The Problem of Stopping Software Tests and the Interpretation of Software Programs

PAOLO ROCCHI
IBM,
via Luigi Stipa 150, 00148 Roma,
ITALY

also with
LUISS University,
via Romania 32, 00197 Roma,
ITALY

Abstract: - We conducted empirical research on the defects found by customers when using six large software programs. The Wakeby (WAK) and Kumaraswamy (KUM) functions, recently discovered by statisticians, proved to be the best in terms of fitting the six time-series of defects. Two analytical reports on this empirical research have already been published; we do not repeat the details of the outcomes here, but instead illustrate the consequences deriving from the published outcomes. The abstract and applied properties of WAK and KUM provide new answers to the following managerial and theoretical problems.
- The first problem concerns managers who make the decision to stop software testing. The literature proposes various directional criteria which leave margins for subjectivity and incertitude. WAK and KUM provide a mathematical answer to the problem of terminating a software test.
- Most researchers have assigned software defects to the personal inabilities of developers, and the descriptions of software programs usually neglect the objective mechanisms for errors. Based on WAK and KUM, this paper suggests a territory model that illustrates the behavior of a software program with erroneous instructions.

This paper aims to raise a discussion of the original ideas generated by WAK and KUM. These ideas need to be further verified, and in its current form, this work can therefore be considered a *position paper*.

Key-Words: - Software defects; Management of software test; Modelling software programs execution; Wakeby and Kumaraswamy functions.

Received: March 17, 2024. Revised: August 21, 2024. Accepted: October 7, 2024. Available online: November 11, 2024.

1 Introduction

This paper reports on the third phase of a research project. In this section, we briefly review the first two phases for the convenience of the reader, and more details can be found in [1], [2].

1.1 First Phase

We first investigated a large-scale product from IBM that is used to monitor the performance and availability of all of the hardware (e.g. disks, memory, I/O, CPU, network) and software (e.g. application programs, management programs, etc.) resources of a large-scale computer installation. IBM developed four versions of this product with increasing complexity over time; each release passed alpha and beta tests and was released on the market, but had residual software defects that clients detected during normal operation. Requests for change (RFCs) by the clients covered a range from severe errors, such as abnormal termination of the program, to suggestions that vary or add simple functions. Each RFC was sent by a customer to the triage system in charge of handling software defects, and teams of software specialists promptly fixed the defects identified by the clients. It may be said that both the clients and the software developers jointly carried out an effective gamma test.

In a previous research work [1], we examined various aspects of this process, such as the discovery of software defects over time, the severity of the impact of a defect on the system, the times required to 'open' and 'close' a change (that are the times to initiate the fixing process and to terminate it), and other parameters. The time-series data on the defects, which relate the number of detected software defects to time, constitute the most important outcome of this study, in fact histograms exhibit very different shapes: concave, convex, oscillating and so on (Figure 1, Figure 2 and Figure 3). The variety of trends made it challenging to find the best statistical function fitting the empirical data.

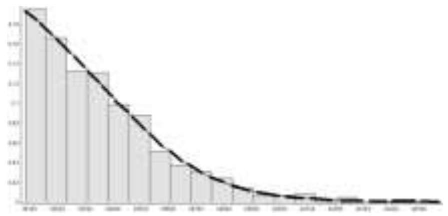
Fig. 1: Rapidly decreasing histogram
Source:[1]

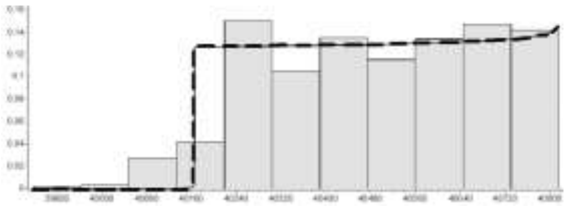
Fig. 2: Increasing histogram
Source:[1]

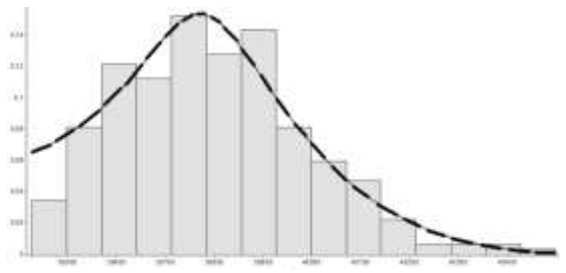
Fig. 3: Bell histogram
Source:[1]

We used a program called EasyFit to try forty candidate functions [3], and finally found that the Wakeby (WAK) was best adapted to the four time-series using the Kolmogorov-Smirnov test, which was accepted at the 99% significance level. This result shows the following features:

- The literature presents the power functions (e.g., Pareto, gamma, log-normal and other traditional functions) as best fitting to time-series data on software bugs [4], [5]; so WAK turns out to be unusual and surprising from the statistical viewpoint.
- WAK is one of the most recent distributions (Appendix A) defined for the purpose of modeling hydrological phenomena, [6], [7]. It is also employed in other research fields, such as sociology [8], energy management [9], and reliability but has been generally ignored in software engineering. To be more precise, we explored various repositories (ACM, IEEE, arXiv and others), and found only one work employs WAK for the purpose of developing a machine learning application, [10].

1.2 Second Phase

We therefore wondered whether the fitting with WAK a particular case or a strange coincidence was. In the second phase, we examined two *open-source packages*, Ubuntu and Android [2], whereas the first phase of research had studied four *proprietary software packages*. From the Ubuntu bug repository, we identified all the defects registered over six years. The errors noticed by Android customers were reported over about five years. An Anderson–Darling test (accepted at the 99% significance level) showed that the WAK function fitted the defects reported in Ubuntu, while a Kolmogorov–Smirnov test proved that the Kumaraswamy distribution (KUM) was the best fit for Android. The KUM distribution was devised to explore rainfalls and floods from a statistical perspective (Appendix B), and the results of the second phase are therefore in line with the first results.

This pair of research stages provides consistent results. The functions WAK and KUM have similar characteristics from the application stance, moreover they suggest new insights about computer science. In detail, they provide in-depth views about stopping software tests, that is a practical problem (Section 2), and software program modeling, representing a theoretical problem (Section 3).

2 The Problem of Stopping Software Tests

Error detection and correction require a great deal of time and remain some of the costliest aspects of software development.

2.1 State of the Art

Testing managers encounter significant difficulties, because software tests indicate how a code works only in certain controlled cases, and an erroneous logical condition can remain hidden for a long time, until it is executed. Program testing proves to be a very effective way to demonstrate the presence of software defects, but it is hopelessly inadequate for demonstrating their absence. Consequently, the more tests that are conducted, the more defects are discovered, but this method proves to be expensive, and determining a suitable number of tests is the central issue to be resolved.

Researchers have put forward several ways to follow. The authors of [11], [12] use software tools, while the researchers in [13] set up a graph, and those in [14] resort to artificial intelligence. A group of researchers exploited the theory of reliability in

[15], [16]. Testing managers are often inclined to find a trade-off between costs and benefits [17]; on the one hand, they aim to exploit the benefits of an earlier market introduction, while on the other, they prefer to defer the release of a product in order to enhance its functionalities or to improve its quality. Despite great efforts, solid criteria are lacking, [18]. The paper [19] concludes: "In current practice, deciding when to close a crowd-testing task is largely done by guesswork due to lack of decision support".

2.2 Separation Effect

WAK and KUM provide a precise answer to why have researchers have missed the mark so far. These two functions share a special mathematical property: the right and left tails of the function are independent of each other (Appendix A). This property, called the "separation effect," implies that *the final course of the curve cannot be predicted based on the initial data*. In the practice of software testing, this implies that it is not possible to accurately determine when to stop a test based on the faults found so far. WAK and KUM disprove in mathematical terms those who theorize about predictions, and prove that it is impossible to forecast when 100% of the defects will be discovered. The problem of stopping a software test has no rigorous solution, meaning that *the difficulties encountered by test managers do not come from professional inabilities but from the very software defects*, the number of which increases over time or decreases, is asymmetric or constant, etc. (Figure 1, Figure 2 and Figure 3).

2.3 Managerial Tactics

Although WAK and KUM demonstrate on a theoretical level that the problem of stopping a software test has no rigorous solution, we can adopt the following pragmatic criteria.

There are two mechanisms underpinning the test processes analyzed here: the discovery of defects and their correction. The former raises the time curve, while the latter lowers the curve. Together, they create variable and unpredictable time distributions of defects. Thus, the best that a testing manager can do is to analyze the most recent results and apply the following rules:

(1) If the number of the latest found defects is steadily decreasing, then the validation process can be stopped in conformity with the mathematical projections and the available resources (economic, organizational, technical etc.).

(2) If the temporal distribution has a counter-intuitive trend because the curve remains constant or is even going up instead of down, then debugging should proceed until condition (1) is met. The test manager may have to ask for more resources in view of the trend in the number of defects discovered at a given time.

As practical examples, the decreasing trend in Figure 1 indicates that the testing process is approaching its end, while the growing curve in Figure 2 indicates that tests must go on. Figure 3 shows a rapidly increasing trend which later drops down, indicating that the condition for stopping testing is not far away from being met.

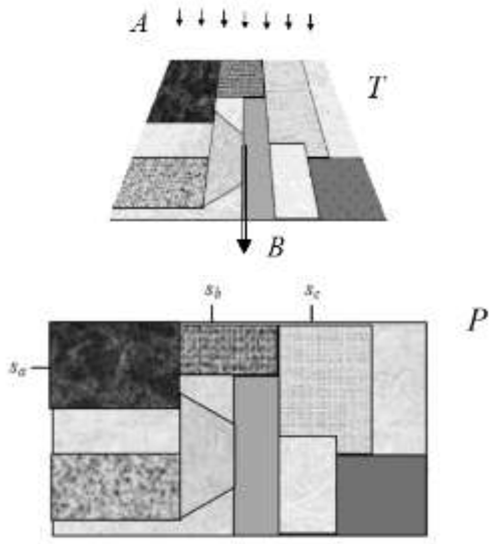

Fig. 4: The awash terrain T (top), and the territory model (bottom)

3 Interpretation of Computer Programs

For theorists, a computer program is a set of instructions [20] that can be formalized using propositional logic, set algebra and other mathematical tools, [21]. Methodological studies of software programming provide further accurate descriptions of programs. Block diagrams, pseudo-code, hierarchical diagrams, module diagrams, Voronoi maps, sequence diagrams, class diagrams and so on are used to illustrate special aspects of the software and to aid experts to implement software products, [22], [23], [24]. Both theoretical and professional models share a common characteristic: they describe a software program that functions correctly, and do not account for the software program that fails or is terminated abnormally. WAK and KUM are original statistical distributions that help us fill this gap.

3.1 Territory Model

A methodological premise needs to be established. Suppose that topics X and Y belong to two distinct fields, and that X is clear whereas Y is new and mysterious. If a mathematical model that applies to X also governs Y, then scientists can exploit the attributes of X to interpret Y. For instance, physicists have used the concept of water flow (X) to describe an electrical current (Y), which was difficult to understand in the early stages.

Both the proprietary programs and the open-source programs studied in the introduction are system programs which ordinarily run without interruption for a long period; hence, we can assume that on a global basis, the defect distributions discovered in [1] and [2] are not influenced by occasional events, human interferences or random factors. We can reasonably conclude that WAK and KUM relate to *the objective behavior of incorrect software programs.*

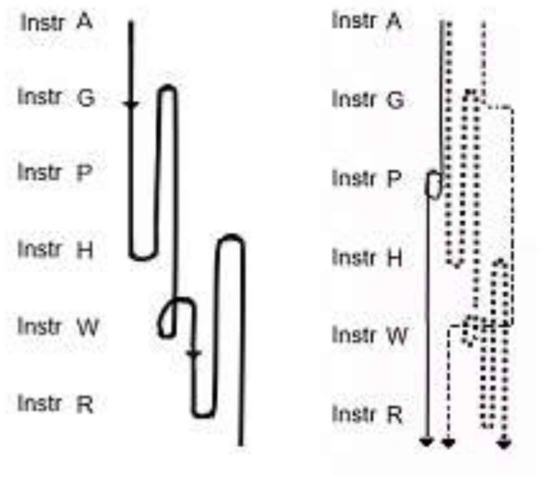

Fig. 5: Software instructions executed once (left) and three times (right)

The Wakeby and Kumaraswamy functions were developed to qualify a geographical territory deluged by floods, overflows and inundations; we can therefore think of software in terms of hydrology/geology. This is suitable because the execution of a program also involves eddies and flows, like water. Hence, we can use a territory T consisting of several plots flooded by water (Figure 4 top) to represent the software program P comprising modules affected by errors. The diagram representing both *T* and *P* is called *Territory Model* (TM) (Figure 4, bottom). We draw analogies between hydrology and software programming in order to better understand defective software programs.

3.2 Running Programs

Turing's machine carries out software operations one after the other, and the program execution can be understood as a continuous flow that conforms to the logic of the algorithm (Figure 5, left). When the program is activated several times, the processor creates a flow with several eddies due to loops, jumps, goto commands, etc. (Figure 5, right).

Suppose the mountain territory *T*, awash with rainfall A, includes areas *Sa, Sb, Sc* etc. which have different shapes, extensions, orientations and textures. The land consists of permeable and impermeable soils, cultivated and inhabited zones, etc. In a similar way, input data A are used to launch the program *P* that is subdivided into sectors *Sa, Sb, Sc* etc. which perform a variety of functions. These sectors are the *routines* and *modules* in procedural programming, and the *methods* in object-oriented programming. Some professional diagrams visualize these sectors in a manner very similar to the TM. For instance, the Nassi–Shneiderman diagram illustrates the sequence, selection and iteration of a structured program using special geometric shapes, [25]. The labels indicate the functions of each section and make the logic of the algorithm explicit (Figure 6).

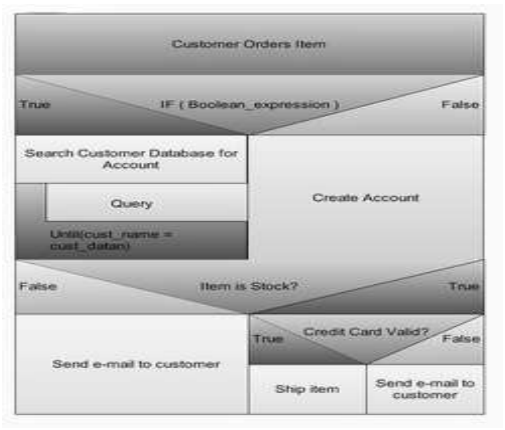

Fig. 6: Nassi–Shneiderman diagram for a program handling a customer order

3.3 The Advantages of TM

In similarity to the terrain *T* which gives out the regular creek B or undergoes a flood (Figure 4 left), the software package *P*, executed several times (Figure 5 right), provides the correct result or otherwise runs into failure. We can use the Territory Model to explicate the performances of *P* using the parallel characteristics of the terrain *T*. The following list pinpoints the symmetries:

1. The terrain T covers a certain geographical area ↔ P occupies a portion of the computer memory.

2. The assortment of sectors characterizes the geographical territory ↔ The number of modules indirectly indicates the complexity of the algorithm.

3. The territorial sections vary significantly in a large area whereas diversities are negligible if T is small ↔ Subdivision into modules is necessary when P is large and negligible in a small program.

4. Each geographic sector has a special geomorphological property: inhabited, arid, cultivated etc. that facilitates or hinders waterways ↔ Each software sector executes a specific function that runs linearly or makes intricate the program running.

5. The rain A falls on the various zones of T ↔ Input-data activate the functions of the software sectors.

6. The borders establish the precise relations amongst the territorial sectors ↔ Software sectors *Sa, Sb, Sc* etc. interact through well-defined interfaces for data exchange.

7. Heavy rains hit every spot on the ground and can result in overflowing ↔ The repeated executions of P check every part of the algorithm and reveal software defects.

8. A regular area of T conveys water into B; instead, obstructed and rocky sectors cause overflows or inundations ↔ The software module brings forth the correct output; otherwise, it has defects or even causes an abnormal end.

9. The flatter the sector of T, the better the rain is absorbed and flows ↔ The more linear the logic of the software module, the less affected by errors.

10. Excess water can generate an abnormal ‘cascade effect’ on T ↔ A software error can multiply the number of subsequent faults and creates a ‘cascade effect’.

11. Inundation has very different behaviors in the initial and final stages ↔ The temporal series of bugs present apparent dissimilarities in the left and right tails.

In conclusion, the Territory Model illustrates the failure proneness of software programs. TM shows how errors deviate the correct execution of the algorithm and can even result in a multiplicative effect (points 8, 10 and 11). TM proves that the extension and complexity of the algorithm are the objective root-causes of software failures (points 2, 3, 4 and 9).

Authors normally ascribe the origin of software errors to human factors, [26], [27]. The cause-effect relationships are identified as cognitive problems, misunderstandings, mistakes and subjective inabilities, [28], [29], [30]; instead, TM highlights mechanisms which are independent of human factors. TM assigns the origins of the software defects to objective situations characterized by extension and complexity of logic. This also match with recent inquiries which seek for the ‘error prone regions’ that are identified using genetic algorithms, [31].

The concept of software engineering has been the subject of vigorous debate over the years. Researchers have raised doubts about software engineering that seems pointless and not a real engineering field, [32]. Troubled software projects and the theories of computing which addressing abstract issues support this criticism in a way. In fact, there is a certain divide between the professional practice and the theories that sometimes are fragmentary and abstract, [33]. The Territory Model, underpinned by WAK and KUM, enhances our knowledge about the fault-proneness of programs in scientific terms and supports the engineering status of software technology.

4 Discussion and Conclusion

The Wakeby and Kumaraswamy functions best fit six temporal distributions of software defects and constitute the first unexpected finding of this research project since they are new in the literature of software debugging. Moreover, WAK and KUM provide innovative answers to a pair of long-discussed issues.

The tails of WAK and KUM are independent each other this property implies that the history of a software test process does not influence the final phase. The two distributions demonstrate in mathematical terms that the problem of stopping test cannot be resolved in a rigorous manner and this conclusion matches with universal experience. As a consequence, a manager should close testing based on the current trend of bug series, and this method also corresponds to and improves the ITIL guidelines about the so called ‘problem management’, [34].

WAK and KUM establish a logical parallel between hydrology and the software, especially between an inundated territory and executed software programs. To make this concrete, we propose the Territory Model, which expounds analytical analogies and improves our understanding of incorrect software. TM highlights how the risks of software failures increase in proportion to the complexity and extent of the software package. This reading key, focusing on objective features, provides insights rather distant from the current subjective views of programming errors and supports the engineering status of software technology.

The original ideas that WAK and KUM spawn should be further verified. It would be useful to examine various time-series of defects, but this inquiry takes much time. The present report is a position paper which aims to quickly inform the scientific community of the results, while further initiatives will start in the future.

References:

[1] Rocchi P., Spagnoletti P., Datta S. - An ecological model for digital platforms maintenance and evolution - In *Organizational Innovation and Change*, Springer (2016), pp. 1-18, [Online]. https://ink.library.smu.edu.sg/sis_research/6007 (Accessed Date: October 1, 2024).

[2] Rocchi P. - Statistical distributions of software bugs and testing management - *Proc. Intl. Conference on Computational Science and Computational Intelligence* (2018), pp. 835-840. DOI: 10.1109/CSCI46756.2018.00167.

[3] Dansereau S., Hruby J., Wakefield D. - *Easy Fit: A Simple Guide on How to Look and Feel Great* - (2011) Author House Publisher.

[4] Shriram C.K., Muthukumaran K., Bhanu Murthy N.L. (2018) - Empirical study on the distribution of bugs in software systems - *Intl. J. of Software Engineering and Knowledge Engineering*, 28(1), (2011), pp. 97-122. DOI: 10.1142/S0218194018500055.

[5] Marchesi M., Murgia A., Tonelli R. - On the distribution of bugs in the eclipse system - *IEEE Trans. on Software Engineering*, 37(6), (2011), pp. 872-877, [Online]. https://hdl.handle.net/11584/104704 (Accessed Date: October 1, 2024).

[6] Rao A.R., Hamed K.H. - *Flood Frequency Analysis* - (2019) CRC Press.

[7] Nadarajah S., Eljabri S. - The Kumaraswamy GP distribution - *Journal of Data Science*, 11, (2013), pp. 739-766. DOI: 10.6339/JDS.2013.11(4).1189.

[8] Sadiq N. - Wakeby distribution modelling of rainfall and thunderstorm over northern areas of Pakistan - *Proc. of the Pakistan Academy of Sciences*, 53 (2), (2016), pp. 121–128, [Online]. https://www.paspk.org/wp-content/uploads/2016/06/Wakeby-Distribution-Modelling.pdf (Accessed Date: October 1, 2024).

[9] Opere A., Mkhandi S., Willems P. - At site flood frequency analysis for the Nile Equatorial basins - *Physics and Chemistry of the Earth*, 31(16-16), (2006), pp. 919-927. DOI: 10.1016/j.pce.2006.08.018.

[10] Heth C.D., Cornell E. - Characteristics of travel by persons lost in Albertan wilderness areas - *Journal of Environmental Psychology*, 18, (1998), pp. 223-235. DOI: 10.1006/jevp.1998.0093.

[11] Jung C., Schindler D. - The role of air density in wind energy assessment: A case study from Germany - *Energy*, 171, (2019), pp.385-392. DOI: 10.1016/j.energy.2019.01.041.

[12] Dehbi Y., Plümer L. (2011) - Learning grammar rules of building parts from precise models and noisy observations - *ISPRS Journal of Photogrammetry and Remote Sensing*, 66, pp. 166–176. DOI: 10.1016/J.ISPRSJPRS.2010.10.001.

[13] Alghmadi H.M., Syer M.D., Shang W., Hassan A.E. - An Automated Approach for Recommending When to Stop Performance Tests - *IEEE Intl. Conf. on Software Maintenance and Evolution*, (2016), pp. 279-289. DOI: 10.1109/ICSME.2016.46.

[14] Loll V. - Developing and testing algorithms for stopping testing, screening, run-in of large systems or programs - Proc. of *Annual Reliability and Maintainability Symposium*, (2000), pp. 124-130. DOI: 10.1109/RAMS.2000.816295.

[15] Dalal S.R., Mallows C.L. - Some graphical aids for deciding when to stop testing software - *IEEE Journal on Selected Areas in Communications*, 8(2), (1990), pp. 169-175. DOI: 10.1109/49.46868.

[16] Syahana N., Saharudin A., Wei K.T., Na K.S. - Machine learning techniques for software bug prediction: a systematic review - *Journal of Computer Science*,16(11), (2020), pp. 1558–1569. DOI: 10.3844/jcssp.2020.1558.1569.

[17] Yang M.C.K., Chao A. - Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs - *IEEE Trans. on Reliability*, 44(2),

(1995), pp. 315-321. DOI: 10.1109/24.387388.
[18] Ross S.M. - Software reliability: The stopping rule problem - *IEEE Trans. on Software Engineering*, SE-11(12), (1985), pp. 1472-1476, [Online]. https://api.semanticscholar.org/CorpusID:19751154 (Accessed Date: October 1, 2024).
[19] Huang Chin-Yu; Lyu M.R. - Optimal release time for software systems considering cost, testing-effort, and test efficiency - *IEEE Trans. on Reliability*, 54(4), (2005), pp. 583-591. DOI: 10.1109/TR.2005.859230.
[20] Couto C., Silva C., Valente M.T., Bigonha R., Anquetil N. - Uncovering causal relationships between software metrics and bugs - *Proc. 16th European Conference on Software Maintenance and Reengineering*, (2012), pp. 223-232, [Online]. https://inria.hal.science/hal-00668151v1 (Accessed Date: October 1, 2024).
[21] Wang J., Yang Y., Yu Z., Menzies T., Wang Q. - Crowdtesting: When is the party over? - (2018) DOI: 10.48550/arXiv.1805.03218.
[22] Agazzi E. (ed) - *The legacy of A. M. Turing* - (2013) Franco Angeli Editore.
[23] Watumull J. - A Turing program for linguistic theory - *Biolinguistics*, 6(2), (2012), pp. 222-245. DOI: 10.5964/bioling.8907.
[24] Blokdyk G. - *Functional Flow Block Diagram Standard Requirements* - (2018) Emereo Pty Limited.
[25] Tenzer J., Stevens P. - Modelling recursive calls with UML state diagrams - *Proc. of the 6th intl. Conf. on Fundamental Approaches to Software Engineering*, (2003), pp. 135-149. DOI: 10.1007/3-540-36578-8_10.
[26] Fowler M. - *UML Distilled* - (2018) Pearson Education Inc.
[27] [25] Nassi I.; Shneiderman B. - Flowchart techniques for structured programming - *SIGPLAN Notices XII*, (1973) pp.12-26, [Online]. http://www.cs.umd.edu/hcil/members/bshneiderman/nsd/1973.pdf (Accessed Date: October 1, 2024).
[28] Davies S., Roper M., Wood M. - A preliminary evaluation of text-based and dependency-based techniques for determining the origins of bugs - *Proc. 18th Working Conference on Reverse Engineering*, (2011), pp. 201-210. DOI: 10.1109/WCRE.2011.32
[29] Zeller A. - *Why Programs Fail* - (2009) Elsevier.
[30] Tao Y., Chen Z., Liu Y., Xuan J., Xu Z., Qin S. - Demystifying "bad" error messages in data science libraries – *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (2021), pp. 818–829. DOI: 10.1145/3468264.3468560.
[31] Huang F., Strigini L. - Predicting software defects based on cognitive error theories - *Proc. IEEE Int. Symposium on Software Reliability Engineering*, (2018), pp. 134-135. DOI: 10.1109/ISSREW.2018.00-16.
[32] Dhillon B.S. - *Computer System Reliability: Safety and Usability* - (2013) CRC Press.
[33] Birt J. - *Software Reliability Enhancement Through Error-prone Path Identification Using Genetic Algorithms* - (2006) Thesis of Griffith University, Australia, [Online]. https://research.bond.edu.au/en/publications/software-reliability-enhancement-through-error-prone-path-identif (Accessed Date: October 1, 2024).
[34] Capretz L.F., Ahmed F. - Why do we need personality diversity in software engineering? - *ACM SIGSOFT Software Engineering Notes*, 35(2), (2010), pp. 1-11, [Online]. https://api.semanticscholar.org/CorpusID:13035695 (Accessed Date: October 1, 2024).
[35] Rocchi P. - Guest Editorial: Informatics and electronics: Some educational remarks - *IEEE Transactions on Education*, 59(3), (2016), pp. 233-239. DOI: 10.1109/TE.2016.2528891.
[36] Howard D. - *IT Release Management: A Hands-on Guide* - (2011) CRC Press.
[37] Houghton C.J. - Birth of a parent: The Wakeby distribution for modeling flood flows - *Water Resources Research*, 14(6), (1978), pp.1105-1109. DOI: 10.1029/WR014i006p01105.
[38] Gyeong-Deok K., Jun-Haeng H., Won-Cheol j. - A study on separation effect of rainfall data using a Wakeby distribution - *Proc. of the Korea Water Resources Ass. Conf.*, (1995), pp. 303-308, [Online]. https://koreascience.kr/article/CFKO199511920081384.pdf (Accessed Date: October 1, 2024).
[39] Kumaraswamy P. - A generalized probability density function for double bounded random-processes - *Journal of Hydrology*, 46, (1980), pp. 79-88. http://dx.doi.org/10.1016/0022-1694(80)90036-0.
[40] Cordeiro G.M., de Castro M. - A new family of generalized distributions - *Journal of*

Statistical Computation and Simulation, 81, (2011), pp. 883-898. DOI: 10.1080/00949650903530745.

[41] Abbas S., Muhammad M., Jamal F., Chesneau C., Muhammad I., Bouchane M. - A new extension of the Kumaraswamy generated family of distributions with applications to real data - *Computation,* 11(2), (2023), pp.26. https://doi.org/10.3390/computation11020026

[42] de Oliveira H.M., Cintra R.J. - A new information theoretical concept: information-weighted heavy-tailed distributions - (2016) DOI: 10.48550/arXiv.1601.06412.

[43] Wang X., McCallum A. - Topics over time: a non-Markov continuous-time model of topical trends - *Proc. 12th Intl. Conf. on Knowledge Discovery and Data Mining*, (2006), pp. 424–433, [Online]. https://people.cs.umass.edu/~mccallum/papers/tot-kdd06.pdf (Accessed Date: October 1, 2024).

[44] Zhang Y., Jiang T., Yang T., Li X., Wang S. - HTKG: Deep keyphrase generation with neural hierarchical topic guidance - *Proc. 45th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval,* (2022), pp. 1044–1054. DOI: 10.1145/3477495.3531990.

[45] Wang G., Guo R., Sakurai Y., Babar A.M., Guo M. - Mechanism design for public projects via neural networks - *Proc. 20th Intl. Conf on Autonomous Agents and MultiAgent Systems*, (2021), pp. 1380–1388. DOI: 10.48550/arXiv.2002.11382.

[46] Tomczak J.M. - Improving neural networks with bunches of neurons modeled by Kumaraswamy units: Preliminary study - (2015) arXiv:1505.02581.

[47] Garcin M., Stéphan S. - Credit scoring using neural networks and SURE posterior probability calibration - (2021) DOI: 10.48550/arXiv.2107.07206.

Appendix A

Climate change cause abnormal rainfalls and unusual water floods. Extreme values and apparent skewness characterize hydrologic phenomena, so statisticians have made considerable efforts to set up suitable mathematical tools to address such special events, say new indices, accurate distribution functions, a theory of extremes etc. Distribution functions, such as GEV (Generalized Extreme Value), TCEV (Two Components Extreme Value) and WAK (Wakeby), characterized by three or more parameters, are able, better than traditional distributions, to reproduce the statistical behaviors of floods.

Wakeby is a five parameters function, more than most of the common distributions; and was originally introduced in 1978 by John C. Houghton who named it after Wakeby pond on Cape Cod, [35]. Houghton parameterized WAK by the following quantile function

$$x(F) = \xi + \frac{\alpha}{\beta}\left[1-(1-F)^{\beta}\right] - \frac{\gamma}{\delta}\left[1-(1-F)^{\delta}\right]. \quad (A.1)$$

Where $x(F)$ is the probability of occurrence of x, that can be regarded as a function to calculate percentile. For example, the value $x(0.5)$ would correspond to the 50th percentile. The value ξ is the location parameter; α and γ are the scale parameters; β and δ are the shape parameters of the left end-tail and of the right-end tail respectively. In fact, the *Wakeby can be thought of in two parts, the right-hand tail* $-(1-F)^{\delta}$ *and the left-hand tail* $(1-F)^{\beta}$. *WAK mimics the final trend of data that is independent of the initial trend (and vice versa) by means of δ. This 'separation effect' ordinarily draws the attention of experts in hydrology,* [36]. When $\delta > 0$, WAK has a heavy upper tail and can model occasional high outliers.

The parameters of WAK are constrained by the following conditions:

(a) If $\alpha = 0$, then $\beta = 0$.
(b) If $\gamma = 0$, then $\delta = 0$.
(c) $\gamma \geq 0, (\alpha + \gamma) \geq 0$.
(d) Either $(\beta + \delta) > 0$ or $(\beta + \gamma) = \delta = 0$.

The moments of all orders exist, provided $\delta \leq 0$ [36]. If δ is positive, $E[Xr]$ exists for $0 \leq r < (\delta - 1)$. The probability moment can be defined this way

$$\alpha_r = E\{X[1 - F(X)]^r\} \quad (A.2)$$

And has this simple form for $r \geq 0$

$$\alpha_r = (r+1)^{-1}\{\xi + \alpha\,(r+\beta+1)^{-1} + \gamma\,(r-\delta+1)^{-1}\} \qquad (A.3)$$

In particular, if $r = 0$ we get from (A.2)

$$\alpha_0 = \xi + \alpha\,(\beta+1)^{-1} + \gamma\,(-\delta+1)^{-1} = E[X] \quad (A.4)$$

All this allows for a wider variety of shapes and the distribution is well suited to simulation of intricate physical phenomena. The Wakeby distribution exhibits more stability under small perturbations when compared to the Beta distribution and other more common distributions. In summary, WAK is highly general; it can describe complex events whose final behaviours dislike the initial trends; it is robust against outliers, and it has a closed functional form for determining percentiles.

Here the outcomes obtained in the first research phase, [1]. Table A1 includes the number of defects discovered by users, the time interval covered by the gamma test of each release, the histograms that relate the number of defects to time and the dimension of the bars. Table A2 shows the parameters that fit with the four releases of the IBM software package. Columns D and P indicate the statistic and the p-values.

Table A3 exhibits the parameters depicting the right and left tails of time-series which demonstrate far different trends. Table A4 shows the values of the parameters fitting with the empirical data of Ubuntu, [2].

Table A1. Data about the time-series

Release	Number of defects	Time interval (days)	Figure	Histogram Segment (days)
1	838	1471	1	73.5
2	322	1074	3	71.6
3	495	975	N.A.	75.0
4	593	939	2	85.36

Table A2. Parameters that optimize the Wakeby distribution for each release

Release	D	P	α	β	γ	δ	ξ
1	0.01986	0.88908	261.35	0.82955	121.3	0.10885	39082.0
2	0.02816	0.95414	803.23	4.4198	228.77	-0.21498	39460.0
3	0.0205	0.98272	742.2	5.5649	333.95	-0.44585	39749.0
4	0.04052	0.27695	6.7581E+8	16826.0	676.17	-1.0297	0

Table A3. Parameters that detail the tails of each distribution

Release	Tail	D	P	α	β	γ	δ	ξ
1	Left	0.02256	0.91524	-97.495	3.8297	330.08	-0.85276	39085.0
1	Right	0.07706	0.97820	1.5900E+9	39906.0	226.47	-0.18278	0
2	Left	0.05626	0.65205	1008.8	12.49	304.76	-1.11	39427.0
2	Right	0.06502	0.90994	1.0533E+10	2.6363E+5	122.36	-0.00949	0
3	Left	0.05435	0.62191	396.72	6.8342	242.63	-1.1144	39733.0
3	Right	0.04618	0.62882	6.9669E+9	1.7384E+5	240.32	-0.42067	0
4	Left	0.09203	0.68535	7030.3	27.326	269.83	-1.5908	39813.0
4	Right	0.0579	0.2075	407.44	1.1267	0	0	40452.0

Table A4. Parameters that optimize the Wakeby distribution for Ubuntu

	R	D	P	α	β	γ	δ	ξ
Ubuntu	4	0.09182	0.17134	1021.8	0.3368	0.0	0.0	38330.0

Appendix B

Poondi Kumaraswamy, an engineer and hydrologist from India, developed a two-parametric function for double bounded random processes, [37]. Following the approach of [38], the cdf of Kumaraswamy is as follows for any baseline cumulative distribution function *G*(*x*):

$$F(x) = 1 - [1 - G(x)^{a}]^{b}, \qquad a > 0, b > 0. \quad \text{(B.1)}$$

and the corresponding probability density function (pdf) is

$$f(x) = ab\square g(x) G(x)^{a-1}[1 - G(x)^{a}]^{b-1}. \quad \text{(B.2)}$$

Where *g*(*x*) = *dG*(*x*)*/dx* is the baseline pdf; a and b are shape parameters whose role is partly to introduce skewness and to vary tail weights. The pdf can be unimodal, increasing, decreasing or constant, depending on *G*(*x*) and the two parameters that is why this distribution is applicable to many natural phenomena, [39].

KUM is used to analyze rainfall time series, regional flood frequencies, wind extremes, river flow modeling, atmospheric temperatures etc. KUM is also used in computing literature, for example, about Shannon information [40], data mining [41], and especially in the design of neural networks [42], [43], [44], [45].

The simplest cdf and pdf of Kumaraswamy are the followings:

$$F(x) = 1 - (1 - x^{h})^{(g-1)} \quad h > 0, g > 0; \quad 0 < x < 1. \quad \text{(B.3)}$$

$$f(x) = hg\square x^{(h-1)}(1 - x^{m})^{(g-1)}. \quad \text{(B.4)}$$

Where *η* and *γ* are the shape parameters. EasyFit utilizes the following pdf version:

$$f(x) = \frac{a_1 a_2 z^{(a_1 - 1)}\left(1 - z^{a_1}\right)^{(a_2 - 1)}}{(b - a)}, \quad \text{where } z = \frac{x - a}{b - a}. \quad \text{(B.5)}$$

Table B1 exhibits the values fitting the Kumaraswamy distribution with the timeseries of Android defects [2].

Table B1. Parameters of KUM

Sample Size Statistic P-Value Rank	1,016 0.05482 0.00428 1				
□	0.2	0.1	0.05	0.02	0.01
Critical Value	0.03366	0.03837	0.0426	0.04762	0.05111
a	*b*	α_1		α_2	
1.0	430.38	1021.8		0.3368	

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.