A Systematic Study on Empirical Software Engineering Research in 2018

Yuqing Yang*
161250179
Junda Chen†
161250010
Ao Zhang†
161250193
Aiwei Liu†
161250071
Chen Lei
161250054
Kangqiao Zhou
161070096
Yuanxiang Lian
1612501065
Xiangyu Zhang
161250202
Ruihua Wang
161250143
***Writer of paper**
†**Presenter**

Abstract—**This paper serves as a periodical report of the assignments from Empirical Software Engineering(henceforth referred to as ESE) course in 2019 spring.**

In this paper, we made an analysis through the paper published in 2018 of three main ESE conferences and journals: Empirical Software Engineering(a.k.a. EMSE), Empirical Software Engineering and Management(a.k.a. ESEM), and Evaluation and Assessment in Software Engineering(a.k.a. EASE), respectively.

We arrange this paper into two main parts, in the first section, we present the workflow and difficulty in analyzing and categorizing the papers, whereas in the second section, we present our research results from the data we obtained.

I. Data Retrieval

A. Search Process

1) Preparations: We are to analyze the papers published in 2018 in EMSE, ESEM, and EASE. Therefore we are performing an exhaustive search throughout the database. To obtain the paper and relative informations, our three sub-groups provided different approaches:

- EASE, ESEM:
  - Obtain the article information and article type(full-paper or workshop) from the conference website
  - Query the paper by name of full-paper articles in dblp [?]
  - Jump to ACM online library to obtain the paper
- EMSE:
  - Obtain the Contents of EMSE 2018 in springer
  - Download the full-paper of EMSE

In the process of paper searching, we found a total number of 180 papers from three venures, among which EMSE along contains 126 papers, making the task assignments challenging. To re-balance the workload, we altered the allocation strategy by evenly distributing works to each group, therefore each group have 60 papers to analyze. The final task allocation is shown below.

G1: Y.Q. Yang, C. Lei, K.Q. Zhou	EASE01-12,EMSE061-108
G2: J.D. Chen, A.W Liu, X.Y. Zhang	ESEM01-42,EMSE109-126
G3: A. Zhang, Y.X. Lian, R.H. Wang	EMSE001-060

To minimize the bias of personal judgement, each member of the sub-group will analyze 2/3 of the number of papers assigned, making sure that each paper is double-checked.

2) TAC-IE: efficient paper reading technique: We present TAC-IE, an efficient paper reading technique, aiming to easily identify and classify research methods of a paper. This paper is based on the structure of research paper. According to our observations, a good paper should be structured like:

- **T**itle
- **A**bstract
- **I**ntroduction
- Related work & background
- **E**xperiment
- Result
- **C**onclusion

Based on such observations, we go through papers by using methods below to optimize our efficiency, making us able to complete vast reading tasks in short period of time.

- First read the **T**itle, **A**bstract,**C**onclusion, trying to understand the problem and the experiment
- Then read the last two paragraphs of **I**ntroduction and the **E**xperiment

3) Criterias: We have two different criterias. One is for filtering out papers, the other is for identifying methods.

Filtering criteria. Due to the huge workload of searching, we adopt a few criterias to filter out unnecessary papers to save time.

- C1: Only full-paper is considered, excluding short paper and workshops
- C2: Non-paper and withdrawn paper is excluded

Classifying criteria. Moreover, a criteria of classification is also proposed after intense trail-and-error and discussions, thus augmenting our categorizing process. The main idea is that we capture keywords and analyze certain patterns of pre-defined methods, and additional discussion is required if new difficult case encountered.

- Experiment
  - presence of **Experiment** keyword
  - focus on changing controlled variants and comparing experiment results
  - a new method is proposed and compared to the baseline to prove its advantage
  - 'quasi-experiment'
- Case Study
  - presence of **case study** keyword
  - a specific case is proveded, e.g. some companies, projects etc.
- Survey
  - presence of **survey** keyword
  - usage of interview, and questionaire...
  - Literature surveys
- Ethnography
  - observations in certain people working routine
- Action Research
  - iterative
  - a feedback and adjust after each iterations

B. Challenges and Solutions

During the reading process, we overcomed the difficulties and managed to present an analysis over the 180 papers.

Timing technique Timing is always a problem in group working. Many group project is cancelled or postponed due to unexpected timing problems. To avoid such risks, we adopt a 'loose-deadline' strategy, setting deadlines slightly earlier than the real deadline, making even somehow postponed works able to finish before the real deadline.

Large amount of reading In 2018, there are 126 articles published in EMSE, whereas in 2012, the number of articles is 27. Moreover, the total reading workload of 2012 is 73 articles, however for 2018, the workload rises up to 180. Therefore a better reading method is needed. To solve this problem, we proposed the TAC-IE reading pattern, making even those who had never read research papers able to read quickly.

Research method ambiguity Some of the research methods are hard to identify, and the discriptions remain descriptive and empirical. Hence we discuss and work out a table of patterns of each methods, and provide examples of each of them. For some of the hard-to-define papers, each member of the group will use the descriptions of the paper and our example to decide which category that article falls into.

Complex research methods Some of the papers contain hybrid methods, we identify the most elicit ones and take them all into account.

Data Representation Due to the complex existence of hybrid research methods, we develop different diagrams to integrate our result. Such diagrams include var chart for an overview of methods used, fan charts of each venure for frequency analysis, and a venn diagram to show the existence of hybrid method.

II. Data Analysis

A. Data Processing

After the data collection, we do following analysis:

- identify and exclude the non-papers and short-papers
- identify the methods used, whether hybrid or not
- give each paper a description of the contents for further discussion
- record the bibtex of each paper

After the data is processed, we make following diagrams to show our result:

- three fan charts of each venures, to see the frequency of each method
- a bar chart of each venures, to show the amount of method usage
- a venn diagram, to show the combination of methods

During such process, 8 papers are excluded for non-papers, 3 papers are identified as systematic review, and the diagrams will be shown in the following section.

B. Diagrams and Explanations

Fig. 1. Total research method usage.

As shown in Fig. 1[1], due to the relative vast number of articles, the bar of EMSE is significantly higher than any others. Nevertheless we can still figure out the top-3 frequent method, which refers to experimet, case study, and survey. This does make sense in that usually ethnography is time-consuming and hard to perform, and action research require effort and iterations. With all the respect, such methods appear rarer than the top-3 frequent methods.

As show in Fig. 2[2], the distribution of methods are similar to each other: case study, experiment and survey are frequently uesd, and usage of ethnography is usually much fewer.

One of the interesting phonomenon is that we find that EASE seem to appeal more to case study researches, and case study takes up 40% of total usages. EMSE likes experiment research more, among all the appearances of research methods, 39.33% of them are delivered in experiment. ESEM, different to both, prefer survey more, with a precentage of 36.21%.

On top of the two analysis, we also calculate the hybrid method usage in a venn's diagram. As shown in Fig. 3[3], among the 180 papers, 65 combined two methods, and

[1]Papers with hybrid methods included
[2]With respect to total method appearances
[3]Consider only the top-3 frequent methods

Fig. 2. Research method usage ratios.

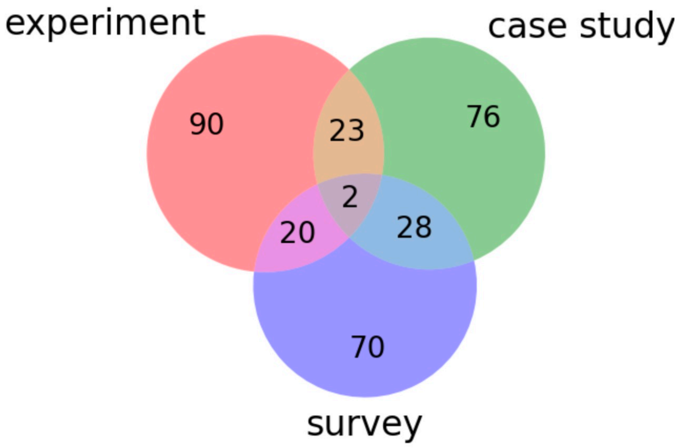


Fig. 3. Total research method usage

there are 2 of them combined 3 methods. Therefore there are over one third of papers adopted hybrid methods, which is an intriguing result.

C. Conclusion

After our preliminary works, we now present several conclusions upon the papers of three venures as follows.

- Among the five methods, experiment, case study, and survey are the three most common methods.
- Different conferences/journals has different taste. EASE prefers case study, EMSE prefers experiment, and ESEM seem to appeal more to survey.
- Papers sometimes integrate different methods in their researches, and papers adopting multiple methods take up one third of all the papers examined.

References

[1] Abad, Z. S. H., Karras, O., Schneider, K., Barker, K., and Bauer, M. Task interruption in software development projects: What makes some interruptions more disruptive than others? In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 122–132.

[2] Accioly, P., Borba, P., and Cavalcanti, G. Understanding semi-structured merge conflict characteristics in open-source java projects. *Empirical Software Engineering 23*, 4 (Aug 2018), 2051–2085.

[3] Ahmed, S., and Bagherzadeh, M. What do concurrency developers ask about?: A large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 30:1–30:10.

[4] Ajienka, N., Capiluppi, A., and Counsell, S. An empirical study on the interplay between semantic coupling and co-change of software classes. *Empirical Software Engineering 23*, 3 (2018), 1791–1825.

[5] Ali, N., Baker, S., O'Crowley, R., Herold, S., and Buckley, J. Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering 23*, 1 (Feb 2018), 224–258.

[6] Ali, N., Baker, S., O' Crowley, R., Herold, S., and Buckley, J. Erratum to: Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering 23*, 3 (2018), 1868–1869.

[7] Aljarallah, S., and Lock, R. An exploratory study of software sustainability dimensions and characteristics: end user perspectives in the kingdom of saudi arabia (ksa). In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 14.

[8] Aniche, M., Bavota, G., Treude, C., Gerosa, M. A., and van Deursen, A. Code smells for model-view-controller architectures. *Empirical Software Engineering 23*, 4 (Aug 2018), 2121–2157.

[9] Arcuri, A. An experience report on applying software testing academic results in industry: We need usable automated test generation. *Empirical Software Engineering 23*, 4 (2018), 1959–1981.

[10] Arif, M. M., Shang, W., and Shihab, E. Empirical study on the discrepancy between performance testing results from virtual and physical environments. *Empirical Software Engineering* (2018), 1–29.

[11] Ayala, C., Nguyen-Duc, A., Franch, X., Höst, M., Conradi, R., Cruzes, D., and Babar, M. A. System requirements-oss components: matching and mismatch resolution practices – an empirical study. *Empirical Software Engineering 23*, 6 (Dec 2018), 3073–3128.

[12] Bagherzadeh, M., Kahani, N., Bezemer, C.-P., Hassan, A. E., Dingel, J., and Cordy, J. R. Analyzing a decade of linux system calls. *Empirical Software Engineering 23*, 3 (2018), 1519–1551.

[13] Baltes, S., and Diehl, S. Usage and attribution of stack overflow code snippets in github projects. *Empirical Software Engineering* (2018), 1–37.

[14] Bao, L., Xing, Z., Xia, X., Lo, D., and Hassan, A. E. Inference of development activities from interaction with uninstrumented applications. *Empirical Software Engineering 23*, 3 (2018), 1313–1351.

[15] Bastarrica, M. C., Espinoza, G., and Marín, J. Implementing agile practices: the experience of tsol. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 38.

[16] Behnamghader, P., Meemeng, P., Fostiropoulos, I., Huang, D., Srisopha, K., and Boehm, B. A scalable and efficient approach for compiling and analyzing commit history. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 27:1–27:10.

[17] Bernardi, S., Domínguez, J. L., Gómez, A., Joubert, C., Merseguer, J., Perez-Palacin, D., Requeno, J. I., and Romeu, A. A systematic approach for performance assessment using process mining. *Empirical Software Engineering 23*, 6 (Dec 2018), 3394–3441.

[18] Binkley, D., Lawrie, D., and Morrell, C. The need for software specific natural language techniques. *Empirical Software Engineering 23*, 4 (Aug 2018), 2398–2425.

[19] Blincoe, K., Dehghan, A., Salaou, A.-D., Neal, A., Linaker, J., and Damian, D. High-level software requirements and iteration changes: a predictive model. *Empirical Software Engineering* (Oct 2018).

[20] Borle, N. C., Feghhi, M., Stroulia, E., Greiner, R., and Hindle, A. Analyzing the effects of test driven development in github. *Empirical Software Engineering 23*, 4 (Aug 2018), 1931–1958.

[21] Brings, J., Daun, M., Kempe, M., and Weyer, T. On different search methods for systematic literature reviews and maps: Experiences from a literature search on validation and verification of emergent behavior. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018* (New York, NY, USA, 2018), EASE'18, ACM, pp. 35–45.

[22] Calefato, F., Lanubile, F., Maiorano, F., and Novielli, N. Sentiment polarity detection for software development. *Empirical Software Engineering 23*, 3 (2018), 1352–1382.

[23] Cartaxo, B., Pinto, G., and Soares, S. The role of rapid reviews in supporting decision-making in software engineering practice. In *EASE* (2018), pp. 24–34.

[24] Castelluccio, M., An, L., and Khomh, F. An empirical study of patch uplift in rapid release development pipelines. *Empirical Software Engineering* (Nov 2018).

[25] Chakraborty, P., Shahriyar, R., Iqbal, A., and Bosu, A. Understanding the software development practices of blockchain projects: A survey. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 28:1–28:10.

[26] Chari, K., and Agrawal, M. Impact of incorrect and new requirements on waterfall software project outcomes. *Empirical Software Engineering 23*, 1 (2018), 165–185.

[27] Chen, C., Xing, Z., and Liu, Y. What' s spain' s paris? mining analogical libraries from q&a discussions. *Empirical Software Engineering* (2018), 1–40.

[28] Chowdhury, S., Borle, S., Romansky, S., and Hindle, A. Greenscaler: training software energy models with automatic test generation. *Empirical Software Engineering* (2018), 1–44.

[29] Chowdhury, S., Di Nardo, S., Hindle, A., and Jiang, Z. M. J. An exploratory study on assessing the energy impact of logging on android applications. *Empirical Software Engineering 23*, 3 (2018), 1422–1456.

[30] Coelho, J., Valente, M. T., Silva, L. L., and Shihab, E. Identifying unmaintained projects in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 15:1–15:10.

[31] Costa, D. A. d., McIntosh, S., Treude, C., Kulesza, U., and Hassan, A. E. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering 23*, 2 (Apr 2018), 835–904.

[32] Coviello, C., Romano, S., and Scanniello, G. An empirical study of inadequate and adequate test suite reduction approaches. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 12:1–12:10.

[33] Da Costa, D. A., McIntosh, S., Kulesza, U., Hassan, A. E., and Abebe, S. L. An empirical study of the integration time of fixed issues. *Empirical Software Engineering 23*, 1 (2018), 334–383.

[34] Danglot, B., Preux, P., Baudry, B., and Monperrus, M. Correctness attraction: a study of stability of software behavior under runtime perturbation. *Empirical Software Engineering 23*, 4 (Aug 2018), 2086–2119.

[35] Datta, S. How does developer interaction relate to software quality? an examination of product development data. *Empirical Software Engineering 23*, 3 (Jun 2018), 1153–1187.

[36] Dingsøyr, T., Moe, N. B., Fægri, T. E., and Seim, E. A. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering 23*, 1 (2018), 490–520.

[37] Dintzner, N., van Deursen, A., and Pinzger, M. Fever: An approach to analyze feature-oriented changes and artefact co-evolution in highly configurable systems. *Empirical Software Engineering 23*, 2 (Apr 2018), 905–952.

[38] El Zanaty, F., Hirao, T., McIntosh, S., Ihara, A., and Matsumoto, K. An empirical study of design discussions in code review. *management 9*, 2011 (2018).

[39] Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., and Oivo, M. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering 23*, 1 (2018), 452–489.

[40] Fan, Y., Xia, X., Lo, D., and Li, S. Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering 23* (2018), 3346–3393.

[41] Feng, Q., Cai, Y., Kazman, R., and Mo, R. The birth, growth, death and rejuvenation of software maintenance communities. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 5.

[42] Fernández-Sáez, A. M., Chaudron, M. R., and Genero, M. An industrial case study on the use of uml in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering 23*, 6 (2018), 3281–3345.

[43] Ferrari, A., Gori, G., Rosadini, B., Trotta, I., Bacherini, S., Fantechi, A., and Gnesi, S. Detecting requirements defects with nlp patterns: an industrial experience in the railway domain. *Empirical Software Engineering* (2018), 1–50.

[44] Fucci, D., Palomares, C., Franch, X., Costal, D., Raatikainen, M., Stettinger, M., Kurtanovic, Z., Kojo, T., Koenig, L., Falkner, A., Schenner, G., Brasca, F., Männistö, T., Felfernig, A., and Maalej, W. Needs and challenges for a platform to support large-scale requirements engineering: A multiple-case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 19:1–19:10.

[45] Fucci, D., Romano, S., Baldassarre, M. T., Caivano, D., Scanniello, G., Turhan, B., and Juristo, N. A longitudinal cohort study on the retainment of test-driven development. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 18:1–18:10.

[46] Gadient, P., Ghafari, M., Frischknecht, P., and Nierstrasz, O. Security code smells in android icc. *Empirical Software Engineering* (2018), 1–31.

[47] Gharehyazie, M., Ray, B., and Filkov, V. Some from here, some from there: Cross-project code reuse in github. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)* (2017), 291–301.

[48] Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., and Yu, H. Data-efficient performance learning for configurable systems. *Empirical Software Engineering 23*, 3 (2018), 1826–1867.

[49] Gupta, M., Asadullah, A., Padmanabhuni, S., and Serebrenik, A. Reducing user input requests to improve it support ticket resolution process. *Empirical Software Engineering* (2018), 1–40.

[50] Hadar, I., Hasson, T., Ayalon, O., Toch, E., Birnhack, M., Sherman, S., and Balissa, A. Privacy by designers: software developers' privacy mindset. *Empirical Software Engineering 23*, 1 (Feb 2018), 259–289.

[51] Hannebauer, C., Hesenius, M., and Gruhn, V. Does syntax highlighting help programming novices? *Empirical Software Engineering 23*, 5 (Oct 2018), 2795–2828.

[52] Hassan, S., Tantithamthavorn, C., Bezemer, C.-P., and Hassan, A. E. Studying the dialogue between users and developers of free apps in the google play store. *Empirical Software Engineering 23*, 3 (2018), 1275–1312.

[53] Hassani, M., Shang, W., Shihab, E., and Tsantalis, N. Studying and detecting log-related issues. *Empirical Software Engineering 23*, 6 (2018), 3248–3280.

[54] Hofer, F. Architecture, technologies and challenges for cyber-physical systems in industry 4.0: a systematic mapping study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ESEM '18, Association for Computing Machinery, p. 1–10.

[55] Hu, H., Bezemer, C.-P., and Hassan, A. E. Studying the consistency of star ratings and the complaints in 1 & 2-star user reviews for top free cross-platform android and ios apps. *Empirical Software Engineering 23* (2018), 3442–3475.

[56] Hu, W., Carver, J. C., Anu, V., Walia, G. S., and Bradshaw, G. L. Using human error information for error prevention. *Empirical Software Engineering 23*, 6 (2018), 3768–3800.

[57] Huang, Q., Shihab, E., Xia, X., Lo, D., and Li, S. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering 23*, 1 (2018), 418–451.

[58] Huang, Q., Xia, X., and Lo, D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering* (Oct 2018).

[59] Huijgens, H., Spadini, D., Stevens, D., Visser, N., and van Deursen, A. Software analytics in continuous delivery: A case study on success factors. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 25:1–25:10.

[60] Jha, N., and Mahmoud, A. Using frame semantics for classifying and summarizing application store reviews. *Empirical Software Engineering 23*, 6 (Dec 2018), 3734–3767.

[61] Jimenez, M., Checkam, T. T., Cordy, M., Papadakis, M., Kintis, M., Traon, Y. L., and Harman, M. Are mutants really natural?: A study on how "naturalness" helps mutant selection. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 3:1–3:10.

[62] Kabinna, S., Bezemer, C.-P., Shang, W., Syer, M. D., and Hassan, A. E. Examining the stability of logging statements. *Empirical Software Engineering 23*, 1 (2018), 290–333.

[63] Karras, O., Risch, A., and Schneider, K. Interrelating use cases and associated requirements by links: An eye tracking study on the impact of different linking variants on the reading behavior. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 2–12.

[64] Kintis, M., Papadakis, M., Papadopoulos, A., Valvis, E., Malevris, N., and Le Traon, Y. How effective are mutation testing tools? an empirical analysis of java mutation testing tools with manual analysis and real faults. *Empirical Software Engineering 23*, 4 (2018), 2426–2463.

[65] Kopeć, W., Balcerzak, B., Nielek, R., Kowalik, G., Wierzbicki, A., and Casati, F. Older adults and hackathons: a qualitative study. *Empirical Software Engineering 23*, 4 (Aug 2018), 1895–1930.

[66] Kosar, T., Gaberc, S., Carver, J. C., and Mernik, M. Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. *Empirical Software Engineering 23* (2017), 2734–2763.

[67] Kropp, M., Meier, A., Anslow, C., and Biddle, R. Satisfaction, practices, and influences in agile software development. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 112–121.

[68] Kuchta, T., Lutellier, T., Wong, E., Tan, L., and Cadar, C. On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in pdf readers and files. *Empirical Software Engineering 23*, 6 (Dec 2018), 3187–3220.

[69] Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. Do developers update their library dependencies? *Empirical Software Engineering 23*, 1 (2018), 384–417.

[70] Kuutila, M., Mäntylä, M., Claes, M., Elovainio, M., and Adams, B. Using experience sampling to link software repositories with emotions and work well-being. *arXiv preprint arXiv:1808.05409* (2018).

[71] Labunets, K. No search allowed: what risk modeling notation to choose? In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 20.

[72] Laukkanen, E., Paasivaara, M., Itkonen, J., and Lassenius, C. Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering* (2018), 1–43.

[73] Le, X. B., Thung, F., Lo, D., and Goues, C. L. [journal first] overfitting in semantics-based automated program repair. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), 163–163.

[74] Li, H., Chen, T.-H., Shang, W., and Hassan, A. E. Studying software logging using topic models. *Empirical Software Engineering 23* (2018), 2655–2694.

[75] Li, X., Wong, W. E., Gao, R., Hu, L., and Hosono, S. Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. *Empirical Software Engineering 23*, 1 (2018), 1–51.

[76] Licorish, S. A., Zolduoarrati, E., and Stanger, N. Linking user requests, developer responses and code changes: Android os case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 79–89.

[77] Lin, D., Bezemer, C.-P., and Hassan, A. E. An empirical study of early access games on the steam platform. *Empirical Software Engineering 23*, 2 (2018), 771–799.

[78] Luz, W. P., Pinto, G., and Bonifácio, R. Building a collaborative culture: A grounded theory of well succeeded devops adoption in practice. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 6:1–6:10.

[79] Madeyski, L., and Kitchenham, B. Effect sizes and their variance for ab/ba crossover design studies. *Empirical Software Engineering 23*, 4 (Aug 2018), 1982–2017.

[80] McChesney, I. R., and Bond, R. Eye tracking analysis of computer program comprehension in programmers with dyslexia. *Empirical Software Engineering* (2018), 1–46.

[81] Medeiros, F., Lima, G., Amaral, G., Apel, S., Kästner, C., Ribeiro, M., and Gheyi, R. An investigation of misunderstanding code patterns in c open-source software projects. *Empirical Software Engineering* (Nov 2018).

[82] Mendes, E., Perkusich, M., Freitas, V., and Nunes, J. Using bayesian network to estimate the value of decisions within the context of value-based software engineering. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 90–100.

[83] Mendonça, D. S., da Silva, T. G., de Oliveira, D. F., Brandão, J. S., Lopes, H., Barbosa, S. D. J., Kalinowski, M., and von Staa, A. Applying pattern-driven maintenance: A method to prevent latent unhandled exceptions in web applications. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 31:1–31:10.

[84] Mengerink, J. G. M., Noten, J., and Serebrenik, A. Empowering ocl research: a large-scale corpus of open-source data.

[85] Mezouar, M. E., Zhang, F., and Zou, Y. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering 23*, 3 (Jun 2018), 1704–1742.

[86] Mondal, M., Rahman, M. S., Roy, C. K., and Schneider, K. A. Is cloned code really stable? *Empirical Software Engineering 23*, 2 (Apr 2018), 693–770.

[87] Moonen, L., Rolfsnes, T., Binkley, D., and Di Alesio, S. What are the effects of history length and age on mining software change impact? *Empirical Software Engineering 23*, 4 (Aug 2018), 2362–2397.

[88] Morrison, P. J., Pandita, R., Xiao, X., Chillarege, R., and Williams, L. Are vulnerabilities discovered and resolved like other defects? *Empirical Software Engineering 23*, 3 (Jun 2018), 1383–1421.

[89] Motwani, M., Sankaranarayanan, S., Just, R., and Brun, Y. Do automated program repair techniques repair hard and important bugs? *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2017), 25–25.

[90] Mujahid, S., Sierra, G., Abdalkareem, R., Shihab, E., and Shang, W. An empirical study of android wear user complaints. *Empirical Software Engineering 23* (2018), 3476–3502.

[91] Munir, H., Linåker, J., Wnuk, K., Runeson, P., and Regnell, B. Open innovation using open source tools: a case study at sony mobile. *Empirical Software Engineering 23*, 1 (Feb 2018), 186–223.

[92] Murgia, A., Ortu, M., Tourani, P., Adams, B., and Demeyer, S. An exploratory qualitative and quantitative analysis of emotions in issue report comments of open source systems. *Empirical Software Engineering 23*, 1 (2018), 521–564.

[93] Narayanan, A., Chandramohan, M., Chen, L., and Liu, Y. A multi-view context-aware approach to android malware detection and malicious code localization. *Empirical Software Engineering* (2018), 1–53.

[94] Nayebi, M., Cho, H., and Ruhe, G. App store mining is not enough for app improvement. *Empirical Software Engineering 23*, 5 (Oct 2018), 2764–2794.

[95] Nielebock, S., Krolikowski, D., Krüger, J., Leich, T., and Ortmeier, F. Commenting source code: is it worth it for small programming tasks? *Empirical Software Engineering* (2018), 1–40.

[96] Oliveira, V. P. L., de Souza, E. F., Goues, C. L., and Camilo-Junior, C. G. Improved representation and genetic operators for linear genetic programming for automated program repair. *Empirical Software Engineering 23* (2017), 2980–3006.

[97] Paasivaara, M., Behm, B., Lassenius, C., and Hallikainen, M. Large-scale agile transformation at ericsson: a case study. *Empirical Software Engineering 23*, 5 (Oct 2018), 2550–2596.

[98] Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., and De Lucia, A. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering 23*, 3 (2018), 1188–1221.

[99] Pano, A., Graziotin, D., and Abrahamsson, P. Factors and actors leading to the adoption of a javascript framework. *Empirical Software Engineering* (2018), 1–32.

[100] Pashchenko, I., Plate, H., Ponta, S. E., Sabetta, A., and Massacci, F. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 42:1–42:10.

[101] Peitek, N., Siegmund, J., Parnin, C., Apel, S., Hofmeister, J. C., and Brechmann, A. Simultaneous measurement of program comprehension with fmri and eye tracking: A case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 24:1–24:10.

[102] Pinto, G., Steinmacher, I., Dias, L. F., and Gerosa, M. On the challenges of open-sourcing proprietary software projects. *Empirical Software Engineering 23*, 6 (2018), 3221–3247.

[103] Port, D., and Taber, B. An empirical study of process policies and metrics to manage productivity and quality for maintenance of critical software systems at the jet propulsion laboratory. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 37:1–37:10.

[104] Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., and Lo, D. Categorizing the content of github readme files. *Empirical Software Engineering* (Oct 2018).

[105] Przybyłek, A. An empirical study on the impact of aspectj on software evolvability. *Empirical Software Engineering 23*, 4 (Aug 2018), 2018–2050.

[106] Qi, K., Hira, A., Venson, E., and Boehm, B. W. Calibrating use case points using bayesian analysis. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 7:1–7:10.

[107] Quirchmayr, T., Paech, B., Kohl, R., Karey, H., and Kasdepke, G. Semi-automatic rule-based domain terminology and software feature-relevant information extraction from natural language user manuals. *Empirical Software Engineering* (2018), 1–54.

[108] Ragkhitwetsagul, C., Krinke, J., and Clark, D. A comparison of code similarity analysers. *Empirical Software Engineering 23*, 4 (Aug 2018), 2464–2519.

[109] Rahimi, M., and Cleland-Huang, J. Evolving software trace links between requirements and source code. *Empirical Software Engineering 23*, 4 (Aug 2018), 2198–2231.

[110] Rakha, M. S., Bezemer, C.-P., and Hassan, A. E. Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval. *Empirical Software Engineering 23*, 5 (Oct 2018), 2597–2621.

[111] Ralph, P., and Tempero, E. Construct validity in software engineering research and software metrics. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 13–23.

[112] Rastogi, A., Nagappan, N., Gousios, G., and van der Hoek, A. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 22:1–22:8.

[113] Ribeiro, T. V., Massollar, J., and Travassos, G. H. Challenges and pitfalls on surveying evidence in the software engineering technical literature: an exploratory study with novices. *Empirical Software Engineering 23*, 3 (Jun 2018), 1594–1663.

[114] Ricca, F., Torchiano, M., Leotta, M., Tiso, A., Guerrini, G., and Reggio, G. On the impact of state-based model-driven development on maintainability: a family of experiments using unimod. *Empirical Software Engineering 23*, 3 (Jun 2018), 1743–1790.

[115] Rios, N., Spínola, R. O., Mendonça, M., and Seaman, C. The most common causes and effects of technical debt: First results from a global family of industrial surveys. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 39:1–39:10.

[116] Rolfsnes, T., Moonen, L., Alesio, S. D., Behjati, R., and Binkley, D. Aggregating association rules to improve change recommendation. *Empirical Software Engineering 23*, 2 (Apr 2018), 987–1035.

[117] Romano, S., Scanniello, G., Fucci, D., Juristo, N., and Turhan, B. The effect of noise on software engineers' performance. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 9:1–9:10.

[118] Rosenberg, C. M., and Moonen, L. Improving problem identification via automated log clustering using dimensionality reduction. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 16:1–16:10.

[119] Roy, A., and Pham, H. Toward the development of a conventional time series based web error forecasting framework. *Empirical Software Engineering 23*, 2 (Apr 2018), 570–644.

[120] Saborido, R., Morales, R., Khomh, F., Guéhéneuc, Y.-G., and Antoniol, G. Getting the most from map data structures in android. *Empirical Software Engineering 23*, 5 (Oct 2018), 2829–2864.

[121] Saini, V., Sajnani, H., and Lopes, C. Cloned and non-cloned java methods: a comparative study. *Empirical Software Engineering 23*, 4 (Aug 2018), 2232–2278.

[122] Salman, I., Turhan, B., and Vegas, S. A controlled experiment on time pressure and confirmation bias in functional software testing. *Empirical Software Engineering* (2018), 1–35.

[123] Santos, A., and Juristo, N. Comparing techniques for aggregating interrelated replications in software engineering. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 8.

[124] Santos, A. R., do Carmo Machado, I., de Almeida, E. S., Siegmund, J., and Apel, S. Comparing the influence of using feature-oriented programming and conditional compilation on

comprehending feature-oriented software. *Empirical Software Engineering* (2018), 1–33.

[125] Santos, R. E. S., Magalhães, C. V. C., Capretz, L. F., Correia-Neto, J. S., da Silva, F. Q. B., and Saher, A. Computer games are serious business and so is their quality: Particularities of software testing in game development from the perspective of practitioners. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 33:1–33:10.

[126] Sawant, A. A., Robbes, R., and Bacchelli, A. On the reaction to deprecation of clients of 4 + 1 popular java apis and the jdk. *Empirical Software Engineering 23*, 4 (Aug 2018), 2158–2197.

[127] Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J. A., Tortora, G., Risi, M., and Dodero, G. Do software models based on the uml aid in source-code comprehensibility? aggregating evidence from 12 controlled experiments. *Empirical Software Engineering 23* (2017), 2695–2733.

[128] Scavuzzo, M., Di Nitto, E., and Ardagna, D. Experiences and challenges in building a data intensive system for data migration. *Empirical Software Engineering 23*, 1 (2018), 52–86.

[129] Schnappinger, M., Osman, M. H., Pretschner, A., Pizka, M., and Fietzke, A. Software quality assessment in practice: A hypothesis-driven framework. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 40:1–40:6.

[130] Senapathi, M., Buchan, J., and Osman, H. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 57–67.

[131] Shahin, M., Zahedi, M., Babar, M. A., and Zhu, L. An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering* (2018), 1–48.

[132] Shepperd, M. J. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empirical Software Engineering 23* (2018), 3801–3820.

[133] Shimagaki, J., Kamei, Y., Ubayashi, N., and Hindle, A. Automatic topic classification of test cases using text mining at an android smartphone vendor. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 32.

[134] Sirres, R., Bissyandé, T. F., Kim, D., Lo, D., Klein, J., Kim, K., and Traon, Y. L. Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering 23*, 5 (Oct 2018), 2622–2654.

[135] Sjøberg, D. I. K. An empirical study of wip in kanban teams. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 13:1–13:8.

[136] Soh, Z., Khomh, F., Guéhéneuc, Y.-G., and Antoniol, G. Noise in mylyn interaction traces and its impact on developers and recommendation systems. *Empirical Software Engineering 23*, 2 (Apr 2018), 645–692.

[137] Spadini, D., Aniche, M., Bruntink, M., and Bacchelli, A. Mock objects for testing java systems. *Empirical Software Engineering* (Nov 2018).

[138] Squire, M. Data sets describing the circle of life in ruby hosting, 2003–2016. *Empirical Software Engineering 23*, 2 (Apr 2018), 1123–1152.

[139] Stevanetic, S., and Zdun, U. Supporting the analyzability of architectural component models-empirical findings and tool support. *Empirical Software Engineering 23*, 6 (2018), 3578–3625.

[140] Strandberg, P. E., Afzal, W., and Sundmark, D. Decision making and visualizations based on test results. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 34:1–34:10.

[141] Stratis, P., Yaneva, V., and Rajan, A. Assessing the effect of data transformations on test suite compilation. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 4:1–4:10.

[142] Tahir, A., Bennin, K. E., MacDonell, S. G., and Marsland, S. Revisiting the size effect in software fault prediction models. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 23.

[143] Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., and Counsell, S. Can you tell me if it smells?: A study on how developers discuss code smells and anti-patterns in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 68–78.

[144] Tamburri, D. A., and Kazman, R. General methods for software architecture recovery: a potential approach and its evaluation. *Empirical Software Engineering 23*, 3 (Jun 2018), 1457–1489.

[145] Tamburri, D. A., Palomba, F., Serebrenik, A., and Zaidman, A. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering* (Nov 2018).

[146] Trautsch, F., Herbold, S., Makedonski, P., and Grabowski, J. Addressing problems with replicability and validity of repository mining studies through a smart data platform. *Empirical Software Engineering 23*, 2 (Apr 2018), 1036–1083.

[147] Trinkenreich, B., Conte, T., Barcellos, M. P., and Santos, G. Defining, measuring and monitoring it service goals and strategies: preliminary results and pitfalls from a qualitative study with it service managers. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 35.

[148] Tsikerdekis, M. Persistent code contribution: a ranking algorithm for code contribution in crowdsourced software. *Empirical Software Engineering 23*, 4 (Aug 2018), 1871–1894.

[149] Vale, T., and de Almeida, E. S. Experimenting with information retrieval methods in the recovery of feature-code spl traces. *Empirical Software Engineering* (Nov 2018).

[150] Vera-Pérez, O. L., Danglot, B., Monperrus, M., and Baudry, B. A comprehensive study of pseudo-tested methods. *CoRR abs/1807.05030* (2018).

[151] Vercammen, S., Demeyer, S., Borg, M., and Eldh, S. Speeding up mutation testing via the cloud: lessons learned for further optimisations. In *ESEM'18: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Oktober 11-12, 2018, Oulu, Finland* (2018).

[152] Walkinshaw, N., and Minku, L. Are 20% of files responsible for 80% of defects? In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 2.

[153] Wang, C., Cui, P., Daneva, M., and Kassab, M. Understanding what industry wants from requirements engineers: an exploration of re jobs in canada. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 41.

[154] Wang, D., and Galster, M. Development processes and practices in a small but growing software industry: A practitioner survey in new zealand. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 36:1–36:10.

[155] Wang, J., Wang, S., and Wang, Q. Is there a golden feature set for static warning identification?: an experimental evaluation. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), ACM, p. 17.

[156] Wang, S., Chen, T.-H., and Hassan, A. E. Understanding the factors for fast answers in technical q&a websites. *Empirical Software Engineering 23*, 3 (2018), 1552–1593.

[157] Wang, S., Lo, D., Vasilescu, B., and Serebrenik, A. Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering 23*, 2 (Apr 2018), 800–832.

[158] Washizaki, H., Guéhéneuc, Y.-G., and Khomh, F. Prometa: a taxonomy for program metamodels in program reverse engineering. *Empirical Software Engineering 23*, 4 (2018), 2323–2358.

[159] Williams, A. Using reasoning markers to select the more rigorous software practitioners' online content when searching for grey literature. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 46–56.

[160] Wu, R., Wen, M., Cheung, S.-C., and Zhang, H. Changelocator: locate crash-inducing changes based on crash reports. *Empirical Software Engineering 23*, 5 (Oct 2018), 2866–2900.

[161] Xiao, Y., Keung, J., Mi, Q., and Bennin, K. E. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 101–111.

[162] Xu, B., Shirani, A., Lo, D., and Alipour, M. A. Prediction of relatedness in stack overflow: Deep learning vs. svm: A reproducibility study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 21:1–21:10.

[163] Xu, B., Xing, Z., Xia, X., Lo, D., Wang, Q., and Li, S. Domain-specific cross-language relevant question retrieval. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)* (2016), IEEE, pp. 413–424.

[164] Ye, D., Bao, L., Xing, Z., and Lin, S.-W. Apireal: an api recognition and linking approach for online developer forums. *Empirical Software Engineering 23* (2018), 3129–3160.

[165] Yi, J., Tan, S. H., Mechtaev, S., Böhme, M., and Roychoudhury, A. A correlation study between automated program repair and test-suite metrics. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2017), 24–24.

[166] Yu, T., and Pradel, M. Pinpointing and repairing performance bottlenecks in concurrent programs. *Empirical Software Engineering 23* (2017), 3034–3071.

[167] Yu, Z., Kraft, N. A., and Menzies, T. Finding better active learners for faster literature reviews. *Empirical Software Engineering 23* (2017), 3161–3186.

[168] Zagalsky, A., German, D. M., Storey, M.-A., Teshima, C. G., and Poo-Caamaño, G. How the r community creates and curates knowledge: an extended study of stack overflow and mailing lists. *Empirical Software Engineering 23*, 2 (Apr 2018), 953–986.

[169] Zahedi, M., Babar, M. A., and Cooper, B. An empirical investigation of transferring research to software technology innovation: A case of data-driven national security software. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 10:1–10:10.

[170] Zhang, H., Wang, S., Yue, T., Ali, S., and Liu, C. Search and similarity based selection of use case scenarios: An empirical study. *Empirical Software Engineering 23*, 1 (Feb 2018), 87–164.

[171] Zhang, Y., Lo, D., Xia, X., Scanniello, G., Le, T.-D. B., and Sun, J. Fusing multi-abstraction vector space models for concern localization. *Empirical Software Engineering 23*, 4 (Aug 2018), 2279–2322.

[172] Zhong, H., and Meng, N. [journal first] towards reusing hints from past fixes: An exploratory study on thousands of real samples. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), 885–885.

A Systematic Study on Empirical Software Engineering Research in 2018: Experiment

Yuqing Yang[*†]
161250179
Junda Chen[*]
161250010
Ao Zhang
161250193

Aiwei Liu
161250071
Chen Lei[*†]
161250054
Kangqiao Zhou
161070096

Yuanxiang Lian[*†]
1612501065
Xiangyu Zhang
161250202
Ruihua Wang
161250143

[*]**Writer of paper**
[†]**Presenter**

***Abstract*—This paper is the second periodical report of the assignments from Empirical Software Engineering(henceforth referred to as ESE) course in 2019 spring.**

In this paper, we made an analysis through the paper published in 2018 of thethree main ESE conferences and journals, with a focus on researches that involve Experiment as their main research methods. We identify and compare the methods involved.

I. Paper Selection

A. Overview

Among the research papers we looked into, we mainly classify them into two categories.

One of the category is researches on a certain algorithm. Papers of this usually analyze the performances of some algorithms on computers, trying to prove that their algorithm works better than the baseline. Algorithms of these are usually solid on computer, whose performances are seldom affected by human-related factors, therefore it is quite different from the traditionally defined experiments.

The other category of research is more empirical: by observing how people act under controlled circumstances. Such experiments often requires more fine-grained controlls of experiments, and is the more standard form of the experiments as introduced on class.

For both of these categories, we identify them as within-experiments or between-experiments, and compare the differences.

Additionally, during the review, we find that in some cases the title of the paper itself also contains critical information of the experiment type. We select two of these papers, and a standard experiment paper, combined with the three papers in the presentation, together to form this paper.

B. Other Procedures

We made error correction on previous classification data. During the process, 3 experiments was corrected into case study in that we misattributed the experimantal methods in case study to experiment method.

C. Definition Rediscover

During the study we found that the idea of between experiment and within experiment might be **misleading**. It differs from the definition on wikipedia, which focus on the experiment of people, ignoring that there are some experiments of algorithms.

To solve this problem, we proposed the question on the group presentation for consultation and finally decided to use the definition of both people and algorithm, whereas the definition of algorthim experiment is refined under our study.

D. Paper Abstract Info

We propose six experiments as follows:

1) EMSE65 [2]:A standard exp: Since its inception in 1996, aspect-oriented programming (AOP) has been believed to reduce the effort required to maintain software systems by replacing cross-cutting code with aspects. However, little convincing empirical evidence exists to support this claim, while several studies suggest that AOP brings new obstacles to maintainability. This paper discusses two controlled experiments conducted to evaluate the impact of AspectJ (the most mature and popular aspect-oriented programming language) versus Java on software evolvability. We consider evolvability as the ease with which a software system can be updated to fulfill new requirements. Since a minor language was compared to the mainstream, the experiments were designed so as to anticipate that the participants were much more experienced in one of the treatments. The first experiment was performed on 35 student subjects who were asked to comprehend either Java or AspectJ implementation of the same system, and perform the corresponding comprehension tasks. Participants of both groups achieved a high rate of correct answers without a statistically significant difference between the groups. Nevertheless, the Java group significantly outperformed the AspectJ group with respect to the average completion time. In the second experiment, 24 student subjects were asked to implement (in a noninvasive way) two extension scenarios to the system that they had already known. Each subject evolved either the Java version using Java or the AspectJ version using AspectJ. We found out that a typical AspectJ programmer needs significantly fewer atomic changes to implement the change scenarios than a typical Java programmer, but we did not observe a significant difference in completion time. The overall result indicates that AspectJ has a different effect on two sub-characteristics of the evolvability: understandability and changeability. While AspectJ decreases the former, it improves one aspect of the latter.

2) ESEM4 [4]:Within-exp of algo: **Background.** The requirements and responsibilities assumed by software has increasingly rendered it to be large and complex. Testing to ensure that software meets all its requirements and is free from failures is a difficult and time-consuming task that necessitates the use of large test suites, containing many tests. Large test suites result in a corresponding increase in the size of the test code that sets up, exercises and verifies the tests. Time needed to compile and optimise the test code becomes prohibitive for large test code sizes.

Aims. In this paper we demonstrate for the first time optimisations to speedup compilation of test code. Reducing the compilation time of test code for large and complex systems will allow additional tests to be compiled and executed, while also enabling more frequent and rigorous testing.

Methods. We propose transformations that reduce the number of instructions in the test code, which in turn reduces compilation time. Using two well known compilers, GCC and Clang, we conduct empirical evaluations using subject programs from industry standard benchmarks and an industry provided program. We evaluate compilation speedup, execution time, scalability and correctness of the proposed test code transformation.

Results. Our approach resulted in significant compilation speedups in the range of 1.3× to 69×. Execution of the test code was just as fast with our transformation when compared to the original while also preserving correctness of execution. Finally, our experiments show that the gains in compilation time allow significantly more tests to be included in a single binary, improving scalability of test code compilation.

Conclusions. The proposed transformation results in faster test code compilation for all the programs in our experiment, with more significant speedups for larger case studies and larger numbers of tests. As systems get more complex requiring frequent and extensive testing, we believe our approach provides a safe and efficient means of compiling test code.

3) EASE11 [5]:Between-exp of algo: **Background:** Correctly localizing buggy files for bug reports together with their semantic and structural information is a crucial task, which would essentially improve the accuracy of bug localization techniques. **Aims:** To empirically evaluate and demonstrate the effects of both semantic and structural information in bug reports and source files on improving the performance of bug localization, we propose CNN Forest involving convolutional neural network and ensemble of random forests that have excellent performance in the tasks of semantic parsing and structural information extraction. **Method:** We first employ convolutional neural network with multiple filters and an ensemble of random forests with multi-grained scanning to extract semantic and structural features from the word vectors derived from bug reports and source files. And a subsequent cascade forest (a cascade of ensembles of random forests) is used to further extract deeper features and observe the correlated relationships between bug reports and source files. CNN Forest is then empirically evaluated over 10,754 bug reports extracted from AspectJ, Eclipse UI, JDT, SWT, and Tomcat projects. **Results:** The experiments empirically demonstrate the significance of including semantic and structural information in bug localization, and further show that the proposed CNN Forest achieves higher Mean Average Precision and Mean Reciprocal Rank measures than the best results of the four current state-of-the-art approaches (NPCNN, LR+WE, DNNLOC, and BugLocator). Conclusion: CNN Forest is capable of defining the correlated relationships between bug reports and source files, and we empirically show that semantic and structural information in bug reports and source files are crucial in improving bug localization

4) EMSE109 [3]:A quasi-exp of people: Interaction traces (ITs) are developers' logs collected while developers maintain or evolve software systems. Researchers use ITs to study developers'editing styles and recommend relevant program entities when developers perform changes on source code. However, when using ITs, they make assumptions that may not necessarily be true. This article assesses the extent to which researchers' assumptions are true and examines noise in ITs. It also investigates the impact of noise on previous studies. This article describes a quasi-experiment collecting both Mylyn ITs and video-screen captures while 15 participants performed four realistic software maintenance tasks. It assesses the noise in ITs by comparing Mylyn ITs and the ITs obtained from the video captures. It proposes an approach to correct noise and uses this approach to revisit previous studies. The collected data show that Mylyn ITs can miss, on average, about 6% of the time spent by participants performing tasks and can contain, on average, about 85% of false edit events, which are not real changes to the source code. The approach to correct noise reveals about 45% of misclassification of ITs. It can improve the precision and recall of recommendation systems from the literature by up to 56% and 62%, respectively. Mylyn ITs include noise that biases subsequentstudies and, thus, can prevent researchers from assisting developers effectively. They must be cleaned before use in studies and recommendation systems. The results on Mylyn ITs open new perspectives for the investigation of noise in ITs generated by other monitoring tools such as DFlow, FeedBag, and Mimec, and for future studies based on ITs.

5) EMSE41 [1]:A paper with title of exp method: t Domain-specific languages (DSLs) allow developers to write code at a higher level of abstraction compared with general-purpose languages (GPLs). Developers often use DSLs to reduce the complexity of GPLs. Our previous study found that developers performed program comprehension tasks more accurately and efficiently with DSLs than with corresponding APIs in GPLs. This study replicates our previous study to validate and extend the results when developers use IDEs to perform program comprehension tasks. We performed a dependent replication of a family of experiments. We made two specific changes to the original study: (1) participants used IDEs to perform the program comprehension tasks, to address a threat to validity in the original experiment and (2) each participant performed program comprehension tasks on either DSLs or GPLs, not both as in the original experiment. The results of the replication are consistent with and expanded the results of the original study. Developers are significantly more effective and efficient in tool-based program comprehension when using a DSL than when using a corresponding API in a GPL. The results indicate that, where a DSL is available, developers will perform program comprehension better using the DSL than when using the corresponding API in a GPL.

6) ESEM21 [6]:A paper with title of exp method: **Background** Xu et al. used a deep neural network (DNN) technique to classify the degree of relatedness between two knowledge units (question-answer threads) on Stack Overflow. More recently, extending Xu et al.' s work, Fu and Menzies proposed a simpler classification technique based on a fine-tuned support vector machine (SVM) that achieves similar performance but in a much shorter time. Thus, they suggested that researchers need to compare their sophisticated methods against simpler alternatives. **Aim** The aim of this work is to replicate the previous studies and further investigate the validity of Fu and Menzies' claim by evaluating the DNN- and SVM-based approaches on a larger dataset. We also compare the effectiveness of these two approaches against SimBow, a lightweight SVM-based method that was previously used for general community question-answering. **Method** We (1) collect a large dataset containing knowledge units from Stack Overflow, (2) show the value of the new dataset addressing shortcomings of the original one, (3) re-evaluate both the DNNand SVM-based approaches on the new dataset, and (4) compare the performance of the two approaches against that of SimBow. **Results** We find that: (1) there are several limitations in the original dataset used in the previous studies, (2) effectiveness of both Xu et al.' s and Fu and Menzies' approaches (as measured using F1-score) drop sharply on the new dataset, (3) similar to the previous finding, performance of SVM-based approaches (Fu and Menzies' approach and SimBow) are slightly better than the DNN-based approach, (4) contrary to the previous findings, Fu and Menzies' approach runs much slower than DNN-based approach on the larger dataset – its runtime grows sharply with increase in dataset size, and (5) SimBow outperforms both X

II. Information Collected of Papers

A. EMSE65

To ensure an empirical study on the impact of AspectJ on software evolvability, the article give out following experiments to measure the impact, which act as Between-subject design: Comparisons are made between units that are treated with different approaches .

- Experimental unit:
  - Languages: aspect-oriented programming , java
  - For software understandability: 35 students from the OOSD course held in the 4th year of study in Computer Science
  - For software changeability: 24 students with the same characteristics as those in the comprehension experiment
- Treatment:
  - For software understandability: One group: Java source code of Telecom, another group: AspectJ source code of Telecom
  - For software changeability: One group: Java source code of Telecom, another group: AspectJ source code of Telecom
- Response:
  - For software understandability: the time needed to comprehend the system, the comprehension accuracy
  - For software changeability: the time (in minutes) that the subjects spent to implement the change scenarios, the volume of changes made to the source code

  The reason we chose this:
  One reason is in which that Experiment Execution First, the subjects were introduced to the overall format of the experiment. they were randomly assigned to two groups (G_OO, G_AO). Group G_OO had 18 members, and group G_AO had 17 members. This definitely shows that experiment is random as subjects in assignment. Another one factor is variable responds which is abundant in quantities as well as statistics. They are shown in Completion Time and changes measure in the part of 7 Also, we find that the experiment is designed as two groups when measure the impact of aop, which is typtically Between-subject design

B. ESEM4

- Experimental unit: Test cases from industry standard benchmark suites, EEMBC and SPEC, and plus an industrial application, ComputeCPP, from Codeplay.

- Treatment: Applying the transformation algorithm to the test case or not
- Response: The compilation speedup, execution time, scalability and correctness of the proposed test code transformation.

This one is a typical within-subject design: all the responses are observed before and after the transformation, which is the treatment of this experiment. Moreover, this is also an experiment conducted on an algorithm as introduced before: this paper is typical and obvious to be classified to the group which mainly discussees about the performance of algorithm on computer.

C. EASE11

In this paper, the authors propose a new approach called CNN_Forest to improve current bug localization techniques.

Like ESEM04, the experiment conducted in the paper focuses on empirical evidences for the effectiveness of the newly devised method. Therefore, We also classify this paper into the algorithm type experiment. Unlike ESEM04, in the treatment part of this experiment, the authors primary goal is not simply to confirm the effectiveness of the method, but rather to confirm the superiority of the new method over pre-existed methods. This distinction can be best summarized with the help of the concept “within-subject” and “between-subjects” . The similarities and distinction between the two experiments make them a suitable pair for comparison, thus we purposefully choose both these experiments to elaborate on those points.

- Experimental unit: The authors choose five open-source Java projects(AspectJ, Eclipse UI, JDT, SWT and Tomcat). Each one of them with a specific treatment is an experiment unit.
- Treatment: The treatment of this experiment is the different techniques to utilize bug report files and source files in each project to localize bugs. Different treatment falls into either of the intrinsic and extrinsic categories. The intrinsic category compare CNN_Forest with CNN_CNN and Forest_Forest, which are both devised by the authors. The extrinsic category compare CNN_Forest with techniques proposed in previous studies, including NP_CNN, LR+WE, DNNLOC, BugLocator
- Response: The experiment uses two indicators for performance. MRR is the mean of the accumulations of the inverse of the ranks of the first correctly-located buggy file for each bug. This indicator is based on the rank of first correctly-located buggy file. MAP is the mean of average precision values for all Q bug reports. This indicator takes into consideration all the correctly-located bug files.
- result: Both the intrinsic and extrinsic experiment reveals that CNN_Forest ranks first for all but one indicator in one project.

Compared with ESEM04, As both experiments are algorithm type, they share similarities in the following ways:

- Both of the experiment units come from static, duplicable data.
- Both of the treatments are conducted automatically by computer, without human variance.
- Both of the responses are measured without human factors. The major difference between them:
- ESEM04 is within-subject. It focuses on the difference between pre-test and post-test performance EASE11 is between-subject. It compares different performances of groups applying different methods. In light of EASE11, it calls for new consideration whether ESEM04 is within-subject or not. Due to the duplicable nature of digital data and the lack of human involvement, ESEM04 could also be labeled between-subject, if we consider the untreated data as a control group. But the fact that digital data never undergoes any implicit or unforeseeable changes as their human counterparts, and considering the purpose of EASE04, it is still better suited to be assigned as within-subject.

D. EMSE109

This paper deals with the problem of whether there are noises in trace informations. When the researchers want to collect data to observe use behaviors, noises can be misleading the researchers.

- Experimental unit:15 peoples' maintenance work on 4 programs
- Treatment: Mylyn program collected data vs. screen-recorder collected and manually analyzied data
- Response: Noise at trace-level and event level.

We can see something interesting that this paper clearly claim that they did not do a random assignment in that they want to manually assign the people with different profiles on same systems to minimise threat to internal validity.

Hence in this case we can come to the conclusion that **sometimes quasi-experiment may be better for the experiment. Whether to use quasi-experiment depends on the research question.**

E. EMSE41

This paper replicates and extends their own study on the efficiency of programmers doing program comprehension tasks using Domain Specific Languages (DSL) or General Programming Languages (GPL).

The participants were randomly assigned, did program comprehension tasks using different languages in different domains, and measurements like efficiency, correctness and time were compared.

The paper proves that developers are significantly more effective and efficient in program comprehension with tools when using DSL compared to using GPL.

- Experimental unit: different groups of students with similar backgrounds
- Treatment: Different groups use different language types (DSL or GPL), work in different domains (FD, GD, GUI), have different questions (Learn, Understand, Evolve) and with different experiences. The first two are most important.
- Response: Correctness, Time, Efficiency, Simplicity of Use, Questionnaire Complexity between current and previous experiments

This is a between-subject design: comparisons are made between the outcomes of each groups, and there are no blind designs, which indicate that it is a quasi-experiment.Both participants and researchers know everything in the experiment, but participants are indeed randomly assigned.

F. ESEM21

This paper replicates previous studies about classifying the degree of relatedness between two knowledge units (question-answer threads) on Stack Overflow using different techniques proposed prior to this paper by other researchers (deep neural network, DNN by Xu et al. and Support Vector Machine, SVM by Fu and Menzies), and tries to investigate the validity and reproducibility of Fu and Menzies' claim that SVM could achieve similar performance but in a shorter time. In this study, the researchers did the following steps:

- collect a new large knowledge units dataset from Stack Overflow and compared the new dataset with the original one
- re-evaluate both approaches on the new dataset
- compare their performances again SimBow's.

Finally, the study came to the following results:

- the original dataset has limitations
- both DNN and SVM are way less effective on the new dataset;
- SVM approaches are slightly better, but much slower in larger dataset than DNN approaches;
- SimBow's outperforms both SVM and DNN approaches

Comparisons about performances between different algorithms are usually based on experiments, during which variables are controlled and the performances of algorithms can be compared in a sensible way. This article is not an exception. This paper is a typical **algorithm performance comparison** article, which are **explicitly indicated** in its title: Deep Learning vs SVM. A deep investigation into the content of the paper also proves its experimental nature, since it includes hypothesis, independent variables that are manipulated and dependent variables that are measured.

- Experimental unit: a new large knowledge units dataset from Stack Overflow
- Treatment: Evaluate the relatedness on the units with DNN based, SVM based approaches and SimBow
- Response: Precision, Recall and F1-Score, the measures that are used in previous works

This is a between subject experiment, comparisons are made between units that are treated with different approaches, yet it is a quasi-experiment in that participants understand what group uses what approaches and there are no randomized group allocation.

References

[1] Kosar, T., Gaberc, S., Carver, J. C., and Mernik, M. Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. *Empirical Software Engineering 23* (2017), 2734–2763.

[2] Przybyłek, A. An empirical study on the impact of aspectj on software evolvability. *Empirical Software Engineering 23*, 4 (Aug 2018), 2018–2050.

[3] Soh, Z., Khomh, F., Guéhéneuc, Y.-G., and Antoniol, G. Noise in mylyn interaction traces and its impact on developers and recommendation systems. *Empirical Software Engineering 23*, 2 (Apr 2018), 645–692.

[4] Stratis, P., Yaneva, V., and Rajan, A. Assessing the effect of data transformations on test suite compilation. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 4:1–4:10.

[5] Xiao, Y., Keung, J., Mi, Q., and Bennin, K. E. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 101–111.

[6] Xu, B., Shirani, A., Lo, D., and Alipour, M. A. Prediction of relatedness in stack overflow: Deep learning vs. svm: A reproducibility study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 21:1–21:10.

Report No.03: Empirical Study Comparison: Case Study
Year: 2018
Presenter: Ruihua Wang, Kangqiao Zhou, Xiangyu Zhang
Writer of paper: Yuqing Yang, Ruihua Wang, Junda Chen,
Kangqiao Zhou, Xiangyu Zhang

A Systematic Study on Empirical Software Engineering Research in 2018: Case Study

Yuqing Yang[*]
161250179
Junda Chen[*]
161250010
Ao Zhang
161250193

Aiwei Liu
161250071
Chen Lei[*]
161250054
Kangqiao Zhou[*†]
161070096

Yuanxiang Lian
1612501065
Xiangyu Zhang[*†]
161250202
Ruihua Wang[*†]
161250143

[*]**Writer of paper**
[†]**Presenter**

***Abstract*—This paper is the second periodical report of the assignments from Empirical Software Engineering(henceforth referred to as ESE) course in 2019 spring.**

In this paper, we made an analysis through the paper published in 2018 of thethree main ESE conferences and journals, with a focus on researches that involve case study as their main research methods. We identify and compare the methods involved.

I. Paper Selection

A. Overview

We choose six different types of papers, three for presentation, and another three for report.

For presentation, the first paper we choose is a standard one, which includes, according to our discussion, a standard form of what a case study looks like. Moreover, there is a 'case study' in the title, which more or less suggests that the paper used a case study method in their research. The other two papers are rather interesting. One of the two involves merely theoretically replication upon the research topic, and is a rather rare one, which argues against a previously accepted idea. The last one of the three, is characterized in its quantitative benchmark for evaluation.

We include another three papers in our report. One is a complex research project with multiple approaches embedded. Another one is also multiple-approach-related, more surprisingly, the paper adopts ethnography methods in their research. The last one is another interesting one which only uses interview for their method.

B. Other Procedures

We also made error correction on previous classification data. During the process, 1 research was found to be falsefully categorized.

C. Paper Abstract Info

We propose six experiments as follows:

1) EMSE118 [3]:A standard case study: Despite growing interest of Open Innovation (OI) in Software Engineering (SE), little is known about what triggers software organizations to adopt it and how this affects SE practices. OI can be realized in numerous of ways, including Open Source Software (OSS) involvement. Outcomes from OI are not restricted to product innovation but also include process innovation, e.g. improved SE practices and methods. This study explores the involvement of a software organization (Sony Mobile) in OSS communities from an OI perspective and what SE practices (requirements engineering and testing) have been adapted in relation to OI. It also highlights the innovative outcomes resulting from OI. An exploratory embedded case study investigates how Sony Mobile use and contribute to Jenkins and Gerrit; the two central OSS tools in their continuous integration tool chain. Quantitative analysis was performed on change log data from source code repositories in order to identify the top contributors and triangulated with the results from five semi-structured interviews to explore the nature of the commits. The findings of the case study include five major themes:

- The process of opening up towards the tool communities correlates in time with a general adoption of OSS in the organization.
- Assets not seen as competitive advantage nor a source of revenue are made open to OSS communities, and gradually, the organization turns more open.
- The requirements engineering process towards the community is informal and based on engagement.
- The need for systematic and automated testing is still in its infancy, but the needs are identified.
- The innovation outcomes included free features and maintenance, and were believed to increase speed and quality in development. Adopting OI was a result of a paradigm shift of moving from Windows to Linux. This shift enabled Sony Mobile to utilize the Jenkins and Gerrit communities to make their internal development process better for its software developers and

testers.

2) EASE8 [1]:Theoretical replication: Multitasking has always been an inherent part of so ware development and is known as the primary source of interruptions due to task switching in so ware development teams. Developing so - ware involves a mix of analytical and creative work, and requires a signi cant load on brain functions, such as working memory and decision making. us, task switching in the context of so - ware development imposes a cognitive load that causes so ware developers to lose focus and concentration while working thereby taking a toll on productivity. To investigate the disruptiveness of task switching and interruptions in so ware development projects, and to understand the reasons for and perceptions of the disruptiveness of task switching we used a mixed-methods approach including a longitudinal data analysis on 4,910 recorded tasks of 17 professional so ware developers, and a survey of 132 so ware developers. We found that, compared to task-speci c factors (e.g. priority, level, and temporal stage), contextual factors such as interruption type (e.g. self/external), time of day, and task type and context are a more potent determinant of task switching disruptiveness in so ware development tasks. Furthermore, while most survey respondents believe external interruptions are more disruptive than self-interruptions, the results of our retrospective analysis reveals otherwise. We found that self-interruptions (i.e. voluntary task switchings) are more disruptive than external interruptions and have a negative e ect on the performance of the interrupted tasks. Finally, we use the results of both studies to provide a set of comparative vulnerability and interaction pa erns which can be used as a mean to guide decision-making and forecasting the consequences of task switching in so ware development teams

3) ESEM13 [5]:Quantitative: **Background:** Limiting the amount of Work-In-Progress (WIP) is considered a fundamental principle in Kanban software development. However, no published studies from real cases exist that indicate what an optimal WIP limit should be. **Aims:** The primary aim is to study the effect of WIP on the performance of a Kanban team. The secondary aim is to illustrate methodological challenges when attempting to identify an optimal or appropriate WIP limit. **Method:** A quantitative case study was conducted in a software company that provided information about more than 8,000 work items developed over four years by five teams. Relationships between WIP, lead time and productivity were analyzed. **Results:** WIP correlates with lead time; that is, lower WIP indicates shorter lead times, which is consistent with claims in the literature. However, WIP also correlates with productivity, which is inconsistent with the claim in the literature that a low WIP (still above a certain threshold) will improve productivity. The collected data set did not include sufficient information to measure aspects of quality. There are several threats to the way productivity was measured. **Conclusions:** Indicating an optimal WIP limit is difficult in the studied company because a changing WIP gives contrasting results on different team performance variables. Because the effect of WIP has not been quantitatively examined before, this study clearly needs to be replicated in other contexts. In addition, studies that include other team performance variables, such as various aspects of quality, are requested. The methodologica

4) EMSE47 [2]:Multiple embedded: Software developers insert logging statements in their source code to record important runtime information; such logged information is valuable for understanding system usage in production and debugging system failures. However, providing proper logging statements remains a manual and challenging task. Missing an important logging statement may increase the difficulty of debugging a system failure, while too much logging can increase system overhead and mask the truly important information. Intuitively, the actual functionality of a software component is one of the major drivers behind logging decisions. For instance, a method maintaining network communications is more likely to be logged than getters and setters. In this paper, we used automatically-computed topics of a code snippet to approximate the functionality of a code snippet. We studied the relationship between the topics of a code snippet and the likelihood of a code snippet being logged (i.e., to contain a logging statement). Our driving intuition is that certain topics in the source code are more likely to be logged than others. To validate our intuition, we conducted a case study on six open source systems, and we found that i) there exists a small number of "log-intensive" topics that are more likely to be logged than other topics; ii) each pair of the studied systems share 12% to 62% common topics, and the likelihood of logging such common topics has a statistically significant correlation of 0.35 to 0.62 among all the studied systems; and iii) our topic-based metrics help explain the likelihood of a code snippet being logged, providing an improvement of 3% to 13% on AUC and 6% to 16% on balanced accuracy over a set of baseline metrics that capture the structural information of a code snippet. Our findings highlight that topics contain valuable information that can help guide and drive developers' logging decisions.

5) ESEM34 [6]:Multiple embedded with ethnography: **Background:** Testing is one of the main methods for quality assurance in the development of embedded software, as well as in software engineering in general. Consequently, test results (and how they are reported and visualized) may substantially influence business decisions in software-intensive organizations. **Aims:** This case study examines the role of test results from automated nightly software testing and the visualizations for decision making they enable at an embedded systems company in Sweden. In particular, we want to identify the use of the visualizations for supporting decisions from three aspects: in daily work, at feature branch merge, and at release time. **Method:** We conducted an embedded case study with multiple units of analysis by conducting interviews, questionnaires, using archival data and participant observations. **Results:** Several visualizations and reports built on top of the test results database are utilized in supporting daily work, merging a feature branch to the master and at release time. Some important visualizations are: lists of failing test cases, easy access to logs, and heatmap trend plots. The industrial practitioners perceived the visualizations and reporting as valuable, however they also mentioned several areas of improvement such as better ways of visualizing test coverage in a functional area as well as better navigation between different views. **Conclusions:** We conclude that visualizations of test results are a vital decision making tool for a variety of roles and tasks in embedded software development, however the visualizations need to be

6) EASE7 [4]: Interview: DevOps is a set of principles and practices to improve collaboration between development and IT Operations. Against the backdrop of the growing adoption of DevOps in a variety of software development domains, this paper describes empirical research into factors influencing its implementation. It presents findings of an in-depth exploratory case study that explored DevOps implementation in a New Zealand product development organisation. The study involved interviewing six experienced software engineers who continuously monitored and reflected on the gradual implementation of DevOps principles and practices. For this case study the use of DevOps practices led to significant benefits, including increase in deployment frequency from about 30 releases a month to an average of 120 releases per month, as well as improved natural communication and collaboration between IT development and operations personnel. We found that the support of a number of technological enablers, such as implementing an automation pipeline and cross functional organisational structures, were critical to delivering the expected benefits of DevOps.

II. Information Collected from Papers

A. EMSE118

This is a typical case study using multiple kinds of methods, with ‘case study’ included in the title. Therefore, we regard this paper as a standard one case study example for further categorizing.

Case study design:

- Type: single case, embedded design.
- Triangulation:
  - Data(source) triangulation: true(Jenkins&Gerrit)
  - Observer triangulation: false
  - Methodological triangulation: true(qualitative & quantitative)
  - Theory triangulation: true

Research questions:

There is only one case in this study, all the questions are asked for the individual case.

TABLE I
Research questions

RQ1: How and to what extent is Sony Mobile involved in the communities of Jenkins and Gerrit?	To characterize Sony Mobile’s involvement and identify potential interviewees.
RQ2: What is the motivation for Sony Mobile to adopt OI?	To explore the transition from a closed innovation process to an OI process.
RQ3: How does Sony Mobile take a decision to make a project or feature open source?	To investigate what factors affect the decision process when determining whether or not Sony Mobile should contribute functionality.
RQ4: What are the innovation outcomes as a result of OI participation?	To explore the vested interest of Sony Mobile as they moved from a closed innovation model to an OI model.
RQ5: How do the requirements engineering and testing processes interplay with the OI adoption?	To investigate the requirements engineering and testing processes and how they deal with the special complexities and challenges involved due to OI.

Case study protocol:

- Preliminary investigation of Jenkins and Gerrit repositories.
- Mine the identified project repositories.
- Extract the change log data from the source code repositories.
- Analyze the change log data (i.e. stakeholders, commits etc).
- Summarize the findings from the change log data to answer RQ1.
- Prepare and conduct semi-structured interviews to answer RQ2–RQ5.
- Synthesize data.
- Answer the research questions RQ1–RQ5.

Data Collection Source: From an organization, or about an organization.

Collecting data:

- Method: direct method
- Data source: archival records or interviews

Three candidates were identified and contacted by e-mail ,Interviewees 4 and 5 were proposed during the initial three interviews.

Analyzing data:

The quantitative analysis part: basically using descriptive statistics.

The qualitative analysis part: Theory generation/theory confirmation

- Transcribe the interviewed data from the five interviewees (see Table 3).
- Identify and define five distinct themes in the data (see Table 7).
- Classify the interview statements based on the defined themes.
- Summarize the findings and answers to the RQs.

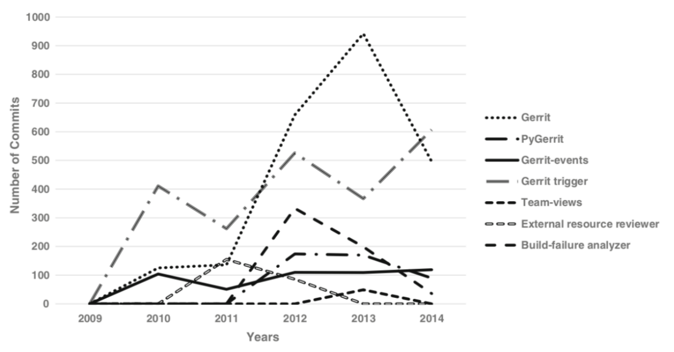


Fig. 1. Quantitative evaluation

Validity: Validity: Construct validity/Internal validity/External validity

Reporting case study:

- Linear-analytic: standard reporting structure
- Using a strict step-sequence

B. EASE8

Study design:

- multiple cases studies
- embedded case studies
- theoretical replications
- **RQ1- Task-speci c Vulnerability:** How do various interruption characteristics impact the vulnerability of programming, testing, architecture design, UI design, and deployment tasks?
- **RQ2- Comparative Vulnerability:** Which types of development tasks are more vulnerable to task switching/interruptions?
- **RQ3- Two-way Impact:** How does the interaction between various interruption characteristics (iv1—8) influence the vulnerability of development tasks to interruptions?

Data collection: The research mainly used survey and records for data collection. They integrated both direct and indirect methods, including a longitudinal data analysis on 4910 recorded tasks of 17 professional software developers, and a user survey of 132 practitioners to complement the quantitative results with developer perception on task switching and interruptions. **Data analysis:** Various factors are included in the process. **Conclusion & validation:**

- Threats solved:
  - Different levels and various countries of survey objects
  - Various projects, different business domains, different levels of exp.
  - Cohen's Kappa statistic: Kappa value=0.87
- Proposition
  - Measure and characterizing the cost of task switching and interruptions
  - Can be a productive task switching

C. ESEM13

This is an Empirical Study of WIP in Kanban Teams:

- WIP: work-in-process
- kanban
- lead time
- productivity

Research question:

- **RQ1:** Which WIP optimizes team performance?
- **RQ2:** What are the challenges of quantifying the effect of WIP limit on the performance of Kanban teams?

Case study design:

- simple case study(one company)
- embedded case study (unit of analysis: lead time, productivity)
- this is not a replication.

Data collection: This collection approach involves physical artifacts: 8,505 work items that were developed by the five teams over a period of 3.5 years (from the second half of 2010, when Kanban was introduced, until the end of 2013).

Results:

- The data shows that an increasing WIP (increasing from 2.8 in 2010 and 3.1 in 2011 to 4.1 in 2012 and 4.0 in 2013) corresponds to a longer lead time (increasing from 7.8 days in 2010 to 9.3 days in 2013).
- the high correlation between productivity and WIP is clearly statistically significant (rho = 0.71, p l.t. 0.01).

Validity:

- construct validity: used code churn as the measurements of the size of a work item?
- internal validity: variation in work item size is controlled?
- external validity: other company?

D. EMSE47

This is an example of Multiple-case&Embedded-Design

Research design: This is a research with multiple case, and embedded design.

Research question:

- **RQ1:** Which topics are more likely to be logged?
- **RQ2:** Are common topics logged similarly across different systems?
- **RQ3:** Can topics provide additional explanatory power for the likelihood of a code snippet being logged?

Case study steps:

- studied systems
- data extraction
- source code preprocessing and LDA
- explain the results

Sources for Data Collection:

- Method: direct method

- Sources: documentation/archival records (from github)

Analyzing data:

- Method: Quantitative analysis
- Descriptive statistics
- Formalism:Editing approaches/Template approaches
- Validity: Construct validity+Internal validity+External validity

Reporting case study: Used comparative way, comparing alternative cases.

E. ESEM34

This is a unique research because it adopts ethnography method for case study. It uses embedded method on single case study, and integrated multiple directed and qualitative methods, with quantitative methods for augmenting the research.

The authors want to know how the test results(and visualized data and reports based on that) influences the interest-related participants. Then they did a case study on a embedded system company in Swiss, using both interview and participant obserations. In section 3.2 the paper illustrated that the researchers participated in two meetings and made records in both.

It is:

- a single case study: on a company called Westermo Research and Development AB
- an embedded case study: units of analysis include interviews & questionnaires to different roles in the company and participant observations (for qualitative research) as well as archival data (for quantitive research).
- not a replication

Data triangulation: people and archival data

Methodological triangulation: qualitative and quantitive

Observer triangulation: all interviews were conducted by multiple researchers

Case study protocol:

- objective: identify the role and the use of test results and their visualizations on supporting decision making.
- the case and observations and selection strategy: data (interviews, questionnaires, observations and archival data) of a company called Westermo Research and Development AB
- theory: test results and their reporting and visualizations, as a main method to assure software qualitym, may influence business decisions.

Research questions:

- **RQ1:** How are visualizations used for making decisions in daily work, at merging a code branch to a master branch and at release time?
- **RQ2:** How do industrial practitioners perceive the value of visualizations, both in general and with respect to decision making in daily work, at merging a code branch to master branch and at release time?

Research methods:

- direct methods: interview and questionnaire
- indirect methods: observation
- independent analysis: using archival data
- semi-structured interview
- funnel interview: background to specific
- observing meetings

Data collection:data are visualized to help authors get familiarized with context and understand interview/questionnaire data.

Theory confirmation and generations:

- visualization do help developers in decision making.
- the theory of how visualization helps developers in decision making are generated.

Validity:

- Construction: limit of scope.
- External validity: the study was conducted at a specific company.
- The linear-analytic report structure was adopted in the report process.

F. EASE7

This is a case study of the use of DevOps. In this case study, authors only take interviews as their data collecting methods.

Case study design:

- single case study: the paper explored DevOps implementation in a product development organisation in New Zealand.
- embedded case study, the unit of analysis is techonological and capability enabler.

Data collection:

- **protocol:** the paper used Smeds model for interview protocol.
- the data collection involved a series of six in-depth semi-structured one-on-one interviews.

The paper adopted direct methods in interviews, which was semi-structured: the data collection involved six one-on-one interviews.

Data analysis: The paper starts from qualitative analysis and ends in a theory confirmation.

Reporting case study: The paper used a linear-analytic way to report.

III. Data correction

During the process there is one paper which was wrongly assigned as case study. The original reason for categorizing that as case study was because there was case study in the title, however it was merely a method used in the research process. This phonomenon indicates that it might be not appropriate to categorize merely by recognizing key words in the title. Hence the lesson was learned for us.

TABLE II
Result Tabular

ResearchID	Research Design	Data Collection	Data Source	Data Analysis	Result Report
EMSE118	Single Embedded	Direct	Archival/Semi-struct.	Quantitative/Qualitative	Linear-analytic
EASE008	Multiple Embedded	Direct/Indirect	Records/survey	Quantitative	
ESEM013	Single Embedded	Direct	Physical artifacts	Quantitative	
EMSE047	Multiple Embedded	Direct	Documentation/Archival	Quantitative/Descriptive	Comparative
ESEM034	Single Embedded	Direct/Indirect	Interview/Observation	Discriptive	Linear-analytic
EASE007	Single Embedded	Direct	Semi-struct.	Qualitative	Linear-analytic

IV. Conclusion and result aggregation

In this study, we make investigations into over 74 papers identified as case study, and choose 6 out of them in this paper. The result is presented as above in TABLE II.

References

[1] Abad, Z. S. H., Karras, O., Schneider, K., Barker, K., and Bauer, M. Task interruption in software development projects: What makes some interruptions more disruptive than others? In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 122–132.

[2] Li, H., Chen, T.-H., Shang, W., and Hassan, A. E. Studying software logging using topic models. *Empirical Software Engineering 23* (2018), 2655–2694.

[3] Munir, H., Linåker, J., Wnuk, K., Runeson, P., and Regnell, B. Open innovation using open source tools: a case study at sony mobile. *Empirical Software Engineering 23*, 1 (Feb 2018), 186–223.

[4] Senapathi, M., Buchan, J., and Osman, H. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (2018), ACM, pp. 57–67.

[5] Sjøberg, D. I. K. An empirical study of wip in kanban teams. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 13:1–13:8.

[6] Strandberg, P. E., Afzal, W., and Sundmark, D. Decision making and visualizations based on test results. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM '18, ACM, pp. 34:1–34:10.

A Systematic Study on Empirical Software Engineering Research in 2018: Survey

Yuqing Yang[*]
161250179
Junda Chen[*†]
161250010
Ao Zhang[*†]
161250193

Aiwei Liu
161250071
Chen Lei[*†]
161250054
Kangqiao Zhou
161070096
[*]**Writer of paper**
[†]**Presenter**

Yuanxiang Lian
1612501065
Xiangyu Zhang
161250202
Ruihua Wang
161250143

Abstract—**This paper is the third periodical report of the assignments from Empirical Software Engineering(henceforth referred to as ESE) course in 2019 spring.**

In this paper, we made an analysis through the paper published in 2018 of thethree main ESE conferences and journals, with a focus on researches that involve survey as their main research methods. We identify and compare the methods and properies involved.

I. Paper Selection

A. Overview

We choose six different types of papers, three for presentation, and another three for report.

We deveoped three identifying criterias: the role survey techniques serves, survey method included, and purposes of the survey.

The survey may serve exclulsively as the main body of an empirical study, or as a part whithin other methods, e.g. case study.

Survey method includes interview and questionaire. The research may use certain or both.

The purposes of a survey usually falls into three main categories:

- To evaluate the **rate** of certain characteristics related to the research question.
- To assess the **severity** of certain characteristics or conditions.
- To find out the **factors** related to the characteristics or conditions.

For the presentations, we include three papers of different elements. ESEM28, as main method, with questionaire, to calculate the rate; ENSE105, as main method, with interview and questionaire, to assess the severity; and EMSE31, as part of case study, uses interview to find out the factors.

We include another three papers in our report. One is a complex research project with multiple approaches embedded. Another one is also multiple-approach-related, more surprisingly, the paper adopts ethnography methods in their research. The last one is another interesting one which only uses interview for their method.

B. Other Procedures

We also made error correction on previous classification data. During the process, 1 research was found to be falsefully categorized.

C. Paper Abstract Info

We propose six experiments as follows:

1) ESEM28 [4]: Background: Researchers oftentimes measure quality metrics only in the changed files when analyzing software evolution over commit-history. This approach is not suitable for compilation and using program analysis techniques that require byte-code. At the same time, compiling the whole software not only is costly but may also leave us with many uncompilable and unanalyzed revisions. Aims: We intend to demonstrate if analyzing changes in a module results in achieving a high compilation ratio and a better understanding of software quality evolution. Method: We conduct a large-scale multi-perspective empirical study on 37838 distinct revisions of the core module of 68 systems across Apache, Google, and Netflix to assess their compilability and identify when the software is uncompilable as a result of a developer's fault. We study the characteristics of uncompilable revisions and analyze compilable ones to understand the impact of developers on software quality. Results: We achieve high compilation ratios: 98.4% for Apache, 99.0% for Google, and 94.3% for Netflix. We identify 303 sequences of uncompile commits and create a model to predict uncompilability based on commit metadata with an F1-score of 0.89 and an AUC of 0.96. We identify statistical differences between the impact of affiliated and external developers of organizations. Conclusions: Focusing on a module results in a more complete and accurate software evolution analysis, reduces the cost and complexity, and facilitates manual inspection.

2) EMSE105 [5]: The release frequency of software projects has increased in recent years. Adopters of so-called rapid releases—short release cycles, often on the order of weeks, days, or even hours—claim that they can deliver fixed issues (i.e., implemented bug fixes and new features) to users more quickly. However, there is little empirical evidence to support these claims. In fact, our prior work shows that code integration phases may introduce delays for rapidly releasing projects—98% of the fixed issues in the rapidly releasing Firefox project had their integration delayed by at least one release. To better understand the impact that rapid release cycles have on the integration delay of fixed issues, we perform a comparative study of traditional and rapid release cycles. Our comparative study has two parts: (i) a quantitative empirical analysis of 72,114 issue reports from the Firefox project, and a (ii) qualitative study involving 37 participants, who are contributors of the Firefox, Eclipse, and ArgoUML projects. Our study is divided into quantitative and qualitative analyses. Quantitative analyses reveal that, surprisingly, fixed issues take a median of 54% (57 days) longer to be integrated in rapid Firefox releases than the traditional ones. To investigate the factors that are related to integration delay in traditional and rapid release cycles, we train regression models that model whether a fixed issue will have its integration delayed or not. Our explanatory models achieve good discrimination (ROC areas of 0.80–0.84) and calibration scores (Brier scores of 0.05–0.16) for rapid and traditional releases. Our explanatory models indicate that (i) traditional releases prioritize the integration of backlog issues, while (ii) rapid releases prioritize issues that were fixed in the current release cycle. Complementary qualitative analyses reveal that participants' perception about integration delay is tightly related to activities that involve decision making, risk management, and team collaboration. Moreover, the allure of shipping fixed issues faster is a main motivator for adopting rapid release cycles among participants (although this motivation is not supported by our quantitative analysis). Furthermore, to explain why traditional releases deliver fixed issues more quickly, our participants point out the rush for integration in traditional releases and the increased time that is invested on polishing issues in rapid releases. Our results suggest that rapid release cycles may not be a silver bullet for the rapid delivery of new content to users. Instead, our results suggest that the benefits of rapid releases are increased software stability and user feedback.

3) EMSE31 [3]: Knowing whether a software feature will be completed in its planned iteration can help with release planning decisions. However, existing research has focused on predictions of only low-level software tasks, like bug fixes. In this paper, we describe a mixed-method empirical study on three large IBM projects. We investigated the types of iteration changes that occur. We show that up to 54% of high-level requirements do not make their planned iteration. Requirements are most often pushed out to the next iteration, but high-level requirements are also commonly moved to the next minor or major release or returned to the product or release backlog. We developed and evaluated a model that uses machine learning to predict if a high-level requirement will be completed within its planned iteration. The model includes 29 features that were engineered based on prior work, interviews with IBM developers, and domain knowledge. Predictions were made at four different stages of the requirement lifetime. Our model is able to achieve up to 100% precision. We ranked the importance of our model features and found that some features are highly dependent on project and prediction stage. However, some features (e.g., the time remaining in the iteration and creator of the requirement) emerge as important across all projects and stages. We conclude with a discussion on future research directions.

4) EMSE124 [1]: Architecture Consistency (AC) aims to align implemented systems with their intended architectures. Several AC approaches and tools have been proposed and empirically evaluated, suggesting favourable results. In this paper, we empirically examine the state of practice with respect to Architecture Consistency, through interviews with nineteen experienced software engineers. Our goal is to identify 1) any practises that the companies these architects work for, currently undertake to achieve AC; 2) any barriers to undertaking explicit AC approaches in these companies; 3) software development situations where practitioners perceive AC approaches would be useful, and 4) AC tool needs, as perceived by practitioners. We also assess current commercial AC tool offerings in terms of these perceived needs. The study reveals that many practitioners apply informal AC approaches as there are barriers for adopting more formal and explicit approaches. These barriers are: 1) Difficulty in quantifying architectural inconsistency effects, and thus justifying the allocation of resources to fix them to senior management, 2) The near invisibility of architectural inconsistency to customers, 3) Practitioners' reluctance towards fixing architectural inconsistencies, and 4) Practitioners perception that huge effort is required to map the system to the architecture when using more formal AC approaches and tools. Practitioners still believe that AC would be useful in supporting several of the software development activities such as auditing, evolution and ensuring quality attributes. After reviewing several commercial tools, we posit that AC tool vendors need to work on their ability to support analysis of systems made up of different technologies, that AC tools need to enhance their capabilities with respect to artefacts such as services and meta-data, and to focus more on non-maintainability architectural concerns.

5) EMSE73 [2]: Previous studies have shown the negative effects that low-quality code can have on maintainability proxies, such as code change- and defect-proneness. One of the symptoms of low-quality code are code smells, defined as sub-optimal implementation choices. While this definition is quite general and seems to suggest a wide spectrum of smells that can affect software systems, the research literature mostly focuses on the set of smells defined in the catalog by Fowler and Beck, reporting design issues that can potentially affect any kind of system, regardless of their architecture (e.g., Complex Class). However, systems adopting a specific architecture (e.g., the Model-View-Controller pattern) can be affected by other types of poor practices that only manifest themselves in the chosen architecture. We present a catalog of six smells tailored to MVC applications and defined by surveying/interviewing 53 MVC developers. We validate our catalog from different perspectives. First, we assess the relationship between the defined smells and the code change- and defect-proneness. Second, we investigate when these smells are introduced and how long they survive. Third, we survey 21 developers to verify their perception of the defined smells. Fourth, since our catalog has been mainly defined together with developers adopting a specific Java framework in their MVC applications (e.g., Spring), we interview four expert developers working with different technologies for the implementation of their MVC applications to check the generalizability of our catalog. The achieved results show that the defined Web MVC smells (i) more often than not, have more chances of being subject to changes and defects, (ii) are mostly introduced when the affected file (i.e., the file containing the smell) is committed for the first time in the repository and survive for long time in the system, (iii) are perceived by developers as severe problems, and (iv) generalize to other languages/frameworks.

6) EMSE36 [6]: A map is a data structure that is commonly used to store data as key–value pairs and retrieve data as keys, values, or key–value pairs. Although Java offers different map implementation classes, Android SDK offers other implementations supposed to be more efficient than HashMap: ArrayMap and SparseArray variants (SparseArray, LongSparseArray, SparseIntArray, SparseLongArray, and SparseBooleanArray). Yet, the performance of these implementations in terms of CPU time, memory usage, and energy consumption is lacking in the official Android documentation; although saving CPU, memory, and energy is a major concern of users wanting to increase battery life. Consequently, we study the use of map implementations by Android developers in two ways. First, we perform an observational study of 5713 Android apps in GitHub. Second, we conduct a survey to assess developers' perspective on Java and Android map implementations. Then, we perform an experimental study comparing HashMap, ArrayMap, and SparseArray variants map implementations in terms of CPU time, memory usage, and energy consumption. We conclude with guidelines for choosing among the map implementations: HashMap is preferable over ArrayMap to improve energy efficiency of apps, and SparseArray variants should be used instead of HashMap and ArrayMap when keys are primitive types.

II. Information Collected from Papers

A. *esem28*

This is a pure survey, adopting questionaire as its main method, to find out the rate ofcharacteristics.

1) Introduction:

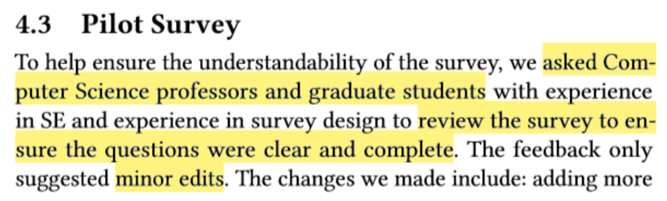
- It tries to understand the Software Development Practices of Blockchain Projects
- Uses online survey
  - Towards blockchain software (BCS) developers on GitHub
- Explores the software engineering practices adopted in BCS
- The first formal survey

2) Design:

- Cross-sectional
- Both open and closed questions
- Including demographic questions

3) Evaluation:

- Pilot survey
  - CS Prof. & graduates
  - Minor changes
- Focus groups
- Face validity



4.3 Pilot Survey

To help ensure the understandability of the survey, we asked Computer Science professors and graduate students with experience in SE and experience in survey design to review the survey to ensure the questions were clear and complete. The feedback only suggested minor edits. The changes we made include: adding more

4) Data Collection:

- Non-probabilistic sampling
  - Convenience sampling
  - Send to all qualified developers and get responses from whom was available at the time
- Sample Size: 156
- Response rate: 156/1604

5) Data Analysis: It adopts a systematic qualitative data analysis process:

- **Extract** general themes
- **Reach** agreed-upon coding scheme
- **Code** the responses with CAT
- **Calculate inter-rater reliability**
- **Resolve** discrepancies
- **Transfer** data to SPSS

6) Result:

- 6 RQs
- Data Visualization
- Data Description
- Analyze causes

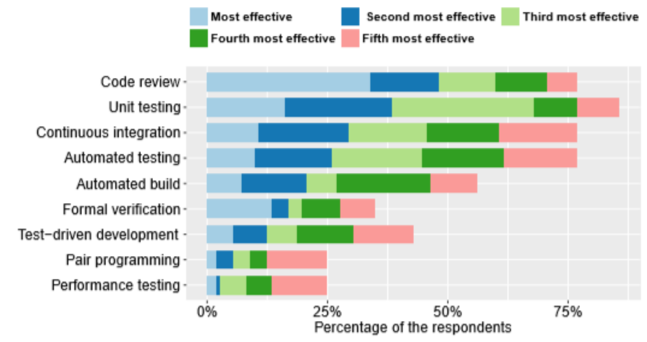


Figure 5: Ranks of software development practices based on effectiveness

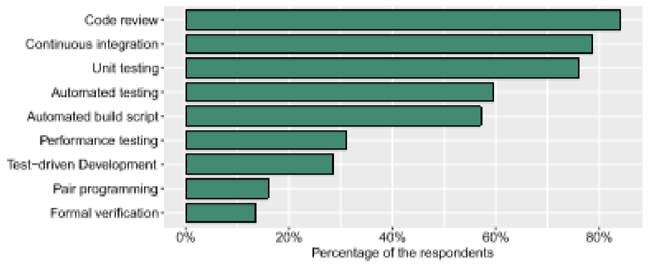


Figure 4: SE practices of BCS developers

B. EMSE105

1) Objective&Method: Objective: to assess the severity of some characteristic or condition that occurs in a population.

Method: Qualitative

2) Design: Research design: This study not only has a survey, but also has an interview.

Survey design: Cross-sectional

3) Survey detail:

- Self-administrated questionnaires: web-based
- Interviews: semi-structured, assumed to be one-to-one
- Does not use existing instruments

4) Sampling:

- non-probabilistic sampling, web-based survey
- convenience sampling
- 37 responses received, 5% rate.

5) Data Analysis: Participant experiences:

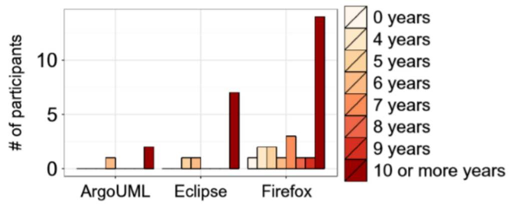


Fig. 5 Software development experience of the participants

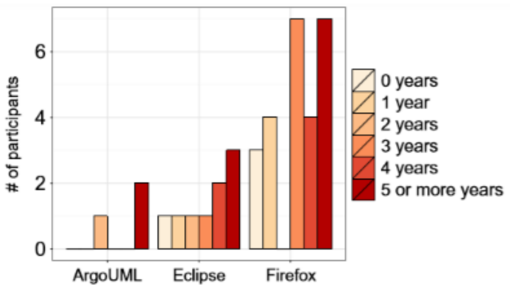


Fig. 6 Development experience of the participants in the respective project

C. EMSE31

1) Motivation:

- Previous works: mainly on tracing changes of low-level project features, e.g. bug fixes.
- This paper: on the change of high-level feature:requirement.

2) Contribution:

- Show an emperical study of whether requirements can be finished in planned iteration.
- build a predictive model to predict whether requirements can be finished in planned iteration.
- All the work was based on three IBM project

3) Survey Administration:

- Interview: face to face
  - 16 semi-structured 30-60 minute interview
- Interviewee:
  - IBM program director
  - Analytics architect
  - 3 other IBM practitioners

4) Roles of interview:

- Drive the study design
- Validate understanding of the repository data
- Find factors which matter in building a prediction model
  - Prediction model features: features proposed in previous works + new features through interviewing

5) Purpose&Design: **Objective:** find factors

Design:

- longitudinal: Multiple iterations to find out related factors
- Involves different people, do not emphasize differences.

Sampling

- No probability methods
- Uses convinience sampling
- Mainly rely on interviewee's experience rather than data analysis

Analyzing

- Directly uses the data.

- Includes:
  - proposals of previous works
  - suggestions of IBM interviewers

D. EMSE124

Architecture consistency: State of the practice, challenges and requirements The artilcle focus on architecture consistency informations by interviewing 19 developers, suc as the approaches to achieve AC, obstacles encountered, and tools needed. Meanwhile the article also makes research into the current AC tools.

- Merely used interview.
- cross-sectional: there are no observations about changes over time.
- one-to-one interviews
- existing instruments: e.g. some existing empirical studies have explore the needs of software architecture, including one in 2013 in which 48 practitioners were surveyed to understand how architecture descrption languages (ADL) were adopted in practice.

Differences to existing instruments:

- 1. they refer to AC across ADL based views, not AC between software system and the architecture.
- 2. They used survey, not semi-structured interviews. (data collection method different in some way from the original instrument’s)

The paper didnt mention evaluating method, at least it is not a questionnaire

Sampling: 3.1 purposive sampling on top of this opportunistic sampling

- external validity: the number of participants are limited
- reliability: despite being audio recorded, most interviews are performed by one researcher only
- construct validity: notes and transcripts were individually analyzed and refined, raising the confidence in construct validity and reliability.

E. EMSE73

This paper categorized the code smells in MVC applications into six categories. During the discovery process, Step1 and 2 are survey, step3 is an interview. It seems that except for that, the rest of the paper mainly uses case study.

Types of Survey Research:

- Questionnaire-based survey：
  - Step 1: cross-sectional survey
  - Step 2: cross-sectional survey
- Interview:
  - Step 3: (2.2)unstructured

Survey Administration:

- self-administrated questionnaire:
  - Step 1: web-based:
  - Step 2: email
- interviews:
  - Step 3: one-to-one

Developing instrument: not mentioned whether using using existing instruments or not, 但猜测为 not using existing instruments

Sampling:

- Step 1: non-probabilistic sampling: convenience sampling
- Step 2: non-probabilistic sampling: convenience sampling

Analyzing data

- response data: partitioning responses
- ordinal data：analyzing ordinal data

F. EMSE36

Usage of questionnaire: to know user preferences

Objective: quantitative

Process: Contacted 656 developers and received118 (18%) completed surveys. Most developers (86%) would replaceHashMapif they hadmore concrete information about the performance of other implementations

Design: cross-sectional: horizontal

Survey Administration: questionnaire: website

Sample: random analyze: quantitative, by calculating percentage

References

[1] Ali, N., Baker, S., O’ Crowley, R., Herold, S., and Buckley, J. Erratum to: Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering 23*, 3 (2018), 1868–1869.

[2] Aniche, M., Bavota, G., Treude, C., Gerosa, M. A., and van Deursen, A. Code smells for model-view-controller architectures. *Empirical Software Engineering 23*, 4 (Aug 2018), 2121–2157.

[3] Blincoe, K., Dehghan, A., Salaou, A.-D., Neal, A., Linaker, J., and Damian, D. High-level software requirements and iteration changes: a predictive model. *Empirical Software Engineering* (Oct 2018).

[4] Chakraborty, P., Shahriyar, R., Iqbal, A., and Bosu, A. Understanding the software development practices of blockchain projects: A survey. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2018), ESEM ’18, ACM, pp. 28:1–28:10.

[5] Costa, D. A. d., McIntosh, S., Treude, C., Kulesza, U., and Hassan, A. E. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering 23*, 2 (Apr 2018), 835–904.

[6] Saborido, R., Morales, R., Khomh, F., Guéhéneuc, Y.-G., and Antoniol, G. Getting the most from map data structures in android. *Empirical Software Engineering 23*, 5 (Oct 2018), 2829–2864.

An empirical study of ARDOC Systems

Empirical Engineering Final Report

Yuqing Yang[*†]
161250179
Software Institute
Nanjing University
Nanjing, China

Cheng Lei
161250054
Software Institute
Nanjing University
Nanjing, China

Kangqiao Zhou
161070096
Software Institute
Nanjing University
Nanjing, China

[*]**Writer of paper**
[†]**Presenter**

***Abstract*—Researchers have proposed multiple approaches to extract useful informations from vast sources of user generated values related to certain systems over the past few years. A group of researchers have been focusing on the user comments on android apps by categorizing and providing useful informations, such as app reviews for developers to improve or fix their products. They claimed that this ARDOC system worked quite well. However, the evaluation process remains superficial, unbound, and lack of persuasive metrics. In this paper, we design, craft and implement an experiment trying to answer the question of whether the ARDOC system really helps developers on their works.**

I. Introduction

Nowadays, the maintainence and development works have been increasingly frequent and fast-paced. Such tendency creates demand of adept developers and requirements of better understandings of the projects under development. Such increasing demand of developers, however, makes the barrier of entering a new project increasingly high, making newcomers harder to begin their work, thus slowing down the development pace. The project organizers or main developers have to allocate lots of time and effort trying to instruct the newcomers, or answering frequently-asked questions repeatedly, making the situation even tougher. Meanwhile, as the project grow large and complex, pinpointing bugs and reproducing anomalies require high amount of time, which is quite a waste. It seems that an automatic machanism to help everyone in the development is becoming essential.

The main problem of such condition is that it lacks a useful tools for developers to filter, classify and aggregate user feedback to guide the development. Researchers have proposed different approaches to fullfill such requirement from the developers. One of the successful and pioneering attempt is ARDOC [3]. In the system, the researchers mainly focus on utilizing user comments under certain android apps. By classifying the comments into different categories defined by ad-hoc rules, the system provides information for developers to improve the apps. Good as it may look like, the categories seem arbitrary and vulnerable. A following-up research [2] makes the determination of categories more reasonable by integrating HDP process, the hierachical dirichlet process, to find out the actual themes of the comments, therefore augmenting the improvement of the project.

However, none of such researches involve actual developers to evaluate their works. They mainly evaluate by coherence values, which may deviate from actual human thoughts. We are curious that whether such ARDOC-like systems can really live up to their claimed performances, therefore in this paper, we:

- design an empirical experiment integrated with survey techniques based on CHANGEADVISOR
- evaluate the actual performances of the systems
- provide the first investigations of issue information retrieval empirically

II. Related Work

A. ARDOC

User comments may contain complaints, wishes or praises of the system, which provides useful information guiding the further development. The researchers proposed the App Reviews Development Oriented Classifier process, managing to distinguish the user comments manually by putting them to four main categories: *information giving, information seeking, feature request, problem discovery.* After that the user comments are parsed under the pre-defined rules of each categories, and classified. The system further on gives suggestion of category and key informations of the user comment.

B. CHANGEADVISOR

With the intension of improving the accuracy and flexibility, CHANGEADVISOR utilizes hdp model to learn and categorize the themes on user comments. Based on ARDOC [3], CHANGEADVISOR combines natural language processing, sentimental analysis, and text analysis techniques, and seeks to categorize the comments more precisely into categories as defined in ARDOC. CHANGEADVISOR benchmarked three theme topic models: LDA [1], LDA_GA, and HDP [5], and figured out that HDP works better than any of the other two methods. After the theme is learned, the system provides suggestions of categories each comment belongs to.

C. HDP

CHANGEADVISOR uses hdp as its theme topic extractor. As an extended version of LDA, HDP is a hierachical theme extraction algorithm, which is different from LDA. HDP is better than LDA in that it does not require the topic num as a hyperparameter. The hdp has a topic num parameter, though, which is set to 150 by default, by setting truncate number of topics, the HDP can automatically select the best topic num to aggregate the themes.

D. Coherence

To evaluate the quality of topic model, coherence act as a metric to measure how tight the topics bound to each other within a category. [1] Normally, we use c_v as our evaluation metric in that although it is slower than the fastest u_mass [4], it achieves the best performance on LDA topics.

III. Approach Selection

To begin the design of our approach, we distinguished the three main methods: experiment, case study, and survey. According to Prof. Zhang, the case study usually happens when it is hard to collect randomly-selected data as done in survey, we declined this method immediately. Since our test subject is concrete: the ARDOC-like system, it is different from the scenes for case study. According to our research goal, it is better to use experiment combined with questionaire to proceed with our study. The users can be invited to test our system, and certain metrics and expert can be used to evaluate the difference bettwen experimental and comparison group. For some other properties that may hard to quantify, we use questionaire to standardize them, thus providing a following-up qualitative evaluation.

IV. Research Overview

A. Research Questions

In this experiment, we want to answer following research questions:

- RQ1. How well is the clustering results of the ARDOC-like process?
- RQ2. Does the ARDOC-like process benefits both development efficiency and experience?

B. Research Protocol

We mainly use the CHANGEADVISOR and the follwing code dependency analysis tools as the augmenting system. The research is divided into five parts.

- Model training. We train our model under the process defined in CHANGEADVISOR, judging by coherence model for the best performances.
- Pre-Assessment. Twenty actual users are invited to assess whether the results of CHANGEADVISOR actually fits the user demands of the system.
- Development. Since the experience is gained after the development, we adopt the between-subject design. A total number of sixteen developers are invited in our experiment on a maintainence task on an unfamiliar project, grouped into two subgroups. Both groups are assigned an review of problem discovery. While the control group starts the work without a suggestion, the experimental group is provided with the ARDOC system to give them suggestions on topics of user review.
- In-Assessment. Five experienced project contributor are invited as experts to assess the time and quality of maintainence tasks. Both factors are taken into account. Moreover, the actual users are also invited to evaluate the performances of the developers' quality of task.
- Post-assessment. Each of the new-commer developers are required to do a questionaire commenting on the using experiences of the system, answering whether it really helps with the development.

C. Potential Threats

Before going further, we analyze the potential threats of the selection of the subjects.

- User Personal Bias: for the actual users, bias may exist since they ususally are amateurs to the system. We select 20 of them trying to prevent such problem.
- Before-after Bias: After the development, the developers may gain knowledges of the project, therefore making latter developments easier. Therefore we separate the developers into two groups, each of them working on the same task on the same project.
- Developer Selection Bias: Different developers have different background, which may affect their performances. We develop a questionaire for each developers to find out their level of experiences and try to balance their backgrounds in our between-experiments. Before the experiment, the target project is kept credential, and before the experiment starts, the project will be announced, and the developers affiliated to the project will be removed from our study.
- Evaluation bias: The evaluation of quality may be affected. We adopt a double-blind experiment to tackle with this problem.

V. Research Details

A. Model training

We train the model according to the description in CHANGEADVISOR. Then we evaluate the performances of each run, calculating the coherence. Empirically, we expect the ideal coherence to be around 0.7-0.75. After the model is trained, we push our experiment to the next phase.

B. Pre-Assessment

In this phase, twenty actual users are invited. The actual users are not required to have developing skills, in that roles as programmers may affect their points of view on certain apps. A total number of 500 random-selected comments are selected and presented to the users and the system separately. Then we collect the results from the users and system, and learn the difference between the categorizing, since under such circumstance, the idea of the human is the ultimate judgement.

C. Development

In this phase, 16 developers that are new to the project are invited. The backgrounds of these subects of the experiment are collected with questionaires, and they are balanced into two groups:

- How long have you been programming?
- Your familiar programming language?
- What's your eduacation status?
- What's your current job title?

Then they are required to perform maintainence task on a *problem discovery* task. We select this task because it is the most vital and easy to observe kind of task. One group begins directly, and the others are provided with the CHANGEADVISOR with them. Then their time used on fixing the problem is recorded.

D. In-Assessment

In this phase, we invite five main developers from the app, they will evaluate the maintainence task quality of each developer, scoring from 1 to 10. Meanwhile, the users will also test the fixed app and check whether the apps really fixed the problem, scoring from 1 to 10.

E. Post-Assessment

In this phase, the developers are asked to fill a questionaire, judging their development experiences with CHANGEADVISOR, commenting on its performance on the following properties:

- Clustering accuracy
- Code Correlation

VI. Expected Result

If we want to prove that such systems is indeed beneficial, results are expected to be:

Item	Exp.(with)	Compar.(without)
Development time	high	low
Development quality	high	low
Developer experience	high	low

TABLE I
Expected result

References

[1] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *Journal of machine Learning research 3*, Jan (2003), 993–1022.

[2] PALOMBA, F., SALZA, P., CIURUMELEA, A., PANICHELLA, S., GALL, H., FERRUCCI, F., AND DE LUCIA, A. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th international conference on software engineering* (2017), IEEE Press, pp. 106–117.

[3] PANICHELLA, S., DI SORBO, A., GUZMAN, E., VISAGGIO, C. A., CANFORA, G., AND GALL, H. C. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2016), ACM, pp. 1023–1027.

[4] RÖDER, M., BOTH, A., AND HINNEBURG, A. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining* (2015), ACM, pp. 399–408.

[5] TEH, Y. W., JORDAN, M. I., BEAL, M. J., AND BLEI, D. M. Sharing clusters among related groups: Hierarchical dirichlet processes. In *Advances in neural information processing systems* (2005), pp. 1385–1392.