Reflecting on the use of the Policy-Process-Product Theory in Empirical Software Engineering

Kelechi G. Kalu
Purdue University
West Lafayette, Indiana, USA
kalu@purdue.edu

Taylor R. Schorlemmer
Purdue University
West Lafayette, Indiana, USA
kalu@purdue.edu

Sophie Chen
Michigan
Michigan, USA
kalu@purdue.edu

Kyle Robinson
Purdue University
West Lafayette, Indiana, USA
kalu@purdue.edu

Erik Kocinare
Purdue University
West Lafayette, Indiana, USA
kalu@purdue.edu

James C. Davis
Purdue University
West Lafayette, Indiana, USA
davisjam@purdue.edu

ABSTRACT

The primary theory of software engineering is that an organization's Policies and Processes influence the quality of its Products. We call this the *PPP Theory*. Although empirical software engineering research has grown common, it is unclear whether researchers are trying to evaluate the PPP Theory. To assess this, we analyzed one-third (22) of the empirical works published over the last two years in three software engineering venues . Of the observed papers, 77% focus solely on either processes/policies or products. Furthermore, 54% of observed empirical papers remain silent about the relationships present between processes/policies and products. We therefore recommend that future empirical software engineering research consider the broader context of their work in terms of the PPP Theory. Research results are in context of and with respect to the relationship between software products, processes, and policies.

ACM Reference Format:
Kelechi G. Kalu, Taylor R. Schorlemmer, Sophie Chen, Kyle Robinson, Erik Kocinare, and James C. Davis. 2023. Reflecting on the use of the Policy-Process-Product Theory in Empirical Software Engineering. In *Proceedings of The 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Empirical software engineering research analyzes data to improve software products and engineering processes [1, 2]. International standards organizations [3], industry consortia [4], and professional organizations [5] all assert that the Policies and Processes of software engineering influence the quality of the software Product (the *PPP Theory*). Various studies support some of the relationships predicted by the PPP Theory [6, 7]. Nevertheless, it remains unclear which policies and processes are most effective in achieving high-quality products, and how these vary by context [8]. To address this, experts have recommended that empirical software engineering researchers incorporate the PPP Theory, either as contextual information in case studies or as part of a controlled experiment [9, 10]. Doing so could resolve widely remarked-upon challenges with the generalizability and replicability of empirical software engineering research results [9, 11–16]. However, the extent to which the research community has taken this advice is unclear.

This reflection paper examines whether empirical software engineering researchers are considering the relationship between policies, processes, and software products. To achieve this, we reviewed empirical software research works published in 3 software engineering venues (ICSE, ASE, and ESEC/FSE) in 2021 and 2022. We identified the primary aspects of the PPP Theory considered by each work, and the extent to which the PPP Theory was incorporated into the work. We report that empirical studies consider a subset of the PPP Theory and are usually focused on individual theoretical concepts rather than the relationships of the theory. We challenge the Empirical Software Engineering research community to consciously consider engineering theory in their study designs.

ESEC/FSE 2023, 11 - 17 November 2023, San Francisco, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND: THE PPP THEORY

We define the key constructs, then the PPP Theory relating them.

2.1 Theoretical Constructs

Policy: Policy has many meanings, including processes, artifacts, discourses, and bodies of knowledge about a field [17–19]. Definitions from the software engineering literature include both organizational strategies [20–22], and technical system behaviors [23–25]. In the context of the PPP Theory, we define **policy** as *an official statement for an organization's software engineering practices, derived from the organization's goals.*

Process: A process consists of steps to be followed by users, system operations personnel, or others to accomplish a particular task (e.g., preparing new user accounts and assigning the appropriate privileges) [6, 20]. In the context of the PPP Theory, we define **process** as *the methods used by software engineers to accomplish their tasks.* Processes may be informed by policy (top-down) or *ad hoc* (bottom-up).

Since *process* and *policy* typically have overlapping definitions, we lump them together into a single **process/policy** term as shown in Figure 1.

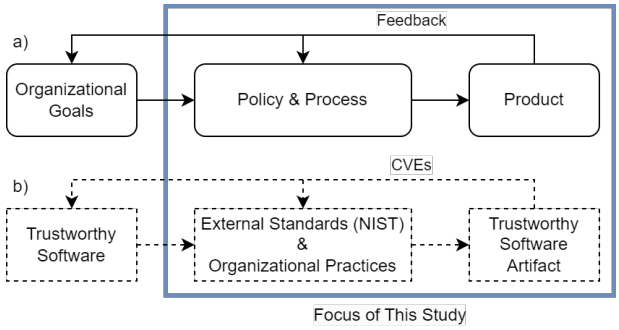

Figure 1: a) Policy-Process-Product (PPP) Relationship Model. Organizational goals influence the policies and processes adopted by software engineers. In-turn, policies and process influence the creation of a product. Feedback from a product can modify policies and processes or the original goals themselves. We have illustrated the often overlapped entities of Policy and Process as a single entity in our model. The constructs and relationships assessed in this study are boxed in blue. b) Example of the PPP model illustrated using a theoretical software engineering goal of producing trustworthy software.

Product: A software product is a set of software and associated documentation, designed and developed to meet a specific set of user needs [6, 26, 27]. In the context of the PPP Theory, we define a **product** as *the artifacts produced by a software engineering process.* What comprises a product is context-dependent; some teams produce libraries, others web services, others mobile applications, and so on. There are many measures of software product quality, including the cost to produce and maintain it and the risks in using it (defect likelihood and severity [28]).

2.2 Policy-Process-Product Relationship

Figure 1 shows the PPP Theory: these constructs and the relationships between them. Organizational goals are iteratively refined into policies, processes, and finally products. This theory is propounded by documents from international standards organizations [3], industry consortia [4], professional organizations [5], governments [20], and the academic literature [6, 21–23, 29, 30].

The PPP Theory predicts bi-directional relationships between the three constructs. A software team's policy informs how its processes are defined, and a team's process influences the quality of the product. In the reverse direction, retrospectives and postmortems provide feedback to modify processes and policies.

An example of the PPP Theory is demonstrated in Figure 1(b). An organization has the goal of securing their artifact's supply chain [31]. Organizational leaders create a policy: "Follow the NIST standards" [32]. An engineering team complies with this policy through several process elements, such as code review (for code vulnerability inspection) and the use of provenance certification tools (*e.g.,* Sigstore [33]). The desired product quality, a secure supply chain, is assessed: defects (*e.g.,* CVEs) provide feedback to improve the process.

3 QUESTION AND METHODS

We ask: *To what extent does the PPP Theory inform modern empirical software engineering research?* To answer this question, we assessed 22 papers from top software engineering research venues. This section describes the selection of those papers, the initial assessment approach used in our pilot study, and our revised assessment approach. Our final methodology is summarized in Figure 2.

3.1 Paper selection

We collected recent (2021-2022) empirical software engineering papers published at three top-tier venues (ICSE, ASE, and ESEC/FSE). We obtained full-length papers included in the conference proceedings (Research/Technical track). There were 65 papers that included "empirical" in the title or keywords. We selected a random sample of 22 papers to analyze.

3.2 Analysis process

Our goal was to assess the presence of PPP relationships in our selected papers. We iteratively refined an analysis instrument through a pilot study. Ultimately, we assessed two distinct aspects of each work: its *construct focus* and its *relationship prevalence.*

3.2.1 Pilot study. In our pilot study, we established a complex classification scheme to rate the appearance of PPP Theory in empirical research papers. This includes a set of rating metrics and a method for scoring each paper on those metrics.

Metrics: Our initial metrics attempted to measure the occurrence of process-product and policy-product relationships in each paper. Each of these section-relationship combinations was evaluated on a four-point scale: (1) *Silent* (no mention of relationships), (2) *Implicit* (acknowledges relationship without discussion of impact), (3) *Descriptive* (describes extensively the relationship between process/policy/product), and (4) *Experimental* (describes and controls for these relationships in their experiment).

Analysis Process: In our initial approach, raters focused on the Methodology, Results/Discussion, and Threats to Validity sections because we found these are where the use of PPP Theory tends to be documented. After reading through a paper, raters classified the section-relationship combinations according to our four-point scale.

Refinements: We used this approach on 13 papers in our pilot study (20% of the available data). We identified two flaws. First, the raters struggled to differentiate between levels of our four-point scale, and we found that the collected data was too detailed for interpretation (6 four-point observations per paper). This also made inter-rater disagreements frequent and hard to resolve. Second, the paper sections targeted in our analysis were too specific — some papers mention theoretical constructs and relationships in their Introduction, others in the Background, and so on.

3.2.2 Final Analysis Approach. Here is the final analysis approach we used. First, we clarified definitions to make categories easier to differentiate. Second, we characterized each paper holistically rather than considering each of its sections in turn. Lastly, given the relatively rare use of PPP Theory relationships in the pilot papers, we reduced the scope of the measurement to simply reporting

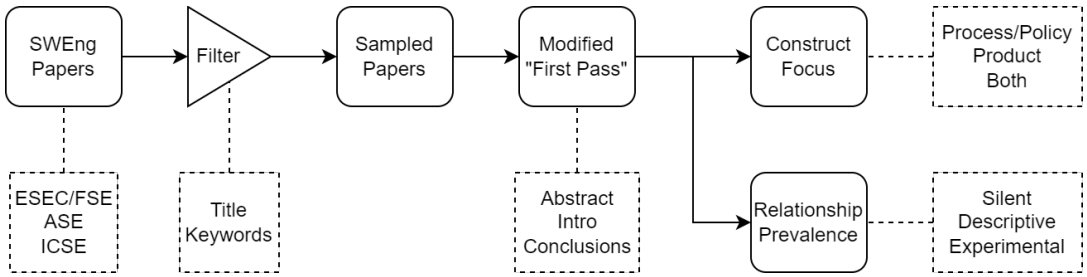

Figure 2: Our refined method for analyzing papers. First, we search for full length papers from top software engineering conferences and filter those papers by their title and keywords. We are then left with empirical studies. After randomly sampling from these empirical studies, we then conduct a "First Pass," similar to the methods proposed by Keshav [34], to gain an understanding of the paper content. Finally, we determine which PPP Theory construct(s) the paper focuses on and assess if PPP Theory relationships are present in the paper.

whether process/product relationships were considered at all. Figure 2 provides an overview of the final analysis approach we used to assess each paper.

Metrics: We simplified our measurement of PPP Theory in each paper down to two metrics.

(1) *Construct Focus*: Which PPP Theory construct(s) did the paper focus on?
(2) *Relationship Prevalence:* Did the paper identify relationships between PPP Theory constructs?

These metrics allow us to categorize what a study is *about* and whether it *considers* PPP Theory.

For the construct focus metric, we categorize each paper into one of the following three types:

(1) *Process/Policy:* The paper focuses on measuring or observing process/policy. For example, He *et al.* measures the library migration process in the Java ecosystem [35].
(2) *Product:* The paper focuses on measuring or observing a product. For example, Shen *et al.* study root causes and symptoms of deep learning compiler bugs [36].
(3) *Both:* The paper considers both. For example, Grazia *et al.* measure the adoption and use of Python type annotations (process) *and* the resulting statically-detectable type errors in Python projects (product) [37].

For the relationship prevalence metric, we categorize each paper into one of the following three types:

(1) *Silent:* The paper makes no mention of PPP Theory relationships. For example, Shen *et al.* report the characteristics of deep learning compiler bugs, but do not explicitly describe how software engineering processes or policies can cause these bugs or should be influenced by bugs.
(2) *Descriptive:* The paper *mentions* a relationship between PPP Theory constructs. For example, He *et al.* measure the library migration process and describe the importance of this process with respect to Java software products, but do not directly measure this relationship.
(3) *Experimental:* The paper *measures* a relationship between PPP Theory constructs. For example, Grazia *et al.* measures the relationship between using type annotations and the resulting number of type errors.

Analysis Process: Our raters followed a modified version of Keshav's "First Pass" to quickly assess the PPP-Theoretic content of selected papers [34]. Raters proceeded as follows:

- Read title, abstract, introduction, and research questions.
- Read section headings.
- Read the findings — this includes highlighted key results, the discussion, and the Conclusion section.

After performing this "First Pass," raters categorized the construct focus and relationship prevalence metrics for the paper according to the descriptions mentioned above.

Each paper was evaluated by two raters. Inter-rater agreement was measured using Cohen's Kappa score [38], and all disagreements were settled through discussion. Prior to settling disagreements, our process produced a Kappa score of 0.848 for the construct classifications and 0.684 for rating the PPP relationships in each work.

4 RESULTS

In this section, we identify our results from assessing 22 empirical software engineering papers. Specifically, we report how many papers focus on each PPP Theory construct, and to what degree each paper recognizes PPP Theory relationships.

4.1 Construct Focus

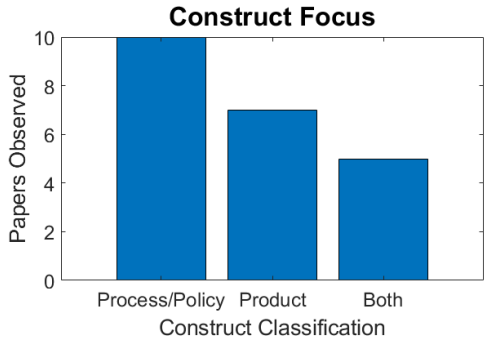

Figure 3: Distribution of empirical software engineering papers that focus on process/policy, products, and both.

We find that most observed papers (45%) focus on the policy/processes. These papers observe the actions taken by software engineers and organizations.

The second most common category of paper (32%) focuses on products. These types of papers tend to focus on measuring information about software artifacts.

The smallest category (23%) measured both policy/process and products. These works provide empirical results on software artifacts and processes/policies employed by software engineers.

Figure 3 shows the distribution of papers that focus on each PPP Theory construct. The majority of the empirical works studied are focused on a single construct (77%).

4.2 Relationship Prevalence

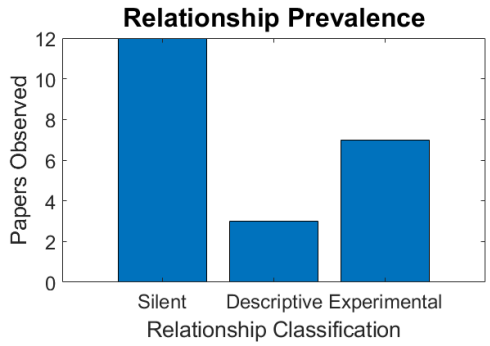

Figure 4: Distribution of papers in each of the three relationship prevalence categories.

Figure 4 demonstrates how prevalent PPP Theory relationships are in each paper. We observe:

- All works that provide a description of the PPP relationship focused on the Process construct. This suggests that works under this construct acknowledge the existence of certain relationships but choose not to measure them for some reason. Works focused on the Product construct did not do so.
- Works that experimentally measured a relationship between the PPP relationship they identified, included all (but one) works that fall under the "Both" construct category (Figure 3).

5 IMPLICATIONS FOR RESEARCH

We suggest three implications for the Empirical Software Engineering research community.

Incorporate PPP Theory into study designs: A surprising fraction of papers (54%) did not consider these PPP relationships. We do not wish to criticize these works; there is value in characterizing processes and in characterizing products, whether or not relationships are demonstrated between these constructs. However, we wonder if the Empirical Software Engineering research community would benefit from a greater focus on the PPP-Theoretic basis for their measurements. This was the original vision of empirical software engineering introduced in the 1980s and 1990s [10, 39, 40]. This could provide a meaningful way to address concerns about the interpretability and generalizability of our community's empirical research [9, 13, 15, 16]. As a step towards this, the SIGSOFT Empirical Standards [41] could be extended to provide guidance about epistemology (*What is software engineering knowledge?* [42]), not just about methodology (*How to obtain knowledge?*). At the least, some thoughtfulness about the PPP Theory would guide authors in analyzing the Threats to Validity of their work, without resorting to vague statements about generalizability in "other contexts".

Study the feedback relationship: Among the papers that did consider a relationship between policy/process and product, we note that there was a bias toward measuring the "forward" direction of Figure 1. In our sample it is unusual for researchers to characterize and measure the role of feedback in the engineering process. Although many works have observed the opportunity for failures to inform future engineering approaches [43, 44], this appears to still be a gap in the literature.

Study more feed-forward relationships: Lastly, although the "forward" direction of relationships was more commonly examined, there are large classes of societally-relevant constructs whose relationships were not examined in our sample. Papers in this category tended to consider topics like software organizational structure, software evolution, and software maintenance. We did not see any papers that examine topics such as the effect of cybersecurity policies, regulations (*e.g.,* GDPR), or industry standards (*e.g.,* MISRA). Greater industry collaboration might facilitate the study of such relationships. Empirical software engineering research often examines open-source software — those engineers lack the liability that motivates organizations to promote such policies and processes.

Disentangle Policy and Process: The PPP Theory predicts separate roles of policy and process. Policy interfaces with organizational goals, while process interfaces with the engineered product. These constructs are generally entangled in the empirical software engineering literature, so in our model, we combined them Figure 2. Considering them as separate constructs may help the community develop a richer theory of software engineering.

6 THREATS TO VALIDITY

Construct: We rely on constructs and relationships defined by the PPP Theory (§1)). Our specific operationalizations of these was derived from literature (§2), but we acknowledge that discerning these can be difficult because they are often entangled. We used interrater agreement to mitigate this concern. We reached a relatively high Kappa score for construct classification (0.848) and an acceptable level for the PPP relationships between constructs (0.684).

Internal: We make no claims of cause and effect.

External: We focused on "modern empirical software engineering research", with a sample of one-third of works from the past two years of ICSE, ESEC/FSE, and ASE. A longer timespan or additional venues might affect our results. Based on our understanding of the recent research literature, we do not think time is a crucial variable. These three venues are large general venues, so considering other venues seems unlikely to substantially shift our result.

7 CONCLUSION

In this paper, we have investigated the degree to which current empirical software engineering works consider the relationship

between policies, processes, and software products (PPP relationship). We have reviewed 22 published works. Our results show that: (1) *most empirical software engineering works are focussed on single constructs of the PPP theory model*, and (2) *32% of the reviewed works provided a description for the PPP relationships, while 54% of the works were silent on this PPP relationship.*

Consequent to our results, we have made 4 suggestions to the software engineering research community on the significance of adopting the PPP theory in future empirical studies.

DATA AVAILABILITY

Our analysis is available in an anonymous spreadsheet: https://tinyurl.com/3zwd8vnx.

ACKNOWLEDGMENTS

Kazerouni

NSF regex award

REFERENCES

[1] V. R. Basili, "The Role of Empirical Study in Software Engineering," in *2006 30th Annual IEEE/NASA Software Engineering Workshop*. Columbia, MD, USA: IEEE, Apr. 2006, pp. 3–6. [Online]. Available: https://ieeexplore.ieee.org/document/4090238/

[2] S. Xu, "Empirical research methods for software engineering: Keynote address," in *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, Jun. 2017, pp. 1–1.

[3] "Quality management systems – requirements," International Organization for Standardization, September 2015, iSO Standard 9001. [Online]. Available: https://www.iso.org/standard/62085.html

[4] "Misra c:2012 guidelines for the use of the c language in critical systems," MISRA, March 2012, mISRA C:2012.

[5] "Ieee standard for software quality assurance processes," IEEE Standards Association, June 2014, iEEE Std 730-2014. [Online]. Available: https://ieeexplore.ieee.org/document/6838485

[6] I. Sommerville, *Software engineering*, 9th ed. Boston: Pearson, 2011, oCLC: ocn462909026.

[7] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: how developers see it," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 1028–1038. [Online]. Available: https://dl.acm.org/doi/10.1145/2884781.2884840

[8] C. Hobbs, "Software development standards," in *Embedded Software Development for Safety-Critical Systems*, 2nd ed., C. Hobbs, Ed. Elsevier, 2016, ch. 3, pp. 65–91. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780081005995000037

[9] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, Aug. 2002, conference Name: IEEE Transactions on Software Engineering.

[10] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 04, pp. 456–473, Jul. 1999, publisher: IEEE Computer Society. [Online]. Available: https://www.computer.org/csdl/journal/ts/1999/04/e0456/13rRUx0xPVD

[11] B. Dit, E. Moritz, M. Linares-Vasquez, and D. Poshyvanyk, "Supporting and Accelerating Reproducible Research in Software Maintenance Using TraceLab Component Library," in *2013 IEEE International Conference on Software Maintenance*. Eindhoven, Netherlands: IEEE, Sep. 2013, pp. 330–339. [Online]. Available: http://ieeexplore.ieee.org/document/6676904/

[12] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. Saint Petersburg Russia: ACM, Aug. 2013, pp. 466–476. [Online]. Available: https://dl.acm.org/doi/10.1145/2491411.2491415

[13] L. Madeyski and B. Kitchenham, "Would wider adoption of reproducible research be beneficial for empirical software engineering research?" *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 2, pp. 1509–1521, Jan. 2017. [Online]. Available: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/JIFS-169146

[14] E. Murphy-Hill, G. C. Murphy, and W. G. Griswold, "Understanding context: creating a lasting impact in experimental software engineering research," in *FSE/SDP workshop on Future of SWEng research*, ser. FoSER '10. New York, NY, USA: Association for Computing Machinery, Nov. 2010, pp. 255–258. [Online]. Available: https://dl.acm.org/doi/10.1145/1882362.1882415

[15] J. Siegmund, N. Siegmund, and S. Apel, "Views on Internal and External Validity in Empirical Software Engineering," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, May 2015, pp. 9–19, iSSN: 1558-1225.

[16] J. M. González-Barahona and G. Robles, "On the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 75–89, Feb. 2012. [Online]. Available: http://link.springer.com/10.1007/s10664-011-9181-9

[17] H. Colebatch, "Introduction to the Handbook on Policy, Process and Governing," in *Handbook on Policy, Process and Governing*. Edward Elgar Publishing, 2018, pp. 1–13. [Online]. Available: https://www.elgaronline.com/view/edcoll/9781784714864/9781784714864.00005.xml

[18] S. J. Ball, "WHAT IS POLICY? TEXTS, TRAJECTORIES AND TOOLBOXES," *Discourse: Studies in the Cultural Politics of Education*, vol. 13, no. 2, pp. 10–17, Apr. 1993. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/0159630930130203

[19] ——, "What is policy? 21 years later: reflections on the possibilities of policy research," *Discourse: Studies in the Cultural Politics of Education*, vol. 36, no. 3, pp. 306–313, May 2015. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/01596306.2015.1015279

[20] "NIST SP 800-12: Chapter 5 - Computer Security Policy." [Online]. Available: https://csrc.nist.rip/publications/nistpubs/800-12/800-12-html/chapter5.html

[21] R. Wies, "Using a Classification of Management Policies for Policy Specification and Policy Transformation," in *Integrated Network Management IV*, A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Boston, MA: Springer US, 1995, pp. 44–56. [Online]. Available: http://link.springer.com/10.1007/978-0-387-34890-2_4

[22] Kafali, J. Jones, M. Petruso, L. Williams, and M. P. Singh, "How Good Is a Security Policy against Real Breaches? A HIPAA Case Study," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 530–540, iSSN: 1558-1225.

[23] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, no. 4, pp. 333–360, Dec. 1994. [Online]. Available: http://link.springer.com/10.1007/BF02283186

[24] P. Naldurg and R. H. Campbell, "Modeling insecurity: policy engineering for survivability," in *ACM workshop on Survivable and self-regenerative systems*, ser. SSRS '03. New York, NY, USA: Association for Computing Machinery, Oct. 2003, pp. 91–98. [Online]. Available: https://doi.org/10.1145/1036921.1036931

[25] N. Dulay, E. Lupu, M. Sloman, and N. Damianou, *A policy deployment model for the Ponder language*, ser. IEEE/IFIP International Symposium on Integrated Network Management (IM'2001). Seattle: IEEE Press., Feb. 2001, pages: 543.

[26] R. S. Pressman, *Software engineering: a practitioner's approach*. McGraw-Hill Education, 2014.

[27] ISO/IEC, *Systems and software engineering–Software life cycle processes*, Std., 2017. [Online]. Available: https://www.iso.org/standard/63711.html

[28] G. Fairbanks, *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd, 2010.

[29] D. Anandayuvaraj and J. C. Davis, "Reflecting on Recurring Failures in IoT Development," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 1–5. [Online]. Available: https://dl.acm.org/doi/10.1145/3551349.3559545

[30] P. C. Amusuo, A. Sharma, S. R. Rao, A. Vincent, and J. C. Davis, "Reflections on software failure analysis," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 1615–1620. [Online]. Available: https://dl.acm.org/doi/10.1145/3540250.3560879

[31] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "Sok: Analysis of software supply chain security by establishing secure design properties," in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022, pp. 15–24.

[32] NIST, "Software security and supply chains guidance," *National Institute of Standards and Technology*, 2021. [Online]. Available: https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-guidance

[33] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: software signing for everybody," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2353–2367.

[34] S. Keshav, "How to read a paper," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, p. 83–84, jul 2007. [Online]. Available: https://doi.org/10.1145/1273445.1273458

[35] H. He, R. He, H. Gu, and M. Zhou, "A large-scale empirical study on Java library migrations: prevalence, trends, and rationales," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 478–490. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468571

[36] Q. Shen, H. Ma, J. Chen, Y. Tian, S.-C. Cheung, and X. Chen, "A comprehensive study of deep learning compiler bugs," in *ESEC/FSE*, ser. ESEC/FSE 2021. New

York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 968–980. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468591

[37] L. Di Grazia and M. Pradel, "The evolution of type annotations in python: an empirical study," in *ESEC/FSE*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 209–220. [Online]. Available: https://dl.acm.org/doi/10.1145/3540250.3549114

[38] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[39] S. S. Brilliant, J. C. Knight, and N. G. Leveson, "Analysis of faults in an n-version software experiment," *IEEE Transactions on software engineering*, vol. 16, no. 2, pp. 238–247, 1990.

[40] A. J. Perlis, F. Sayward, and M. Shaw, *Software metrics: an analysis and evaluation*. Mit Press, 1981, vol. 5.

[41] ACM SIGSOFT, "Acm sigsoft empirical standards," https://github.com/acmsigsoft/EmpiricalStandards, 2021, accessed: May 3, 2023.

[42] IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Press, 2014.

[43] D. Anandayuvaraj and J. C. Davis, "Reflecting on recurring failures in iot development," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[44] P. C. Amusuo, A. Sharma, S. R. Rao, A. Vincent, and J. C. Davis, "Reflections on software failure analysis," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1615–1620.