Fast Feedback Cycles in Empirical Software Engineering Research

Antonio Vetrò, Saahil Ognawala, Daniel Méndez Fernández
Technische Universität München
Email: vetro—ognawala—mendezfe@in.tum.de
Stefan Wagner
University of Stuttgart
Email: stefan.wagner@informatik.uni-stuttgart.de

***Abstract*—*Background/Context:* Gathering empirical knowledge is a time consuming task and the results from empirical studies often are soon outdated by new technological solutions. As a result, the impact of empirical results on software engineering practice is often not guaranteed.**

***Objective/Aim:* In this paper, we summarize the ongoing discussion on "Empirical Software Engineering 2.0" as a way to improve the impact of empirical results on industrial practices. We propose a way to combine data mining and analysis with domain knowledge to enable fast feedback cycles in empirical software engineering research.**

***Method:* We identify the key concepts on gathering fast feedback in empirical software engineering by following an experience-based line of reasoning by argument. Based on the identified key concepts, we design and execute a small proof of concept with a company to demonstrate potential benefits of the approach.**

***Results:* In our example, we observed that a simple double feedback mechanism notably increased the precision of the data analysis and improved the quality of the knowledge gathered.**

***Conclusion:* Our results serve as a basis to foster discussion and collaboration within the research community for a development of the idea.**

Keywords: Empirical methods, Research methods, Data mining, Knowledge transfer

I. Introduction

In recent years, the contributions made in empirical software engineering enabled a shift in our discipline from a more design-science-driven engineering, where we applied scientific methods to isolated practical problems, to a more epistemology-driven and insight-oriented science [2]. That is, over the years, we could establish a reliable software engineering body of knowledge which supports the problem-driven development and evaluation of various methods and tools, thus, supporting scientific progress in our field. A common parallelism is often drawn with physics, where experimental physics is devoted to conduct research upon the theorems and proofs provided by theoretical physics. However, whereas physics is governed by precise laws which we can express and objectively interpret in mathematical forms, software engineering laws are less structured and more difficult to model, because they rely on the cognitive abilities of people [14]. In addition, software engineering laws are often valid only within specific contexts, whose boundaries are difficult to draw because of the multitude and incertitude of the human, economical, technological, and cultural factors involved. The aforementioned intrinsic difficulties and the young age of the discipline make theory building and scientific knowledge acquisition often slow and not always in tune with the current speed of industrial practices and innovation. Techniques are often not tested in practical settings for many years after they were invented [5].

These issues are well-know in the empirical community, and since a few years, a need for change has pulsed under the surface: the traditional empirical techniques might need to be complemented by new technologies and new ways in which we treat knowledge today [16]. In particular, data mining and analysis of software engineering data has captured a lot of attention in technical briefs, panel sessions and editorials, and the related concepts have often been identified under the umbrella of "Empirical Software Engineering (EMSE) 2.0".

We will summarize the ongoing discussion and evolve it with our preposition, which is: Empirical Software Engineering research should steer towards the automatic collection of domain knowledge as driver and corrector for further automatic analysis and as facilitator for fast feedback cycles. We can combine the speed of data mining with the human domain expertise and knowledge. The expected benefits are an iterative knowledge-value chain and an iterative pattern discovery process which allows us a fast transfer into practice and provides input for follow-up studies.

II. EMSE 2.0

The term *EMSE 2.0* was coined first by Thomas Zimmermann and appeared in Andreas Zeller's 2007 keynote of the *Mining Software Repositories* conference [20]. He underlined that, although empirical studies and their results are valuable, collecting the proper data takes a high amount of effort and time and their analysis brings results that are limited in scope and time. In that keynote, Zeller suggested a scenario based on the transposition of the techniques and concepts of *Web 2.0* into the empirical software engineering community. He envisioned data being available with no effort and instantaneous results aligned to the current situation of the software project. The vision was grounded on a simple technological solution that, in fact, has become current practice in the later years: software archives. Nowadays, we have a large amount of software repositories (especially open source) which contain not only source code but also additional information about artifacts and processes: for example, bugs, requirements, developers' mails, or change requests. The empirical community is also

making an effort to share this kind of data, spread over various sources, through the shared *PROMISE* repository[1] and the related conference.

The keyword *EMSE 2.0* appeared again during an *ICSE 2011* technical brief by Menzies and Shull [10]. The motivations of the brief resided in the same problems of slowness and weak impact of empirical studies. The proposed idea was a scalable empirical research approach based on the combination of automated analysis of data with human domain expertise and knowledge. This approach breaks the narrow technological view of the original idea of Zeller, including also the domain knowledge as driver and corrector of the automatic analysis. This piece of the puzzle has become more and more central in the later follow-ups. In an editorial introduction of the *IEEE Software* magazine in 2012 [16], Shull wrote about *Research 2.0* taking inspiration from the idea of *Science 2.0* coined by Shneiderman [15]. Shull stressed again the need of a hybrid approach that combines the cognitive power of manual hypothesis testing with the speed of automatic analysis. The vision was to have tools which would enable practitioners to take data-driven decisions linked to the business and strategic goals of an organization. A few months later, in the introduction for the special issue on Software Analytics for the same magazine [17], the focus on the human intuition behind massive data analysis was more explicit and enriched with an accent on collaborative effort in the hypothesis testing process. In that special issue, we also have found several practical applications that indicate that we, as a community, are already heading in this direction.

We are still far from an end point, however. The approach is promising but not easy, and the original picture drawn by Zeller might be too optimistic. First, combining data from different sources is not an easy task: data mining opportunities can be neutralized by poor quality of the data itself, like a lack of common semantics, low accuracy, or low degree of completeness. In addition, even when data is of high quality, a big amount of data contains big amounts of useless data as well as statistical noise. Therefore, applying statistical techniques with human qualitative analysis of input data is essential [21]. Yet, one more important aspect emerging from success stories in software analytics is to incorporate domain knowledge and to enable a close relationship between researchers and practitioners, interactively and iteratively [22]. Also in the *International Workshop of Conducting Empirical Studies in Industry* co-located with ICSE'13[2], part of the discussion focused on the value of feedback with stakeholders. The importance of feedback is also stressed by Basili in his personal perspective on the Empirical Software Engineering story [2] and it is embedded in the cyclic process for technology transfer in Software Engineering proposed by Gorshek et al. [7]. Recent work has even suggested the use of minimum viable products, which is built around the feedback concept, in industry-academia collaborations [11] .

Finally, past work showed that learning and flexibility has positive impact on process decisions [6]. In the same way, we think that applying fast iterations of feedback in the empirical cycle, we can minimze risks of failure in industry-research collaborations and focus on value.

[1]https://code.google.com/p/promisedata/

[2]http://www.essi.upc.edu/ franch/cesi2013/program.html

Therefore, we center our idea around the role of feedback and our proposition is that data mining can speed up the feedback cycles with stakeholders at any of the steps of the traditional empirical research cycle[3].

III. Enabling fast feedback cycles

Feedback can be collected informally, for example in retrospective meetings, and also using more formal empirical methods like surveys or interviews. The Web 2.0 technologies, however, have revealed mechanisms and tools to collect fast feedback, both explicitly and implicitly. Some straightforward examples: *Facebook* and *Google* use "like" , "+1" and recently even "emoticons" to gather opinion and sentiments. *Stackoverflow* has a mechanism of arrows up and down to rate proposed solutions. The *New York Times* traces users navigation to suggest similar interesting readings. Amazon does even more: It tracks user purchases and navigations on the website to build customized recommendations and after purchases asks the buyer to leave reviews. These examples[4] show that simple but effective mechanisms can easily and quickly collect a large amount of feedback to extract knowledge. In addition, when it comes to combining a large amount of data with human feedback, results are even more promising: *Google Translator*[5] refines its probabilistic models based on millions of digitalized books with human feedback. Another project, *reCAPTCHA*[6], instead, reverts this cycle: it uses a large amount of human feedback (crowd wisdom based on independent judgements) to build knowledge for text recognition tasks. We can take inspiration from these mechanisms and elaborate them for the following goals:

1) Shorten our feedback time towards industry collaborators,
2) fasten feedback among scientists, and
3) tune the empirical approach in any of its phases from design to data analysis and interpretation.

In our vision, the data sources include data from the development, execution, and maintenance of software projects. As stated in the introduction, collecting data from software production has become a normal practice in the last years. Yet, understanding which data to collect, what is good data, and how to get it is not an easy task at all. For this reason, we think that domain knowledge is still an important missing piece in EMSE 2.0 [21]. In addition, tuning of the data mining techniques for empirical software engineering research purposes is still an open issue [18].

These two concepts form the basic of the proposed approach: tune data analysis techniques with automatic injection of stakeholders' feedback. For a first test of this approach, we built a fast feedback cycle with a local company as described next.

IV. Proof of Concept

We developed a small proof of concept with an industrial partner to investigate our proposed approach. The industrial

[3]Herein, we will refer to the cycle reported by Jedlitschka [8] which is based on the Quality Improvement Paradigm and Experience Factory [3].

[4]Some of these examples are from [9]

[5]http://translate.google.com/

[6]https://www.google.com/recaptcha

partner follows the *Scrum* development process with tool support for the process steps by *Atlassian Jira*. Examples of the available data, by phase, are:

- *Sprint planning*: Story points, tasks, features, dependencies between features, characteristics of stories
- *Coding*: Implementation time, code, structural metrics from code, bugs, changes
- *User acceptance*: Outcome, bugs, customer comments
- *Retrospectives*: Notes, problems

For the proof of concept, we concentrated on the user stories to learn more about the context of the requirements and recurring patterns within them. The main practical goal was to discover "scope creeps", i.e. discrepancies in the mapping of user stories and project goals. As a secondary use, a prediction mechanism could reveal recurring topics for user stories which were wrongly estimated or which code exposed the most bugs.

All the user stories used in this study were marked as "100% complete" or "Unresolved". For our analysis, we only used those stories marked as completed and belonging to a single project. To extract semantic information, we followed a process that consisted of two steps:

1) *Text Sanitization*: Pre-processing the text of user stories into a consistent format. This includes removal or conversion of non-ASCII characters, or accents followed by language-specific stemming [12].
2) *Topics Extraction*: A probabilistic method of inferring topic distribution from the set of all user stories. This method is described in more detail in the next sub-section.

A. Topics Extraction: Implementation

We applied a probabilistic approach to inferring topic distribution from a set of documents as described in [19]. Given a list of topics, the collection of documents is assumed to contain a distribution of these topics. Similarly, the topic itself has a distribution over all words in the vocabulary. This can be formalized as below:

$$P(w_i) = \sum_{j=1}^{T} P(w_i \mid z_i = j) P(z_i = j) \tag{1}$$

Here, $P(w_i)$ is the probability of drawing the i^{th} word in the vocabulary. Index j denotes the topic id. z_i is the topic from which w_i is sampled. Therefore, $P(z_i = j)$ denotes the probability of drawing from topic j. The probability $P(w_i)$ is given as the sum of conditional probabilities of drawing word w_i from any of the T topics present in the document corpus.

Every document is assumed to have been generated with these probability distributions (topic distribution in document corpus and word distribution in topics). This is called the *generative model*. The idea of the generative model can be inverted to a *statistical inference* process to learn these probability distributions: This inversion process to learn the topics from a big set of text documents is called *Topic Modeling*. In the set of user stories, we treat each story (indexed by Story ID) as a separate document. On the output from the *text sanitization* step, we perform a statistical inference to get a list of k topics from the list of user stories. We then associate a feature vector to every user story that denotes whether a topic was seen in it or not depending on the top n words (according to the probability distribution) that belong to a topic. The value of k was chosen to be such that the feedback from the stakeholders is fast.

B. Fast Feedback Mechanism

We evaluated the topic extraction step by assuming that the group of words present in a topic indicate a functional or non-functional area of requirements. Examples of such areas are *logistics*, *database*, or *web framework*. We simulated a simple feedback collection mechanism with two iterations. In the first iteration, we presented the list of topics to the partners to get their feedback. In the list of extracted topics, the partners marked the ones where the group of words together clearly point to a functional or non-functional area. Partners also marked the most *influential words* in all topics, i.e. those words that were significant and could be related to functionalities or important functional requirements.

We used this feedback to tune the topic extraction process in the following way: We removed the terms that were *not marked* as influential assuming that these words do not have an effect on the ability to identify a functional or a non-functional requirement. After this, we ran the topic extraction process in a second iteration to extract the same number of topics as before. The stakeholders give the same manner of feedback on the second iteration. The most important knowledge that we obtained from the second iteration, which was tuned with the stockholders' feedback, was that the number of topics that were marked with one or more functional or non-functional area of requirement increased after the second iteration. Specifically, this number increased from *3 out of 10* to *9 out of 10* from the first iteration to the second iteration. Concretely, this means that incorporating the feedback from the first iteration helped to improve the topic extraction stage quantitatively by having more related influential terms appear together in topics. Additionally, there were *23* functional or non-functional areas identified within these 9 topics in the second iteration, compared to only *5* in the 3 topics of the first iteration.

Therefore, this proof of concept demonstrates potential benefits of proceeding with fast feedback cycles to gather knowledge. In particular, instead of classifying the user stories with a lot of human effort and with the risk of still being wrong (at least in part), it is possible run quick automatic analysis, get the domain knowledge through feedback mechanisms and automatically refine. In our example we observed that a simple double feedback mechanism (i.e. marking influential work and writing on text box functionalities connected to the automatically extracted topics), notably increased the precision of the topics extraction algorithm from the user stories.

C. Research Roadmap

From a conceptual point of view, our future effort will be devoted to understand under which circumstances and at which points of the empirical cycle [8] it is possible to inject feedback cycles. It is important to understand which techniques to use when to build quantitative analyses upon qualitatively gathered information. Research in such directions has been already done in the interactive machine learning community (e.g., [1], [4]). In particular, the trend towards iterative development with

very short iterations supports in quickly observing results of feedback. Considering this research as a community effort, it will hopefully result in:

1) A coarse artefact model in support of the approach
2) A set of method building blocks and their combination to create the artefacts
3) Application of the building blocks to real problems in the field and collection of preconditions, lessons learnt, and fail conditions

From the implementation point of view, our next step is a more complete evaluation of the proof of concept presented here. To this end, we aim at implementing the feedback mechanism within an interactive web application and test the improvements longitudinally with more iterations. We are also investigating the application of our idea in the field of software energy efficiency research (for details see [13]).

V. Conclusions

The promise of EMSE 2.0 is to speed up the transfer of empirical studies and improve their impact into industrial practice. To this end, we envisioned a way to enable fast feedback cycles in empirical software engineering research. Our proposition is that the analysis of data from software projects can be combined with automatic mechanisms to gather stakeholders feedback and improve precision and applicability of the analyses. We built a proof of concept with two iterations of feedback cycles to tune the extraction of user stories' topics and reported improvements.

VI. Aknowledgements

We thank Forrest Shull, Davide Falessi, Andreas Jedlitschka, and Jens Heidrich for the time spent around this idea at Fraunhofer CESE in College Park. Thanks also to Henning Femmer and Jakob Mund for their feedback.

References

[1] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[2] V. Basili. A personal perspective on the evolution of empirical software engineering. In J. Muench and K. Schmid, editors, *Perspectives on the Future of Software Engineering*, pages 255–273. Springer Berlin Heidelberg, 2013.

[3] V. R. Basili, G. Caldeira, and H. D. Rombach. *Encyclopedia of Software Engineering*, chapter The Experience Factory, pages 469–476. John Wiley & Sons, 1984.

[4] S. Das, T. Moore, W. Wong, S. Stumpf, I. Oberst, K. McIntosh, and M. M. Burnett. End-user feature labeling: Supervised and semi-supervised approaches based on locally-weighted logistic regression. *Artif. Intell.*, 204:56–74, 2013.

[5] O. Dieste, N. Juristo, and M. Martins. Software industry experiments: A systematic literature review. In *Conducting Empirical Studies in Industry (CESI), 2013 1st International Workshop on*, pages 2–8, 2013.

[6] H. Erdogmus. The economic impact of learning and flexibility on process decisions. *Software, IEEE*, 22(6):76–83, Nov 2005.

[7] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson. A model for technology transfer in practice. *Software, IEEE*, 23(6):88–95, Nov 2006.

[8] A. Jedlitschka, L. Guzmán, J. Jung, C. Lampasona, and S. Steinbach. Empirical practice in software engineering. In J. Muench and K. Schmid, editors, *Perspectives on the Future of Software Engineering*, pages 217–233. Springer Berlin Heidelberg, 2013.

[9] V. Mayer-Schönberger and K. Cukier. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt, 2013.

[10] T. Menzies and F. Shull. Empirical software engineering 2.0. http://tinyurl.com/mdjlxhq, Jan. 2011.

[11] J. Muench, F. Fagerholm, P. Johnson, J. Pirttilahti, J. Torkkel, and J. Jarvinen. Creating minimum viable products in industry-academia collaborations. In *Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013)*, LNBIP, Galway, Ireland, 2013. Springer-Verlag, Heidelberg.

[12] M. F. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[13] G. Procaccianti, P. Lago, A. Vetrò, D. Méndez Fernández, and R. Wieringa. The green lab: Experimentation in software energy efficiency. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015. To appear.

[14] D. Rombach. Empirical software engineering models: Can they become the equivalent of physical laws in traditional engineering? In J. Muench and K. Schmid, editors, *Perspectives on the Future of Software Engineering*, pages 1–12. Springer Berlin Heidelberg, 2013.

[15] B. Shneiderman. Science 2.0. *Science*, 2008.

[16] F. Shull. Research 2.0? *IEEE Software*, 29(6), 2012.

[17] F. Shull. Getting an intuition for big data. *IEEE Software*, 30(4):3–6, 2013.

[18] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen. The future of empirical methods in software engineering research. In *2007 Future of Software Engineering*, FOSE '07, pages 358–378, Washington, DC, USA, 2007. IEEE Computer Society.

[19] M. Steyvers and T. Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[20] A. Zeller. Empirical software engineering 2.0: How mining software repositories changes the game for empirical software engineering research. http://tinyurl.com/k7vaj6p, 2007.

[21] A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word: A parody in empirical research. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 5:1–5:7, New York, NY, USA, 2011. ACM.

[22] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie. Software analytics in practice. *Software, IEEE*, 30(5):30–37, 2013.