The Software Aging and Rejuvenation Repository

http://openscience.us/repo/software-aging/

Domenico Cotroneo, Antonio Ken Iannillo, Roberto Natella, Roberto Pietrantuono, Stefano Russo
Università degli Studi di Napoli Federico II, Naples, Italy
{cotroneo, antonioken.iannillo, roberto.natella, roberto.pietrantuono, sterusso}@unina.it

***Abstract*—While Software Aging and Rejuvenation (SAR) research has been steadily increasing, the *artifacts* related to SAR studies (such as software aging measurements and bug datasets) are seldom made available to researchers and practitioners, thus limiting potential improvements of rejuvenation solutions and their practical adoption. We discuss in this paper the role of artifacts in SAR research, and present *SARRY* (the Software Aging and Rejuvenation RepositorY), an open-access support for the SAR community to share research artifacts (available at http://openscience.us/repo/software-aging/). We invite researchers to contribute to SARRY, in order to aid future SAR research and to improve the visibility and impact of their work.**

***Keywords*—Software Aging and Rejuvenation; Research Artifacts; Open Access; Data Repository**

I. Introduction

Several researchers and practitioners have been addressing the phenomenon of *software aging* since the seminal paper “Software rejuvenation: Analysis, module and applications” by Huang *et al.* [1], which was published two decades ago. Research on Software Aging and Rejuvenation (SAR) has steadily increased, gaining attention in several international conferences and journals [2], [3], and led to the WoSAR workshop series [4] and to software rejuvenation solutions deployed in several commercial and open-source systems [5].

Therefore, existing SAR research provides a solid foundation for new researchers and practitioners that want to enhance and put into practice software rejuvenation solutions. Past experiences with software aging are valuable for both *measurement-based* and *model-based* SAR research. For measurement-based solutions, experimental data on resource consumption and performance are required to design and to validate techniques for detecting and forecasting software aging. For model-based solutions, experimental data can support rejuvenation scheduling models, whose applicability is often criticized by practitioners due to the lack of realistic parameters. SAR research and innovation processes would significantly benefit from artifacts produced by previous research, such as software aging measurements and rejuvenation scheduling models. However, new researchers and practitioners must face the scarcity of artifacts from previous SAR research, and they are forced to build new datasets for supporting their efforts, or they have to base their projects on the limited information available in research papers.

To address this gap, our goal is to promote the sharing of artifacts developed by the SAR research community. The sharing of artifacts is a common practice among several areas of science, including computer science and software engineering, to improve the reproducibility of results and to foster new research. Several initiatives are raising awareness of these issues, encouraging researchers to make computer code and data publicly accessible to reuse and to validate the results [6], [7].

Some notable examples of such initiatives are: the *Software-artifacts Infrastructure Repository* (SIR) [8], which supports testing research by providing programs, test cases and real software bugs; the *Amber Data Repository* (ADR) [9], which hosts data from field failure analyses and fault injection experiments; the *Artifact Evaluation Committees* (AECs) [10] established at top computer science conferences to publish experimental data, complete experimental setups, test suites, and tools; and the *Promise Repository of Empirical Software Engineering Data* (PROMISE) [11], which was born for sharing datasets for software defect prediction research, and then expanded to host other artifacts for effort estimation, requirement analysis, testing, social analysis and other types of software engineering research.

We present in this paper *SARRY*, the Software Aging and Rejuvenation RepositorY, which is an open-access support for the SAR community to share research artifacts and to aid future research. We believe that this is a good opportunity both for individual SAR researchers, whose research may achieve a broader visibility and impact, and for the SAR community, which may benefit from previous efforts and build new solutions upon them. Thus, we invite other SAR researchers to join us in this endeavor, by contributing to SARRY with research artifacts, and by referring the reader to these artifacts in their research studies.

In the following of this paper, we first discuss the role of artifacts in SAR research, by providing notable examples that have helped in the past to develop new studies and products (Section II). Then, we provide an overview of the SARRY repository (Section III). Section IV concludes the paper.

II. SAR Artifacts

In software engineering, an artifact is a tangible piece of information that is used or produced by a process. In our case, we define a SAR artifact as a product of SAR research that can be useful for researchers, including experimental data, software, and system models. We analyzed previous SAR research, and identified the following main categories of SAR artifacts for SARRY:

- Measurements on performance, resource consumption and failures collected at run-time from **deployed software systems**;

- Measurements on performance, resource consumption and failures collected during **stress tests of software systems**;
- Datasets on **software aging bugs** in software systems, and their software complexity metrics;
- **Models** for rejuvenation scheduling.

In the following of this section, we analyze each of these categories, by answering the following questions:

- ARTIFACT NATURE: what artifacts can be derived and shared from SAR research?
- RELATED STUDIES: which SAR research studies have produced artifacts of this kind?
- DERIVED STUDIES: how did these artifacts help other SAR research?

A. Measurements on performance, resource consumption and failures collected at run-time from deployed software systems

This category includes datasets with measurements collected from software systems in production. These systems are exercised by the workload imposed by their end-users. Measurements are usually collected using existing monitoring tools, which are installed in the system for troubleshooting purposes.

Measurements collected during the operational phase are an excellent way to analyze the actual behaviour of deployed software systems, since the measurements reflect the breadth and the variability (e.g., seasonal variations) of real workloads. Researchers can use these data to validate and to benchmark techniques for detecting and forecasting aging phenomena in a real environment.

ARTIFACT NATURE: These artifacts consist of one or more time series, where each time series represents the usage of a resource, a performance or workload indicator, or a failure symptom. Examples of relevant measurements are: used/free memory, used/free swap space, process states, occupation of network queues, I/O device throughput, used/free inodes, CPU usage. Each sample in the time series is marked with the current timestamp. These data can be saved in a database or dumped on a file, using a format such as CSV (Comma-Separated Values) or ARFF (Attribute Relationship File Format), where each row represents a sample of a time series.

RELATED STUDIES: Measurements of this kind have been used for investigating statistical trend analysis techniques, to characterize and forecast software aging trends. Garg *et al.* [12] collected system data using SNMP every fifteen minutes, for determining the “health” of the eight heterogeneous UNIX workstations during a period of 53 days. Fig. 1 shows an example of analysis applied on these data: The points represent the amount of free physical memory, which gradually decreases, and the size of the process table, which gradually increases. These data were smoothed to show the trend of resource consumption. Moreover, the data collected in this study provide evidence of aging-related failures caused by resource exhaustion (e.g., *Out-Of-Memory* and *swap space exhaustion* failures). Finally, this study used the *seasonal Kendall test* to detect, with statistical confidence, the presence of a trend in the measurements, and the *Sen procedure* for robust linear regression to forecast the *Time To Exhaustion* (TTE) of system resources.

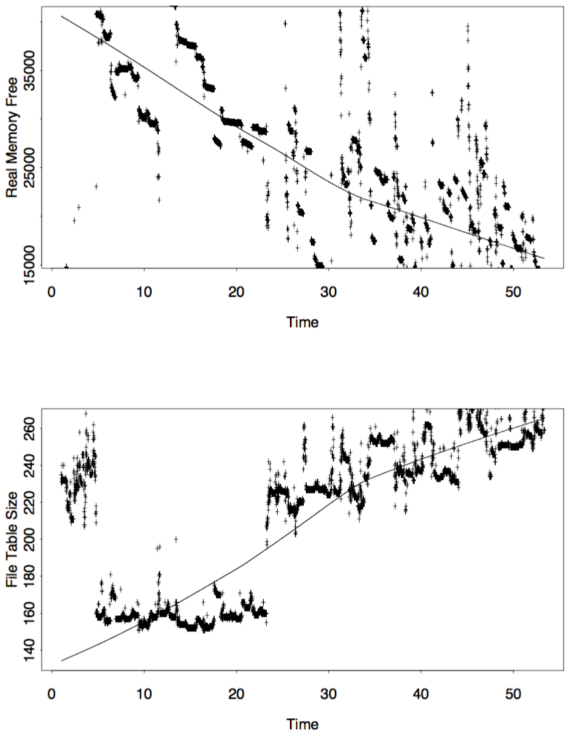


Fig. 1. Resource consumption measurements (free memory and file table size) from a SunOS workstation, and non-parametric regression smoothing to visualize trends [12].

DERIVED STUDIES: Vaidyanathan *et al.* [13] adopted a workload-based approach to improve the forecasting of the time to exhaustion. Their approach uses a clustering algorithm to identify workload clusters, where each cluster represents a state in a semi-Markov reward model, and each cluster is individually analyzed and is attributed a resource consumption rate. The approach has been validated using the measurements collected by Garg *et al.* [12]. Measurements under real workload conditions were important to support the claim that the resource consumption rate varies with the workload, and to accurately evaluate how the new algorithm improves the estimation of the TTE.

B. Measurements on performance, resource consumption and failures collected from stress tests of software systems

Data from production systems are often not available, such as in the case of new systems or technologies under development that have not been yet widely deployed. Thus, researchers often adopt data obtained from laboratory tests, in which the system is put under stressful conditions in order to trigger aging phenomena. The tests use synthetic workloads that stress the system with a constant workload, whose type and volume can vary across experiments. These tests are useful to analyze the relation between workload and software aging, and to investigate aging forecasting and rejuvenation scheduling.

This kind of data are complementary to the data described in Section II-A. Stress tests do not reflect the actual usage of the system by end-users, and thus they may over- or under-represent some workload conditions that are experienced in production. However, the workload is now a controllable factor of stress tests, and it is thus fully known and tunable by the experimenter. For example, the experimenter can evaluate the sensitivity of the system to different types or volumes of workload. Moreover, similarly to the studies discussed in Section II-A, these measurements can be used to investigate techniques for aging detection and forecasting.

ARTIFACT NATURE: Artifacts with this kind of measurements are similar to the ones discussed in Section II-A. Since these data are collected in a laboratory setting, they usually include a more rich set of monitored metrics. Moreover, the synthetic workloads represent a worst-case workload for the system. Therefore, the aging trends in stress testing experiments are more numerous and more pronounced than aging trends experienced in production systems.

RELATED STUDIES: A synthetic workload has been used by Grottke *et al.* [14] to study resource consumption trends for an Apache HTTPD web server, which was stressed at its maximum capacity over a period of 25 days. During the experiment, data on response time, free physical memory, and used swap space were collected. This study further developed the statistical techniques adopted by Garg *et al.* [12] for detecting aging trends with statistical hypothesis testing, and for estimating the TTE with robust linear regression, which confirmed the existence of software aging phenomena in this system. The measurements exhibited noticeable seasonal patterns, which were due to periodical events in the system (in particular, the automated restart of child processes, and the scheduled maintenance for log rotation and filesystem indexing). Therefore, the study developed an autoregressive model to forecast resource consumption in the presence of seasonal patterns. Fig. 2 shows the usage of swap memory over time, and the trend prediction made by the autoregressive model. It is also worth to mention another study on software aging in the Apache HTTPD web server by Matias jr. and Filho [15], which stressed this system by considering several workloads, by varying the size of requested pages, the type of page (static or dynamic), and the request rate. They used the *design of experiments* methodology to identify the degree of influence of these factors on aging trends.

DERIVED STUDIES: El-Shishiny *et al.* [16] further developed techniques for aging forecasting, using the data from the Apache HTTPD experiment that were kindly shared by Grottke *et al.* [14]. This study proposed the use of Artificial Neural Networks for predicting resource consumption, in order to provide a more general and powerful identification of patterns in the time series. The data from [14] were used to train and to evaluate the accuracy of the model in [16], which was able to fit the time series with a low error.

C. Datasets on software aging bugs and complexity metrics

These artifacts provide information on the root cause of software aging. A software system manifests software aging when a special type of software faults, called "Aging-Related Bugs" (ARBs), is triggered. According to Grottke *et al.*, an ARB is a fault that decreases performance and/or increases the failure rate in long-running software [17]. Software rejuvenation is a practical and effective technique to proactively mitigate software aging effects, by scheduling rejuvenation actions when the maintenance downtime is lower than the failure downtime. Moreover, if ARBs are removed before deployment (e.g., by performing stress tests), the downtime can be further reduced. In order to support research on ARB-oriented testing, it is useful to share samples of these faults, along with contextual information, such as software complexity metrics from the faulty software. Studies on ARBs could use these data to predict aging-prone components, and to plan testing activities by focusing on these components.

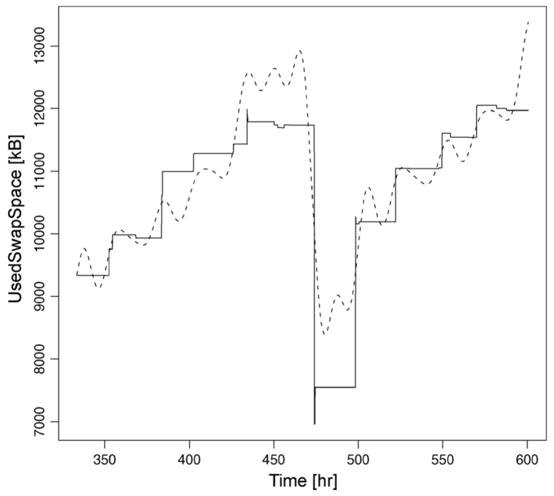


Fig. 2. Used swap space in a stress test experiment on Apache HTTPD, and an autoregressive model fitted on the data (dashed line) [14].

ARTIFACT NATURE: Information on ARBs of complex software projects can be obtained by analyzing *bug reports* issued by users and developers. These bug reports can be retrieved from the issue trackers of both proprietary and open-source projects. Artifacts of this category are datasets that classify bug reports according to the conditions that trigger the bug, to identify ARBs among the bug reports. The artifacts can also provide further information about the nature of the fault, such as the *resource* affected by the ARB, the *location* of the fault in the code base (e.g., a file, a class, a package, or a module affected by the fault), and the *complexity* of the component containing the fault (in terms of number of statements, paths, external dependencies, etc.). These data can be stored in a file formatted as CSV or ARFF, and containing one row per bug, or one row per software module, and their related information.

RELATED STUDIES: Cotroneo *et al.* [18] investigated how to predict the location of ARBs in order to support verification activities, and considered ARBs found in three complex software systems, namely the Linux kernel, the MySQL DBMS, and the CARDAMOM middleware. They first identified ARBs by manually analyzing the bug reports from these projects. Then, they extracted a set of software complexity metrics (both traditional metrics, such as McCabe's and Halstead's, and "aging-related" metrics proposed in the study) for each file and module in these projects, in order to build an ARB dataset. Finally, they used this dataset to train and to validate *machine learning* algorithms for the prediction of ARB-prone locations. The prediction has been validated by training the predictor using data respectively from the same component, from other components of the same project, and from external projects. The experimental evaluation allowed to assess the potentialities and the limitations of bug prediction: while intra-project prediction is feasible (especially when using aging-related metrics), inter-project prediction can be inaccurate when projects are heterogeneous.

The classification of ARBs was refined in [19]. ARBs (along with other types of Mandelbugs) in open-source projects were further classified in subtypes, by considering the type of software aging symptom: accumulation of memory management errors (e.g., memory leaks), accumulation of storage errors (e.g., disk block leaks), accumulation of system-dependent resources (e.g., sockets), accumulation of numerical errors, and other bugs with no accumulation but influenced by the total system runtime. The study analyzed the occurrence of ARBs over time, and the relationships between, on the one hand, the bug type/subtype and, on the other hand, the type of project, the severity, and the time-to-fix. Besides the insights on real ARBs, and the verification and fault-tolerance strategies to address them, this study provided a dataset for future analysis on ARBs, including, but not limiting to, defect classification and prediction approaches.

DERIVED STUDIES: The data on ARBs from open-source projects [18], [19] were used by Xia *et al.* [20] to develop and validate an approach for the *automated classification* of bug reports, by using a text mining algorithm to analyze the description of the bug in natural language. This approach creates a set of terms suitable to classify a bug into a Bohrbug or Mandelbug, and builds a classifier on these terms. The dataset from open-source projects was used to identify the classifier and the terms which achieve the best performance.

Another type of derived studies is represented by model-based studies on software systems that are affected by Mandelbugs and ARBs. Two studies of this type by Grottke *et al.* [21], [22] analyzed, respectively, the availability obtained with recovery strategies to counteract Mandelbugs [22], and the availability achieved when considering both testing and software rejuvenation [21]. These models are apt to be configured with parameters that reflect the relative frequency of Mandelbugs and ARBs, and of failures caused by these bugs, which can be provided by empirical studies on bug analysis.

D. Models for rejuvenation scheduling

This category includes artifacts from model-based SAR research. This research uses models of the system behaviour, and solves these models to tune software rejuvenation (i.e., the optimal rejuvenation schedule) and to evaluate its effectiveness. The focus is on modelling aging phenomena, in order to achieve an accurate failure prediction, and to suggest practitioners when to plan rejuvenation. Sharing these models is useful for researchers to better understand the state-of-the-art, to configure and use the model for specific systems, and to improve them.

ARTIFACT NATURE: In this case, the artifacts are the *models* used for the analysis and prediction of the system

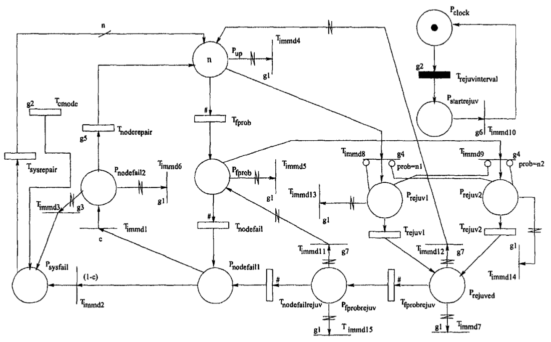
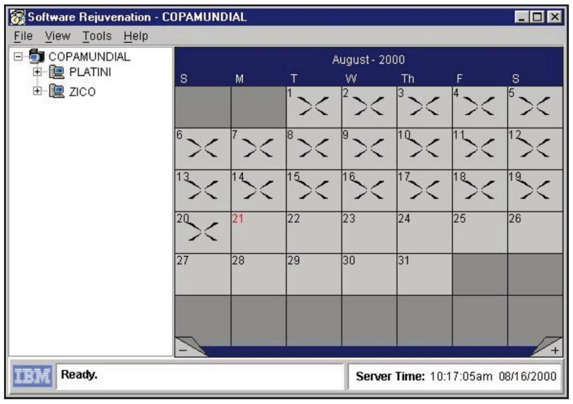


Fig. 3. Cluster system adopting time-based rejuvenation [25].



Ready.

Server Time: 10:17:05am 08/16/2000

Fig. 4. Software rejuvenation calendar [26].

behaviour, including aging failures. The most common models are based on Markov chains and Petri nets. Markov chains are mainly used to analyse a system with multiple rejuvenation strategies, or to describe complex failure manifestations. Petri nets, instead, are useful to define performability metrics under multiple conditions, or to model system with multiple nodes, such as clustered system. In practice, these models are "implemented" in textual configuration files, by using ad-hoc specification languages, to be fed to software tools for reliability modelling. SHARPE [23] and GreatSPN [24] are examples of such tools.

RELATED STUDIES: Vaidyanathan *et al.* [25] used Stochastic Reward Nets (SRNs) to model and analyze cluster systems that adopt software rejuvenation. SRNs are an extension of Petri nets, where the reward rates are specified at the net level. In a SRN, the cardinality of an arc is a function of the number of tokens in a place, and a transition can be enhanced with a guard in addiction to other firing conditions. In a cluster system modelled by SRNs, a token in a place represents a node that is in a certain state (e.g., all nodes in the clusters are initially in a robust state). A transition can fire according to a pre-defined probability, such as the probability of a node to fail when in a failure-prone state. Using an SRN model, researchers implemented rejuvenation approaches on a cluster system and determined the optimal rejuvenation interval with respect to availability and cost. Fig. 3 shows a model of a cluster system that adopts time-based rejuvenation.

DERIVED STUDIES: Castelli *et al.* [26] designed and developed a rejuvenation agent for IBM Director, namely the *xSeries Software Rejuvenation Agent* (SRA), to manage highly-available clustered environments. The SRA monitors consumable resources, estimates the time-to-exhaustion of those resources, and generates alerts to the management infrastructure when the time to exhaustion is less than a user-defined threshold. A calendar interface is presented to the user (Fig. 4) to visually manage rejuvenations. A user has simply to drag-and-drop a node from the left panel in the desired day, in order to plan rejuvenation of that node in that day. By exploiting the previously described SRNs, time- and prediction-based rejuvenation approaches are evaluated to define the best parameters to use. For example, a model for time-based rejuvenation may show two different intervals within the potential rejuvenation periods: one minimising expected costs, the other minimising expected downtime. Thus, a practitioner should make a compromise on whether the expected downtime or the expected cost is more important, while deciding the period of the rejuvenation actions.

III. The SARRY Repository

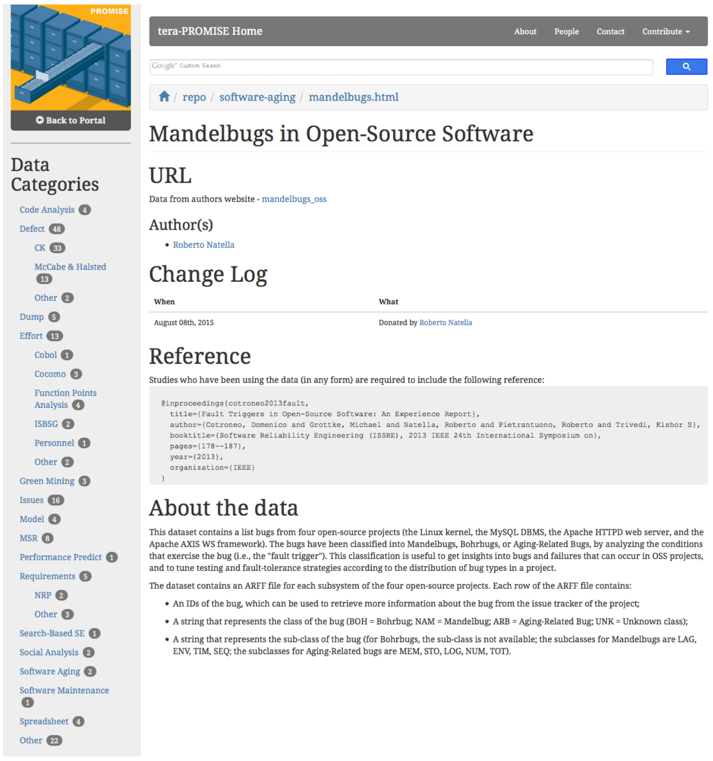
The SARRY repository is meant to collect SAR artifacts like those discussed in the previous section. In addiction to the raw files with the data, the artifacts are accompanied by the following information:

- The category of the artifact, among the four categories previously showed;
- A brief description on the structure of the artifact (e.g., how information is represented in a file), and an overview of possible uses of the data;
- A bibliographic reference (in BibTex format) to the study that produced the artifact, in order to provide a citable reference to users, and to give credit to the authors of the artifact.

The natural way to disseminate SAR artifacts is the World Wide Web. However, creating a new website to host the artifacts, or using a personal website, may not be a wise choice. The main issue is to assure the long-term availability of SAR artifacts for future research: an individual researcher may not be able to assure the availability of artifacts along his/her whole career, and a network of researchers, with the financial and technical support of their institutions, can provide better guarantees.

Therefore, we opted to join with a big and well-established repository, the *OpenScience tera-PROMISE* [11], which provides a long-term storage facility for research artifacts. PROMISE is “a research dataset repository specializing in software engineering research datasets”, which is organized in a hierarchical structure in order to host data from several areas of software engineering (including defect prediction, effort estimation, requirement engineering, testing and debugging, etc.). In agreement with the maintainers of PROMISE, we added a new category for SAR research on the PROMISE repository. This category is the root of the SARRY repository, which will include sub-pages to host artifacts from different research studies. Fig. 5 shows an example of page with a dataset on Mandelbugs.

The repository is open to researchers that want to contribute with new artifacts. To submit an artifact, you can either contact



Mandelbugs in Open-Source Software

URL

Data from authors website - mandelbugs_oss

Author(s)

- Roberto Natella

Change Log

When	What
August 08th, 2015	Donated by Roberto Natella

Reference

Studies who have been using the data (in any form) are required to include the following reference:

```
@inproceedings{cotroneo2013fault,
  title={Fault Triggers in Open-Source Software: An Experience Report},
  author={Cotroneo, Domenico and Grottke, Michael and Natella, Roberto and Pietrantuono, Roberto and Trivedi, Kishor S},
  booktitle={Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on},
  pages={178--187},
  year={2013},
  organization={IEEE}
}
```

About the data

This dataset contains a list bugs from four open-source projects (the Linux kernel, the MySQL DBMS, the Apache HTTPD web server, and the Apache AXIS WS framework). The bugs have been classified into Mandelbugs, Bohrbugs, or Aging-Related Bugs, by analyzing the conditions that exercise the bug (i.e., the "fault trigger"). This classification is useful to get insights into bugs and failures that can occur in OSS projects, and to tune testing and fault-tolerance strategies according to the distribution of bug types in a project.

The dataset contains an ARFF file for each subsystem of the four open-source projects. Each row of the ARFF file contains:

- An IDs of the bug, which can be used to retrieve more information about the bug from the issue tracker of the project;
- A string that represents the class of the bug (BOH = Bohrbug; NAM = Mandelbug; ARB = Aging-Related Bug; UNK = Unknown class);
- A string that represents the sub-class of the bug (for Bohrbugs, the sub-class is not available; the subclasses for Mandelbugs are LAG, ENV, TIM, SEQ; the subclasses for Aging-Related bugs are MEM, STO, LOG, NUM, TOT).

Fig. 5. An example of dataset published on the SARRY repository.

the authors of this paper (see the contacts in the front of this paper), or you can fill the online form on the website, by specifying *Software Aging* as the “PROMISE repo category”. The dataset will be added to the SARRY repository, and will become openly accessible. We remark that the sharing of artifacts is a good opportunity for researchers to improve the visibility and the impact of their work.

The SARRY repository can be accessed at the address: **http://openscience.us/repo/software-aging/**.

IV. Conclusion

Software Aging and Rejuvenation (SAR) is a well-established research area, but it lacks a “sharing culture” to enable new research to benefit from the results of previous studies. We proposed four categories of SAR artifacts to share, and we analyzed past studies that developed notable examples of SAR artifacts, from which later SAR studies have benefited. SAR artifacts include performance, failure, and resource consumption measurements from software systems in production and from stress testing experiments, data on aging-related bugs, and rejuvenation scheduling models. SARRY is an open repository for long-term storage of these SAR artifacts, available at the address **http://openscience.us/repo/software-aging/**. Our hope is that other SAR researchers will join us in populating SARRY, and that much more researchers can benefit from it for their high-quality SAR research.

Acknowledgment

The authors would like to thank the maintainers of the PROMISE repository, and in particular Rahul Krishna and Tim Menzies, for their kind support for the SARRY repository. We would also like to thank Michael Grottke and Kishor Trivedi for sharing their SAR artifacts for the initial version of the SARRY repository. This work was supported by the COSMIC public-private laboratory, projects SVEVIA (PON02_00485_3487758), MINIMINDS (PON02_00485_3164061) and DISPLAY (PON02_00485_3487784), funded by the Italian Ministry of Education, University and Research.

References

[1] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on.* IEEE, 1995, pp. 381–390.

[2] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging and rejuvenation: Where we are and where we are going," in *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on.* IEEE, 2011, pp. 1–6.

[3] ——, "A survey of software aging and rejuvenation studies," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, p. 8, 2014.

[4] "WoSAR 2015 seventh international workshop on software aging and rejuvenation," https://sites.google.com/site/wosar2015/, accessed: 2015-08-01.

[5] J. M. Alonso, A. Bovenzi, J. Li, Y. Wang, S. Russo, and K. Trivedi, "Software rejuvenation: Do it & telco industries use it?" in *Software Reliability Engineering Workshops (ISSREW 2012), 23rd International Symposium on.* IEEE, 2012, pp. 299–304.

[6] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame, "Data sharing by scientists: practices and perceptions," *PloS one*, vol. 6, no. 6, 2011.

[7] "Challenge in irreproducible research : Nature news and comments," http://www.nature.com/news/reproducibility-1.17552, accessed: 2015-08-01.

[8] "SIR software-artifact infrastructure repository," http://sir.unl.edu/portal/index.php, accessed: 2015-08-01.

[9] "ADR amber raw data repository," http://amber-dbserver.dei.uc.pt:8080/repository/main.action, accessed: 2015-08-01.

[10] "Artifact evaluation for software conferences," http://www.artifact-eval.org/, accessed: 2015-08-01.

[11] T. Menzies, M. Rees-Jones, R. Krishna, and C. Pape, "The promise repository of empirical software engineering data," 2015, http://openscience.us/repo. North Carolina State University, Department of Computer Science.

[12] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on.* IEEE, 1998, pp. 283–292.

[13] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on.* IEEE, 1999, pp. 84–93.

[14] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *Reliability, IEEE Transactions on*, vol. 55, no. 3, pp. 411–420, 2006.

[15] R. Matias Jr and J. Paulo Filho, "An experimental study on software aging and rejuvenation in web servers," in *Computer Software and Applications Conference (COMPSAC '06), 30th Annual International*, vol. 1. IEEE, 2006, pp. 189–196.

[16] H. El-Shishiny, S. Deraz, and O. Bahy, "Mining software aging patterns by artificial neural networks," in *Artificial Neural Networks in Pattern Recognition.* Springer, 2008, pp. 252–262.

[17] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on.* IEEE, 2010, pp. 447–456.

[18] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," *Performance Evaluation*, vol. 70, no. 3, pp. 163–178, 2013.

[19] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi, "Fault triggers in open-source software: An experience report," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on.* IEEE, 2013, pp. 178–187.

[20] X. Xia, D. Lo, X. Wang, and B. Zhou, "Automatic defect categorization based on fault triggering conditions," in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on.* IEEE, 2014, pp. 39–48.

[21] M. Grottke and B. Schleich, "How does testing affect the availability of aging software systems?" *Performance Evaluation*, vol. 70, no. 3, pp. 179–196, 2013.

[22] M. Grottke, D. Kim, R. Mansharamani, M. Nambiar, R. Natella, and K. Trivedi, "Recovery From Software Failures Caused by Mandelbugs," *IEEE Transactions on Reliability*, 2015, PrePrint, http://dx.doi.org/10.1109/TR.2015.2452933.

[23] "SHARPE symbolic hierarchical automated reliability and performance evaluator," http://sharpe.pratt.duke.edu/, accessed: 2015-08-01.

[24] "GreatSPN 2.0 graphical editor and analyzer for timed and stochastic petri nets," http://www.di.unito.it/~greatspn/index.html, accessed: 2015-08-01.

[25] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi, "Analysis and implementation of software rejuvenation in cluster systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 62–71, 2001.

[26] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 311–332, 2001.