Empirical Methodologies in Software Engineering

Ray Dawson
Loughborough University
UK
R.J.Dawson@lboro.ac.uk

Phil Bones
University of Canterbury
New Zealand
P.Bones@elec.canterbury.ac.nz

Briony J Oates
University of Teeside
UK
B.J.Oates@tees.ac.uk

Pearl Brereton
Keele University
UK
o.p.brereton@keele.ac.uk

Motoei Azuma
Waseda University
Japan
azuma@azuma.mgmt.waseda.ac.jp

Mary Lou Jackson
Vancouver Island Health Authority, Canada
mljackson@shaw.ca

Abstract

The collection and use of evidence in Software Engineering practice and research are essential elements in the development of the discipline. This paper discusses the need for evidence-based software engineering, the nature of evidence in its various forms and some of the research methodologies used in other disciplines for the collection of evidence, which are also relevant to software engineering. Two frameworks or models are proposed which illustrate the relationships between the methodologies discussed. In particular, the paper highlights the importance and roles of both positivist and interpretivist methods of investigation.

1. Introduction

This paper examines the different approaches that can be taken to the generation of empirical evidence to support the theory and practice of software engineering. First we explain why empirical evidence is needed. The underlying philosophies of different empirical approaches are then explained, the different methodologies are compared and their advantages and difficulties are identified. Two frameworks are devised to show how the different methodologies relate. The paper concludes that all the different methodologies have their place in software engineering and each approach has value for the software engineering practitioner.

2. Why Take an Empirical Approach?

Software engineering is essentially the realm of the practitioner. The discipline aims to enable the successful production of software, where the criteria for success can include such quality characteristics as accuracy, appropriateness, functionality, reliability, usability, efficiency, maintainability and portability, as well as timeliness, cost effectiveness, customer satisfaction or even political expedience.

As software engineering is so dependent on the practitioner it suffers from all the variation and unpredictability associated with people, who have their individual strengths and weaknesses, insights and blind spots. Equally, as software products are produced in the real world, every software project is influenced by the environment in which it takes place. This variation in circumstances means that guiding principles are hard to establish and, consequently, the discipline of software engineering is often referred to as an art or craft. This can lead to individuals forming their own ideas for working practice based on a mixture of their own experiences, hearsay from others and general folklore and myths (e.g. [1]).

The wide variety of software products adds to the problem. Software engineering includes large scale, mission-critical, real-time systems software, interactive off-the-shelf software, Web-based e-commerce software, and embedded software. Each category of software can have different quality requirements, and therefore needs a different approach.

If software engineering is to live up to the word "engineering" in its title it needs to move towards being a rigorous discipline. For this to be the case, observations of working practices need to be collected, theories and hypotheses have to be formed to explain the observations and new ideas must be advanced. These theories, hypotheses and ideas need to be tested to produce evidence of their worth (or lack of worth).

Any observation, such as "project X took longer to complete than project Y", can be regarded as evidence, but it is clear that it is not satisfactory without more detail, and more detail normally requires the measurement and capture of data. For example, it would clearly be beneficial to have an indication of the size and complexity of each project, the number of people working on it, their roles and experience, and the detail of hardware and software tools available. Many of these aspects have measurable attributes which convey a much greater understanding if they are known. For instance, the data that Project X involved twenty five thousand lines of code and fifty database tables whereas Project Y involved two thousand lines of code and six database tables, conveys a much greater understanding of the relative size and complexity of the two projects than a statement that Project X is "much bigger" than Project Y.

A further advantage accruing from the collection of empirical data is that it can lend itself to statistical analysis. When anecdotal evidence is relied upon, the conclusion that projects with more lines of code take more effort to complete could be thrown into doubt if a single exception is found. The application of statistics could, however, give an indication of the significance of the results and show that there is an acceptable degree of confidence in the conclusions drawn. Qualitative data analysis, while lacking the intuitive appeal (to many software engineers) of numerical precision, also plays a role in identifying themes, attitudes and interpretations, which help to describe and justify the practice of software engineering.

3. What is Evidence?

There are many approaches to collecting data. Anecdotes, case studies, action research, surveys and controlled experiments can all yield empirical data, but of what value is that data and could one approach be seen as being superior to another? To answer these questions, it is necessary to have an idea of what we are looking for, and, indeed, how we will know when we have found it. For empirical research, the outcome of a study depends very much on whether the researcher is a positivist or an interpretivist (sometimes called anti-positivist) in their approach. The positivist looks for irrefutable facts and fundamental laws that can be shown to be true regardless of the researcher and the occasion. For example Sir Isaac Newton's laws of gravity have been shown to be true many times by many researchers. The positivist philosophy is the necessary and obvious approach in the pure sciences where the pursuit of such fundamental laws is the norm. In such areas of research it is the general practice to formulate a hypothesis that is tested via controlled experiments that isolate independent variables, enabling a cause and effect to be established. Other researchers then attempt to replicate the experiments, and if the same causal relation is repeatedly established, the hypothesis is accepted as proven and therefore 'true'. Kitchenham *et al* [2] have provided useful guidelines for the conduct of such controlled experiments in software engineering.

Software engineering is not a pure science, however. It is certainly arguable whether a positivist approach can ever be appropriate for a discipline so dependent on people and the environment, where carefully controlled and repeatable experiments, which change only one variable at a time, are often difficult or impossible to design and implement. For this reason many researchers favour an interpretivist approach to software engineering research. Interpretivists believe all research must be interpreted within the context in which it takes place where even the researcher must be considered part of the context. This approach makes absolute truths difficult, if not impossible, to find, as every context is likely to be different. Interpretive studies do not therefore, prove or disprove a hypothesis. Instead they try to understand phenomena through the meanings and values that people themselves assign to them, and produce a rich and detailed description of the phenomenon under investigation. This description can lead to new, empirically grounded theories. Case studies, for example, often fall into this category of research.

The problem with interpretivist research is that it is difficult to prove anything – a problem, at least, for a world where the scientific model of research, and hence the expectation of proof, often dominates. For example, if a new methodology is tried in a case study, the only thing that can be shown for certain is that it *can* work. The methodology can produce impressive results, enabling a process to be completed in a very short time, for example, but it is still not possible to say whether the results are entirely due to the methodology used. It may be that, say, the practitioners were particularly capable in the test performed. This could be due to something as simple as the higher motivation achieved by a pay rise! Furthermore, it is not possible to prove that the tested methodology would even work at all on a different occasion. As Checkland, creator of Soft Systems Methodology, writes:

"… if a reader tells the author 'I have used your methodology and it works', the author will have to reply 'How do you know that better results might not have been obtained by an *ad hoc* approach?' If the assertion is: 'The methodology does not work', the author can reply, ungraciously but with logic, 'How do you know the poor results were not due simply to your incompetence in using the methodology?" [3]

Instead of proving a hypothesis by means of isolating factors and establishing cause and effect, interpretive research seeks to explore and explain how all the factors in the object under study (a software development team, an organisation etc.) are related and interdependent.

So what constitutes evidence? How can empirical data be used to support the teaching, learning and research of software engineering principles and methodologies? Positivism could provide the fundamental truths on which to build a discipline but it is difficult, perhaps impossible, to perform in a software engineering context. Interpretivism is a possible approach but may not be able to provide the generality needed for a widely practised discipline. To overcome this dilemma it would be instructive to examine what other disciplines can teach the software engineering community in this respect.

4. The Approach of Other Disciplines

The medical world has a long standing track record of research in which the highest standards are essential as the potential for tragic disasters is so great. A new drug undergoes extensive laboratory experiments, but still cannot be released until it has had extensive trials with hundreds or thousands of volunteers under strict double blind test conditions. These rigorous experiments and trials are in the positivist vein, aiming to determine the underlying truth of what is or is not safe and effective, and should, in theory, be reproducible. The high numbers of volunteers involved in the trials allow statistical analysis to be performed regarding the effectiveness of a treatment, overcoming the variation found between individuals. However, while this level of rigour is commendable it is not always possible and, as a result, there is also a wealth of other reported cases that involve small samples or even individuals. An example, illustrating this type of research in medicine, is given by the early heart transplant operations. These operations were extensively reported and studied and became an important source of knowledge, yet were clearly subject to the individual patients, the medical team performing the operation and the conditions in which the operations took place. The knowledge gained therefore comes from an interpretivist perspective in these cases, with the rich and detailed understanding of each operation (i.e. case study) gradually accumulating into a body of knowledge about how, and in what circumstances, such operations might succeed or fail.

The legal world has a long established tradition of basing decisions on case law, (i.e. decisions are based on the outcomes of previous "case studies"). This is particularly important in the UK where case precedents have enabled the law to operate without the need for a written constitution. This use of case law shows the value of interpretivist research but also illustrates some of the limitations. In law, if it can be shown that conditions are substantially different, a different decision can be made. The case precedents then become more refined for the future, indicating one possible outcome for the original conditions but with exceptions for the conditions corresponding to the later precedent. This again shows how case studies can be built up over time, enabling us to refine and improve the knowledge they provide.

In crime detection evidence is pieced together to reach an overall conclusion that would not be possible from each individual bit of evidence. A detective needs to show that an accused person had to be in the right place at the right time, to have the right opportunity, the tools and the right motivation to undertake the crime. Often, the evidence when viewed one bit at a time can be described as "circumstantial", but viewed together the evidence can be overwhelming. For the software engineering community this suggests that while individual pieces of evidence regarding, say, the effectiveness of a methodology may not be conclusive, it is important to acknowledge and record the evidence as it could become part of a much bigger picture later.

Closer to the area of software engineering are the worlds of industrial engineering, knowledge management and information systems. Industrial engineering, like software engineering, aims to produce high quality products at the lowest possible cost. Traditionally the target products of industrial engineering have been hardware while software engineering handles software. However, recently many target products of industrial engineering include the software of embedded computers. Examples are cars, driving navigators, cell phones, DVD players etc. Like medicine, industrial engineering uses a combination of positivist approaches such as controlled experiments and statistical analysis, plus observations on real-life projects and case studies.

In knowledge management the advantages of "story telling", have long been accepted. This indicates that the anecdote, a form of interpretivist research that is even less formal than the case study, has value. In knowledge management it is important to capture and then make available all knowledge whether based on a rigorous study or simple anecdote.

Information systems' defining feature, or raison d'être, is the study of the development, use and effects of

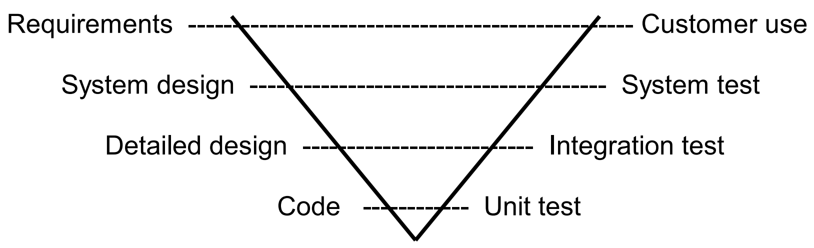

Figure 1. The "V" model for software testing

information systems, usually computer-based, in organisations, groups and society, that is, in their social context. Unlike software engineering, the majority of information systems research is empirical [4]. It initially concentrated on positivist research, but the limitations of such methods for investigating human activities have long been recognised in the information systems discipline [5], and the use of interpretive methods has gradually increased [6,7,8]. Information systems research therefore has a long tradition of both positivist and interpretivist research.

These different disciplines therefore teach the software engineering community that both positivist and interpretivist research have a role to play, and that evidence can be gradually accumulated over a period of time from multiple studies.

The disciplines of software testing and quality assurance give us a further perspective. Testing (and QA) is performed at several levels within a software development project. Four such levels are shown in Fig. 1 in the form of a "V" model.

The life cycle of a software-based system, in its simplest form, tends to follow the V down the left arm and up the right. However, the depth of the V also gives an indication of the 'reality gap' between the testing environment and the target operating environment. At the bottom of the V the unit test involves testing the code outside its planned operational software environment. Without the rest of the code the unit can still undergo some tests but it is impossible to judge how the rest of the code may affect it. Uncovering errors at this level is relatively inexpensive. Travelling up the right arm of the V moves closer to the real operating environment. Integration testing includes more, but not all, of the software. System testing involves all the software, but in an artificial environment where the testers are not the end users. It is only when the software is put to actual customer use that the full operating conditions are experienced and it is only at this level of reality that certain errors (the requirements errors) tend to be found. The cost of uncovering these errors tends to be very high.

The relevance of the software testing perspective to empirical research is in the trade-off between realism and rigour. If such a "V" model were applied to empirical research, the lowest level would correspond to a rigorous experiment from the positivist camp, necessarily performed in a tightly controlled environment, but a long way from representing the real world of software development. At the other extreme (corresponding to high on the "V" model) would lie the truly interpretivist case study, instructional and firmly planted in the real world, but lacking rigour in the scientific sense.

In the next section we extend the idea of providing a model to explain the distinctions between different empirical methods and attempt to incorporate lessons from other disciplines.

5. A Framework for Empirical Methodologies in Software Engineering

In order to appreciate the contributions of the different approaches to empirical research in software engineering it is helpful to create a framework to show how the approaches relate to each other.

Fig. 2 shows a pyramid giving the positivist – interpretivist spectrum with positivist methodologies at the top and interpretivist methodologies at the base.

Why a pyramid? The framework is depicted as a pyramid for two reasons. Firstly, the pinnacle of the pyramid represents a goal to which many researchers may aspire: to discover fundamental, irrefutable truths which other researchers can reproduce and confirm or refute. The base of the pyramid on the other hand is placed firmly on the ground representing the practical constraints within which software engineering research must operate. Secondly, the nature of the research and constraints means that the positivist research at the top of the pyramid is rarely achieved, whereas the interpretivist research, especially anecdotes, is far more common, so the area at each level represents the relative quantity of research undertaken. The intention is not to imply that the research

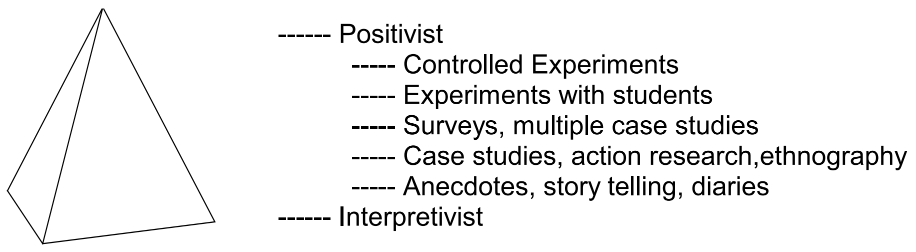
------ Positivist
----- Controlled Experiments
----- Experiments with students
----- Surveys, multiple case studies
----- Case studies, action research,ethnography
----- Anecdotes, story telling, diaries
------ Interpretivist

Figure 2. The pyramid of empirical research types

at the top is in any way superior to that below; all software engineering research has its value.

We can briefly summarise the different research methods in our pyramid. Experiments, as discussed earlier, concentrate on standardising all variables except one, and observe what happens when that single variable is changed, so that cause and effect can be established (see, for example, [2]). For example, in a medical experiment, one group may receive a new drug, and another identical group is given a placebo. Significant variations in outcomes for the two groups should be attributable to the drug. Because of the difficulties of controlling all variables in software engineering or establishing an identical control group, experiments and the confirmation or refutation of hypotheses are hard to achieve, but desirable, so experiments are placed at the top of the pyramid. "Experiments with students" are positioned below them. Tests on students have a number of advantages. There are often large numbers of students available, allowing many tests to be undertaken in parallel, and this can allow alternative methods to be tried and compared. The students themselves are of a known ability (e.g. as measured by their recent grades), a narrow range of experience, and all should have similar motivations (the desire to do well in their course of study). The academic environment can lend itself to the control of experiments, keeping requirements, team membership, hardware and software support constant, for example, as it is not subject to the commercial pressures experienced in industry. However, the same environment that enables controlled experiments to be undertaken in academia, also contributes to the limitations to this type of research. The fact that conditions can be kept constant immediately makes it less real than would be experienced in the very dynamic, changing environment of a typical software company with, probably, a mixture of experience and motivations. Nevertheless, research with students is still important as it provides knowledge which, when taken with other sources, can build an overall picture in which software practitioners can have confidence. The relatively high position on the pyramid indicates both the positivist nature of the experimental design and the relative paucity of such studies reported in the literature.

Surveys are a systematic gathering of information from a large sample, looking for general trends or patterns [9]. They involve wide and inclusive coverage, usually at a specific point in time. The data is analysed using statistics. Careful selection of the sample to be surveyed allows conclusions to be drawn about a wider population than the sample, but the results usually have a confidence level of less than 100%, so they are placed lower than experiments.

Case studies are a rich account of a particular experience, event or situation, often taking a longitudinal view [10,11]. The findings are often dependent on the particular context of study, and may not be transferable to any other setting. They are more common than experiments in software engineering practice, if not in the literature, but unlikely to produce irrefutable truths, so they are lower in our pyramid. Multiple case studies, while still context-dependent, can identify recurring themes, which may eventually become software engineering 'laws', so are higher than single case studies.

Action research involves practitioners researching into their own practice in an iterative cycle of planning, acting and reflecting, with the twin aims of contributing both to the practical concerns of people in an immediate problematic situation and to the goals of science [12,13]. Ethnographic research comes from anthropology where a researcher would spend a significant amount of time in the field. Ethnographers immerse themselves in the lives of the people they are interested in and seek to place the phenomena studied in their social and cultural context [14,15]. Like single case studies, findings from action research and ethnography are dependent on their context and may not be transferable to other settings. They are therefore placed lower in our pyramid.

Finally anecdotes, storytelling and diaries capture the data and interpretations we all can and do discover. Anecdotes are often told when software engineers meet, they are easy to produce, but their insights may be unique to individuals. They are therefore on the ground level of our pyramid.

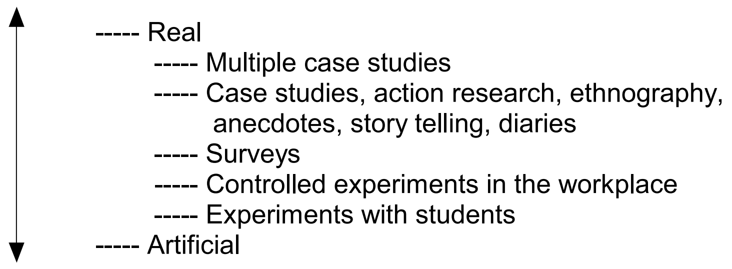
----- Real
----- Multiple case studies
----- Case studies, action research, ethnography, anecdotes, story telling, diaries
----- Surveys
----- Controlled experiments in the workplace
----- Experiments with students
----- Artificial

Figure 3. The relevance of research types to the real world

The placing of the different types of research within the pyramid is, to some extent, subjective, and in reality it may be better to represent each with a range of levels. A case study, for example, may be examining results from the customer base of a particular company. This customer base could be huge, representing a high proportion of the potential population, in which case the findings may be considered to be further up the pyramid than a company with a small, specialised customer base. Similarly, the placing of surveys and multiple case studies will depend on the numbers involved.

An alternative framework is to place the research types on a scale of reality or relevance to the real world as is given in Fig. 3.

The placing of each research type here is even more subjective than the placing within the positivist-interpretivist pyramid. For example, the controlled experiment within the workplace is put low down on the reality scale as the controlled environment is bound to affect its relevance. However, the degree to which this is so will vary from one experiment to another. The positioning of surveys is because of the inevitable biases that occur in the questions or the sample population, but again this is bound to vary. Nevertheless, the scale is useful to highlight the fact that different research types can vary not only in their rigour but also in their relevance and that the most rigorously determined research results may not be useful simply because the rigorous conditions imposed can themselves reduce the relevance to the software engineering practitioner.

7. Conclusion

In this paper we have examined the different approaches that can be taken to gain empirical evidence to support the theory and practice of software engineering. We conclude that all the different research methodologies have their place in software engineering, and each approach has value for the software engineering practitioner. Similar views about the need for multiple approaches and the accumulation of evidence over time are apparent in other disciplines. Our two frameworks show how the different approaches relate to each other and to the real world.

The recognition of two different paradigms of research – positivism and interpretivism – is also important to empirical research in software engineering. Again we argue that both types of research are important if software engineering research is to be both rigorous and relevant.

7. Acknowledgements

This paper has been written as a result of the deliberations at the STEP Workshop on “Where is the Evidence? The Role of Empirical Practices in Software Engineering” held in Amsterdam in September 2003. The authors, who all attended the workshop, would like to acknowledge and thank the other participants for their contributions to the ideas discussed in this paper.

8. References

[1] R. Hirschheim and M. Newman, “Symbolism and information systems development: Myth, metaphor and magic”, *Information Systems Research, 2*(1), 1991, pp. 29-62.
[2] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering”, *I.E.E.E. Trans. Software Eng*., 28(8), 2002, pp. 721-734.
[3] P. Checkland and J. Scholes, *Soft Systems Methodology in Action*, Wiley, Chichester, 1990, p299.
[4] J. Mingers, “The paucity of multimethod research: A review of the information systems literature”, *Information Systems Journal, 13*(3), 2003, pp. 233-249.
[5] E. Mumford, R. Hirschheim, G. Fitzgerald and T. Wood-Harper, *Research Methods in Information Systems: Proceedings of the IFIP WG 8.2 Colloquium, Manchester Business School, 1-3 September, 1984*, North-Holland, Amsterdam, 1985.
[6] D. Avison, “The ‘discipline’ of information systems: Teaching, research and practice”, In J. Mingers & F. A. Stowell (Eds.), *Information Systems: An Emerging Discipline?,* McGraw-Hill, London, 1997, pp. 113-135.
[7] W.J. Orlikowski and J.J. Baroudi, “Studying information technology in organizations: Research approaches and

assumptions", *Information Systems Research, 2*(1), 1991, pp. 1-28.
[8] G. Walsham, "The emergence of interpretivism in IS research. *Information Systems Research, 6*4), 1995, pp. 376-394.
[9] S.L. Pfleeger and B.A. Kitchenham "Principles of Survey Research, Part 1: Turning Lemons into Lemonade", *Software Engineering Notes*, 26(6), 2001, pp16-18
[10] R.K. Yin, Applications of Case Study Research, Sage, Thousand Oaks CA, 2nd edition, 2003a.
[11] R.K. Yin, Case Study Research. Design and Methods. Sage, Thousand Oaks CA, 3rd edition, 2003b.
[12] R.L. Baskerville and A.T. Wood-Harper, "A Critical Perspective on Action Research as a Method for Information Systems Research," *Journal of Information Technology* (11), 1996, pp. 235-246.
[13] P. Checkland, "From framework through experience to learning: the essential nature of action research"" in *Information Systems Research: Contemporary Approaches and Emergent Traditions*, H-E. Nissen, H.K. Klein, R.A. Hirschheim (eds.), North-Holland, Amsterdam, 1991, pp. 397-403.
[14] S.L. Star, *Cultures of Computing*. Blackwell, Oxford, 1995.
[15] J. Van Maanen, *Tales of the Field: On Writing Ethnography*, University of Chicago Press, Chicago, 1988.