Introduction to Empirical Software Engineering

Martin Monperrus
University of Lille
June 25, 2015

This document presents an introduction to empirical software engineering. It has been first prepared for a course at EJCP 2015, a French summer school on software engineering and programming languages.

The main URL of the course is http://www.monperrus.net/martin/ejcp2015

This document is constantly improving. I would appreciate your feedback.

1 Background

1.1 Short bio

I am an associate professor at the University of Lille (France) and a member of INRIA’s research group SPIRALS. In 2008-2011, I was a research associate in Mira Mezini’s group at the Darmstadt University of Technology (Germany). I received a Ph.D. from the University of Rennes (France) in 2008, for which I was supervised by Jean-Marc Jézéquel, Joël Champeau and Brigitte Hoeltzener. In 2005-2008, I was a research assistant at ENSIETA (Brest, France), thanks to a research scholarship from DGA and CNRS. Previously, I worked as a software engineer for CMB. I spent 6 months working with Yoshua Bengio at the University of Montréal (Canada) in 2004 for my master’s thesis. I received a M.Sc. and an engineering degree in computer science from the Compiègne University of Technology (France) in 2004.

1.2 My past work in empirical software engineering

My research has a large empirical flavor. First, because I like to target real-life problems, so I’m always confronted to the complexity and richness of reality. For instance, I’ve contributed to a static analysis for the Android framework [1]. Second, because I’m convinced that we, software engineers, manipulate something (software) that is of different nature and complexity to every other human-made artifacts. We neither understand nor master all aspects of this thing, and we need to better understand its essence and its implication for engineering [10]. Third, because I value curiosity by itself, which sometimes leads to observe just for the sake of observing [12, 3, 11].

1 http://ejcp2015.inria.fr/
2 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)
- 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.

3 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.

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

For example:

- Observation of artifacts: What is distribution of software dependencies?
- Explanation of phenomena: Why is the distribution expolential? [13]
- 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 multiversion programming true? [9]
- Is static typing really good? [8]
- What is the diversity of module usages? [11]
- Does Github change something in open-source processes? [3]
- <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

4 Types of empirical software engineering

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” [4]. For example using a tool vs not using a tool. See the TSE survey on this topic [16].

- an hypothesis
- tasks
- subjects (who?)

A concrete example: “Software Systems as Cities: A Controlled Experiment” [17].

- 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.

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, software engineering?),

- 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)

Even if a controlled experiment is possible:

• it may be very costly (pay real developers for 2 months?)
• it may be 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” [15].

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

• Falsification [6] (excellent paper)

The steps of case studies (Five of [15] plus one):
• 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

An example, “Pair Programming and Software Defects – A Large, Industrial Case Study” [2]

• 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” [6]. Excerpts:

• Misunderstanding: General, theoretical (context-independent) knowledge is more valuable than concrete, practical (context-dependent) knowledge (no, depth and context matters).
• Misunderstanding: One cannot generalize on the basis of an individual case; therefore, the case study cannot contribute to scientific development (no, falsification require 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 case studies. (no, a good narrative can have a much bigger impact).

A survey is the study of the characteristics of a broad population of individuals [4].

• Different from common meaning of “literature survey”.
• A collection of standardized information from a specific population [15]
• Emphasis on large sample (as opposed to case study?)
• Not necessarily by means of a questionnaire or interview [15]
• Usually a synonym of “empirical study”.
• If done in the field, aka field study

The steps of survey research are the same as for case studies. But the main differences are:
Figure 2: Excerpts of the results of Gabel and Su [7]

- 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 [7]:

- 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
- 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

5 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” [14].

• what factors are uncontrolled?
• mitigation: ?

The **reliability** “focuses on whether the study yields the same results if other researchers replicate it” [4]. 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).

### 6 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 higher 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

For computer scientists, all statistics can be understood and validated with Monte-Carlo simulations, e.g. [http://www.monperrus.net/martin/monte-carlo-spearman](http://www.monperrus.net/martin/monte-carlo-spearman).

### 7 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.

• An experimental evaluation of the assumption of independence in multiversion programming [9]
• An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time [8]
• My hairiest bug war stories [5]

### 8 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, 3
engineering, 2
falsification, 6
science, 2
software engineering, 2
validity, 9
actionable, 4
case study, 6
construct validity, 9
controlled experiment, 5
external validity, 9
field study, 7
internal validity, 10
mining software repositories, 3
reliability, 10
software evolution, 3
software maintenance, 3
survey, 7
threats to validity, 9
References


