Introduction to Empirical Software Engineering

Martin Monperrus
University of Lille

Version of June 28, 2016

This documents presents an introduction to empirical software engineering. It has been first prepared for a course at EJCP 2015[1], then reworked for EJCP 2016.

This document will evolve, I would thus appreciate your feedback.
– Martin Monperrus

Background: see `http://www.monperrus.net/martin/about+me`
Previous work in empirical software engineering: see `http://www.monperrus.net/martin/publications`

1 Science and Engineering

The classical difference between science and engineering:

- **Science** is about observation, explanation.
- **Engineering** is about creation and optimization of tools.

There is no hierarchy, both are beautiful:

- Science is curiosity driven.
- Engineering is utility driven.

But there are profound links between both:

- Science: needs new tools (particle collider, Galileo's telescope)

[1] http://ejcp2015.inria.fr/

- Engineering: needs observation and explanation on phenomena resulting from new tools.

Software engineering research sometimes falls short on the utility side.

- Unreasonable assumptions.
- No applicability.
- No empirical validation.

2 A Tentative Definition of Empirical Software Engineering

Software engineering is dual. Literally, software engineering is the creation and maintenance of software. But from a research perspective, software engineering is the body of knowledge about the creation and maintenance of software and about the phenomena underlying and emerging from those two activities.

- software engineering: creation and maintenance of actual software
- software engineering research: tools to create software, understanding of the nature of software and its usage.

Empirical software engineering is a research area concerned with the empirical observation of software engineering artifacts and the empirical validation of software engineering theories and assumptions. Subfields of software engineering that are accustomed to empirical research comprise **software evolution**, **software maintenance** and **mining software repositories**.

- Observation of artifacts.
- Validation of tools.
- Validation of methodologies.
- Validation of assumptions.

For example:

- Observation of artifacts: What is distribution of software dependencies?

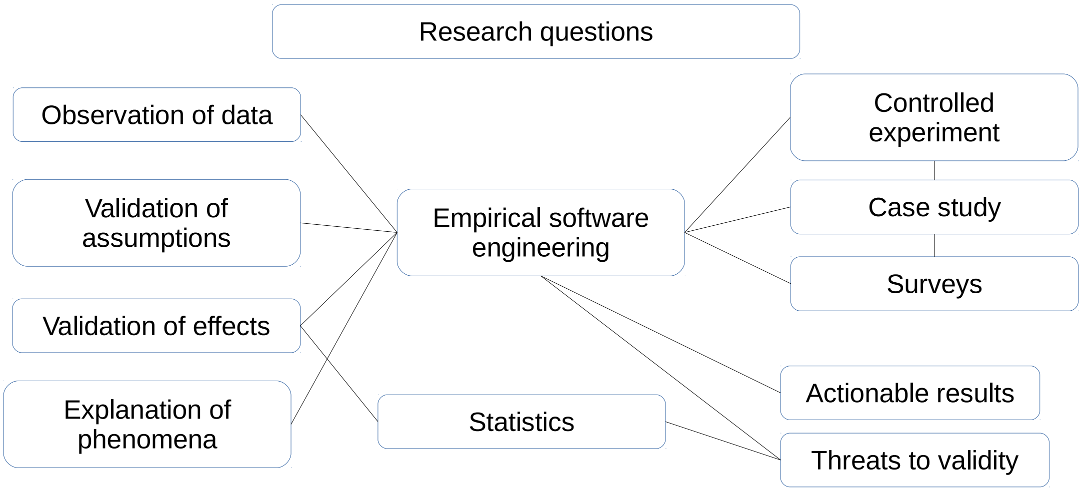

Figure 1: A partial conceptual map of empirical software engineering

- Explanation of phenomena: Why is the distribution exponential? [10]
- Validation of tools: Does static analysis of buffer overflows work in practice?
- Validation of methodologies: Is pair-programming effective?
- Validation of assumptions: Can an average programmer write a specification in linear temporal logic?

Empirical software engineering starts with a good question:

- Is the assumption of independence in N-version programming true? [8]
- Is static typing really good? [7]
- What is the diversity of module usages?[9]
- Does Github change something in open-source processes? [2]
- <Your questions>

There are good, so-what and bad questions. A good question has at least one of:

- a clear answer.
- triggers new questions.

- an **actionable** answer.
- a surprising result (and why not a fascinating one).

You often read reviews like: "Nice paper, well-written and interesting results but the results are not actionable". In ESE, an answer is said to be **actionable** if the answer leads to ("engineering forever"):

- the creation of a new tool
- the improvement of existing tools
- the improvement of existing development and engineering processes

Empirical research can be applied to all artifacts of the software engineering process:

- Code (source, binary)
- Version control systems
- Bug reports
- Documentation
- Communication traces (e.g. emails, forums, Stackoverflow)
- Design documents
- Execution traces
- Configuration files

3 Types of empirical software engineeering

A **controlled experiment** "is an investigation of a testable hypothesis where one or more independent variables (treatment) are manipulated to measure their effect on one or more dependent variables" [3]. For example using a tool vs not using a tool. See the TSE survey on this topic [13].

- a hypothesis
- tasks

- subjects (who?)

A concrete example: “Software Systems as Cities: A Controlled Experiment” [14].

- 5 questions, incl “Does the use of CodeCity increase the correctness of the solutions to program comprehension tasks, compared to non-visual exploration tools, regardless of the object system’s size?”
- 10 tasks, e.g. “Locate all the unit test classes of the system and identify the convention (or lack thereof) used by the developers to organize the tests”
- subjects: 6 locations, e.g. “Bologna I. 8 professionals with 4–10 years of experience.”
- The answer is yes.

“An experimental evaluation of the assumption of independence in multiversion programming” [8]

- RQ: are bugs independent?
- 1 task (implement a missile interception decision system
- subjects: 27 student programmers
- The answer is no: “We conclude that the model does not hold. However, clearly the only potential problem with the model is that it is derived from the assumption of independent failures. Thus, we reject this assumption.”

A controlled experiment is a child of **reductionism**, which advocates to constantly seek and isolate simple laws and effects. But, sometimes the phenomenon under study is too complex, ecology, astronomy

- some effects happen only at a certain size (10 developers, 100000LOC, etc)
- some effects happen only on a certain time frame (code decay after 5 years)
- too many and uncontrolled independent variables (programmer background, application domain) the classical “student subject” problem

Even if a controlled experiment is possible:

- it may be very costly (pay real developers for 2 months?)

- it may clash with the duration and requirements of a PhD (short term results and publications)

A **case study** is “an empirical method aimed at investigating contemporary phenomena in their context” [12].

⚠Much confusion. Different from colloquial meaning “worked example”

- Descriptive case study: portraying a situation or phenomenon.
- Exploratory case study: finding out what is happening, seeking new insights and generating ideas and hypotheses for new research.
- Confirmatory & Falsifying case study: testing existing theories

The steps of for setting up case studies (Five of [12] plus one):

be agile!

- set up objectives and research questions (case study design)
- selection of cases (purposive sampling, extreme/critical/paradigmatic cases)
- prepare what and how the data should be collected
- collect the data
- analyze the data
- reporting

extreme is a revealer

An example, “Pair Programming and Software Defects – A Large, Industrial Case Study” [1]

- project large Italian manufacturing company (application domain?)
- 5 RQs (table 11): Is there a relationship between the usage of pair-programming and the defect rate in the code?
- 6 types of data (Table 12): effort, PP configuration, work item, changelog.
- 39 defects, 144 user story implementations
- Zero-inflated Poisson Regression (ZIPR), Mann-Whitney, Kolmogorov-Smirnov
- There is a slight effect of PP on reducing defects
- My opinion: does it improve code ownership?

On this topic, I recommend the excellent reading "Five misunderstandings about case-study research" [5]. Excerpts:

- Misunderstanding: concrete, ~~practical (context-dependent) knowledge is less valuable~~ than general, theoretical (context-independent) knowledge : no depth only comes with context.
- Misunderstanding: ~~One cannot generalize on the basis of an individual case~~; therefore, the case study cannot contribute to scientific development: no, falsification requires a single example.
- Misunderstanding: ~~The case study contains a bias toward verification~~, that is, a tendency to confirm the researcher's preconceived notions: no, not more than with other techniques.
- Misunderstanding: ~~It is often difficult to summarize and develop general propositions and theories on the basis of specific cases~~: no, a good narrative can have a much bigger impact. (see the excellent paper "My hairiest bug war stories" [4])

A **survey** is the study of the characteristics of a broad population of individuals [3].

- Terminology: ⚠Different from common meaning of "literature survey".
- A collection of standardized information from a specific population [12]
- Emphasis on large sample (as opposed to case study?) what is large? small?
- Not necessarily by means of a questionnaire or interview [12]
- Terminology: usually a synonym of "empirical study".
- Terminology: If done in the field, aka **field study** but what is a field in software engineering?

The steps of survey research are the same as for case studies. But the main differences are:

- Larger number of cases
- Automated data collection vs manual one
- Automated data analysis vs manual one

An example of survey research, "A study of the uniqueness of source code" by Gabel and Su [6]:

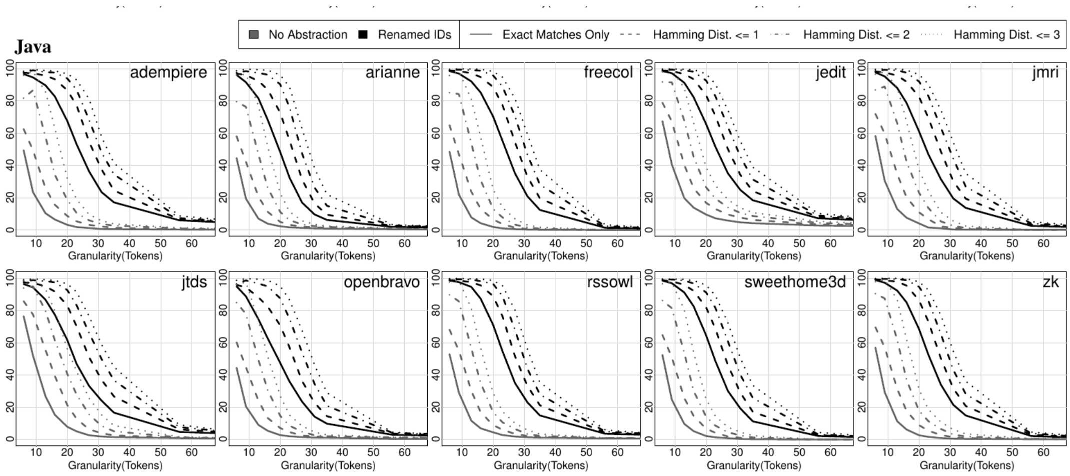

Figure 2: Excerpts of the results of Gabel and Su [6]

- 3,958 C projects, 1571 C++ projects, 437 Java programs (420 Millions lines of code)
- collect token-level n-grams (with and without abstraction / granularity)
- measure uniqueness (syntactic redundancy)
- results are in Figure 2

There are many other expressions and related concepts, but they are less used in the software engineering literature:

- User study
- Action research: do it as opposed to observe it
- Natural experiment: control conditions are determined by nature
- Quasi-experiment
- Ethnography
- Longitudinal study: involves repeated observations of the same variables over long periods of time
- Cohort study or Panel study: a particular form of longitudinal study where a group of subjects (patients) is closely monitored over a span of time.

The magic of version control

4 Validity

Empirical research can be informative or anecdotal, it can contribute to major advances for knowledge or be completely flawed.

- Generalization from two examples?
- Measure the wrong thing?

The systematic study of the **threats to validity** aims at staying on the right side of science. They are different kinds of validity

- External Validity
- Construct Validity
- Internal Validity

The **external validity** is the extent to which the results hold for other subjects.

- for other application domains?
- for other application programming languages?
- for other kinds of programmers?
- mitigation: careful inclusion criteria

The **construct validity** is the extent to which the observed phenomena correspond to what is intended to be observed. Possible threats:

- bugs in tools used for measurement
- bugs in tools used for analysis
- mitigation: validated software, open-source

For controlled or natural experiments, the **internal validity** "means that changes in the dependent variables can be safely attributed to changes in the independent variables" [11].

- what factors are uncontrolled?

- mitigation: ?

The **reliability** “focuses on whether the study yields the same results if other researchers replicate it” [3]. Related to the well-known “experimenter bias”.

The analysis and assessment of threats often/always contains a subjective part (on your side and on the reviewer’s side).

5 Statistics

Having a good statistical analysis is important in some but not all empirical research (for instance for controlled experiments). This course is not a statistics course, so I only remind the core-core concepts.

- a confidence interval indicates the reliability of the result
- a critical value of a measure is the threshold beyond which randomness is not a possible explanatory option.
- the null hypothesis states that the observed phenomena simply occurs by chance.
- a type I error is detecting an effect that is not present
- the p-value is the upper bound on the type 1 error frequency.
- the effect size indicates the magnitude of an effect

Common anti-patterns in SE papers:

- the authors don’t understand what their statistics mean
- the reviewers don’t understand what the statistics mean
- only the p-value are given and not the core metrics
- statistics just for statistics and not enough perspectives, actionable, deep discussions

Hypotheses:

- Null hypothesis: no effect, no relation ship, random

- Alternative hypothesis: there is an effect

For computer scientists, all statistics can be understood and validated with Monte-Carlo simulations.

```
counter=0 # temp variables
for i in range(0,Ntest):
  v1=[x for x in range(0, Nelem)]
  v2=[x for x in range(0, Nelem)]
  # purely random phenomenon, no correlation at all
  random.shuffle(v1)
  random.shuffle(v2)
  spearman=scipy.stats.spearmanr(v1,v2)[0]
  if spearman>critical_value:
    counter=counter+1

p_value = counter*1.0/Ntest
```

6 Conclusion

The best way to truly understand and appreciate empirical software engineering is to read and reread excellent empirical papers. Here is an arbitrary anthology.

7 Appendix

What are the main venues for empirical software engineering research? An answer with DBLP (controlled|study|experiment|empiric in the title):

- Empirical Software Engineering (ESE) (522 publications)
- Journal of Systems and Software (JSS) (284 publications)
- Information & Software Technology (INFSOF) (244 publications)
- IEEE Trans. Software Eng. (TSE) (237 publications)
- ICSE (230 publications)

Important Concepts

empirical software engineering, 2
engineering, 1
science, 1
software engineering, 2
validity, 9
actionable, 4
case study, 6
construct validity, 9
controlled experiment, 4
external validity, 9
field study, 7
internal validity, 9
mining software repositories, 2
reductionism, 5
reliability, 10
software evolution, 2
software maintenance, 2
survey, 7
threats to validity, 9

References

[1] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko. "Pair Programming and Software Defects – A Large, Industrial Case Study". In: *IEEE Transactions on Software Engineering* 39.7 (July 2013), pp. 930–953.

[2] M. Biazzini, M. Monperrus, and B. Baudry. "On Analyzing the Topology of Commit Histories in Decentralized Version Control Systems". In: *Proceedings of the 30th International Conference on Software Maintenance and Evolution.* Canada, 2014.

[3] S. Easterbrook, J. Singer, M. Storey, and D. Damian. *Selecting Empirical Methods for Software Engineering Research.* Ed. by F. Shull and J. Singer. Springer, 2007.

[4] M. Eisenstadt. "My hairiest bug war stories". In: *Communications of the ACM* 40.4 (1997), pp. 30–37.

[5] B. Flyvbjerg. "Five misunderstandings about case-study research". In: *Qualitative inquiry* 12.2 (2006), pp. 219–245.

[6] M. Gabel and Z. Su. "A study of the uniqueness of source code". In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering.* ACM. 2010, pp. 147–156.

[7] S. Hanenberg. "An Experiment About Static and Dynamic Type Systems: Doubts About the Positive Impact of Static Type Systems on Development Time". In: *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications.* 2010, pp. 22–35.

[8] J. C. Knight and N. G. Leveson. "An experimental evaluation of the assumption of independence in multiversion programming". In: *IEEE Trans. on Software Engineering (TSE)* 1 (1986), pp. 96–109.

[9] D. Mendez, B. Baudry, and M. Monperrus. "Empirical Evidence of Large-Scale Diversity in API Usage of Object-Oriented Software". In: *Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM).* 2013.

[10] C. R. Myers. "Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs". In: *Physical Review E* 68 (Oct. 2003), p. 046116.

[11] D. E. Perry, A. A. Porter, and L. G. Votta. "Empirical studies of software engineering: a roadmap". In: *Proceedings of the conference on The future of Software engineering.* ACM. 2000, pp. 345–355.

[12] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2 (2009), pp. 131–164.

[13] D. I. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. "A survey of controlled experiments in software engineering". In: *IEEE Transactions on Software Engineering* 31.9 (2005), pp. 733–753.

[14] R. Wettel, M. Lanza, and R. Robbes. "Software systems as cities: a controlled experiment". In: *Proceedings of the 33rd International Conference on Software Engineering.* ACM. 2011, pp. 551–560.