An assessment of persistent homology tools applied in software reliability analysis

Tin Nanndo Jovanović[1,†], Alan Burić[1,†], Tihana Galinac Grbac[1,*,†], João Pita Costa[2,†] and Neven Grbac[1,†]

[1]Juraj Dobrila University of Pula, Zagrebačka 30, HR-52100 Pula, Croatia
[2]IRCAI – International Research Centre on AI under the auspices of UNESCO, Jamova cesta 39, SI-1000 Ljubljana, Slovenia

Abstract

In recent years, topological data analysis has become an important aspect of modern data science. Persistent homology is one of its popular features, and there are several specialized tools for its calculation. On the other hand, software reliability can be studied as a time-series of discovered software faults and/or occurrences of software failures. Topological data analysis of such time-series reduces to calculation of the persistent homology of certain point-clouds attached to time-series in sufficiently high dimension. The aim of this paper is to investigate which of the tools for computing persistent homology is the most appropriate for applications in software reliability analysis.

Keywords

Software reliability, Topological data analysis, Persistent homology, Software faults and failures, Time-series

1. Introduction

Software becomes a cornerstone of digital society, where software systems are interconnecting infrastructure, transportation, healthcare, and many other domains. Failures in such systems can have far-reaching, even catastrophic, consequences. Software reliability is a crucial system quality attribute, especially for mission-critical systems where failures can have severe consequences for safety, human lives, and the operation of vital infrastructure. Moreover, reliability is also becoming increasingly important for all other domains due to economic, business, and sustainability constraints that may be affected by system failures.

Reliability modeling plays a pivotal role in managing such systems, enabling engineers to quantitatively analyze past failures, predict future reliability, and help risk management to make informed decisions about further management decisions like investments, release readiness, and quality assurance. The current state of the art in software reliability modeling focuses on advanced software reliability growth models (SRGM) that are essential tools for quantitatively assessing and predicting the reliability of software systems throughout their lifecycle [1], [2]. These models are based on nonhomogeneous Poisson processes, such as exponential, logarithmic, and S-shaped models, to analyze the rate of system failure detection over time and forecast reliability improvements throughout the software lifecycle. These models, supported by empirical validation and parameter estimation techniques, are increasingly integrated into modern development workflows, providing a rigorous foundation for managing and certifying software reliability in complex, safety-critical systems. Recent research in SRGMs reflects a significant evolution from traditional failure-counting approaches to more sophisticated models that incorporate real-world operational complexities.

In our previous studies [3], [4], [5], [6], [7], [8], we were motivated from the real industrial context.

SQAMIA 2025: 12th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, September 10–12, 2025, Maribor, Slovenia

*Corresponding author.
†These authors contributed equally.
tin.nanndo.jovanovic@unipu.hr (T. N. Jovanović); alan.buric1@unipu.hr (A. Burić); tihana.galinac@unipu.hr (T. Galinac Grbac); joao.pitacosta@ircai.org (J. Pita Costa); neven.grbac@unipu.hr (N. Grbac)
0000-0002-4351-4082 (T. Galinac Grbac); 0000-0001-5745-1302 (J. Pita Costa); 0000-0001-6657-6297 (N. Grbac)

In particular, the increasing frequency of software releases in complex software systems and the common practice of overlapping multiple projects that present significant challenges for fundamental assumptions underlying traditionally used SRGMs. When development and maintenance activities for different versions of a product occur simultaneously, the fault detection rates from previous revisions can interfere with current reliability measurements, leading to inaccurate predictions if traditional models are applied. Research has shown that in overlapping project environments, a single reliability growth model often fails to fit the observed data accurately. Thus, combining multiple distributions or adapting models becomes necessary to achieve reliable estimations and support effective project management and planning [3].

Some further studies, such as [9] and [10], introduce coverage-based and multi-release models that integrate testing effort functions, change-point analysis, and advanced lifetime distributions to better reflect software behavior over time within a real operational context. The work [11] extended SRGM applications beyond defect modeling to predict microservices' performance degradation using growth theory, highlighting the adaptability of SRGMs to modern architectures.

A comprehensive examination of research trends, model selection practices, and enhancement techniques in SRGMs is provided in [12]. The authors conducted a systematic mapping and literature review covering 142 primary studies published between 1992 and 2020. The study categorizes SRGM selection criteria into model assumptions, application context, and evaluation metrics, and it identifies various enhancement methods such as hybrid modeling, incorporation of testing effort functions, and parameter estimation using metaheuristics (e.g. genetic algorithms and particle swarm optimization). The review reveals a notable shift in research toward more adaptable and context-aware models, while also highlighting gaps in empirical validation and standardization of evaluation practices. The authors conclude by recommending more rigorous experimental setups, broader datasets, and consistent benchmarking to support SRGM reliability and industrial relevance.

Our long term goal is to study software reliability using topological data analysis, as already announced in [13]. Geometry, and in particular topology, has gained in importance in computer science in recent years. Among the most prominent research direction are the geometric approaches to artificial intelligence [14], [15], [16], and the topological approach to quantum computers [17], [18]. The seed of all these new developments and applications of topology lies in topological data analysis [19], [20]. It relies on the toolbox of algebraic topology [21], [22], to extract hidden patterns in data that are out of reach by other, in particular quantitative, methods. Topological data analysis has found several important applications, see the expository papers [23], [24] for an overview of these.

Our previous work [13] is a preliminary study of the application of topological data analysis to software failure time-series, using persistent homology to extract qualitative patterns that enhance the understanding of reliability behaviors and support SRGM selection. Rather than focusing solely on quantitative measures traditionally used in Software Reliability Growth Models (SRGMs), we explore the qualitative, shape-based features of failure data to uncover hidden structural patterns across software versions. Using datasets from two open-source Eclipse projects (JDT and PDE) and an industrial telecom system (MSC), we demonstrate how topological features in failure data can reveal differences and similarities in reliability evolution. The analysis suggests that these topological insights can support early identification of reliability behaviors and inform the selection of appropriate SRGMs, offering a promising direction for future research into modeling complex, evolving software systems.

As the first step in that endeavor, we explore in this paper different tools available for computing the persistent homology of time-series using the Vietoris–Rips simplicial complexes obtained by the sliding window approach to construct the point-clouds in higher dimensions from a given time-series. Although a detailed analysis of the most important tools has already been reported in [25], [26], we undertake here an assessment of computation time of these tools in view of the application in software reliability analysis. The motivation is that the general results reported in the literature may not be appropriate for the special type of time-series arising from the software reliability analysis.

The assessment of the persistent homology tools is performed on 60 simulated software fault time-series. The simulations are made following four different classical reliability growth models, and 120 point clouds are constructed using two different widths of the sliding window. The size of the dataset

and the variability in models and sliding window, allows us to evaluate the considered tools in regard to the software reliability analysis.

The paper is organized as follows. In Section 2, the reliability growth models are recalled and the simulation of software fault time-series is explained. A brief and rough overview of persistent homology and tools for its calculations are the subject of Section 3. The conducted experiment is described and the results are reported in Section 4. Section 5 concludes the paper.

2. Reliability Growth Models

In this section we recall the classical reliability growth models for software reliability modeling. The main reference for these matters are still the books [1], [2]. Although there is a plethora of new models for reliability behavior of software and systems, we prefer to stick with the classical SRGMs for two reasons. Firstly, these models are of general applicability even beyond software reliability, perhaps not the best performing in some special circumstances, but very robust to environmental changes. Secondly, there are analytic expressions for their behavior in time, so that the time-series data can be simulated, which is not the case for many of the modern models.

We use the standard notation, that is, $\mu(t)$ represents the number of faults discovered in a software system from the beginning of testing or operation up to time $t \geq 0$. Here $t = 0$ represents the moment when the testing or operation starts. The four SRGMs considered in this paper are the same as in the papers [27], [28]. These are recalled in the following list with the formula for $\mu(t)$ and the bounds for their parameters:

- Goel–Okumoto or basic Musa model [29]

$$\mu_{\mathrm{GO}}(t) = a\left(1 - e^{-bt}\right),$$

  where $a > 0$, $b > 0$,
- Delayed S-shaped model [30]

$$\mu_{\mathrm{DS}}(t) = a\left[1 - (1 + bt)\, e^{-bt}\right],$$

  where $a > 0$, $b > 0$,
- Gompertz model [31]

$$\mu_G(t) = a\left(b^{c^t}\right),$$

  where $a > 0$, $0 < b < 1$, $0 < c < 1$,
- Yamada model [32]

$$\mu_Y(t) = a\left[1 - e^{-bc\left(1 - e^{-dt}\right)}\right],$$

  where $a > 0$, $bc > 0$, $d > 0$.

All these models are based on the nonhomogeneous Poisson process with different distributions of inter-arrival times of faults. The Goel–Okumoto and Yamada models are concave, that is, the curve $\mu(t)$ is concave. The delayed S-shaped and Gompertz model are S-shaped, which means that the curve $\mu(t)$ exhibits an inflection point at which it changes from convex to concave.

The parameter a represents the total number of faults in the system. In all the models except the Yamada model, this is the same as the number of faults that would be detected as the testing proceeds infinitely. The Yamada model assumes that even if the testing proceeds forever, there would still be some faults left in the system. This fact is taken into account when making simulations of fault counts.

There is an additional subtlety with the Gompertz model, as the faults can be detected at negative time, i.e., for $t < 0$. This happens because the Gompertz model assumes that some faults are detected before the actual testing started.

The simulated time-series following the considered SRGMs are constructed from a uniformly distributed random numbers in the unit interval using the transformations arising from the formulas for $\mu(t)$. The problem with possibly negative values of t in the case of the Gompertz model is resolved in three different ways, thus producing three different types of time-series for the Gompertz model.

To ensure that the simulated time-series exhibit the fault behavior of real world software, and thus increase the generalizability of the findings, the parameters of SRGMs that are used in simulations are estimated from fault data of five industrial large-scale software development projects in the telecommunication domain.

In summary, we constructed the input for the experiment with persistent homology tools of Section 3. It is a dataset with 60 simulated time-series of software fault counts. There are 10 time-series that follow each of the considered SRGMs, including three different types of the Gompertz model.

3. Persistent Homology Tools

This section begins with a very rough overview of persistent homology. The overview is included for completeness, and the reader interested only in applications to software engineering can safely skip this part and consider it as a black box. Due to limited space and the complexity of the subject, we are not in position to provide a complete self-contained account of the topic with examples and/or simple cases to demonstrate the theory. An example of topological analysis of higher dimensional structures in software systems is already published in this conference [33]. For the interested reader, we refer to the comprehensive standard reference [21], in which persistent homology is discussed in detail in Chapter VII. An excellent source for a beginner in algebraic topology and its applications are the lecture notes [34].

Persistent homology is an algebraic object attached to a filtration of simplicial complices. It captures how homology classes, in different degrees, appear and vanish when computing homology of simplicial complices in the filtration. These are called the birth and death times of homology classes (also called topological features from the machine learning point of view), where time refers to the position in the filtration. The results are encoded in either persistent barcodes or persistent diagrams. Both represent the birth and death times of homology classes.

In the application to point-clouds that represent some data, the filtration of simplicial complices arises from the Vietoris–Rips complices associated with the growing radius of balls around points in the cloud. Since point-clouds are finite, there are only finitely many radii at which the Vietoris–Rips complex changes. This gives a required finite filtration.

In our case, the time-series simulated in Section 2 are transformed in point-clouds using a sliding window approach. For each time-series the sliding windows of width five and ten are applied. These produce point-clouds in dimensions five and ten that are the input for persistent homology tools in the experiment described in Section 4. Thus, we have in total 120 point-clouds, two arising from each of the 60 simulated time-series. Among them, 60 point clouds are in dimension five, and 60 in dimension ten.

The tools for computing persistent homology considered in this paper are the following:

- *Dionysus* version 2.0.10 in Python [35],
- *DIPHA* version 2.1.0 in C++ [36], [37],
- *Eirene* version 1.3.6 in Julia, [38], [39],
- *GUDHI* version 3.11.0 in Python, [40], [41],
- *Ripser* version 1.2.1 in C++, [25], [42].

These are the open source tools that all implement the calculation of Vietoris–Rips complex from a point-cloud with respect to growing radius and the calculation of the persistent homology of the filtration of simplicial complices so obtained. There are, however, certain differences in their approach. Dionysus takes advantage of the fact that cohomology is easier to compute than homology, and exploits the duality between homology and cohomology. DIPHA is making the calculation using distributed computing, which is very beneficial for large and high-dimensional data. Eirene is relying on matroid

Table 1
Descriptive Statistics of Computational Times for the Five Tools for Persistent Homology in All Dimensions, and Separately in Dimensions 5 and 10

Point-clouds	Computational time [s]	Dionysus	DIPHA	Eirene	GUDHI	Ripser
All	Mean	369.776	432.893	3.306	16.922	23.820
	Std. dev.	612.145	792.721	7.835	29.910	40.933
	Median	23.772	31.606	0.495	1.163	1.550
	Min.	4.196	9.601	0.076	0.307	0.333
	Max.	2974.993	3251.298	65.944	144.219	202.412
Dimension 5	Mean	195.196	622.618	1.110	6.792	10.895
	Std. dev.	282.340	943.022	1.365	9.038	14.648
	Median	11.928	32.726	0.180	0.722	0.912
	Min.	4.196	9.601	0.076	0.307	0.333
	Max.	986.868	3251.298	4.630	25.670	47.439
Dimension 10	Mean	544.355	168.161	5.502	27.052	36.746
	Std. dev.	783.639	392.065	10.289	38.910	53.152
	Median	33.859	31.606	0.835	1.733	2.468
	Min.	7.618	14.120	0.143	0.515	0.570
	Max.	2974.993	1831.467	65.994	144.219	202.412

Table 2
Descriptive Statistics of Computational Times for All Five Tools for Persistent Homology and Point-clouds Arising from Different SGRMs

Computational time [s]	Goel–Okumoto	Delayed S-shaped	Gompertz	Yamada
Mean	553.490	38.128	6.604	413.854
Std. dev.	867.663	62.572	9.519	600.834
Median	79.488	3.692	0.843	73.645
Min.	2.049	0.234	0.077	1.688
Max.	3251.298	310.971	41.755	1972.463

theory and discrete Morse theory. GUDHI employs new data structures for simplicial complices and their boundary matrices. Ripser is known as one of the best-performing tools due to its special optimizations. For a detailed account of existing tools for calculation of persistent homology see [26].

4. Experiment

The experiment is made on the dataset of 120 point-clouds constructed in Section 3 from 60 time-series simulated as in Section 2. The calculations of persistent homology was done for degrees 0, 1, 2, 3 in homology. Higher degrees are computationally more demanding and would take quite long, at least on our equipment.

The tools for computation of persistent homology listed in Section 3 were deployed on a laptop running Linux Mint 21.2 (kernel 6.8.0), equipped with an AMD Ryzen 7 4800H processor (8 cores / 16 threads), 32 GB of DDR4 RAM at 3200 MHz, and two graphic cards (integrated AMD Radeon RX Vega 6 and a discrete NVIDIA GeForce RTX 3050 Mobile). All tools were executed via the Kitty terminal using the Zsh shell. The GPU was not utilized, as none of the tools support GPU acceleration. Each computation completed within a few seconds, and 32 GB of RAM was sufficient for all processed point-clouds.

For each of 120 point-clouds the computational time in seconds for each of the five tools is recorded. The summary of the descriptive statistics of these times is given in Table 1, for point-clouds in all dimensions, and separately for points-clouds in dimension 5 and dimension 10. In Table 2, the summary of the descriptive statistics is given separately for point-clouds obtained from different SRGMs.

Table 3
Results of t-tests for Dependent Variables between Different Tools in All Dimensions

vs.	DIPHA	Eirene	GUDHI	Ripser
Dionysus	$t = -0.670052$ $p = \mathbf{0.503524}$	$t = 6.616714$ $p < 10^{-6}$	$t = 6.623903$ $p < 10^{-6}$	$t = 608180$ $p < 10^{-6}$
DIPHA	— —	$t = 5.938261$ $p < 10^{-6}$	$t = 5.745596$ $p < 10^{-6}$	$t = 5.646235$ $p < 10^{-6}$
Eirene	— —	— —	$t = -6.19828$ $p < 10^{-6}$	$t = -6.456150$ $p < 10^{-6}$
GUDHI	— —	— —	— —	$t = -6.515680$ $p < 10^{-6}$

Table 4
Results of t-tests for Dependent Variables between Different Tools in Dimension 5

vs.	DIPHA	Eirene	GUDHI	Ripser
Dionysus	$t = -3.36332$ $p = 0.001039$	$t = 5.348114$ $p = 0.000002$	$t = 5.332483$ $p = 0.000002$	$t = 5.314825$ $p = 0.000002$
DIPHA	— —	$t = 5.105054$ $p = 0.000001$	$t = 5.058149$ $p = 0.000002$	$t = 5.024080$ $p = 0.000002$
Eirene	— —	— —	$t = -5.669510$ $p < 10^{-6}$	$t = -5.678280$ $p < 10^{-6}$
GUDHI	— —	— —	— —	$t = -5.364410$ $p = 0.000001$

Table 5
Results of t-tests for Dependent Variables between Different Tools in Dimension 10

vs.	DIPHA	Eirene	GUDHI	Ripser
Dionysus	$t = 2.896106$ $p = 0.004632$	$t = 5.375149$ $p = 0.000001$	$t = 5.36937$ $p = 0.000001$	$t = 5.360293$ $p = 0.000001$
DIPHA	— —	$t = 3.218287$ $p = 0.001734$	$t = 2.774210$ $p = 0.006593$	$t = 2.568501$ $p = 0.011677$
Eirene	— —	— —	$t = -5.332970$ $p = 0.000002$	$t = -5.369745$ $p = 0.000001$
GUDHI	— —	— —	— —	$t = -5.060904$ $p = 0.000004$

The descriptive statistics of computational times for all point-clouds in Table 1 reveals substantial differences between considered tools. All statistics consistently indicate that Eirene is the best performing tool in terms of computational time, followed by GUDHI and then Ripser, while Dionysus is slightly better than DIPHA in the last place. The same conclusion also holds for results in the case of point-clouds in dimension 5 in Table 1. The results of Table 1 in dimension 10 show very similar behavior, except that DIPHA seems to be better than Dionysus in that case.

It is interesting to observe how all the means in Table 1 are considerably greater than medians, maxima are pretty large, and there is high variability in terms of standard deviation. This shows that there is a small portion of point-clouds with very high computational times, and the remaining large portion of point-clouds that are very quickly processed. Checking by hand which of the point-clouds have larger computational times than average, we discovered that all tools perform badly on the same

Table 6
Results of t-tests for Dependent Variables between the Same Tool Applied to Point-clouds in Dimension 5 and 10

	Dionysus	DIPHA	Eirene	GUDHI	Ripser
dim 5 vs dim 10	$t = -4.915500$ $p = 0.000007$	$t = 2.977814$ $p = 0.003635$	$t = -3.501114$ $p = 0.000889$	$t = -5.111223$ $p = 0.000004$	$t = -5.012962$ $p = 0.000005$

Table 7
Results of t-tests for Independent Variables between Different Software Reliability Growth Models

vs.	Delayed S-shaped	Gompertz	Yamada
Goel–Okumoto	$t = 5.924296$ $p < 10^{-6}$	$t = 6.304215$ $p < 10^{-6}$	$t = 1.266813$ $p = \mathbf{0.206850}$
Delayed S-shaped	— —	$t = 4.980730$ $p = 0.000001$	$t = -6.218271$ $p < 10^{-6}$
Gompertz	— —	— —	$t = -6.778711$ $p < 10^{-6}$

point-clouds.

To explore this observation further, we compare in Table 2 the descriptive statistics separately for point-clouds arising from different SRGMs. According to these statistics, the computational times of all tools combined strongly depends on the SRGM from which a point-cloud is constructed. The tools perform best on point-clouds from Gompertz model, closely followed by delayed S-shaped, while they perform significantly slower in the case of Yamada model and the last placed Goel–Okumoto model. This is consistent with the worst point-clouds from computational time point of view observe above.

It is worth noticing in Table 1 that all the tools except DIPHA are faster on point-clouds of dimension 5 than on those of dimension 10. This is as expected, since computational complexity rises with growth of dimension, that is, with the width of the sliding window used in the construction of the point-cloud from a time-series. The reason why DIPHA exhibits the opposite behavior may lie in the fact that it is designed for distributed computing.

To support the conclusions made above based solely on descriptive statistics, we performed the t-test of equal means between different samples of computational times. The t-statistic and the corresponding p-value for compared samples of five considered tools applied to all point-clouds are reported in Table 3, applied to point-clouds in dimension 5 in Table 4 and in dimension 10 in Table 5. On the other hand, the results of t-test for samples of the same tool applied to point-clouds in dimension 5 and 10 are reported in Table 6. Finally, the t-statistic and p-value for samples arising from the application of all tools to point-clouds obtained from time-series of different SRGMs are reported in Table 7. In all these tables, if the null hypothesis of equal means is not rejected at significance level $p = 0.5$, i.e., if the p-value $p > 0.05$, then the p-value is written in boldface for the convenience of the reader.

Browsing through these tables shows that the only cases in which the null hypothesis is not rejected are in the case of comparison of Dionysus and DIPHA tool applied to all point-clouds, and the comparison of all tools applied to point-clouds arising from Goel–Okumoto and Yamada model. In both cases these are the worst performing models. For all other comparisons the null hypothesis can be rejected, thus providing evidence for our conclusions based on descriptive statistics.

5. Conclusion

One of the key aspects of software quality, especially in mission-critical systems, is its reliability. Topological data analysis through persistent homology is one of the modern approaches to the analysis of time-series. There are several tools for computing persistent homology of time-series. We analyzed five of them (Dionysus, DIPHA, Eirene, GUDHI, Ripser) from the point of view of software reliability.

More precisely, we analyzed computational time of different tools when applied to time-series datasets simulated to follow some of the classical software reliability growth models (Goel–Okumoto, Delayed S-shaped, Gompertz, Yamada) with parameters estimated from five industrial large-scale software development projects. The main conclusion is that the best performing tool is the latest version of Eirene, followed by GUDHI, and then Ripser. Dionysus and DIPHA performed substantially slower. However, we also observed high variability in computation time and a small portion of input data with very long computational times, so that the conclusions are not decisive.

It is interesting that the computational times of all tools strongly depend on the underlying software reliability growth model which the time-series follows. We also confirmed the expectation that computational times grow with higher dimensions.

Acknowledgments

This work was supported by the Croatian Science Foundation under the project number HRZZ-2022-10-4615.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-4o in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] J. D. Musa, A. Iannino, K. Okumoto, Software Reliability: Measurement, Prediction, Application (professional ed.), McGraw-Hill, Inc., USA, 1989.

[2] J. D. Musa, Software Reliability Engineering: More Reliable Software Faster and Cheaper (2nd ed.), AuthorHouse, UK, 2004.

[3] T. Galinac, S. Golubić, Project overlapping and its influence on the product quality, in: 8th International Conference on Telecommunications (ConTEL 2005), IEEE, 2005, pp. 655–660.

[4] T. Galinac Grbac, P. Runeson, D. Huljenić, A second replicated quantitative analysis of fault distributions in complex software systems, IEEE Transactions on Software Engineering 39 (2013) 462–476. doi:10.1109/TSE.2012.46.

[5] T. Galinac Grbac, P. Runeson, D. Huljenić, Unit verification effects on reused components in sequential project releases, in: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2017, pp. 74–82. doi:10.1109/SEAA.2017.63.

[6] T. Galinac Grbac, D. Huljenić, On the probability distribution of faults in complex software systems, Information and Software Technology 58 (2015) 250–258. doi:https://doi.org/10.1016/j.infsof.2014.06.014.

[7] T. Galinac Grbac, G. Mauša, On the distribution of software faults in evolution of complex systems, in: Proceedings of the International Colloquium on Software-Intensive Systems-of-Systems at 10th European Conference on Software Architecture, SiSoS@ECSA '16, Association for Computing Machinery, New York, NY, USA, 2016. doi:10.1145/3175731.3176181, Art. no. 2 (7pp).

[8] T. Galinac Grbac, D. Huljenić, A. Grgurić, N. Grbac, The Reed–Jorgensen double Pareto-lognormal distribution as the probability distribution of faults in software systems, preprint (2025).

[9] A. Aggarwal, S. Kumar, R. Gupta, Testing coverage based NHPP software reliability growth modeling with testing effort and change-point, International Journal of System Assurance Engineering and Management 15 (2024) 5157–5166. doi:10.1007/s13198-024-02504-7.

[10] J. Wang, P. Li, J. Hu, C. Zhang, A multi-release reliability model of open source software with fault detection obeying three-parameter lifetime distribution, Scientific Reports 14 (2024) 19576. doi:10.1038/s41598-024-70536-x.

[11] M. Camilli, B. Russo, Modeling performance of microservices systems with growth theory, Empirical Software Engineering 27 (2022). URL: https://doi.org/10.1007/s10664-021-10088-0. doi:10.1007/s10664-021-10088-0.
[12] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, C. Höglund, Selecting software reliability growth models and improving their predictive accuracy using historical projects data, Journal of Systems and Software 98 (2014). doi:10.1016/j.jss.2014.08.033.
[13] J. Pita Costa, T. Galinac Grbac, The topological data analysis of time series failure data in software evolution, in: W. Binder, V. Cortellessa, A. Koziolek, E. Smirni, M. Poess (Eds.), Companion Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017, ACM, 2017, pp. 25–30. URL: https://doi.org/10.1145/3053600.3053604. doi:10.1145/3053600.3053604.
[14] A. Micheletti, A new paradigm for artificial intelligence based on group equivariant non-expansive operators, European Mathematical Society Magazine (2023) 4–12.
[15] R. Balestriero, A. Imtiaz Humayun, R. G. Baraniuk, On the geometry of deep learning, Notices of the American Mathematical Society 72 (2025) 374–385.
[16] Erlangen AI Hub Team, Erlangen AI hub – mathematical foundations of intelligence, 2025. URL: https://erlangenhub.ox.ac.uk/.
[17] S. H. Simon, Topological Quantum, Oxford University Press, Oxford, 2023. URL: https://doi.org/10.1093/oso/9780198886723.001.0001. doi:10.1093/oso/9780198886723.001.0001.
[18] C. Edwards, Tales of topological qubits, Communications of the ACM 66 (2023) 8–10. URL: https://doi.org/10.1145/3624436. doi:10.1145/3624436.
[19] G. Carlsson, Topology and data, Bulletin of the American Mathematical Society (New Series) 46 (2009) 255–308. URL: https://doi.org/10.1090/S0273-0979-09-01249-X. doi:10.1090/S0273-0979-09-01249-X.
[20] P. Dłotko, On the shape that matters—topology and geometry in data science, European Mathematical Society Magazine (2024) 5–13.
[21] H. Edelsbrunner, J. L. Harer, Computational Topology, American Mathematical Society, Providence, RI, 2010. URL: https://doi.org/10.1090/mbk/069. doi:10.1090/mbk/069, an introduction.
[22] J. R. Munkres, Elements of Algebraic Topology, Addison-Wesley Publishing Company, Menlo Park, CA, 1984.
[23] G. Carlsson, Persistent homology and applied homotopy theory, in: H. Miller (Ed.), Handbook of Homotopy Theory, Chapman and Hall/CRC, New York, 2020, pp. 297–329. doi:10.1201/9781351251624.
[24] J. A. Perea, Topological times series analysis, Notices of the American Mathematical Society 66 (2019) 686–694.
[25] U. Bauer, Ripser: efficient computation of Vietoris-Rips persistence barcodes, Journal of Applied and Computational Topology 5 (2021) 391–423. URL: https://doi.org/10.1007/s41468-021-00071-5. doi:10.1007/s41468-021-00071-5.
[26] R. Ceccaroni, L. Di Rocco, U. Ferraro Petrillo, P. Brutti, A distributed approach for persistent homology computation on a large scale, The Journal of Supercomputing 80 (2024) 25510–25532. doi:10.1007/s11227-024-06374-5.
[27] C. Stringfellow, A. A. Andrews, An empirical method for selecting software reliability growth models, Empirical Software Engineering 7 (2002) 319–343. doi:10.1023/A:1020515105175.
[28] C. Andersson, A replicated empirical study of a selection method for software reliability growth models, Empirical Software Engineering 12 (2007) 161–182. doi:10.1007/s10664-006-9018-0.
[29] A. L. Goel, K. Okumoto, Time-dependent error-detection rate model for software reliability and other performance measures, IEEE Transactions on Reliability 28 (1979) 206–211.
[30] S. Yamada, M. Ohba, S. Osaki, S-shaped reliability growth modeling for software error detection, IEEE Transactions on Reliability R-32 (1983) 475–484. doi:10.1109/TR.1983.5221735.
[31] D. B. Kececioglu, Reliability Engineering Handbook, volume 2, Prentice Hall, Englewood Cliffs, NJ, 1991.
[32] S. Yamada, H. Ohtera, H. Narihisa, Software reliability growth models with testing-effort, IEEE

Transactions on Reliability 35 (1986) 19–23. doi:10.1109/TR.1986.4335332.

[33] E. Puh, T. Galinac Grbac, N. Grbac, Preliminary study of higher dimensional software structures, in: Z. Budimac, V. Vranić, J. Lang (Eds.), Proceedings of the 10th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2023, CEUR Workshop Proceedings, volume 3588, 2003, pp. 13–25.

[34] T. Galinac Grbac, E. Puh, N. Grbac, Algorithms for sustainable system topologies (lecture notes, SusTrainable Summer School 2022, Univ. of Rijeka), Lecture Notes in Computer Science, vol. 15670, Springer, Cham, to appear, 2025. URL: http://tania.unipu.hr/$\sim$negrbac/2025_Sustrainable_SS_2022_LNCS_15670_to_appear.pdf.

[35] D. Albrecht, Topology and data analysis with Dionysus, 2017. URL: https://github.com/nonabelian/tda_dionysus.

[36] H. Wagner, C. Chen, E. Vuçini, Efficient computation of persistent homology for cubical data, in: R. Peikert, H. Hauser, H. Carr, R. Fuchs (Eds.), Topological Methods in Data Analysis and Visualization II: Theory, Algorithms, and Applications, Springer, Berlin, Heidelberg, 2012, pp. 91–106. URL: https://doi.org/10.1007/978-3-642-23175-9_7. doi:10.1007/978-3-642-23175-9_7.

[37] DIPHA team, DIPHA tool, 2017. URL: https://github.com/DIPHA.

[38] G. Henselman, R. Ghrist, Matroid filtrations and computational persistent homology, preprint (2017) 16pp. URL: https://arxiv.org/pdf/1606.00199. doi:10.48550/arXiv.1606.00199.

[39] G. Henselman-Petrusek, Eirene: Julia library for homological persistence, 2016–2021. URL: https://github.com/henselman-petrusek/Eirene.jl.

[40] C. Maria, J.-D. Boissonnat, M. Glisse, M. Yvinec, The Gudhi library: Simplicial complexes and persistent homology, in: H. Hong, C. Yap (Eds.), Mathematical Software – ICMS 2014, Springer, Berlin, Heidelberg, 2014, pp. 167–174.

[41] J.-D. Boissonnat, M. Carrière, M. Glisse, C. Maria, V. Rouvreau, A. Stamm (eds.), GUDHI – Geometry Understanding in Higher Dimensions, 2014–2025. URL: https://gudhi.inria.fr/.

[42] U. Bauer, Ripser: tool for efficient computation of Vietoris–Rips persistence barcodes, 2021. URL: https://github.com/Ripser.