The type of evidence produced by empirical software engineers

Judith Segal
Department of Computing
Faculty of Maths and Computing
The Open University
Walton Hall
Milton Keynes
MK7 6AA
UK
+44 (0)1908 659793
j.a.segal@open.ac.uk

Antony Grinyer
CSW Group Ltd
4240 Nash Court
Oxford Business Park South
Oxford
OX4 2RU
ENGLAND
+44 (0) 1865 337400
antony.grinyer@csw.co.uk

Helen Sharp
Department of Computing
Faculty of Maths and Computing
The Open University
Walton Hall
Milton Keynes
MK7 6AA
UK
+44 (0)1908 653638
h.c.sharp@open.ac.uk

ABSTRACT

This paper reports on the research published between the years 1997 and 2003 inclusive in the journal of Empirical Software Engineering, drawing on the taxonomy developed by Glass et al. in [3]. We found that the research was somewhat narrow in topic with about half the papers focusing on measurement/metrics, review and inspection; that researchers were almost as interested in formulating as in evaluating; that hypothesis testing and laboratory experiments dominated evaluations; that research was not very likely to focus on people and extremely unlikely to refer to other disciplines. We discuss our findings in the context of making empirical software engineering more relevant to practitioners.

Categories and Subject Descriptors

D.2. [**Software Engineering**]: *Empirical software engineering, evidence.*

General Terms

Experimentation

Keywords

Empirical software engineering; research taxonomy; evidence; field studies.

1. INTRODUCTION

In [6], we argued that the gap between empirical software engineering and software engineering practice might be lessened if more recognition were afforded to the following two points:

- Evidence from case or field studies of actual software engineering practice is essential in order to understand and inform that practice.
- The nature of evidence should fit the purpose to which the evidence is going to be put. For example, quantitative evidence might be necessary to convince a manager to introduce some change in working practices; a rich case study might persuade developers to accept such a change.

In this paper, we investigate the nature of the evidence published over a period of 7 years in the academic journal, Empirical Software Engineering (see http://journals.kluweronline.com/). Our investigation was inspired in part by questions arising from the argument above – what is the prevalence of case and field studies of software engineering practice? Is there a wide variety in the types of evidence reported in the field of empirical software engineering? – and in part by the work of Glass, Vessey and Ramesh, as reported in [3]. These latter sought to describe the current state of software engineering research by scrutinizing 369 papers representing a sample of those papers published in 6 top software engineering journals over a period of 5 years. The papers were classified along the following dimensions:

- The topic covered (for example, algorithms, data structures; organisational issues);
- The research approach. This was divided into the following categories: descriptive; formulative (for example, of guidelines, methods, algorithms, models, etcetera) and evaluative. This latter included the subcategories of positivist and interpretivist. Positivist research assumes the existence of an objective measurable reality and the independence of the researcher and object of research. It frequently takes the form of hypothesis testing, referred to as 'Evaluative deductive' in this paper. Interpretivist research, on the other hand, argues that our understanding of reality depends on how we interpret our perceptions in the light of our experience: that is, the object of research and the researcher are not separate. A field study in which our understanding of reality emerges through social interactions is a typical interpretivist study.

- The research methods. These included positivist methods such as laboratory experiments and interpretivist methods such as field studies.
- The reference discipline. Just as civil engineering (say) is an applied science informed by the pure sciences (reference disciplines) of mathematics, geology, chemistry, physics etcetera, so software engineering is an applied science potentially informed by pure sciences such as mathematics, psychology, sociology, etcetera.
- Units of analysis. These included categories such as the profession; the group; individuals; computer system, computer element such as program.

Glass et al. found that

- The spread of topics was broad (though a closer look at their results shows that less than 3% of the papers were on organisational and societal topics. It appears that the term 'broad' refers only to technical topics).
- As to research approach, over half the papers were formulative; a further 28% were descriptive and only 14% evaluative (4% evaluative deductive; less than 1% interpretative). This is consistent with the results of Tichy et al. [8], who commented on the lack of experimental evaluation in Computer Science publications in the early 1990s.
- Research methods were dominated by conceptual analysis, proof-of-concept and mathematical analysis, which together accounted for nearly three quarters of all the papers.
- In 98% of the papers surveyed, reference disciplines were not mentioned.
- Most of the units of analysis were on abstract concepts: only 11% of the papers surveyed focused on people.

Glass et al.'s study did not include the journal Empirical Software Engineering, which we assumed had published a good portion of empirical software engineering research since its inception in the middle 1990s. We thus determined to carry out a classification on papers published by this journal, similar to the one carried out by Glass et al.

Given our argument in [6] and described above, we hoped to find many field studies of software engineering practice, and a variety of different types of evidence to fit the variety of purposes to which such evidence could be put. As we shall see, our hopes were not realized.

2. METHODOLOGY

2.1 The material

We classified all those 119 papers which appeared in Empirical Software Engineering between 1997 and 2003 inclusive, excluding only those that were pure polemic. Our classification scheme was based on that of Glass et al. with some amendments to fit our own purposes. These amendments are as follows:

- In 'units of analysis', where people were involved, we determined whether they were practitioners or students; what the nature of their activity was, and whether they were interacting with real-world or artificial systems. It was important for our purposes to differentiate between papers where the focus was on students and 'toy' systems, and those where the focus was on practitioners developing, testing or maintaining systems in the real world.
- We included another category to record whether the authors of a particular paper were academics, practitioners or both. We based our decision on the email addresses provided. We shall comment in section 3 on the inherent ambiguity in this.

2.2 The method

Two coders classified all the 1997 papers and then came together to discuss their individual classifications, thus coming to a shared understanding of the classification scheme. Papers from the years 1998 – 2003 were then coded independently, though the two coders continued to discuss how the coding scheme might be interpreted and extended as issues arose. For each paper, we hoped to gain sufficient information to complete the coding by reading its abstract and conclusions; if this did not suffice, we speed read the paper to determine (for example) whether there was mention of a hypothesis or of statistical testing (both indicative of an evaluative deductive research method), or whether the participants (if any) were students or practitioners. Only if the required information was still not forthcoming did we carefully peruse the paper.

We accept that the resulting coding cannot be completely objective, not least because of the inherent ambiguity of the classification scheme. For example, if a paper describes the use of some metric in order to distinguish between two testing schemes, it is difficult to know if the authors' intention was to focus on the metric or on testing. Nevertheless, the two independently coded sets of papers showed a high measure of agreement, averaging 83% across the classification categories. The papers were then handed to a third coder (who had not been party to the earlier discussions), who recoded those papers on which the first two coders disagreed, according to the agreed classification scheme.

3. RESULTS

Here, we report our findings.

3.1 Research topic

5 topics (at the level of granularity provided in [3]) covered over three quarters of the papers, as shown below.

Table 1. Topic

Software life-cycle/engineering (incl. requirements, design, coding, testing, maintenance)	33%
Measurement/metrics (development and use)	19%
Process management	10%
Tools (incl. compilers, debuggers)	8%

Computing Research (that is, meta-level issues, such as discussions about methodologies)	8%

No other topic was covered in more than 4% of the papers. Analysing the most common topic (software life-cycle/engineering) in more detail, almost half of the papers covering this topic were concerned with maintenance and a further third with review and inspection. Perhaps this bias is a reflection of the workshops held and journal special issues published during the period under consideration.

3.2 Research approach

The research approaches seen in the papers we scrutinised are recorded in Table 2 below.

Table 2. Research Approach

Evaluative deductive	46%
Descriptive	13%
Formulative (process, method, algorithm)	10%
Formulative (model)	8%
Formulative (guidelines/standards)	5%
Evaluative – other	5%
Review of literature	5%
Evaluative – interpretive	2%

No other research approach was seen in more than a single paper. Given that we are considering empirical software engineering, we were surprised that only just over a half of the papers featured evaluation (though mindful that this was about four times the number noted in the general software engineering literature by Glass et al.). Of these, evaluative deductive – testing hypotheses in a very positivist tradition – dominated.

3.3 Research method

Table 3 records the research methods used.

Table 3. Research Method

Laboratory experiment (human subjects)	29%
Data analysis	15%
Case study	13%
Descriptive/exploratory survey	12%
Laboratory experiment (software)	7%
Concept implementation (proof of concept)	7%
Meta-analysis	6%
Literature review/analysis	6%

No other research method was seen in more than 2 of the papers. The dominance of laboratory experiments (in 36% of the papers) reflects the dominance of the evaluative deductive, hypothesis testing, research approach.

3.4 Reference discipline

The next table indicates that empirical software engineers, like software engineers in general as noted in [3], tend to be insular and take little cognisance of research in other disciplines.

Table 4. Reference Discipline

None	85%
Psychology	6%
Statistics/mathematics/computational science	5%
Social Science	4%

3.5 Units of analysis

In table 5, we consider units of analysis, that is, the entities which were the focus of the papers.

Table 5. Units of Analysis

Real-life computer system/data/project	36%
Profession	24%
Individual students	18%
Individual practitioners	7%
Teams of students	4%
Artificial laboratory computer system/data/project	3%
Team of practitioners	2%
Other	6%

Here, we see that only 31% of the papers had people at their focus (and given the results of Table 3, presumably the vast majority of these concerned laboratory experiments, that is, people in an artificial setting). On the other hand, over a third of the papers focussed on real-world systems.

3.6 Who wrote the papers?

Finally, we consider authorship of papers in Table 6.

Table 6. Paper author(s)

Academic authors alone	73%
Mixture of academic and practitioner authors	16%
Practitioners alone	11%

We noted in 2.1, that we determined whether an author was an academic or practitioner simply on the basis of his/her email address. We recognise that this does not take account of practitioners being seconded to academic institutions and vice versa, nor the fact that the distinction between an academic department and a research department in an industrial setting may not be clear-cut. Nonetheless, academic authors clearly predominate.

4. DISCUSSION

Our results reveal that, based on an analysis of 119 papers, comprising nearly all the output of the journal Empirical Software Engineering between the years 1997 and 2003 inclusive, empirical software engineers are

1. Somewhat narrow in topic, with measurement/metrics, maintenance, and review and inspection accounting for about half the papers;
2. Almost as interested in formulating (processes; models; guidelines), describing and reviewing, as in evaluation;
3. Far more likely to evaluate using hypothesis testing than any other method;
4. Likely to do laboratory experiments (described in over a third of the papers);
5. Very unlikely to refer to any other scientific discipline;
6. Not very likely to focus on people.

This last point is, we think, unfortunate, in that software engineers are in general agreement as to the importance of people factors in the successful practice of software engineering ([5], [1], [2]). Given this importance, we feel that software engineers might take more cognizance of reference disciplines in the social sciences such as psychology and sociology (cf. point 5 above).

With respect to point 4 above: cognizant of the fact that software engineering takes place within a context – a particular team of people are involved; a particular application environment; a particular development environment; a particular organisational, social, market environment – we would encourage empirical software engineers to consider using research methods which take account of the complexity of context, such as field studies, rather than methods which factor out the effect of context, such as laboratory studies. Such studies can make use of natural controls to confirm/disconfirm a hypothesis ([4]), cf. point 3.

An argument often made against field studies is that they cannot be replicated – but neither can a software engineering activity in the real world (one cannot dip one's toes into the same river twice!). Validation of such studies can be based not on replication of the study but on replication of the interpretation: the question to ask is, would other researchers from the same scientific cultural tradition as the original researcher(s) and given the same data, come to the same conclusions?

We recognize the practical difficulties of involving practitioners in research and performing field studies ([7]). It is clearly easier to involve students in a laboratory experiment, and as Tichy says in [9], graduate students in computer science can be more technically adept and up to date than practitioners. However, graduate students are *not* practitioners: they do not work in the same organisational and professional context; they are not subject to the same pressures. It is plausible that there are circumstances where laboratory experiments with students might yield results which can inform practice (for example, experiments concerned with individual cognition such as one designed to test whether *this* representation is easier to comprehend (in some sense) than *that*). Nevertheless, we urge researchers to scrutinize the external validity of their laboratory experiments – do the results of their research really have the potential to inform the richly contextualized practice of software engineering?

In the above, we have argued for the importance of field studies in empirical software engineering within the positivist tradition of hypothesis deduction and testing. In [6], we argued for their importance in constructing within the interpretivist tradition an understanding of the actual practice of software engineering. We argued that such an understanding on the part of empirical software engineers may have an important role to play in bridging the gap between them and practitioners.

Although we are disappointed by the lack of field studies and especially interpretive field studies, we are pleased by two facets of current empirical software engineering research. Firstly, we applaud the efforts of many of the researchers surveyed to consider real-life systems, data and projects (see Table 5). Secondly, our analysis presents evidence that empirical software engineers are 'reflective practitioners'; they reflect upon their discipline and how it might be improved (witness the fact that a quarter of the papers surveyed focused on the profession, as recorded in Table 5). It remains to be seen whether this reflection will influence empirical software engineering practice.

5. REFERENCES

[1] Boehm, B., Turner, R.. *Balancing Agility and Discipline.* Addison-Wesley, 2004.

[2] Cockburn, A. *Agile Software Development*. Addison Wesley, 2002.

[3] Glass, R.L., Vessey, I., Ramesh, V. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44, 491-506, 2002.

[4] Lee, A.S. A scientific methodology for MIS case studies. *MISQ*, 33-50, 1989.

[5] Seaman, C. Methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering,* 25(4), 557-572, 1999.

[6] Segal, J. The nature of evidence in empirical software engineering. *Proc. Intl. Workshop on Software Technology and Engineering Practice (STEP 2003),* IEEE Computer Society Press, 40-47, 2003.

[7] Sim, S.E., Singer, J. and Storey, M-A. Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research. *Empirical Software Engineering,* 6, 85-93, 2001.

[8] Tichy, W., Lukowicz, P., Prechelt L., and Heinz, E. Experimental Evaluation in Computer Science: A Quantitative Study. *J.Systems Software,* 28, 9-18, 1995.

[9] Tichy, W. Hints for reviewing empirical work in software engineering. *Empirical Software Engineering,* 5, 309-312, 2000