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

Kelechi G. Kalu
Purdue University, IN, USA
kalu@purdue.edu

Taylor R. Schorlemmer
Purdue University, IN, USA
tschorle@purdue.edu

Sophie Chen
University of Michigan, MI, USA
sophie.cy.chen@gmail.com

Kyle A. Robinson
Purdue University, IN, USA
robin489@purdue.edu

Erik Kocinare
Purdue University, IN, USA
ekocinar@purdue.edu

James C. Davis
Purdue University, IN, 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 half (33) of the empirical works published over the last two years in three prominent software engineering conferences. In this sample, 70% focus on policies/processes or products, not both. Only 33% provided measurements relating policy/process and products. We make four recommendations: (1) Use PPP Theory in study design; (2) Study feedback relationships; (3) Diversify the studied feedforward relationships; and (4) Disentangle policy and process. Let us remember that research results are in the context of, and with respect to, the relationship between software products, processes, and policies.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; • **Software and its engineering**;

KEYWORDS

Empirical Software Engineering, Software Process and Policy

ACM Reference Format:
Kelechi G. Kalu, Taylor R. Schorlemmer, Sophie Chen, Kyle A. 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 '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3611643.3613075

1 INTRODUCTION

Empirical software engineering research analyzes data to improve software products and engineering processes [5, 45]. International standards organizations [20], industry consortia [26], and professional organizations [19] 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 [34, 39]. Nevertheless, it remains unclear which policies and processes are most effective in achieving high-quality products, and how these vary by context [17]. 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 [6, 24]. This could address concerns regarding the generalizability and replicability of empirical software engineering research [13, 15, 24, 25, 27, 28, 34, 37]. 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, ESEC/FSE, and ASE) 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

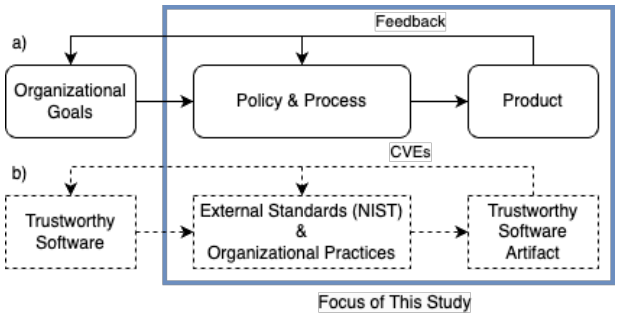

Figure 1: (a) Policy-Process-Product (PPP) Theory. Organizational goals influence the policies and processes adopted by software engineers. Policies and process influence product development. Feedback may modify policies, processes, or the original goals. We treat the (often overlapped) concepts of Policy and Process as a single entity. (b) Example of the PPP Theory for the goal of producing trustworthy software.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3613075

challenge the Empirical Software Engineering research community to consciously consider the PPP Theory in their study designs.

2 BACKGROUND: THE PPP THEORY

2.1 Theoretical Constructs

Policy: Policy has many meanings, including processes, artifacts, discourses, and bodies of knowledge about a field [4, 10, 11]. In the software engineering literature, policy means both organizational strategies [22, 31, 41], and technical system behaviors [14, 29, 38]. For PPP Theory, we define **policy** as *an official statement of an organization's software engineering practices, derived from the organization's goals.*

Process: A process consists of the steps followed to accomplish a task, *e.g.,* performing code review or implementing a new feature [31, 39]. For PPP Theory, we define **process** as *the methods used by software engineers to accomplish their tasks.*

In the software engineering literature, we found that *process* and *policy* typically have overlapping definitions. We lump them together into a single **process/policy** construct as shown in Figure 1.

Product: A software product is a set of software and associated documentation, designed and developed to meet a specific set of user needs [21, 35, 39]. For 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.

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 [20], industry consortia [26], professional organizations [19], governments [31], and the academic literature [2, 3, 22, 34, 38, 39, 41].

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 its artifact's supply chain [32]. Organizational leaders create a policy: "Follow NIST standards" [40]. Engineering teams comply through several process elements, such as code review (for code vulnerability inspection) and using provenance certification tools (*e.g.,* Sigstore [30]). The desired product quality, a secure supply chain, is assessed: defects (*e.g.,* CVEs) provide feedback to improve the process.

Some seminal works explore the relationships between the PPP theory constructs [18, 42–44]. For example, Humphrey *et al.* [18] and Wohlin *et al.* [42] demonstrated the impact of the Personal Software Process (PSP) on the software product (forward direction). In a follow-up study, Wohlin *et al.* showed software defects can be utilized in the Feedback direction to improve the PSP [44].

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 33 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 gathered recent empirical software engineering papers (2021-2022) from all tracks of three prominent conferences (ICSE, ESEC/FSE, and ASE). We retrieved full-length papers, totaling 65, that included the term "empirical" in their title or keywords. Initially, we used the DBLP database for the title match and later cross-verified our findings and expanded our search using the ACM digital library, considering both the title and author's keywords. For analysis, we randomly selected 50% of the collected papers.

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 — we thought any use of PPP Theory would be documented here. 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. (1) Raters struggled to differentiate between levels of our four-point scale. Inter-rater disagreements were common and hard to resolve. (2) Some PPP Theory elements were missed because the paper sections targeted in our analysis were too specific.

3.2.2 Final Analysis Approach. First, we clarified definitions to make categories easier to differentiate. Second, we characterized papers holistically rather than considering individual sections. Lastly, given the relatively rare use of PPP Theory relationships in the pilot papers, we reduced the scope of the measurement to simply reporting whether process/product relationships were considered at all, or actively controlled for in the papers. Our final analysis approach is summarized in Figure 2.

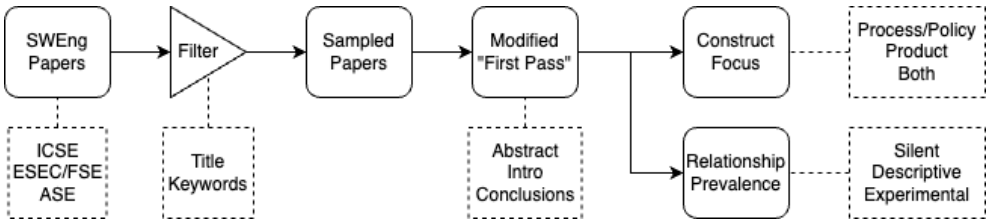

Figure 2: Our streamlined paper analysis approach. We begin by searching for full-length papers in leading software engineering conferences, filtering based on titles and keywords, resulting in empirical studies. We randomly sample half of these studies. This is followed by a "First Pass" analysis akin to Keshav [23] to comprehend paper content. We identify the paper's PPP Theory construct(s) focus and assess the presence of PPP Theory relationships.

Metrics: We assessed the use of PPP Theory with 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 (item 1 above), we categorize each paper into one of the following three types:

(1) *Process/Policy:* The paper observes or measures process/policy. For example, He *et al.* measures the library migration process in the Java ecosystem [16].
(2) *Product:* The paper observes or measures 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, Di Grazia *et al.* measure the adoption and use of Python type annotations (process) *and* the resulting statically-detectable type errors in Python projects (product) [12].

For the relationship prevalence metric (item 2 above), 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.* [36] 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.* [16] 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, Di Grazia *et al.* [12] measures the relationship between using type annotations and the resulting number of type errors.

Also, we categorized each paper based on the ownership of the empirical data employed in the study: *Public* (papers involving publicly accessible data), *Private* (data not accessible/proprietary), and *Both* (papers incorporating both private and public data).

Analysis Process: Our raters followed a modified version of Keshav's "First Pass" to quickly assess the PPP-Theoretic content of selected papers [23]. 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.

Two raters evaluated each paper. Inter-rater agreement was assessed using Cohen's Kappa [9], yielding scores of 0.86, 0.67, and 0.88 for construct classifications, PPP relationships, and data accessibility, respectively. Disagreements were settled through discussion.

4 RESULTS

In this section, we identify our results from assessing 33 empirical software engineering papers. Figure 3 summarizes our findings.

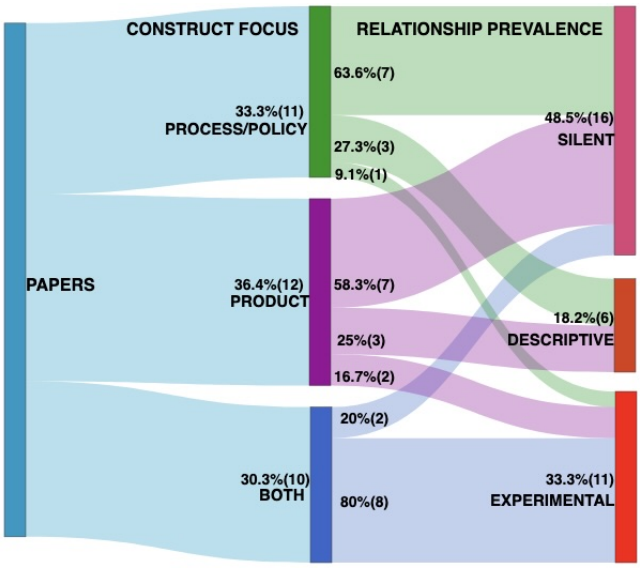

Figure 3: Distribution of analyzed papers based on Construct Focus (process/policy, products, both) and Relationship Prevalence (silent, descriptive, experimental). 30% of papers consider multiple constructs and 49% of papers are silent about relationships between them.

4.1 Construct Focus

The first stage of Figure 3 divides papers by their Construct focus. Papers typically consider products (37%) or policy/processes (33%). The former observes the actions of software engineers and organizations, and the latter measures information about software artifacts. **The smallest set considered both policy/process and products** (30%).

4.2 Relationship Prevalence

The second stage of Figure 3 divides papers by their relationship prevalence. **Work experimenting with or measuring PPP Theory relationships was rare**: 33% or 11 of the papers. About half of the papers (16 papers or 49%) did not discuss relationships from the PPP Theory, and the remaining 6 papers (18%) were descriptive. As expected, considering the construct focus of a paper, the studies focused on a single construct tend to be silent or descriptive, and studies that consider both constructs tend to relate them.

4.3 Other Observations

We considered the ownership of the data used in these works. Most papers used public data (64%), some private (24%), rarely both (12%). We observed one trend: of the four studies that used both data types, three experimentally showed a PPP relationship. Perhaps studies with diverse data provide more insight into PPP relationships.

5 IMPLICATIONS FOR RESEARCH

We suggest four implications for the research community.

(1) Incorporate PPP Theory into Study Designs: A surprising fraction of papers (49%) 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 [6, 8, 33]. This could provide a meaningful way to address concerns about the interpretability and generalizability of our community's empirical research [15, 24, 25, 37]. As a step towards this, the SIGSOFT Empirical Standards [1] could be extended to provide guidance about epistemology (*What is software engineering knowledge?* [7]), not just about methodology (*How to obtain knowledge?*). Some thoughtfulness about the PPP Theory could help authors analyze the Threats to the Validity of their work, without resorting to vague statements about generalizability in "other contexts".

(2) 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 [2, 3], this appears to still be a gap in the literature.

(3) Study More Feed-Forward Relationships: Although the "forward" direction of relationships was more commonly examined, there are classes of constructs whose relationships were not examined in our sample. Papers in this category considered topics like software organizational structure, software evolution, and software maintenance. We did not see any papers on topics such as the effect of regulations (*e.g.,* GDPR), cybersecurity policies (*e.g.,* NIST 8397), 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.

(4) Disentangle Policy and Process: Lastly, 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 in 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. Our specific operationalizations were derived from literature (§2), but distinguishing these can be difficult because they are often entangled. We addressed this concern through inter-rater agreement, achieving reasonable Kappa scores of 0.86 for PPP constructs and 0.67 for relationships between constructs.

Internal: We make no claims of cause and effect.

External: We sampled half of the empirical works from ICSE, ESEC/FSE, and ASE 2021-2022. A longer time span or alternative 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 33 published works. Our results show that: (1) Most empirical software engineering works are focussed on single constructs of the PPP theory model, and (2) 18% of the reviewed works provided a description for the PPP relationships, while 49% 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. Our recommendations center on the need to further study the PPP theory constructs, and their forward and feedback relations.

DATA AVAILABILITY

Data is available on Zenodo: https://doi.org/10.5281/zenodo.8277429.

ACKNOWLEDGMENTS

We thank A. Kazerouni and the reviewers for their feedback. We acknowledge financial support from NSF awards IIS-1813935, SaTC-2135156, and POSE-2229703, as well as Cisco and Rolls Royce.

REFERENCES

[1] ACM SIGSOFT. 2021. ACM SIGSOFT Empirical Standards. https://github.com/acmsigsoft/EmpiricalStandards. Accessed: May 3, 2023.

[2] Paschal C. Amusuo, Aishwarya Sharma, Siddharth R. Rao, Abbey Vincent, and James C. Davis. 2022. Reflections on software failure analysis. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1615–1620. https://doi.org/10.1145/3540250.3560879

[3] Dharun Anandayuvaraj and James C Davis. 2022. Reflecting on Recurring Failures in IoT Development. In *37th IEEE/ACM International Conference on Automated Software Engineering*. IEEE/ACM, 1–5.

[4] Stephen J. Ball. 2015. What is policy? 21 years later: reflections on the possibilities of policy research. *Discourse: Studies in the Cultural Politics of Education* 36, 3 (May 2015), 306–313. https://doi.org/10.1080/01596306.2015.1015279

[5] Victor R. Basili. 2006. The Role of Empirical Study in Software Engineering. In *2006 30th Annual IEEE/NASA Software Engineering Workshop*. IEEE, Columbia, MD, USA, 3–6. https://doi.org/10.1109/SEW.2006.34

[6] Victor R. Basili, Forrest Shull, and Filippo Lanubile. 1999. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering* 25, 04 (July 1999), 456–473. https://doi.org/10.1109/32.799939 Publisher: IEEE Computer Society.

[7] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. 2023. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 4.0* (4th ed.). IEEE Computer Society Press, Washington, DC, USA. https://doi.org/10.1109/9781118823096

[8] Susan S. Brilliant, John C. Knight, and Nancy G. Leveson. 1990. Analysis of faults in an N-version software experiment. *IEEE Transactions on software engineering* 16, 2 (1990), 238–247.

[9] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[10] H.K. Colebatch. 2018. Introduction to the Handbook on Policy, Process and Governing. In *Handbook on Policy, Process and Governing*. Edward Elgar Publishing, 1–13. https://doi.org/10.4337/9781784714871.00005

[11] W Alec Cram, Jeffrey G Proudfoot, and John D'arcy. 2017. Organizational information security policies: a review and research framework. *European Journal of Information Systems* 26 (2017), 605–641.

[12] Luca Di Grazia and Michael Pradel. 2022. The evolution of type annotations in python: an empirical study. In *ESEC/FSE (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 209–220. https://doi.org/10.1145/3540250.3549114

[13] Bogdan Dit, Evan Moritz, Mario Linares-Vasquez, and Denys Poshyvanyk. 2013. Supporting and Accelerating Reproducible Research in Software Maintenance Using TraceLab Component Library. In *2013 IEEE International Conference on Software Maintenance*. IEEE, Eindhoven, Netherlands, 330–339. https://doi.org/10.1109/ICSM.2013.44

[14] Naranker Dulay, Emil Lupu, Morris Sloman, and N. Damianou. 2001. *A policy deployment model for the Ponder language*. IEEE Press., Seattle. https://doi.org/10.1109/INM.2001.918064 Pages: 543.

[15] Jesús M. González-Barahona and Gregorio Robles. 2012. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering* 17, 1-2 (Feb. 2012), 75–89. https://doi.org/10.1007/s10664-011-9181-9

[16] Hao He, Runzhi He, Haiqiao Gu, and Minghui Zhou. 2021. 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) (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 478–490. https://doi.org/10.1145/3468264.3468571

[17] Chris Hobbs. 2016. Software Development Standards. In *Embedded Software Development for Safety-Critical Systems* (2nd ed.), Chris Hobbs (Ed.). Elsevier, Chapter 3, 65–91.

[18] W.S. Humphrey. 1996. Using a defined and measured Personal Software Process. *IEEE Software* 13, 3 (May 1996), 77–88. https://doi.org/10.1109/52.493023

[19] IEEE. 2014. IEEE Standard for Software Quality Assurance Processes. IEEE Standards Association. https://ieeexplore.ieee.org/document/6838485 IEEE Std 730-2014.

[20] ISO. 2015. Quality management systems – Requirements. International Organization for Standardization. https://www.iso.org/standard/62085.html ISO Standard 9001.

[21] ISO/IEC 12207. 2017. Systems and software engineering–Software life cycle processes. https://www.iso.org/standard/63711.html

[22] Özgür Kafali, Jasmine Jones, Megan Petruso, Laurie Williams, and Munindar P. Singh. 2017. How Good Is a Security Policy against Real Breaches? A HIPAA Case Study. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 530–540. https://doi.org/10.1109/ICSE.2017.55

[23] S. Keshav. 2007. How to Read a Paper. *SIGCOMM Comput. Commun. Rev.* 37, 3 (jul 2007), 83–84. https://doi.org/10.1145/1273445.1273458

[24] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28, 8 (Aug. 2002), 721–734. https://doi.org/10.1109/TSE.2002.1027796

[25] Lech Madeyski and Barbara Kitchenham. 2017. Would wider adoption of reproducible research be beneficial for empirical software engineering research? *Journal of Intelligent & Fuzzy Systems* 32, 2 (Jan. 2017), 1509–1521. https://doi.org/10.3233/JIFS-169146

[26] MISRA. 2013. *MISRA C 2012: Guidelines for the Use of the C Language in Critical Systems: March 2013*. Motor Industry Software Research Association.

[27] Emerson Murphy-Hill, Gail C. Murphy, and William G. Griswold. 2010. Understanding context: creating a lasting impact in experimental software engineering research. In *FSE/SDP workshop on Future of SWEng research (FoSER '10)*. Association for Computing Machinery, New York, NY, USA, 255–258. https://doi.org/10.1145/1882362.1882415

[28] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, Saint Petersburg Russia, 466–476. https://doi.org/10.1145/2491411.2491415

[29] Prasad Naldurg and Roy H. Campbell. 2003. Modeling insecurity: policy engineering for survivability. In *ACM workshop on Survivable and self-regenerative systems (SSRS '03)*. Association for Computing Machinery, New York, NY, USA, 91–98. https://doi.org/10.1145/1036921.1036931

[30] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. 2022. Sigstore: software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2353–2367.

[31] NIST. [n. d.]. NIST SP 800-12: Chapter 5 - Computer Security Policy. https://csrc.nist.rip/publications/nistpubs/800-12/800-12-html/chapter5.html

[32] Chinenye Okafor, Taylor R Schorlemmer, Santiago Torres-Arias, and James C Davis. 2022. 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*. 15–24.

[33] Alan J Perlis, Frederick Sayward, and Mary Shaw. 1981. *Software metrics: an analysis and evaluation*. Vol. 5. Mit Press.

[34] Kai Petersen and Claes Wohlin. 2009. Context in industrial software engineering research. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 401–404. https://doi.org/10.1109/ESEM.2009.5316010

[35] Roger S Pressman. 2014. *Software engineering: a practitioner's approach*. McGraw-Hill Education.

[36] Qingchao Shen, Haoyang Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen. 2021. A comprehensive study of deep learning compiler bugs. In *ESEC/FSE (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 968–980. https://doi.org/10.1145/3468264.3468591

[37] Janet Siegmund, Norbert Siegmund, and Sven Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 9–19. https://doi.org/10.1109/ICSE.2015.24 ISSN: 1558-1225.

[38] Morris Sloman. 1994. Policy driven management for distributed systems. *Journal of Network and Systems Management* 2, 4 (Dec. 1994), 333–360. https://doi.org/10.1007/BF02283186

[39] Ian Sommerville. 2011. *Software engineering* (9th ed ed.). Pearson, Boston. OCLC: ocn462909026.

[40] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. 2022. *Secure Software Development Framework (SSDF) version 1.1 recommendations for mitigating the risk of software vulnerabilities*. Number NIST SP 800-218. National Institute of Standards and Technology (U.S.), Gaithersburg, MD. NIST SP 800–218 pages. https://doi.org/10.6028/NIST.SP.800-218

[41] René Wies. 1995. Using a Classification of Management Policies for Policy Specification and Policy Transformation. In *Integrated Network Management IV*, Adarshpal S. Sethi, Yves Raynaud, and Fabienne Faure-Vincent (Eds.). Springer US, Boston, MA, 44–56. https://doi.org/10.1007/978-0-387-34890-2_4

[42] Claes Wohlin, Martin Höst, and Anders Wesslén. 1999. Can the personal software process be used for empirical studies. In *Proceedings ICSE Workshop on Empirical Studies of Software Development and Evolution*.

[43] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

[44] C. Wohlin and A. Wesslen. 1998. Understanding software defect detection in the Personal Software Process. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*. 49–58. https://doi.org/10.1109/ISSRE.1998.730773

[45] Simon Xu. 2017. Empirical research methods for software engineering: Keynote address. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*. 1–1. https://doi.org/10.1109/SERA.2017.7965698