Indonesian Journal of Electrical Engineering and Computer Science
Vol. 32, No. 3, December 2023, pp. 1361~1369
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v32.i3.pp1361-1369

Design and development of frameworks for CPU verification efficiency improvement

Sheetal Singrihalli Hemaraj[1], Shylashree Nagaraja[1], Sunitha Yariyur Narasimhaiah[2], Madhu Patil[3]
[1]Department of Electronics and Communication Engineering, RV College of Engineering, Bengaluru, India
[2]Department of Electronics and Communication Engineering, SJB Institute of Technology, Bengaluru, India
[3]Department of Computer Science and Engineering, BGS College of Engineering and Technology, Bengaluru, India

Article Info

Article history:

Received Jun 30, 2022
Revised Jan 31, 2023
Accepted Oct 8, 2023

Keywords:

Debugging
Functional verification bug triage
PERL
Triaging
YAML

ABSTRACT

Bug finding is a critical component of the verification flow and is resource intensive. In a typical week, a debug engineer writes triages, which take up significant amount of time that could be spent debugging another unique issue, and the lack of standardization in scripting causes maintainability issues in functional verification bug triage. A framework that allows customizable triage script generation is developed based on inputs from the engineer deploying YAML isn't another markup language (YAML) files and practical extraction and report language (PERL) scripting, and this methodology is made automated and is standardized across projects to ensure maximum benefit going forward. The use of auto-triage in the project of functional verification bug triage has contributed to a 18% increase in triaged signatures on average, from 40% before its use to 58% after. A similar earlier project vs. current project comparison shows a 20% uplift. The triaged inputs that are parsed are currently being fed to a machine learning algorithm, which will help further improve the debug efficiency. As part of future work, the information from input YAML files can be used to analyze simulation failure attributes, hence improving the overall efficiency of debugging.

Corresponding Author:

Shylashree Nagaraja
Department of Electronics and Communication Engineering, RV College of Engineering
RV Vidyanikethan Post, 8th Mile, Mysuru Road, Bengaluru, Karnataka, India
Email: shylashreen@rvce.edu.in

1. INTRODUCTION

In any verification industry, source codes, details regarding the project, bugs, or simulation failures are stored in large-scale databases called repositories. One such repository for storing all the details related to bugs or simulation failures is the bug repository. Core verification companies spend a significant amount of time just debugging the issues and filing them. A bug tracking system will be employed by most of the large ventures to support debug details and aid developers in handling debug details. The quantity of bug data is one of the challenges that engineers face. As the complexity of the processor increases, it becomes difficult to handle such large-scale information manually. The other challenge is its quality. The quality is measured in terms of redundancy, which squanders the limited debugging time. The efficiency of the core verification process can be enhanced by reducing the redundancy and introducing automation wherever possible. At the projectpeak, several thousand simulations are run every week for complex central processing unit/processor (CPU) cores. Among them, a few thousand produce simulation failures. The failures produced are not all unique, and most of them can be categorized based on some unique fingerprints that are found in logs. This process is called triaging. For example, there are 100 failures of a particular type called "signature." One is debugged, and there are 99 unassigned fails. To identify the failures that can be associated with the same bug, a triage script is written that parses logs from all the unassigned failures and matches the unique fingerprint linked to that bug and marks them as debugged. In a typical week, a debug engineer writes 3 triages, which take 30 minutes each and eat up the time that could be spent debugging a unique issue. In most cases, triaging is considered as assigning the different bugs to the developer accurately. A predictive model is used to determine which developer is best suited to analyze the bug [1], [2]. Extended techniques were proposed to enhance the accuracy of text classification, such as reduction techniques [3]. Also, formulation of bug reports that are adversarial have helped to some extent [4]. Data reduction methods are adopted to reduce the redundancy and increase the quality of the bug data in the repositories [5]–[7]. Bug triaging has been made automated using numerous machine learning techniques [8]. The graph methods were also adopted for efficient triaging [9]-[25]. The contributions made by this paper primarily are Time consumed due to the manual triaging process has been very high, and this problem has been addressed by automating the same. The problem of lack of stan- dardization and maintainability issues has been addressed. The paper is organized as follows: Section 2 gives the analysis of the triaging process and the manual triaging. In section 3 provides the details on how the triaging gets automated using practical extraction and report language (PERL) scripting and how the issues of standardization and time consumption have been eliminated. In section 4 provides results and analysis. In section 5 contains the conclusion.

2. TRIAGING PROCESS

The flow of the core verification bug triage process is as shown in Figure 1. The fails get assigned to each of the engineers for the purpose of debugging. The status of that fail becomes "assigned." The path for debugging the fail will be present in the database itself. The fail has to be debugged to formulate the bug report. The procedure for debugging any fail in is discussed in this section. The sim.out file, which is the output file of the simulation, has to be read and analyzed. The error due to which the corresponding fail has emerged will be obtained here in the sim out file. Signatures are unique fingerprints that are used to classify failures. Fails that occur due to the same root cause will have similar signatures. These signatures indicate the root cause of the fail. The details of the signature will be present in sim.out files along with the last instruction due to the execution of which the error was encountered. After checking the sim.out file, other configuration and register transfer level (RTL) microcode files have to be checked where the other information required, such as dispatch and retire cycles of each instruction, registe contents after the execution of each instruction, and exceptions caused is checked, thereby finding the root cause for it.

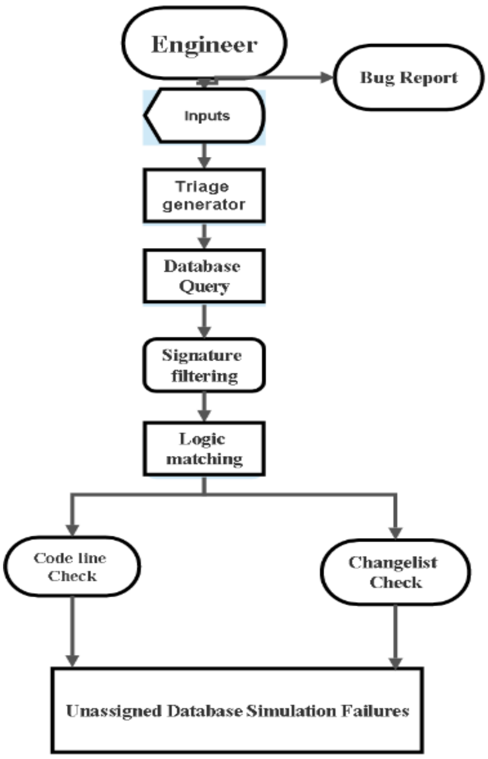


Figure 1. Core verification bug triage process flow

Then the information collected is produced in detail in the form of a bug report. Once the bug report has been filed, a big number will be obtained and the status of that fail in the database will change to 'debugged' as shown in Figure 2. The bug number is used to mark similar fails as "debugged." The next step is to create a triage script. The triage script is written in the PERL language. The PERL scripting language was chosen as it is well known for processing texts and analysis of strings. This script is used to categorize all the failures that occurred due to the same root cause. A triage script contains fields such as the owner's name, batch name, i.e., the regression batch to which it belongs, signature, and other information. The first section of the triage script contains the database query. Simulation results will have simulation fails from all the regressions. It has to be made sure that selected fails are from the current project only.

A database query includes entering into the database and parsing the logs. The database module at the back end is responsible for the previously mentioned operations. Many arguments have to be given to the database query module, such as project name, owner name, regression batch and bug number. All the fails that are of the project other than the one mentioned in the database query will be removed. The next section of the triage script is the signature matching. The script parses the sim.out log file, does string comparison, and checks if the signatures match. If so, then it moves to the next part of the code. The next section is the logic matching for which the RTL code log file has to be parsed. The script looks for any exceptions and the last instructions due to which the fail is encountered. If these match, the next few sections are code line and the change list check. The fails which satisfy all these checks will be marked as "debugged" with the same bug number as they have been caused by the same root cause. Similarly, all the other fails in the database get marked as 'debugged' thereby achieving the faster verification cycles. The manual process of triaging as depicted by Figure 3 includes writing the whole triage script manually, and these triage scripts are unique for each fail. After debugging a specific fail, the engineer writes the entire triage script for that The database query set up function is hard coded within the script itself. So, unless the source code itself is changed, the database query cannot be altered. This dependency makes it harder to port the triage script across different regressions. In a typical week, a debug engineer writes 3 triages, which take around 30 minutes each and eat up the time that could be spent debugging a unique issue. So, automation is required.

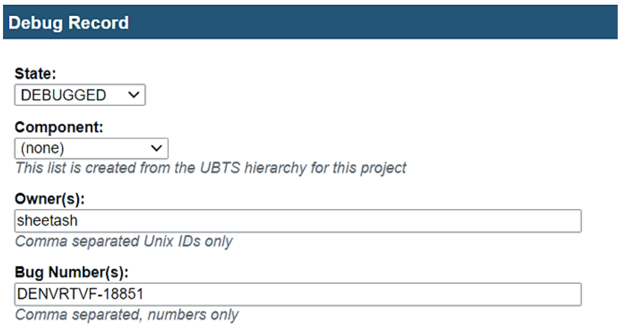


Figure 2. Window showing the status of the fail

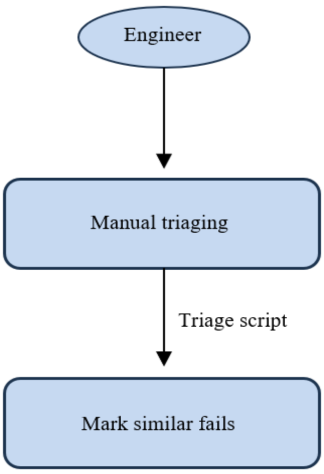


Figure 3. Current triaging process

3. PROPOSED METHOD

The proposed solution replaces the manual effort in triaging with a YAML isn't another markup language (YAML) input-based methodology and is represented in Figure 4. The user needs to add the triage inputs in a key: value format that is then processed and returned in the form of a PERL script that follows a standardized format. The automated triage script that is now generated is independent of the database query attributes. It invokes a specialized PERL module called the triage helper that handles the setup code of the triage so that only the matching logic remains in the generated PERL script. A command line interface is adopted that accepts the query attributes that can range from feature bring up regressions to derivative core regressions, ensuring portability of triages across regressions. To give inputs, a YAML format file is used. Data in YAML is written in key: value pair format. The details that are to be given are just these oneword values. The details to be provided are owner name, error message, bug number, signature, and instruction. These details are given to the framework called triage generator that generates a triage script automatically considering YAML as input file.

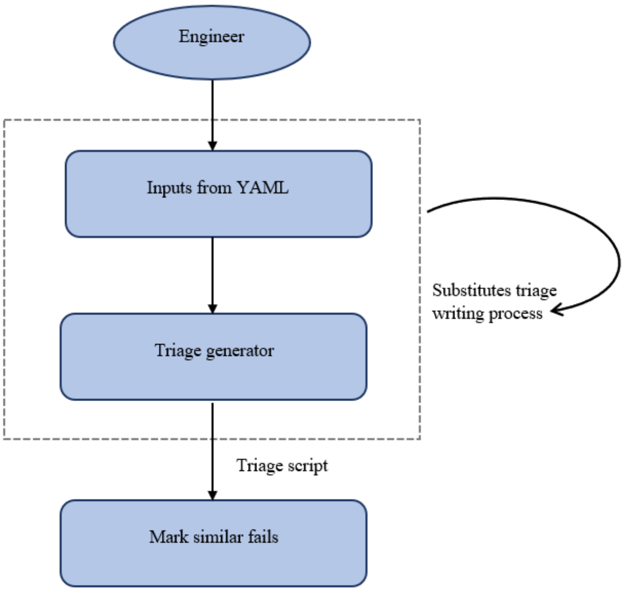


Figure 4. Automated triaging process

3.1. Triage generator

The triage generator plays an important role in the generation of the triage script. The triage gen-erator is built using PERL language that enables it to read the information present in the YAML file and setup the database query on its own. The process in which the triage helper is designed and it proceeds is shown in Figure 5. The signature field from the YAML file is taken as the first argument and is used to parse the sim.out file where the script searches for the string "error:" and reads until the end so as to get the information about the fail. Then it tries to compare the strings obtained from the YAML file with the strings read from the new sim.out file which the script parses. If the signature matches, then it goes to the next section, which is logic matching. The Triage script has to parse the configuration files, RTL files, and simulation files in order to perform the logic matching. For this operation, the script removes all the other strings except the series of strings present in the last instruction cycle. Once the last instruction and the register contents match, then it tries to match the exceptions or interrupts that occurred if there are any, in the YAML file. Then the code line and changelist checks occur. There are separate modules for each of them, and the triage generator automatically generates the entire script required. If the bug number is ABCD001, then the triage script generated by the triage generator will be saved as ABCD001.pl. The format of the triage script is shown in Figure 6. There occurs redundancy problem due to several problems such as side regression, tape out branch and derivativecores triaging. To solve the redundancy problem, the solution involves making the triage smarter.

3.2. Proposed solution to eliminate redundancy

Towards the back end of the project, in order to keep the tape out code line clean, a branch is forked off the trunk called the tape out branch. The tape out branch creates an obstacle in triaging because now the triages need to cater to two different code lines. Previous projects circumvented this problem by creating a clone of each trunk bug on the tape out branch and then using that to create two identical triages that only differ in the bug number they use. This can be observed in Figure 7. The p1 label mentioned in Figures 7-10 indicates parent core name representing the project name. This creates a redundancy in the bug filed since most bugs are not even considered for a tape out branch fix and are outright rejected, while creating two identical triages uses up unnecessary disc space.

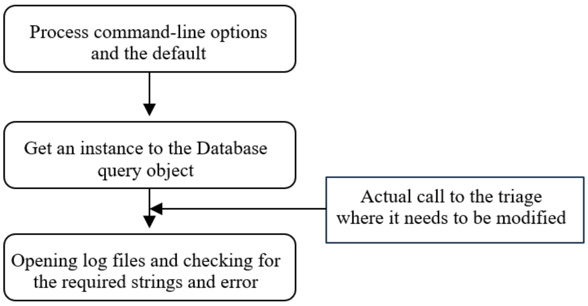
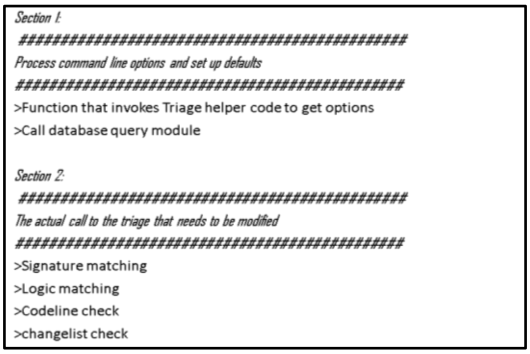


Figure 5. Triage flow



Section 1:
##
Process command line options and set up defaults
##
>Function that invokes Triage helper code to get options
>Call database query module

Section 2:
##
The actual call to the triage that needs to be modified
##
>Signature matching
>Logic matching
>Codeline check
>changelist check

Figure 6. Triage script details

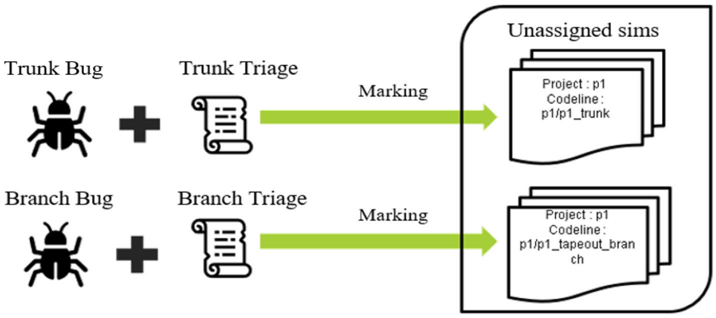


Figure 7. Redundand flow

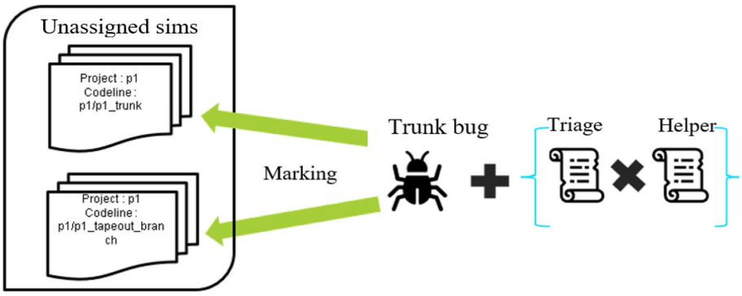


Figure 8. Proposed solution to reduce redundancy

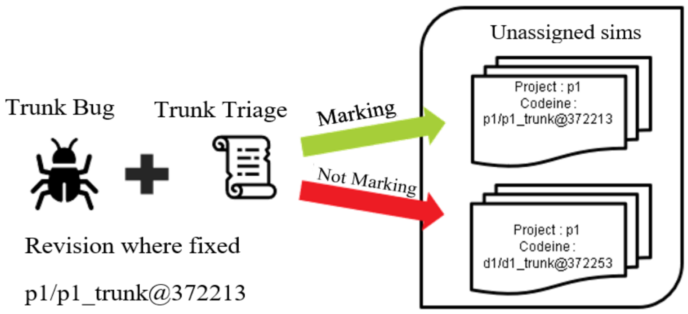


Figure 9. Triage failing at the code line check

Only issues approved for a tape out branch go through a creep process where a bug is automatically filed to track that fix on the branch. The triage helper uses the original bug number to issue an API call where it picks up the bug number and the code line for the automatically filed creep bug. It then creates an interface and leverages the existing database triage code, enabling a single triage script to service both code lines. This is represented in Figure 8. Text mining was also being used to reduce redundancy [20]. In a CPU core that spawns multiple derivative cores, it becomes important to extend the triaging process to those cores that undergo a process of auto integration using the parent core code line. The debug engineer who looks at the simulation failures of derivative cores often runs into issues that were found on the parent core as they are being brought up in parallel, leading to duplication of effort as shown in Figure 9. The label d1mentioned in Figures 9 and 10 indicates the derivative core. The triages need to be able to map the parent code line version onto the derivative core to mark simulation failures. Derivative cores undergo a periodic (often daily) auto merge process. In order to pass the code line check, the triage helper maps the fixed revision in parent core bugs to the relevant code line and change list specified in a runtime argument by making a perforce call as shown in Figure 10.

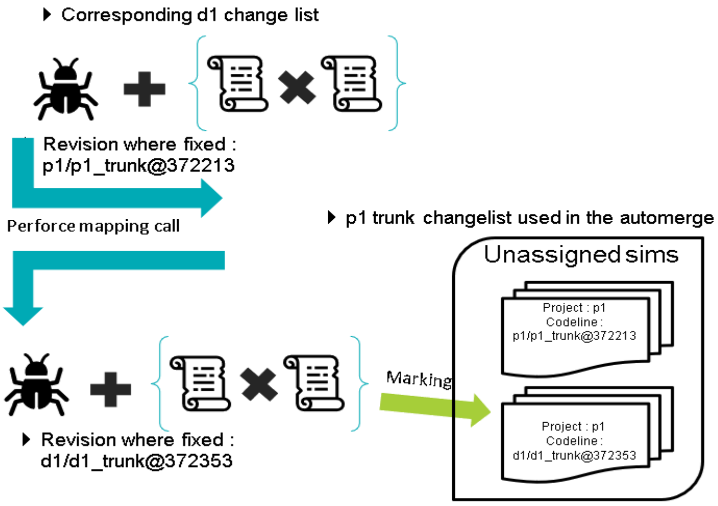


Figure 10. Proposed solution for derivative core triaging

4. RESULTS AND DISCUSSION

The data on number of triaged signatures were noted down for the span of two years and it is represented in Tables 1 and 2. Table 1 represents the number of signatures triaged before the check-in of automated triage process. Table 2 represents the number of signatures triaged after the check-in of automated triage process. The data is represented graphically as shown in Figure 11, before the check-in of automated triaging, the average percentage of signatures that ended up being triaged were around 40%. And once the triaging has been made automated, the average percentage of signatures that are being triaged has become 58%. From this, it can be said that the use of automated triage in the project of functional verification bug triage has contributed to a 18% increase in triaged signatures on average.

Table 1. Data indicating the number of triaged signatures before automation

Month/year	No. of triaged signatures in (%)
Mar/2020	35
Apr/2020	28
May/2020	39
Jun/2020	45
Jul/2020	65
Aug/2020	35
Sep/2020	44
Oct/2020	31
Nov/2020	30
Dec/2020	40
Jan/2021	33
Feb/2021	35

This methodology greatly eases the parsing problem, and the triaged inputs that are now parsed are currently being fed to a machine learning algorithm [21], which will help further improve the debug efficiency. As part of future work, the information from input YAML files can be used to analyze simulation failure attributes. A direct consequence of this is a reductionin duplicate debugs. The comprehensive automation of the triaging framework has helped save engineeringtime that would have otherwise been spent manually coding and porting the triages across projects. There are many disadvantages of approaches used to reduce the redundancy in the bug data, keyword extraction [14]. But eliminating the whole string series and consider only required bug information as done in [15] proved to be efficient. This in turn free sup time that can now be spent on debugging and therefore helps improve debugrates.

Table 2. Data indicating the number of triaged signatures after automation

Month/year	No. of triaged signatures in (%)	Month/year	No. of triaged signatures in (%)
Mar/2021	61	Nov/2021	53
Apr/2021	45	Dec/2021	57
May/2021	55	Jan/2022	64
Jun/2021	57	Feb/2022	59
Jul/2021	59	Mar/2022	52
Aug/2021	62	Apr/2022	67
Sep/2021	58	May/2022	65
Oct/2021	51	Jun/2022	63

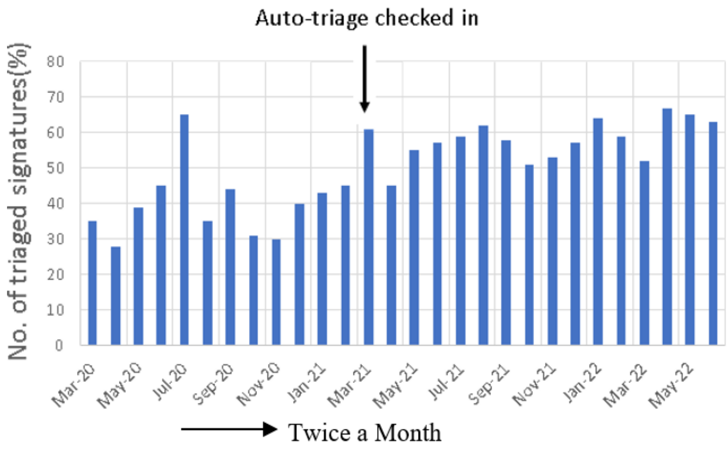


Figure 11. Week over week triaged signatures data

5. CONCLUSION

Triages now take a fraction of the time to write. As a result, problems associated with missing triages, such as duplicate debug effort, are avoided. The proposed methodology reduces duplicate effort in the form of redundancy, by eliminating cloning of bugs and creating multiple copies of a triage, for each tape out branches. The triage helper usage in the script has now extended the triaging mechanism beyond mainline core regressions and into side regressions, tape out branches, and derivative core regressions with no extra effort for the engineer. The automation of the triage writing process ensures standardization of format across projects, which makes the code readable and maintainable. Hence, an automated approach to improve functional verification bug triage has uplifted the efficiency by 18%. There will be a difficulty in comparing the results of the proposed framework with the related models because of the different mode of analysis and metrics used.

REFERENCES

[1] J. Xuan *et al.*, "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 264–280, Jan. 2015, doi: 10.1109/TKDE.2014.2324590.

[2] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, Apr. 2014, doi: 10.1109/TSE.2013.2297712.

[3] N. Sreenivas and S. J. Saritha, "Enhancement towards efficient bug triage with software data reducing methods," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, Aug. 2018, pp. 3649–3652, doi: 10.1109/ICECDS.2017.8390144.

[4] M. H. Kim, D. S. Wang, S. T. Wang, S. H. Park, and C. G. Lee, "Improving the robustness of the bug triage model through adversarial training," in *International Conference on Information Networking*, Jan. 2022, vol. 2022-Janua, pp. 478–481, doi: 10.1109/ICOIN53446.2022.9687279.

[5] A. Goyal, "Effective bug triage for non-reproducible bugs," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, May 2017, pp. 487–488, doi: 10.1109/ICSE-C.2017.41.

[6] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, Nov. 2011, pp. 253–262, doi: 10.1109/ASE.2011.6100061.

[7] V. Jain, A. Rath, and S. Ramaswamy, "Field weighting for automatic bug triaging systems," in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2012, pp. 2845–2848, doi: 10.1109/ICSMC.2012.6378180.

[8] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2190, Sep. 2013, doi: 10.4304/jsw.8.9.2185-2190.

[9] S. F. A. Zaidi and C. G. Lee, "Learning graph representation of bug reports to triage bugs using graph convolution network," in *International Conference on Information Networking*, Jan. 2021, vol. 2021-January, pp. 504–507, doi: 10.1109/ICOIN50884.2021.9333902.

[10] S. F. A. Zaidi, H. Woo, and C. G. Lee, "A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports," *IEEE Access*, vol. 10, pp. 20677–20689, 2022, doi: 10.1109/ACCESS.2022.3153075.

[11] S. F. A. Zaidi and C. G. Lee, "One-class classification based bug triage system to assign a newly added developer," in *International Conference on Information Networking*, Jan. 2021, vol. 2021-January, pp. 738–741, doi: 10.1109/ICOIN50884.2021.9334002.

[12] Z. Li and H. Zhong, "Revisiting textual feature of bug-triage approach," in *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, Nov. 2021, pp. 1183–1185, doi: 10.1109/ASE51524.2021.9678863.

[13] T. Zhang, G. Yang, B. Lee, and E. K. Lua, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Dec. 2014, vol. 1, pp. 223–230, doi: 10.1109/APSEC.2014.43.

[14] T. S. Gadge and N. Mangrulkar, "Approaches for automated bug triaging: a review," in *IEEE International Conference on Innovative Mechanisms for Industry Applications, ICIMIA 2017 - Proceedings*, Feb. 2017, pp. 158–161, doi: 10.1109/ICIMIA.2017.7975592.

[15] V. C. Georgopoulos and C. D. Stylios, "Fuzzy cognitive maps for decision making in triage of non-critical elderly patients," in *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Nov. 2017, vol. 2018-Janua, pp. 225–228, doi: 10.1109/ICIIBMS.2017.8279752.

[16] H. A. Chong and K. B. Gan, "Development of automated triage system for emergency medical service," in *2016 International Conference on Advances in Electrical, Electronic and Systems Engineering, ICAEES 2016*, Nov. 2017, pp. 642–645, doi: 10.1109/ICAEES.2016.7888125.

[17] V. Dedik and B. Rossi, "Automated bug triaging in an industrial context," in *Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016*, Aug. 2016, pp. 363–367, doi: 10.1109/SEAA.2016.20.

[18] A. Oleksiak, S. Cieslak, K. Marcinek, and W. A. Pleskacz, "Design and verification environment for RISC-V processor cores," in *Proceedings of the 26th International Conference "Mixed Design of Integrated Circuits and Systems", MIXDES 2019*, Jun. 2019, pp. 206–209, doi: 10.23919/MIXDES.2019.8787108.

[19] H. Yang and D. Ma, "The research on formal verification of CPU structure based on theorem proving," in *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, Oct. 2019, vol. 2019-October, pp. 139–143, doi: 10.1109/ICSESS47205.2019.9040731.

[20] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, Mar. 2017, doi: 10.1109/TSE.2016.2576454.

[21] D. Amran, M. Frid-Adar, N. Sagie, J. Nassar, A. Kabakovitch, and H. Greenspan, "Automated triage of covid-19 from various lung abnormalities using chest ct features," in *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, Apr. 2021, vol. 2021-April, pp. 155–159, doi: 10.1109/ISBI48211.2021.9433803.

[22] S. Chopade and P. More, "Effective bug triage with prim's algorithm for feature selection," in *Proceedings of IEEE International Conference on Signal Processing and Communication, ICSPC 2017*, Jul. 2018, vol. 2018-January, pp. 217–220, doi: 10.1109/CSPC.2017.8305842.

[23] N. A. Hidayat, P. Megantoro, A. Yurianta, A. Sofiah, S. A. Aldhama, and Y. A. Effendi, "The application of instrumentation system on a contactless robotic triage assistant to detect early transmission on a COVID-19 suspect," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 22, no. 3, pp. 1334–1344, Jun. 2021, doi: 10.11591/ijeecs.v22.i3.pp1334-1344.

[24] M. Mohagheghi and K. Salehi, "Improving graph-based methods for computing qualitative properties of markov decision processes," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 17, no. 3, pp. 1571–1577, 2020, doi: 10.11591/ijeecs.v18.i1.pp1571-1577.

[25] D. Yashas, P. S. H. Babu, and N. Shylashree, "UVM-based logic verification of input output interface," in *2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings*, May 2019, pp. 420–423, doi: 10.1109/RTEICT46194.2019.9016934.

BIOGRAPHIES OF AUTHORS

Sheetal Singrihalli Hemaraj received her B.E. degree in Electronics and Communication Engineering at Visvesvaraya Technological University, and her M.Tech. in VLSI design and embedded systems at RV College of Engineering, India in the year 2022. Her research lines are CPU core verification, memory in-computation. She can be contacted at email: sheeta+lsh62@gmail.com.

Shylashree Nagaraja is currently working as Associate Professor in the Department of Electronics and Communication Engineering at RV College of Engineering, Bengaluru. She is having 17 years of teaching experience. She was a recipient of the best Ph.D. thesis award for the year 2016-2017 in Electronics and Communication Engineering from BITES. She has received the best IEEE researcher award in IEEE-AGM meeting held during 2021 from Bangalore IEEE section. She has also received the best paper award in IEEE-ICERECT held during 2015 at Mandya. She has research publication in 40 International Journals (out of which 12 journals are SCI journals), 6 Springer book chapters and 10 International conferences. She received one US patent grant, two Indian patent grant in cryptography. She has also received two Indian patent grants in the area of VLSI. She is also the co-author of the network theory, engineering statistics and linear algebra and control engineering textbook. She has funded projects consultancy projects and has delivered many technical talks on VLSI. She has delivered lectures as a subject matter expert in VTU e-shikshana and EDUSAT program. She is a recipient of an international travel grant under SERB young research scholar category. She is a life member of ISTE, IETE, fellow member of ISVE, senior member of IEEE and IEEE CAS Secretary, Bangalore section. Her areas of interest include cryptography network security, network analysis, analysis and design of digital circuits, digital VLSI design, analog mixed mode VLSI design, low power VLSI design, statistics and linear algebra and control engineering. She can be contacted at email: shylashreen@rvce.edu.in.

Sunitha Yariyur Narasimhaiah is currently working as Assistant Professor in the Department of Electronics and Communication Engineering at SJB Institute of Technology, Bengaluru. She is having 16 years of teaching experience. She has completed her M.Tech and Ph.D. from VTU, Belagavi in the year 2003 and 2020 repectivley. Her areas of interest are Machine learning algorithm implemented for image fusion applications. She can be contacted at email: sunithayn@sjbit.edu.in.

Madhu Patil received her Ph.D. degree in Communication from University of Mysore. She is currently working as Professor in the department of computer science and Engineering in BGS College of Engineering and Technology, Bengaluru, India. She has 19 years of teaching experience. Her main research interest is in areas of communication and signal processing. She can be contacted at email: p25.madhu@gmail.com.