Documenting and understanding the architecture is very important. By regularly updating the documentation and structure of the software architecture, the system can be designed to be susceptible to change and can be modified to reduce the effects of aging and mitigate architectural degradation while also implementing features and functionalities that are not prone to additional failures or bugs.

We can also rejuvenate the system which increases the sustainability of the software for a longer period. Additionally, planning and designing beforehand helps to identify the problems that may occur before implementing it. When ignored, the aging of the software accelerates and the software becomes obsolete in nature.

7.1 Answer to the Research Question RQ1

As we discussed in Section 6, the research on Software Aging has begun in 1994 and the study has increased gradually since the late 1990s and early 2000. The popularity of Software Aging research works was steadily increasing until 2009. Since 2010 we observe a very high number of research works being published and a number of new venues are created for Software Aging studies. Even though there was a downfall in 2015 due to the reduced number of conferences, there still was a gradual increase in the research conducted. Depending on the number of prominent venues that held conferences that year, there was a significant rise or fall in the publications in the domain of software Aging. Even though there has been instability in the number of publications each year, the domain of software aging has been continuously growing to date.

7.2 Answer to the Research Question RQ2

The research papers we studied have discussed the possible reasons behind software aging. There are many factors identified by the authors that are affecting software quality, maintainability, architecture, and causing software aging. We observed or inferred the factors if they have been identified or discussed by the researchers as one of the responsible factors for either “Software Aging”, or “Architecture Degradation” or both.

We cited the papers where a particular factor has been explicitly mentioned as one of the responsible factors. We are calling it “inferred” when a factor has been indirectly inferred from the other factors. For example, Vaidyanathan and Gross (see Table 4) explicitly mentioned that data corruption is one of the factors causing Software Aging while they have also discussed at the same time that Software Aging happens eventually from Architecture Degradation which indirectly infers that data corruption causes architecture degradation eventually causing software aging.

Table 4 shows that there are several factors contributing to Software Aging as well as Architectural Degradation. The most frequently discussed factors by researchers include Code-Complexity and Maintainability, Data Corruption, Exhaustion of System Resources, Memory Bloating, Memory Leaking, Numerical Error Accumulation, Storage Space Fragmentation, Unplanned System Changes, Unreleased File Locks, and Unterminated Threads.

Table 4. Factors contributing to Software Aging and Architectural Degradation

Factor/s	Software Aging	Architectural Degradation
Complex and Unmaintainable Code	Observed (Parnas, 1994)	Observed (Baabad, 2020)(Andrews and Sheppard, 2020)
Data Corruption	Observed (Vaidyanathan & Gross, 2003) (Cotroneo, 2010)	Inferred
Exhaustion of System Resources	Observed (Li, 2002)	Inferred
Memory Bloating	Observed (Cassidy, 2002)(Cotroneo, 2016)(Cotroneo & Iannillo, 2020)	Inferred
Memory Leaking	Observed (Cassidy, 2002)(Cotroneo, 2016)(Cotroneo & Iannillo, 2020)	Inferred
Numerical Error Accumulation	Observed (Li, 2002)	Inferred
Storage Space Fragmentation	Observed (Garg, 1998)	Inferred
Unplanned System Changes	Observed (Jiang & Xu, 2007)	Observed (Baabad, 2020)(Andrews and Sheppard, 2020)
Unreleased File Locks	Observed (Garg, 1998)(Vaidyanathan and Trivedi, 2005)	Inferred
Unterminated Threads	Observed (Cassidy, 2002)(Alonso, 2010)(Cotroneo & Matias, 2020)	Inferred

Table containing the factors that have been identified by the researchers contributing to Software Aging, also factors that have been identified by the researchers contributing to architectural degradation.

7.3 Answer to the Research Question RQ3

We observe that Software Aging is a big concern for software systems. The aging of software and the deterioration of architecture are inextricably linked. Hence, we can determine that one of the major contributors to Software Aging is Architectural Degradation. Both of these can be mitigated to some extent by upgrading the architecture regularly to reflect system changes and modifications. When modifying or updating a software system, the changes must be planned and implemented in the architecture. It must also follow architectural design guidelines, which makes the software system more practical, viable, and sustainable in the long run.

7.4 Answer to the Research Question RQ4

One of the major challenges in the area of Software Aging is that it occurs in the software system over a long period of time and hard to see symptoms when the system is young. The moment the symptoms start, they are very negligible and on many occasions, developers ignore them and do not care for taking a big initiative to revise or investigate the current architecture of the system. The researchers also face difficulties producing real-time scenarios to identify all the factors that contribute to the problem.

8. Future Work and Conclusion

This study aims to provide insight into the current trends in the research and practice of software aging. Particularly as it relates to the problems, publication trends, and factors that contribute to software aging. We also provide a critical perspective on the relationship between software aging and architectural degradation. We believe that insight into the problems, trends, and contributing factors to software aging would help practitioners to design effective solutions. We found that the more our practitioners are growing with software development, the more researchers studying the phenomenon of software aging. In many cases researchers tend to intertwine the terms software aging and architecture degradation, in many cases, researchers tend to consider architecture degradation as a path to aging. Many factors have been identified in the literature that accelerates software aging and degrades the architecture at the same time. However, from this study, it is very clear that architecture degradation is a shadow of software aging. Many of the factors mentioned by researchers eventually degrade the architecture and when the architecture is degraded, software becomes unwelcoming to the new changes and at some point it becomes unmaintainable.

The takeaway from this study is that besides many other factors, software architecture is also one of the prominent factors that expedite software aging. Although practitioners take lots of initiatives to improve code quality, avoid data leaks, data corruption, and refactor code to avoid code complexity. We observed that revisiting architecture at a regular interval is a necessary task to watch for architecture degradation and forecasting software aging. Our future work as a follow-up to this study will build a tool to visualize the current architecture of a software application at run-time. We will quantify the architectural drift by comparing the previous versions of the software system so that developers understand not just the difference between the conceptual/initial architecture and the current architecture but also they will be able to measure how much the architecture has drifted from the previous version. Additionally, the user can find the impact of the newly added changes they made to the architecture. This will help document the system better and mitigate Software Aging and Architectural Degradation.

References

Alonso, J., Torres, J., Berral, J. L., & Gavalda, R. (2010). *Adaptive on-line software aging pre- diction based on machine learning.* In Proceedings of 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp. 507-516.

Andrade, E., & Machida, F. (2019). *Analysis of software aging impacts on plant anomaly detection with edge computing.* In Proceedings of 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 204-210.

Andrews, S., & Sheppard, M. (2020). Software architecture erosion: Impacts, causes, and management. *International Journal of Computer Science and Security (IJCSS), 14*(2), 82-94.

Baabad, A., Zulzalil, H. B., & Baharom, S. B., et al. (2020). Software architecture degradation in open source software: A systematic literature review. *IEEE Access, 8,* 173681173709.

Cassidy, K. J., Gross, K. C., & Malekpour, A. (2002). *Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers.* In Proceedings international conference on dependable systems and networks, pp. 478482.

Castelli, V., Harper, R. E., Heidelberger, P., Hunter, S. W., Trivedi, K. S., Vaidyanathan, K., & Zeggert, W. P. (2001). Proactive management of software aging. *IBM Journal of Research and Development, 45*(2), 311-332.

Cotroneo, D., Fucci, F., Iannillo, A. K., Natella, R., & Pietrantuono, R. (2016). *Software aging analysis of the android mobile os.* In Proceedings of 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pp. 478-489.

Cotroneo, D., Iannillo, A. K., Natella, R., & Pietrantuono, R. (2020). A comprehensive study on software aging across android versions and vendors. *Empirical Software Engineering, 25*(5), 3357-3395.

Cotroneo, D., Matias Jr, R., & Natella, R. (2020). *Fundamentals of software aging.* In Handbook of software aging and rejuvenation: fundamentals, methods, applications, and future directions, pp. 21-39. World Scientific.

Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2010). *Software aging analysis of the linux operating system.* In Proceedings of 2010 IEEE 21st International Symposium on Software Reliability Engineering, pp. 71-80.

Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC), 10*(1), 1-34.

De Silva, L., & Balasubramaniam, D. (2012). Controlling software architecture erosion: A survey. *Journal of Systems and Software, 85*(1), 132-151.

Garg, S., Van Moorsel, A., Vaidyanathan, K., & Trivedi, K. S. (1998). *A methodology for detection and estimation of software aging.* In Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257), pp. 283-292.

Heinkens, R., & de Graaf, O. An introduction to software aging: How to deal with it? Computing Science University of Groningen, pp. 44.

Jiang, L., & Xu, G. (2007). Modeling and analysis of software aging and software failure. *Journal of systems and software, 80*(4), 590-595.

Li, L., Vaidyanathan, K., & Trivedi, K. S. (2002). *An approach for estimation of software aging in a web server.* In Proceedings of the International Symposium on Empirical Software Engineering, pp. 91-100.

Li, Z., & Long, J. (2011). *A case study of measuring degeneration of software architectures from a defect perspective.* In Proceedings of the 18th Asia-Pacific Software Engineering Conference 2011, pp. 242-249.

Liu, J., & Meng, L. (2019). Integrating artificial bee colony algorithm and BP neural network for software aging prediction in IOT environment. *IEEE Access, 7,* 3294132948.

Liu, J., Tan, X., & Wang, Y. (2019). *Cssap: software aging prediction for cloud services based on arima-lstm hybrid model.* In Proceedings of the 2019 IEEE International Conference on Web Services (ICWS), pp. 283-290.

Parnas, D. L. (1994). *Software aging.* In Proceedings of 16th International Conference on Software Engineering, pp. 279-287.

Pietrantuono, R., & Russo, S. (2020). A survey on software aging and rejuvenation in the cloud. *Software Quality Journal, 28*(1), 7-38.

Rahman, M. T., Rigby, P. C., & Shihab, E. (2019). The modular and feature toggle architectures of google chrome. *Empirical Software Engineering, 24*(2), 826-853.

Rosik, J., Le Gear, A., Buckley, J., Babar, M. A., & Connolly, D. (2011). Assessing architectural drift in commercial software development: a case study. *Software: Practice and Experience, 41*(1), 63-86.

Saraswat, S., & Yadava, G. (2008). An overview on reliability, availability, maintainability and supportability (rams) engineering. *International Journal of Quality & Reliability Management.*

Terra, R., Valente, M. T., Czarnecki, K., & Bigonha, R. S. (2012). *Recommending refactorings to reverse software architecture erosion.* In Proceedings of 2012 16th European Conference on Software Maintenance and Reengineering, pp. 335-340.

Torquato, M., Araujo, J., Umesh, I., & Maciel, P. (2018). Sware: a methodology for software aging and rejuvenation experiments. *Journal of Information Systems Engineering and Management, 3*(2), 15.

Vaidyanathan, K., & Gross, K. (2003). *Mset performance optimization for detection of software aging.* In Proceedings of ISSRE. Citeseer.

Vaidyanathan, K., & Trivedi, K. S. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing, 2*(2), 124-137.

Van Gurp, J., & Bosch, J. (2002). Design erosion: problems and causes. *Journal of systems and software, 61*(2), 105-119.

Whiting, E., & Andrews, S. (2020). *Drift and erosion in software architecture: Summary and prevention strategies.* In Proceedings of the 2020 the 4th International Conference on Information System and Data Mining, pages 132-138.

Yakhchi, M., Alonso, J., Fazeli, M., Bitaraf, A. A., & Patooqhy, A. (2015). Neural network based approach for time to crash prediction to cope with software aging. *Journal of Systems Engineering and Electronics, 26*(2), 407-414.

Appendix A

Research Papers Used in the Survey

Abbas, H. M., Zwolinski, M., & Halak, B. (2020). *Aging mitigation techniques for microprocessors using anti-aging software.* In Ageing of Integrated Circuits, pp. 6789. Springer.

Ahamad, S. (2016). Study of software aging issues and prevention solutions. *Int. J. Comput. Sci. Inf. Secur, 14*(8), 307-313.

Ahmad, A. (2016). Predicting software aging-related bugs from imbalanced datasets by using data mining techniques. *IOSR J. Comput. Eng., 18,* 27-35.

Aker, T. E. (2007). *Discovering architectural drift through evaluation.* Masters thesis, Institutt for datateknikk og informasjonsvitenskap.

Alencar, F., Santos, M., Santana, M., & Fernandes, S. (2014). *How software aging affects sdn: A view on the controllers.* In 2014 Global Information Infrastructure and Networking Symposium (GIIS), pp. 16. IEEE.

Alonso, J., Goiri, I., Guitart, J., Gavalda, R., & Torres, J. (2011). *Optimal resource allocation in a virtualized software aging platform with software rejuvenation.* In 2011 IEEE 22nd International Symposium on Software Reliability Engineering, pp. 250259. IEEE.

Alonso, J., Torres, J., Berral, J. L., & Gavalda, R. (2010a). *Adaptive on-line software aging prediction based on machine learning.* In 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp. 507516. IEEE.

Alonso, J., Torres, J., Berral, J. L., & Gavalda, R. (2010b). *J2EE instrumentation for software aging root cause application com- ponent determination with AspectJ.* In 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 18. IEEE.

Andrade, E., & Machida, F. (2019). *Analysis of software aging impacts on plant anomaly detection with edge computing.* In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 204210. IEEE.

Andrade, E., Machida, F., Pietrantuono, R., & Cotroneo, D. (2020). *Software aging in image classification systems on cloud and edge.* In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 342-348. IEEE.

Andrews, S., & Sheppard, M. (2020). Software architecture erosion: Impacts, causes, and management. *International Journal of Computer Science and Security (IJCSS), 14*(2), 82-94.

Andrzejak, A., & Silva, L. (2007). *Deterministic models of software aging and optimal rejuvenation schedules.* In 2007 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 159168. IEEE.

Andrzejak, A., & Silva, L. (2008). *Using machine learning for non-intrusive modeling and prediction of software aging.* In NOMS 2008-2008 IEEE Network Operations and Management Symposium, pp. 2532. IEEE.

Araujo, J., Alves, V., Oliveira, D., Dias, P., Silva, B., & Maciel, P. (2013). *An investigative approach to software aging in android applications.* In 2013 IEEE international conference on systems, man, and cybernetics, pp. 12291234. IEEE.

Araujo, J., Braga, C., Neto, J. B., Costa, A., Junior, R. d. S. M., & Maciel, P. R. M. (2016a). An integrated platform for distributed resources monitoring and software aging mitigation in private clouds. *J. Softw., 11*(10), 976-993.

Araujo, J., Matos, R., Alves, V., Maciel, P., Souza, F. V. d., Jr, R. M., & Trivedi, K. S. (2014). Software aging in the eucalyptus cloud computing infrastructure: characterization and rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems (JETC), 10*(1), 1-22.

Araujo, J., Matos, R., Maciel, P., & Matias, R. (2011a). *Software aging issues on the eucalyptus cloud computing infrastructure.* In 2011 IEEE international conference on systems, man, and cybernetics, pp. 1411-1416. IEEE.

Araujo, J., Matos, R., Maciel, P., Matias, R., & Beicker, I. (2011b). *Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure.* In Proceedings of the middleware 2011 industry track workshop, pp. 17.

Araujo, J., Melo, C., Oliveira, F., Pereira, P., & Matos, R. (2021). *A software maintenance methodology: An approach applied to software aging.* In 2021 IEEE International Systems Conference (SysCon), pp. 1-8. IEEE.

Araujo, J., Oliveira, F., Junior, R. d. S. M., Torquato, M., Ferreira, J., & Maciel, P. R. M. (2016b). Software aging issues in streaming video player. *J. Softw., 11*(6), 554568.

Avritzer, A., Cole, R. G., & Weyuker, E. J. (2010). Methods and opportunities for rejuvenation in aging distributed software systems. *Journal of Systems and Software, 83*(9), 1568-1578.

Avritzer, A., Cotroneo, D., Huang, Y., & Trivedi, K. (2020a). *Software aging and rejuvenation: A genesis.* In Handbook Of Software Aging And Rejuvenation: Fundamentals, Methods, Applications, And Future Directions, pp. 319. World Scientific.

Avritzer, A., Czekster, R. M., Distefano, S., & Trivedi, K. S. (2012). *Software aging and rejuvenation for increased resilience: modeling, analysis and applications.* In Resilience assessment and evaluation of computing systems, pp. 167-183. Springer.

Avritzer, A., Grottke, M., & Menasche, D. S. (2020b). *Software aging monitoring and rejuvenation for the assessment of high availability systems.* In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 327327. IEEE Computer Society.

Avritzer, A., Grottke, M., & Menasche, D. S. (2020c). *Using software aging monitoring and rejuvenation for the assessment of high availability systems.* In Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions, pp. 197228. World Scientific.

Avritzer, A., Pietrantuono, R., & Trivedi, K. (2020d). *Future directions for software aging and rejuvenation research.* In Handbook Of Software Aging And Rejuvenation: Fundamentals, Methods, Applications, And Future Directions, pp. 355362. World Scientific.

Ayyaz, S., Rehman, S., & Qamar, U. (2015). A four method framework for fighting software architecture erosion. *International Journal of Computer and Systems Engineering, 9*(1), 133139.

Baabad, A., Zulzalil, H. B., & Baharom, S. B., et al. (2020). Software architecture degradation in open source software:

A systematic literature review. *IEEE Access, 8,* 173681173709.

Bandara, V., & Perera, I. (2018). *Identifying software architecture erosion through code comments.* In 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), pp. 6269. IEEE.

Bao, Y., Sun, X., & Trivedi, K. S. (2005). A workloadbased analysis of software aging, and rejuvenation. *IEEE Transactions on Reliability, 54*(3), 541548.

Barada, S. & Swain, S. K. (2014). A survey report on software aging and rejuvenation studies in virtualized environment. *Int J Comput Eng Technol (IJCSET), 5*(5), 541546.

Beierlieb, L., Ifflander, L., Milenkoski, A., Goncalves, C. F., Antunes, N., & Kounev, S. (2019). *Towards testing the software aging behavior of hypervisor hypercall interfaces.* In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 218224. IEEE.

Bobbio, A., Puliafito, A., Scarpa, M., & Telek, M. (2020). *Stochastic modeling techniques in software aging and rejuvenation phenomena.* In HANDBOOK OF SOFTWARE AGING AND REJUVENATION: Fundamentals, Methods, Applications, and Future Directions, pp. 363401. World Scientific.

Bombardieri, M., & Fontana, F. A. (2009). *Software aging assessment through a specialization of the square quality model.* In 2009 ICSE Workshop on Software Quality, pp. 3338. IEEE.

Bovenzi, A., Cotroneo, D., Pietrantuono, R., & Russo, S. (2011). *Workload characterization for software aging analysis.* In 2011 IEEE 22nd International Symposium on Software Reliability Engineering, pp. 240249. IEEE.

Britcher, R. N., & Craig, J. J. (1986). Using modem design practices to upgrade aging software systems. *IEEE Software, 3*(3), 16.

Cassidy, K. J., Gross, K. C., & Malekpour, A. (2002). *Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers.* In Proceedings international conference on dependable systems and networks, pp. 478482. IEEE.

Castelli, V., Harper, R. E., Heidelberger, P., Hunter, S. W., Trivedi, K. S., Vaidyanathan, K., & Zeggert, W. P. (2001). Proactive management of software aging. *IBM Journal of Research and Development, 45*(2), 311332.

Chen, P., Qi, Y., Zheng, P., Zhan, J., & Wu, Y. (2013). *Multi-scale entropy: One metric of software aging.* In 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, pp. 162169. IEEE.

Chen, X.-E., Quan, Q., Jia, Y. F., & Cai, K. Y. (2006). *A threshold autoregressive model for software aging.* In 2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE06), pp. 34-40. IEEE.

Chouhan, S. S., & Rathore, S. S. (2021). Generative adversarial networks-based imbalance learning in software aging-related bug prediction. *IEEE Transactions on Reliability, 70*(2), 626642.

Chouhan, S. S., Rathore, S. S., & Choudhary, R. (2021). *A study of aging-related bugs prediction in software system.* In Proceedings of the International Conference on Paradigms of Computing, Communication and Data Sciences, pp. 4961. Springer.

Constantinides, C., & Arnaoudova, V. (2009). *Prolonging the aging of software systems.* In Encyclopedia of Information Science and Technology, Second Edition, pp. 31523160. IGI Global.

Cotroneo, D., De Simone, L., Natella, R., Pietrantuono, R., & Russo, S. (2019). *A configurable software aging detection and rejuvenation agent for android.* In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 239245. IEEE.

Cotroneo, D., Fucci, F., Iannillo, A. K., Natella, R., & Pietrantuono, R. (2016). *Software aging analysis of the android mobile os.* In 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pp. 478489. IEEE.

Cotroneo, D., Iannillo, A. K., Natella, R., & Pietrantuono, R. (2020). A comprehensive study on software aging across android versions and vendors. *Empirical Software Engineering, 25*(5), 33573395.

Cotroneo, D., Iannillo, A. K., Natella, R., Pietrantuono, R., & Russo, S. (2015). *The software aging and rejuvenation repository: http://openscience. us/repo/software-aging.* In 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 108113. IEEE.

Cotroneo, D., & Natella, R. (2012). *Monitoring of aging software systems affected by integer overflows.* In 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops, pp. 265270. Ieee.

Cotroneo, D., Natella, R., & Pietrantuono, R. (2010a). *Is software aging related to software metrics?* In 2010 IEEE Second international workshop on software aging and rejuvenation, pp. 16. IEEE.

Cotroneo, D., Natella, R., & Pietrantuono, R. (2013). Predicting aging-related bugs using software complexity metrics. *Performance Evaluation, 70*(3), 163178.

Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2010b). *Software aging analysis of the linux operating system.* In 2010 IEEE 21st International Symposium on Software Reliability Engineering, pp. 7180. IEEE.

Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2011). *Software aging and rejuvenation: Where we are and where we are going.* In 2011 IEEE Third International Workshop on Software Aging and Rejuvenation, pp. 16. IEEE.

Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC), 10*(1), 134.

Cui, L., Li, B., Li, J., Hardy, J., & Liu, L. (2012). *Software aging in virtualized environments: detection and prediction.* In 2012 IEEE 18th International Conference on Parallel and Distributed Systems, pp. 718-719. IEEE.

Da Costa, J. T., Matos, R. d. S., de Araujo, J. C., & Maciel, P. R. (2021). *Systematic mapping of literature on software aging and rejuvenation research trends.* In 2021 Annual Reliability and Maintainability Symposium (RAMS), pp. 16. IEEE.

Dabiri, F., & Potkonjak, M. (2009). *Hardware aging-based software metering.* In 2009 Design, Automation & Test in Europe Conference & Exhibition, pp. 460465. IEEE.

Das, A., Kumar, A., & Veeravalli, B. (2013). *Aging-aware hardware-software task partitioning for reliable reconfigurable multiprocessor systems.* In 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), pp. 110. IEEE.

De Melo, M. D. T., Araujo, J., Umesh, I., & Maciel, P. R. M. (2017). Sware: an approach to support software aging and rejuvenation experiments. *Journal on Advances in Theoretical and Applied Informatics, 3*(1), 3138.

De Silva, L., & Balasubramaniam, D. (2012). Controlling software architecture erosion: A survey. *Journal of Systems and Software, 85*(1), 132151.

De Silva, M., & Perera, I. (2015). *Preventing software architecture erosion through static architecture conformance checking.* In 2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS), pp. 4348. IEEE.

Du, X., Lu, H., & Liu, G. (2013). Software aging prediction based on extreme learning machine. *TELKOMNIKA Indonesian Journal of Electrical Engineering, 11*(11), 65476555.

Du, X., Qi, Y., Lu, H., & He, X. (2010). *A method for software aging state evaluation.* In 2010 6th International Conference on Advanced Information Management and Service (IMS), pp. 4853. IEEE.

Du, X., Xu, C., Hou, D., & Qi, Y. (2009). *Software aging estimation and prediction of a real vod system based on pca and neural networks.* In 2009 International Conference on Information and Automation, pp. 111-116. IEEE.

El-Shishiny, H., Deraz, S., & Bahy, O. (2008a). *Mining software aging patterns by artificial neural networks.* In Iapr workshop on artificial neural networks in pattern recognition, pp. 252262. Springer.

El-Shishiny, H., Deraz, S. S., & Badreddin, O. B. (2008b). *Mining software aging: A neural network approach.* In 2008 IEEE Symposium on Computers and Communications, pp. 182187. IEEE.

Escheikh, M., Tayachi, Z., & Barkaoui, K. (2016). *Workload-dependent software aging impact on performance and energy consumption in server virtualized systems.* In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 111118. IEEE.

Fontana, F. A., Roveda, R., Zanoni, M., Raibulet, C., & Capilla, R. (2016). *An experience report on detecting and repairing software architecture erosion.* In 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 2130. IEEE.

Garg, S., Van Moorsel, A., Vaidyanathan, K., & Trivedi, K. S. (1998). *A methodology for detection and estimation of software aging.* In Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257), pp. 283292. IEEE.

Ghoneim, S. A., & Fahmy, H. (2003). Evaluation of the drm and the time for preventive maintenance for aging software.

Software Quality Journal, 11(1), 5775.

Glass, R. (2011). On the aging of software. *IS Management, 28,* 184-185.

Gross, K. (2012). *Integration of electronic prognostics with software aging and rejuvenation for business-critical enterprise servers.* In IEEE Intnl Workshop on Software Aging and Rejuvenation.

Gross, K. C., Bhardwaj, V., & Bickford, R. (2002). *Proactive detection of software aging mechanisms in performance critical computers.* In 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings., pp. 1723. IEEE.

Grottke, M., Li, L., Vaidyanathan, K., & Trivedi, K. S. (2006). Analysis of software aging in a web server. *IEEE Transactions on reliability, 55*(3), 411420.

Grottke, M., Matias, R., & Trivedi, K. S. (2008). *The fundamentals of software aging.* In 2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp), pp. 16. Ieee.

Grottke, M., & Schleich, B. (2013). How does testing affect the availability of aging software systems? *Performance Evaluation, 70*(3), 179196.

Guo, C., Wu, H., Hua, X., Lautner, D., & Ren, S. (2015). *Use two-level rejuvenation to combat software aging and maximize average resource performance.* In 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp. 11601165. IEEE.

Guo, J., Ju, Y., Wang, Y., Li, X., & Zhang, B. (2010). *The prediction of software aging trend based on user intention.* In 2010 IEEE Youth Conference on Information, Computing and Telecommunications, pp. 206209. IEEE.

Guo, J., Song, X. Y., Wang, Y. S., Zhang, B., & Li, X. L. (2012). The measurement of software aging damage and rejuvenation strategy for discrete web services. *Advanced Materials Research, 433,* 432-437. Trans Tech Publ.

Guo, J., Zhang, G. X., Liu, F., & Zhang, B. (2013). Forecasting web application software aging damage based on user behavior. *Applied Mechanics and Materials, 427,* 2523-2526. Trans Tech Publ.

Gurgel, A., Macia, I., Garcia, A., von Staa, A., Mezini, M., Eichberg, M., & Mitschke, R. (2014). *Blending and reusing rules for architectural degradation prevention.* In Proceedings of the 13th international conference on Modularity, pp. 61-72.

Hannemann, A., & Klamma, R. (2013). *Community dynamics in open source software projects: Aging and social reshaping.* In IFIP International Conference on Open Source Systems, pp. 80-96. Springer.

Hao, Z., & Liu, J. (2020). *Gan-asd: Precise software aging state detection for android system based on began model and state clustering.* In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 212-221. IEEE.

Heimbigner, D. (2005). *Client-side deception using architectural degradation.* In Security and Management, pp. 187193. Citeseer.

Herold, S. (2020). *An initial study on the association between architectural smells and degradation.* In European Conference on Software Architecture, pp. 193-201. Springer.

Herold, S., Buckley, J., & Rausch, A. (2015). *Second workshop on software architecture erosion and architectural consistency (saerocon 2015).* In Proceedings of the 2015 European Conference on Software Architecture Workshops, pp. 12.

Herold, S., Knieke, C., Schindler, M., & Rausch, A. (2020). *Towards improving software architecture degradation mitigation by machine learning.* In The Twelfth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2020), Nice, France, October, 26-29 2020.

Herold, S., & Rausch, A. (2013). *Complementing model-driven development for the detection of software architecture erosion.* In 2013 5th International Workshop on Modeling in Software Engineering (MiSE), pp. 2430. IEEE.

Huang, B., & Qin, Y. (2015). *Software aging analysis based on manmachineenvironment system engineering.* In International Conference on Man-Machine-Environment System Engineering, pp. 681687. Springer.

Huo, S., Zhao, D., Liu, X., Xiang, J., Zhong, Y., & Yu, H. (2018). *Using machine learning for software aging detection in android system.* In 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), pp. 741746. IEEE.

Jaktman, C. B. (1998). *Detecting architectural erosion in an evolving product-line architecture.* In Submitted paper Metrics 98 Symposium.

Jia, S., Hou, C., & Wang, J. (2017). *Software aging analysis and prediction in a web server based on multiple linear regression algorithm.* In 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), pp. 14521456. IEEE.

Jia, Y. F., Chen, X. E., & Cai, K. Y. (2006). *Chaotic analysis of software aging in web server.* In 2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE06), pp. 117-120. IEEE.

Jia, Y. F., Chen, X. E., Zhao, L., and Cai, K. Y. (2008a). *On the relationship between software aging and related parameters (short paper).* In 2008 The Eighth International Conference on Quality Software, pp. 241246. IEEE.

Jia, Y. F., Xu, H., & Wu, R. (2014). A nonlinear dynamic model for software aging in service-oriented software. *J. Softw., 9*(9), 2260-2266.

Jia, Y. F., Zhao, L., & Cai, K. Y. (2008b). *A nonlinear approach to modeling of software aging in a web server.* In 2008 15th AsiaPacific Software Engineering Conference, pp. 7784. IEEE.

Jia, Y.-F., Zhou, Z. Q., & Wu, R. (2019). A feedback control approach for preventing system resource exhaustion caused by software aging. *Journal of Internet Technology, 20*(5), 15131522.

Jiang, L., & Xu, G. (2007). Modeling and analysis of software aging and software failure. *Journal of systems and software, 80*(4), 590-595.

Jun, G., Bo, W., Yunsheng, W., Bin, Z., & Jiaojiao, W. (2011). *Research of the software aging regeneration strategy based on components.* In Proceedings of the 2011, International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011) November 1920, 2011, Melbourne, Australia, pp. 601608. Springer.

Kawamoto, S., & Tashiro, D. (2005). Proposal of dependable j2ee application serverprevention of failure and performance degradation caused by software aging. *IEICE Technical Report; IEICE Tech. Rep., 105*(339), 1924.

Koopman, P., & Shelton, C. (2002). *Using architectural properties to model and measure system-wide graceful degradation.*

Kula, R. G., German, D. M., Ishio, T., Ouni, A., & Inoue, K. (2017). *An exploratory study on library aging by monitoring client usage in a software ecosystem.* In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 407-411. IEEE.

Kwon, K., Lee, S., & Kim, S. (2018). Estimation of aging properties for plastic bonded explosives using akts thermokinetic software. *Journal of the Korean Society of Propulsion Engineers, 22,* 66-71.

Langner, F., & Andrzejak, A. (2013a). *Detecting software aging in a cloud computing framework by comparing development versions.* In 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 896899. IEEE.

Langner, F., & Andrzejak, A. (2013b). *Detection and root cause analysis of memory-related software aging defects by automated tests.* In 2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 365369. IEEE.

Lavallee, M., & Robillard, P. N. (2011). *Causes of premature aging during software development: an observational study.* In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, pp. 6170.

Lei, X., Xue, K., & Jia, Y. (2010). A comprehensive metric to evaluating software aging. *2010 International Conference on Computer Application and System Modeling (ICCASM 2010), 11,* V11518. IEEE.

Li, H. R., Guo, J., Li, W. Y., Zhang, B., & Wang, Y. S. (2011). Dynamic load-balance strategy based on software aging in web server cluster system. *Key Engineering Materials, 460,* 237245. Trans Tech Publ.

Li, J., Qi, Y., & Cai, L. (2018). *A hybrid approach for predicting aging-related failures of software systems.* In 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 96105. IEEE.

Li, J., Qi, Y., Wang, G., & Lin, J. (2020). Planning optimal rejuvenation policy for aging software systems via a two-layer model. *IEEE Access, 8,* 136725136735.

Li, L., He, H., Wang, Q., & Zhang, H. (2017). *Aberrant software-aging server detection and analysis using sliding window over lof.* In 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp.

652656. IEEE.

Li, L., Vaidyanathan, K., & Trivedi, K. S. (2002). *An approach for estimation of software aging in a web server.* In Proceedings International Symposium on Empirical Software Engineering, pp. 91100. IEEE.

Li, R., Liang, P., Soliman, M., & Avgeriou, P. (2022). Understanding software architecture erosion: A systematic mapping study. *Journal of Software: Evolution and Process, 34*(3), e2423.

Li, S., & Yong, Q. (2012). *Software aging detection based on narx model.* In 2012 Ninth Web Information Systems and Applications Conference, pp. 105110. IEEE.

Liang, H., Gu, P., Tang, J., Chen, W., & Gao, F. (2016). Discussion on software aging management of nuclear power plant safety digital control system. *SpringerPlus, 5*(1), 15.

Liu, F., Cao, J., Guo, J., & Zhang, B. (2013). Research the measurement method of software aging in cloud. *Applied Mechanics and Materials, 392,* 779782. Trans Tech Publ.

Liu, J., & Meng, L. (2019). Integrating artificial bee colony algorithm and bp neural network for software aging prediction in iot environment. *IEEE Access, 7,* 3294132948.

Liu, J., Tan, X., & Wang, Y. (2019). *Cssap: software aging prediction for cloud services based on arima-lstm hybrid model.* In 2019 IEEE International Conference on Web Services (ICWS), pp. 283290. IEEE.

Macedo, A., Ferreira, T. B., & Matias, R. (2010). *The mechanics of memory-related software aging.* In 2010 IEEE Second International Workshop on Software Aging and Rejuvenation, pp. 15. Ieee.

Macedo, R. M. B. E. A. *Monitoring memory-related software aging: An exploratory study.*

Machida, F., Andrzejak, A., Matias, R., & Vicente, E. (2013). *On the effectiveness of mann-kendall test for detection of software aging.* In 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 269274. IEEE.

Machida, F., Nicola, V. F., & Trivedi, K. S. (2011). *Job completion time on a virtualized server subject to software aging and rejuvenation.* In 2011 IEEE Third International Workshop on Software Aging and Rejuvenation, pp. 4449. IEEE.

Machida, F., Xiang, J., Tadano, K., & Maeno, Y. (2012a). *Aging-related bugs in cloud computing software.* In 2012 IEEE 23rd international symposium on software reliability engineering workshops, pp. 287292. IEEE.

Machida, F., Xiang, J., Tadano, K., & Maeno, Y. (2012b). *Software life-extension: a new countermeasure to software aging.* In 2012 IEEE 23rd International Symposium on Software Reliability Engineering, pp. 131140. IEEE.

Machida, F., Xiang, J., Tadano, K., & Maeno, Y. (2016). Lifetime extension of software execution subject to aging. *IEEE Transactions on Reliability, 66*(1), 123134.

Magalhaes, J. P., & Silva, L. M. (2010). *Prediction of performance anomalies in web-applications based-on software aging scenarios.* In 2010 IEEE second international workshop on software aging and rejuvenation, pp. 17. IEEE.

Mair, M., & Herold, S. (2013). *Towards extensive software architecture erosion repairs.* In European Conference on Software Architecture, pp. 299306. Springer.

Mair, M., Herold, S., & Rausch, A. (2014). *Towards flexible automated software architecture erosion diagnosis and treatment.* In Proceedings of the WICSA 2014 Companion Volume, pp. 16.

Marmsoler, D., & Petrovska, A. (2019). *Detecting architectural erosion using runtime verification.* arXiv preprint arXiv:1909.05973.

Matias, R., Andrzejak, A., Machida, F., Elias, D., & Trivedi, K. (2014). *A systematic differential analysis for fast and robust detection of software aging.* In 2014 IEEE 33rd International Symposium on Reliable Distributed Systems, pp. 311320. IEEE.

Matias, R., Barbetta, P. A., Trivedi, K. S., & Freitas Filho, P. J. (2009). Accelerated degradation tests applied to software aging experiments. *IEEE Transactions on reliability, 59*(1), 102114.

Matias, R., Beicker, I., Leitao, B., & Maciel, P. R. (2010). *Measuring software aging effects through os kernel instrumentation.* In 2010 IEEE Second International Workshop on Software Aging and Rejuvenation, pp. 16. IEEE.

Matias, R., Costa, B. E., & Macedo, A. (2012). *Monitoring memory-related software aging: An exploratory study.* In 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops, pp. 247252. IEEE.

Matias, R., de Sena, G. O., Andrzejak, A., & Trivedi, K. S. (2016). *Software aging detection based on differential analysis: an experimental study.* In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 7177. IEEE.

Matias, R., & Paulo Filho, J. (2006). *An experimental study on software aging and rejuvenation in web servers.* In 30th Annual International Computer Software and Applications Conference (COMPSAC06), volume 1, pp. 189196. IEEE.

Matias Jr, R. & Trivedi, K. S. (2011). *Accelerated life tests and software aging.* In Adaptive Control Approach For Software Quality Improvement, pp. 267299. World Scientific.

Matias Jr, R., Trivedi, K. S., & Maciel, P. R. (2010). *Using accelerated life tests to estimate time to software aging failure.* 2010 IEEE 21st International Symposium on Software Reliability Engineering, pp. 211219. IEEE.

Matos, R., Araujo, J., Alves, V., & Maciel, P. (2012a). *Characterization of software aging effects in elastic storage mechanisms for private clouds.* In 2012 IEEE 23rd international symposium on software reliability engineering workshops (ISSREW), pp. 293298. IEEE Computer society.

Matos, R., Araujo, J., Alves, V., & Maciel, P. (2012b). *Experimental evaluation of software aging effects in the eucalyptus elastic block storage.* In 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 11031108. IEEE.

Medvidovic, N., Egyed, A., & Gruenbacher, P. (2003). Stemming architectural erosion by coupling architectural discovery and recovery. *STRAW, 3,* 6168.

Melo, C., Araujo, J., Alves, V., & Maciel, P. R. M. (2017). Investigation of software aging effects on the openstack cloud computing platform. J. Softw., 12(2), 125137.

Meng, H., Hei, X., Zhang, J., Liu, J., & Sui, L. (2016). Software aging and rejuvenation in a j2ee application server. *Quality and Reliability Engineering International, 32*(1), 8997.

Meng, H. N., Qi, Y., Hou, D., & Chen, Y. (2007a). *Forecasting software aging of service-oriented application server based on wavelet network with adaptive genetic algorithm.* In Fourth International Conference on Autonomic Computing (ICAC07), pp. 1010. IEEE.

Meng, H. N., Qi, Y., Hou, D., Zhang, Y., & Liu, L. (2007b). Software aging forecasting model of service-oriented application server based on wavelet network with adaptive genetic algorithm. *Third International Conference on Natural Computation (ICNC 2007), 3,* 353357. IEEE.

Merkle, B. (2009). *Stopping (and reversing) the architectural erosion of software systems. an industrial case study.* Software Engineering 2009.

Merkle, B. (2010). *Stop the software architecture erosion: building better software systems.* In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, pp. 129138.

Miller, L., Johnson, L., Ning, J. Q., Quilici, A., & Devanbu, P. (1992). *Program understanding-does it offer hope for aging software?* In Proceedings of the Seventh Knowledge-Based Software Engineering Conference, pp. 238239. IEEE Computer Society.

Mirakhorli, M. (2013). *Preventing erosion of architectural tactics through their strategic implementation, preservation, and visualization.* In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 762765. IEEE.

Mirakhorli, M., & Cleland-Huang, J. (2011). *Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance.* In 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp. 123132. IEEE.

Mitra, D., Arora, M., Rakhra, M., Kumar, C. R., Reddy, M. L., Reddy, S. P. K., Kumar, C., & Shabaz, M. (2021). *A hybrid framework to control software architecture erosion for addressing maintenance issues.* Annals of the Romanian Society for Cell Biology, pp. 29742989.

Mohan, B. R., & Reddy, G. R. M. (2014). *Software aging trend analysis of server virtualized system.* In The International Conference on Information Networking 2014 (ICOIN2014), pp. 260263. IEEE.

Mohan, B. R., & Reddy, G. R. M. (2015). *The effect of software aging on power usage.* In 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), pp. 13. IEEE.

Monden, A., Sato, S. I., Matsumoto, K. I., & Inoue, K. (2000). *Modeling and analysis of software aging process.* In International Conference on Product Focused Software Process Improvement, pp. 140153. Springer.

Morshidi, M. K., Nor, R. N. H., & Mahdin, H. (2017). Initial review on icts governance for software anti-aging. *JOIV: International Journal on Informatics Visualization, 1*(4-2), 232235.

Natella, R., & Andrzejak, A. (2020). *Experimental tools for software aging analysis.* In Handbook Of Software Aging And Rejuvenation: Fundamentals, Methods, Applications, And Future Directions, pp. 285326. World Scientific.

Neto, V. V. G., Manzano, W., Garces, L., Guessi, M., Oliveira, B., Volpato, T., & Nakagawa, E. Y. (2018). *Back-sos: towards a model-based approach to address architectural drift in systems-of-systems.* In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, pp. 14611463.

Ning, M. H., Yong, Q., Di, H., & Ying, C. (2007). *Forecasting software aging of service-oriented application server based on wavelet network with adaptive genetic algorithm.* In Fourth International Conference on Autonomic Computing (ICAC07).

Ning, M. H., Yong, Q., Di, H., Ying, C., & Zhong, Z. J. (2006). *Software aging prediction model based on fuzzy wavelet network with adaptive genetic algorithm.* In 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI06), pp. 659666. IEEE.

Okamura, H., Luo, C., & Dohi, T. (2013). *Estimating response time distribution of server application in software aging phenomenon.* In 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 281284. IEEE.

Okamura, H., Zheng, J., & Dohi, T. (2017). *A statistical framework on software aging modeling with continuous-time hidden markov model.* In 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), pp. 114123. IEEE.

Oliveira, F., Araujo, J., Matos, R., Lins, L., Rodrigues, A., & Maciel, P. (2020). *Experimental evaluation of software aging effects in a container-based virtualization platform.* In 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 414419. IEEE.

Oliveira, F., Araujo, J., Matos, R., & Maciel, P. (2021). *Software aging in container-based virtualization: an experimental analysis on docker platform.* In 2021 16th Iberian Conference on Information Systems and Technologies (CISTI), pp. 17. IEEE.

OReilly, C., Morrow, P., & Bustard, D. (2003). *Lightweight prevention of architectural erosion.* In Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings., pp. 5964. IEEE.

Paing, A. M. M. (2020). *Analysis of availability model based on software aging in sdn controllers with rejuvenation.* In 2020 IEEE Conference on Computer Applications (ICCA), pp. 17. IEEE.

Parnas, D. L. (1994). *Software aging.* In Proceedings of 16th International Conference on Software Engineering, pp. 279287. IEEE.

Patel, J., & Das, O. (2007). *A new model for evaluating performability under the effects of software aging and rejuvenation.* In Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, pp. 2530.

Pietrantuono, R., & Russo, S. (2018). *Software aging and rejuvenation in the cloud: a literature review.* In 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 257263. IEEE.

Pietrantuono, R., & Russo, S. (2020). A survey on software aging and rejuvenation in the cloud. *Software Quality Journal, 28*(1), 738.

Qiao, Y., Zheng, Z., & Fang, Y. (2018). *An empirical study on software aging indicators prediction in android mobile.* In 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 271277. IEEE.

Qiao, Y., Zheng, Z., & Qin, F. (2016). *An empirical study of software aging manifestations in android.* In 2016 IEEE international symposium on software reliability engineering workshops (ISSREW), pp. 8490. IEEE.

Raju, R., Sumathi, G., & Vidhyashankardevi, J. (2012). Processing software aging through swarm intelligence. *Software Engineering and Technology, 4*(4).

Rosik, J., Le Gear, A., Buckley, J., Babar, M. A., & Connolly, D. (2011). Assessing architectural drift in commercial software development: a case study. *Software: Practice and Experience, 41*(1), 6386.

Russo, S. (2014). *The dual nature of software aging: Twenty years of software aging research.* In 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pp. 431432. IEEE.

Sabino, M. E., Merabti, M., Llewellyn-Jones, D., & Bouhafs, F. (2013). *Detecting software aging in safety-critical infrastuctures.* In 2013 Science and Information Conference, pp. 7885. IEEE.

Saravakos, P., Gravvanis, G., Koutras, V., & Platis, A. (2009). *A comprehensive approach to software aging and rejuvenation on a single node software system.* In Proceedings of the 9th Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA 2009).

Shaikh, A. S., & Sangani, S. (2019). *Software aging analysis and prediction using aknn algorithm.* In 2019 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), pp. 16. IEEE.

Shao, Q., Gou, X., Huang, T., & Yang, S. (2020). Antiaging analysis for software reliability design modes in the context of singleevent effect. *Software Quality Journal, 28*(1), 221243.

Sharma, R., & Kumar, G. (2019). Availability modelling of cluster-based system with software aging and optional rejuvenation policy. *Cybernetics and Information Technologies, 19*(4), 90100.

Sharma, S., & Kumar, S. (2018). *Analysis of ensemble models for aging related bug prediction in software systems.* In ICSOFT, pp. 290297.

Shelton, C., & Koopman, P. (2003). *Using architectural properties to model and measure graceful degradation.* In Architecting dependable systems, pp. 267289. Springer.

Shereshevsky, M., Crowell, J., Cukic, B., Gandikota, V., & Liu, Y. (2003). Software aging and multifractality of memory resources. *DSN, 3,* 721730.

Shruthi, P., & Cholli, N. G. (2020). An analysis of software aging in cloud environment. *International Journal of Electrical and Computer Engineering (IJECE), 10*(6), 59855991.

Silva, L., Madeira, H., & Silva, J. G. (2006). *Software aging and rejuvenation in a soap-based server.* In Fifth IEEE International Symposium on Network Computing and Applications (NCA06), pp. 5665. IEEE.

Song, J. Y., & Jang, J. S. (2005). *Aging test and software reliability analysis method for pc-based controller.* In Proceedings of the Korean Society of Precision Engineering Conference, pp. 969973. Korean Society for Precision Engineering.

Strasser, A., Cool, B., Gernert, C., Knieke, C., Korner, M., Niebuhr, D., Peters, H., Rausch, A., Brox, O., & Jauns-Seyfried, S., et al. (2014). *Mastering erosion of software architecture in automotive software product lines.* In International Conference on Current Trends in Theory and Practice of Informatics, pp. 491502. Springer.

Su, L., Chen, P., Qi, Y., & Wu, Y. (2013). A software aging detection method based on bayesian framework using least squares support vector machine. *Journal of Xi an Jiaotong University, 47*(8), 12-18.

Sukhwani, H., Matias, R., Trivedi, K. S., & Rindos, A. (2017). *Monitoring and mitigating software aging on ibm cloud controller system.* In 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 266272. IEEE.

Sumathi, G., & Raju, R. (2012). *Software aging analysis of web server using neural networks.* arXiv preprint arXiv:1206.1534.

Sumathi, G., Raju, R., & Shankardevi, J. V. (2012). Processing software aging analysis of web server through machine learning. *Software Engineering and Technology, 4*(7).

Tayachi, Z., Escheikh, M., & Barkaoui, K. (2019). Performability modelling and analysis of server virtualised systems subject to workload-dependent software aging. *International Journal of Critical Computer-Based Systems, 9*(3), 248292.

Tekinerdogan, B. (2016). *Architectural drift analysis using architecture reflexion viewpoint and design structure reflexion matrices.* In Software Quality Assurance, pp. 221236. Elsevier.

Terra, R., Valente, M. T., Czarnecki, K., & Bigonha, R. S. (2012). *Recommending refactorings to reverse software architecture erosion.* In 2012 16th European Conference on Software Maintenance and Reengineering, pp. 335340. IEEE.

Torquato, M., Araujo, J., Umesh, I., & Maciel, P. (2018). Sware: a methodology for software aging and rejuvenation experiments. *Journal of Information Systems Engineering and Management, 3*(2), 15.

Torquato, M., Maciel, P., Araujo, J., & Umesh, I. (2017). *An approach to investigate aging symptoms and rejuvenation effectiveness on software systems.* In 2017 12th iberian conference on Information systems and technologies (CISTI), pp. 16. IEEE.

Torquato, M., & Vieira, M. (2019). *An experimental study of software aging and rejuvenation in dockerd.* In 2019 15th European Dependable Computing Conference (EDCC), pp. 16. IEEE.

Trivedi, K. (2019). *Software aging and software rejuvenation: Keynote.* In Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, pp. 11.

Trivedi, K. S., & Alonso, J. (2011). *Software aging and rejuvenation part ii of ii.*

Trivedi, K. S., Vaidyanathan, K., & GosevaPopstojanova, K. (2000). *Modeling and analysis of software aging and rejuvenation.* In Proceedings 33rd annual simulation symposium (SS 2000), pp. 270279. IEEE.

Umesh, I., & Srinivasan, G. (2016). Optimum software aging prediction and rejuvenation model for virtualized environment. *Indonesian Journal of Electrical Engineering and Computer Science, 3*(3), 572578.

Umesh, I., Srinivasan, G., & Torquato, M. (2017). Software aging forecasting using time series model. *Indonesian Journal of Electrical Engineering and Computer Science, 7*(3), 839845.

Umesh, I., & Srinivasan, G. N. (2017). *Dynamic software aging detection-based fault tolerant software rejuvenation model for virtualized environment.* In Proceedings of the International Conference on Data Engineering and Communication Technology, pp. 779787. Springer.

Vaidyanathan, K., & Gross, K. (2003). *Mset performance optimization for detection of software aging.* In Proc. of ISSRE. Citeseer.

Vaidyanathan, K., & Trivedi, K. S. (2001). *Extended classification of software faults based on aging.* In Fast Abstract, Int. Symp. Software Reliability Eng., Hong Kong. Citeseer.

Valentim, N. A., Macedo, A., & Matias, R. (2016). *A systematic mapping review of the first 20 years of software aging and rejuvenation research.* In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 5763. IEEE.

Weng, C., Xiang, J., Xiong, S., Zhao, D., & Yang, C. (2016). *Analysis of software aging in android.* In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 7883. IEEE.

Whiting, E., & Andrews, S. (2020). *Drift and erosion in software architecture: Summary and prevention strategies.* In Proceedings of the 2020 the 4th International Conference on Information System and Data Mining, pp. 132138.

Wu, H., & Wolter, K. (2015). *Software aging in mobile devices: Partial computation offloading as a solution.* In 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 125131. IEEE.

Wu, Q., Han, Z., & Guo, T. (2009). *Application of an uncertain reasoning approach to software aging detection.* In 2009 Fifth International Joint Conference on INC, IMS and IDC, pp. 15921597. IEEE.

Wu, Q., Qi, Y., Du, X., & Han, Z. (2010). *A new rejuvenation approach of software aging.* In 2010 International Conference on Optoelectronics and Image Processing, volume 1, pp. 7174. IEEE.

Wu, X., Zheng, W., Pu, M., Chen, J., & Mu, D. (2020). Invalid bug reports complicate the software aging situation. *Software Quality Journal, 28*(1), 195220.

Xiang, J., Weng, C., Zhao, D., Andrzejak, A., Xiong, S., Li, L., & Tian, J. (2020). Software aging and rejuvenation in android: new models and metrics. *Software Quality Journal, 28*(1), 85106.

Xiao-zhi, D., Yong, Q., & Hui-min, L., et al. (2011). Software aging pattern analysis of the video on demand system. *Journal of Computer Research and Development, 48*(11), 21392146.

Xu, J., You, J., & Zhang, K. (2005). *A neural-wavelet based methodology for software aging forecasting.* In 2005 IEEE International Conference on Systems, Man and Cybernetics, volume 1, pp. 5963. IEEE.

Yakhchi, M., Alonso, J., Fazeli, M., Bitaraf, A. A., & Patooqhy, A. (2015). Neural network based approach for time to crash prediction to cope with software aging. *Journal of Systems Engineering and Electronics, 26*(2), 407-414.

Yakovyna, V., & Uhrynovskyi, B. (2020). *User-perceived response metrics in android os for software aging detection.* In 2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), volume 1, pp. 436439. IEEE.

Yan, Y. (2018). Performance analysis of software aging prediction. *International Journal of Performability Engineering, 14*(11), 2692.

Yan, Y. (2019). Novel method to forecast software aging problems. *The Journal of Engineering, 2019*(10), 72377243.

Yan, Y. (2020). Software aging forecast using recurrent som with local model. *Journal of Information Technology Research (JITR), 13*(1), 30-43.

Yan, Y., & Guo, P. (2016). A practice guide of software aging prediction in a web server based on machine learning. *China Communications, 13*(6), 225235.

Yan, Y., Guo, P., & Liu, L. (2014). *A practice of forecasting software aging in an iis web server using svm.* In 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pp. 443448. IEEE.

Yan, Y., Li, Y., & Cheng, B. (2021). Predicting software aging with a hybrid weight-based method. *Journal of Information Technology Research (JITR), 14*(4), 58-69.

Yang, H., Liang, Y., Tan, C., & Fu, J. (2011). *Detecting software aging of web servers with real-valued negative selection algorithm.* In 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 298302. IEEE.

Yang, T., Wu, Q., & Bao, J. (2012). *Research of a resource to influence on the software aging and rejuvenation cycle.* In 2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 13491351. IEEE.

Yoo, Y., & Yoo, W. S. (2021). Digital image comparisons for investigating aging effects and artificial modifications using image analysis software. *Journal of Conservation Science, 37*(1), 1-12.

Zhang, L., Sun, Y., Song, H., Chauvel, F., & Mei, H. (2011). *Detecting architecture erosion by design decision of architectural pattern.* In SEKE, pp. 758763.

Zhao, J., Trivedi, K. S., Wang, Y., & Chen, X. (2010). *Evaluation of software performance affected by aging.* In 2010 IEEE Second International Workshop on Software Aging and Rejuvenation, pp. 16. IEEE.

Zhao, J. F. (2016). *Modeling of software aging based on nonstationary time series.* In 2016 International Conference on Information System and Artificial Intelligence (ISAI), pp. 176180. IEEE.

Zhao, L., Song, Q., & Zhu, L. (2008). *Common softwareaging-related faults in fault-tolerant systems.* In 2008 International conference on computational intelligence for modelling control & automation, pp. 327331. IEEE.

Zhao, Y., Xiang, J., Xiong, S., Wu, Y., An, J., Wang, S., & Yu, X. (2015). *An experimental study on software aging in android operating system.* In 2015 2nd International Symposium on Dependable Computing and Internet of Things (DCIT), pp. 148150. IEEE.

Zheng, P., Xu, Q., & Qi, Y. (2012). *An advanced methodology for measuring and characterizing software aging.* In 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops, pp. 253258. IEEE.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).