Philosophies **2016**, *1*, 87-101; doi:10.3390/philosophies1010087

OPEN ACCESS

philosophies

ISSN 2409-9287

http://www.mdpi.com/journal/philosophies

Article

Rethinking Experiments in a Socio-Technical Perspective: The Case of Software Engineering

Viola Schiaffonati [1,†] and Mario Verdicchio [2,†,*]

[1] DEIB, Politecnico di Milano, Piazza Leonardo Da Vinci 32, Milano 20133, Italy
[2] DIGIP, Università degli Studi di Bergamo, Via Salvecchio 19, Bergamo 24129, Italy

[†] These authors contributed equally to this work.

* Author to whom correspondence should be addressed; E-Mail: mario.verdicchio@unibg.it; Tel.: +39-035-205-2358.

Academic Editor: Vincent C. Müller

Received: 27 July 2015 / Accepted: 9 December 2015 / Published: 11 December 2015

Abstract: Experiments in computing share many characteristics with the traditional experimental method, but also present significant differences from a practical perspective, due to their aim at producing software artifacts and the central role played by human actors and organizations (e.g., programmers, project teams, software houses) involved in the software development process. By analyzing some of the most significant experiments in the subfield of software engineering, we aim at showing how the conceptual framework that supports experimental methodology in this context needs an extension in a socio-technical perspective.

Keywords: experiments; empirical software engineering; socio-technical systems

1. Introduction

The investigation on the disciplinary nature of computing has been conducted from two different perspectives: the subject matter, on one side, and the methodology, on the other. They provide interesting suggestions on how to position computing with respect to traditional sciences (e.g., physics) and engineering disciplines (e.g., electronic engineering). We shift the focus from the status of computing to what we consider a related critical issue, which is indeed connected with the methodological perspective: the lack of a proper conceptual framework dealing with the nature of experiments in computing.

We have already argued that, although there is a general agreement on the fundamental features of the scientific method, the straightforward application of such a framework to computing can lead to oversimplifications that prevent us from achieving a complete picture of the practices in this discipline; see, for instance [1]. Experiments in computing indeed have much in common with more traditional sciences (e.g., the call for replication [2]), but also present significant differences from a practical perspective, due to their aim at producing artifacts (*i.e.*, computing systems, programs, *etc.*) and the key role played by human actors and organizations (e.g., programmers, project teams, software houses) in such production.

If the focus on artifacts in computing has shed light on the need to borrow conceptual instruments from the domain of engineering (besides those of science), the central role played by the human actors involved in the artifact creation process seems to call for a further extension of the relevant framework. Indeed, it has been recently pointed out by [3] that when it comes to socio-technical systems, that is systems whose function depends not only on technological devices, but also on human agents and social institutions, traditional taxonomies of experimental forms may be inadequate.

In this paper, we continue a discussion on computing and experiments narrowing our analysis down to how experiments are conducted in a subfield of computing, namely software engineering (SE), with the aim to show how the traditional experimental framework needs to be revised in an engineering context. We believe there are good reasons that justify our choice. Firstly, SE aims at the study and application of techniques for the design, development, operation and maintenance of software. As any computer system must rely on software, it is clear that SE is in its own right a vast discipline that intersects many, if not all other subfields in computing. Hence, we hope that any findings that we obtain in our endeavor in this context can be seen as having a significant connection to what is carried out in terms of experiments in the whole field of computing. Moreover, contrary to what was stated by some critics (e.g., "Evidence to the decline of scientific methods is found in textbooks on software engineering" [4] (p. 148), more and more attention has been devoted in SE to methodological and experimental issues, as shown by some general works by [5] and the birth of conferences and journals about these topics, like [6] and [7].

We analyzed the 50 most cited papers (according to [8]) from "Empirical Software Engineering: An International Journal" (ESE) in the 2003–2012 decade and selected the most representative papers of the characteristics that make experimentation in SE so interesting from a philosophical perspective, in that they call for a revision of the conceptual framework of the traditional experimental method (We are aware that this is a limited sample and that several experimental results in SE are currently presented in conference proceedings; still, the papers we analyzed suffice for the purposes of this work.). Our focus is on those endeavors that adopted the most rigorous methodology from the scientific tradition, namely the controlled experiment, so that we are able to observe the widest range of instruments for scientific investigation put to the test in the context of SE. Some of these papers [9–12] explicitly refer to the presented work as a "controlled experiment"; others [13–15] only speak of an "experiment", but due to the rigorous procedures adopted, can be considered equivalent from a methodological point of view. In particular, we selected these works because they present the results of controlled experiments performed with human subjects, which makes them examples of one of the most challenging methodological endeavors, where the most rigorous (and traditional) experimental procedures are applied in a context where the human factor plays a primary role and brings in its typical unforeseeable and disruptive effects.

As the aim of this work is to start a discussion on such effects, four papers that present controlled experiments without human subjects [16–19] are out of our scope and were not considered. The rest of the papers consists of 12 case studies [20–31], 11 reviews [32–42], 11 empirical analyzes [43–53], three comparative analyzes ([54–56]) and two field studies [57,58].

Obviously, this analysis is not to be considered exhaustive at all, but it is meant as the first step of a more comprehensive study on methodology in computing. We focus on the experimental method, and we argue that such a method is not sufficient to tackle all of the issues rising in this discipline from a socio-technical perspective. Our hope is to show that there exists a very interesting approach worth exploring, taking the various subfields of computing as starting points. The paper is structured as follows: In Section 2, we analyze a group of selected experiments illustrated in the ESE journal with a particular focus on the issues that may prevent a straightforward generalization of their results. In Section 3, we show how such issues are connected with the nature of SE as a discipline, which includes not only technological artifacts, but also features a significant social dimension. Finally, Section 4 concludes.

2. Social Issues in the Lab

Let us analyze the experiments presented in the papers we are considering, with a specific focus on social issues. In this context, we use the adjective "social" to indicate two different aspects that characterize all of the experiments: the context they refer to and the human actors they involve.

We must begin by taking the context of the software industry into account (by "industry" we mean the complex of organizations devoted to the production of software for commercial use), because it provides the main motivation for the researchers to set up their experiments in the first place: to discover the best practices to create quality software. The criteria that are used to judge such quality seem to refer mainly to what is currently practiced in the context of the software industry.

Here follows a brief description of the controlled experiments, which we present in decreasing order of the number of citations and refer to by the surname of the first author of the relevant paper, accompanied by the motivations for such endeavors, as declared by the authors themselves.

The Genero experiment aimed at investigating whether a collection of measures can be a good predictor of the maintainability of Unified Modeling Language (UML) class diagrams, which are meant to represent graphically the functional structure of a complex program. These indicators would allow software designers to make better decisions in the early stages of the software development life cycle, contributing to the creation of better quality products. The choice to focus the experimental endeavor specifically on the class diagram out of nine diagrams defined in the UML standard is motivated by the fact that surveys in the industry show that "[the class diagram] is perceived by practitioners as the most important diagram type." [10] (p. 518). Moreover, by looking for good predictors of class diagram maintainability, the proponents of the Genero experiment aim to "allow [object-oriented] software designers to make better decisions early in the software development life cycle, thus contributing to the development of better quality [object-oriented] software." [10] (p. 519). The Vokáč experiment replicated a previous experiment aimed at verifying the beneficial effects of design patterns (*i.e.*, proven solutions to design problems organized into reusable software modules). The main motivation is "to support the industrial use of design patterns." [12] (p. 149). The Shull experiment consisted of two replications on software requirements reviewing techniques to verify the advantages of introducing laboratory packages (*i.e.*, structured descriptions of experiments), with the long-term goal of building a "body of knowledge that identifies the benefits and costs of various techniques and tools to support the engineering of software." [15] (p. 112). The Karlsson experiment compared a number of techniques to introduce a priority hierarchy among different requirements when implementing a software system. The aim was to establish which technique showed the least time consumption, the greatest ease of use and the best accuracy. Furthermore, this effort on requirement prioritization is driven by considerations connected with the industry: "When requirements are elicited from several stakeholders, it often yields more requirements than can be implemented at once. (...) [The requirements] need to be prioritised so that the most significant ones are met by the earliest product releases." [11] (p. 4). The De Lucia experiment aimed at assessing the usefulness of an automatic tool in support of traceability recovery, that is the process of identifying functional links between different parts of a complex software project, such as requirements and implemented software modules. The assessment was based on the measurement of the time needed by software engineers to accomplish the assigned tasks with respect to manual identification. This experiment was carried out with two perspectives, one related to the researchers' interest to evaluate the benefit of a tool-based support in the task and the other belonging to "a project manager, who wants to evaluate the possibility of adopting the tool within his/her own organisation, depending on the skills of the involved human resources." [9] (p. 67). The Fenton experiment aimed at developing a causal model (in the form of a Bayesian net) for predicting the number of defects likely to be found in a piece of software during testing or usage. This endeavor is part of a larger research endeavor in software metrics, whose "ultimate goal (...) is to help project managers make decisions under uncertainty." [13] (p. 500). Finally, the Reinhartz–Berger experiment compared two modeling techniques, namely object-process methodology (OPM) and UML, to establish the level of comprehension and the quality of the constructed models in the context of web applications. The motivation for this effort came from "the exponential growth of the Web and the progress of Internet-based architectures, [which] have set the stage for the proliferation of a variety of Web applications." [14] (p. 57).

The experimenters care for what is carried out by the professionals in the context of the SE industry, because their aim is to improve their practice, and experiments seem to provide an argument for the adoption of some techniques rather than others. For such an argument to be more convincing, the experiments need to be as close as possible to the reality of SE practice. This means that whatever the experimental conditions, they must capture all of the significant factors that affect the processes of real industrial software development, if researchers want their experiments to be meaningful.

In the analyzed papers, we can find several factors that the researchers took into account while trying to recreate a realistic experimental environment.

The proponents of the Genero experiment have "used material that is more representative of real cases" [10] (p. 523) compared to previous endeavors of theirs. Moreover, they "improved the experimental tasks by trying to specify [tasks] that are similar to those required in real projects." [10] (p. 523).

The Vokáč experiment was meant to improve on a previous experience by other researchers. The authors "replicated their experiment (...) using the same programs in a real programming environment, instead of pen and paper. This increases the experimental realism and, thereby, the applicability of the results." [12] (p. 150).

The search for realism has found some obstacles in the intrinsic differences between a controlled experimental environment and the resource-constrained reality of the SE industry. From the Vokáč experiment: "[a] difference is that it is possible to walk away from a nonworking solution, as actually happened with several subjects. In industry that option is simply unavailable. When it happens, projects tend to become highly visible failures." [12] (p. 190). Moreover, "the experimental design required all participants to be present at the same time, for two consecutive days. This made it much more difficult to get a sufficient number than for previous experiments that could be done on each company's premises, one company at a time." [12] (p. 185).

Sometimes the resource constraints are on the experiment's side. Again, from the Vokáč experiment: "Real-world programs are much larger than those used in the experiment. With a restricted time and money budget, this is a limitation that is difficult to overcome." [12] (p. 189). The De Lucia experiment presents similar issues: "Concerning the artefact repository used in the experiment, it is worth noting that it is not comparable to industrial projects (...) However, the type of experimentation has to be conducted in a controlled way and in a limited amount of time. For this reason, it is not easy to use repositories of larger size and even if larger repositories are used, there might anyway be the need to select a subset of (...) artefacts (...)." [9] (p. 82).

In some cases, like in the Karlsson experiment, such constraints are part of the motivation for the whole effort: "Since the major disadvantage of [a technique based on an exhaustive list of requirement comparisons] is the time-consumption for large problems, different investigations have been performed in order to decrease the number of comparisons, and thus the time needed." [11] (p. 7). Similarity to the industrial context plays such a primary role in this experiment that a technique that was "ranked rather low" in previous studies is still considered interesting to investigate, since it is used in a very active subgroup of the SE community.

In the effort to maintain similarity to actual SE practices, the proponents of these experiments must take several factors into account, but one plays a particularly important role: The human factor. SE research aims at helping make the best choices in terms of selecting, with respect to a given problem, the most suited requirement specification language, the most skilled programmer, the best fitting programming paradigm and language, *etc*. These choices have a social component in that they are strongly dependent on human actors: requirements consist of a call for a software solution to a problem expressed by groups of people or even entire organizations [59,60]; programmers are human beings endowed with honed technical skills, but also characterized by idiosyncrasies and subjective preferences with respect to programming paradigms and languages [61,62]. Thus, experiments intended to investigate these aspects of SE must make use of human subjects, and so did our chosen examples.

- The Genero experiment [10] was divided into two replicated rounds: The first involved 38 computer science students in the third year at a Spanish university, the second 23 computer science students in their final year at an Italian university.

- The Vokáč experiment [12] replicated a previous experiment, and according to the authors, experimental realism was increased by using a real programming environment instead of pen and paper; 44 paid professionals from multiple companies were the subjects instead of 29 volunteers from a single organization, like in the previous attempt.
- Each of the two replications in the Shull experiment [15] involved 18 undergraduate students from a different Brazilian university, all with an experience of slightly more than one year of software development in class.
- The Karlsson experiment [11] consisted of two sessions: one with a group of 16 subjects, including Ph.D. students, and one professor as part of a research methodology course, the other with 30 MSc students in their final year, taking an optional requirements engineering course.
- The De Lucia experiment [9] was executed twice in an Italian university, with 20 first-year MSc students attending an advanced SE course and with 12 second-year MSc students from a course on software project management, respectively.
- The causal model used in the Fenton experiment [13] was comprised of variables and relationships identified by means of questionnaires distributed to a number of managers involved in 31 software projects in the consumer electronics industry.
- Finally, the Reinhartz–Berger experiment [14] involved 81 third-year students in a four-year information systems engineering program in the final four hours of a 13-week “Specification and Analysis of Information Systems” course.

Issues arise from the fact that experiments are conducted with groups of human subjects. People come with different prior knowledge, tastes, habits, skills, and so on, which means that they are likely to tackle the tasks in the experiments with results that may be very varied and make the researchers’ attempt to extract scientifically-proven facts more complicated. We have found that some of these issues derived from a less-than-ideal experiment design. In the two replications of the Genero experiment, for instance, the subjects in Spain were given material in their native language, whereas what was provided to their counterparts in Italy had been translated into English. The experimenters found out that not all of the subjects were sufficiently skilled in English and needed extra time to ask about the meaning of some statements. Another example is given by the Reinhartz–Berger experiment: some fatigue and order factors may have interfered with the results, since in all of the texts asking the subjects to create two models, one for a project management system and one for a book ordering system, the former always appeared first, so that it is likely that the subjects started working on it first.

Naturally, more uniform experimental conditions (possibly including the use of the subjects’ native language or a proper randomization in the sequences of experimental tasks) would enable experimenters to overcome such issues, but others are more intrinsic to the use of subjects and do not depend on experimental design.

One of the objectives of the Genero experiment was to verify that some structural properties of diagrams have an impact on their cognitive complexity, defined as the mental burden posed by the diagram on the persons who are studying it. Such complexity was measured in the experiment by means of the subjective judgment of the participants through a scale of five linguistic labels, ranging from “very simple” to “very complex”. In the Karlsson experiment on requirement prioritization, the investigation is about the beneficial effect of the introduction of automated tools, but the decision making process on whether one requirement is more important than another ultimately rests on the subjects' shoulders. The causal model to predict software defects that was analyzed in the Fenton experiment was created on the basis of the expertise of professionals gathered by means of a questionnaire.

These examples show that a significant part of the results in the experiments relies on the personal point of view of the subjects involved. The very nature of this perspective strongly contrasts with the universality and objectivity expected of the results of scientific experimentation. An obvious, although not always possible, way to deal with this issue is to enlarge the pool of subjects involved in the experiments. Indeed, all of the experimenters are well aware of the issue of sample significance, especially since they relied on students attending an SE-related course, because in most cases, the experimenters are professors or lecturers at a university. Apart from concerns about the sample size, such a specific context would not be considered adequate for a general sociological or psychological study, but as clearly shown in the quotes below, all SE experimenters have indeed a more restricted aim: they perform experiments having the SE industry in mind: "Because of the difficulty of obtaining professional subjects, we used students from a software engineering course." [10] (p. 539); "Some [professional] maintainers may have more experience with design patterns than did our subjects." [12] (p. 189). "The generalisability of the study is limited due to the rather small sample and specific context. (...) the subjects may have opinions similar to decision makers in the industry." [11] (pp. 27–28). "All students are master students who either had some professional experiences or worked on industrial projects (...) This makes these students comparable to industry junior developers." [9] (p. 25). Let us explore the connection between experiments and the industry in the following section.

3. From the Lab to the Real World

The generalization of experimental results is a highly desirable property for any experimenter, and SE practitioners are no exception in this respect. In most of the controlled experiments involving human subjects we have considered in our analysis, the authors express concern regarding the possibility of generalizing their results by discussing threats to external validity and possible solutions. External validity is the degree to which the results can be generalized and transferred to other situations.

In the Shull experiment, for instance, we can find a warning about "the sensitivity of experimental designs involving human subjects, where small variations in the execution of the experiment can have large effects on results." [15] (p. 132). More specifically, what is usually recognized in these articles as a common threat is the use of students (mostly MSc and/or Ph.D.) instead of real professionals as experimental subjects. As said in the De Lucia experiment, for instance, "this kind of threat is always present when experimenting with students." [9] (p. 81). Moreover, as remarked in the Karlsson experiment, although improving the validity of the conclusions, the homogeneity of a subject group (like students from a specific SE class) makes it difficult to generalize the results to a broader population. However, the experimenters rely on a previous study by Tichy [63] showing that if a student experiment shows that one technique is better than another, it is rather unlikely that professionals would come to the opposite conclusion. Although by means of specifically-designed courses, the students' modeling and programming skills can be brought up to the level of industry junior developers and strategies can be implemented to compensate for their limited familiarity with the application domains and the software systems used in the experiments (such as organizing meetings before the experiment to provide students with an acceptable knowledge level), the necessity to replicate the same experiment in an industrial context is clearly recognized. In the Genero experiment, for instance, further research is said to have to include "a family of experiments [which] should also use professionals as subjects and different experimental models and tasks." [10] (pp. 543–544).

A comprehensive analysis of the threats to external validity can be found in the Vokáč experiment, where several differences between the experimental environment and an actual software maintenance setting are considered, such as the fact that maintainers in the experiment were different from the programmers who created the software, so that the results are not applicable to maintenance by the original designers, who are normally "expected to remember not only the actual design, but also much of the motivation for it." [12] (p. 189). Another difference is the fact that some maintainers may have more experience than the experimental subjects. With respect to these and a number of other issues, the authors successfully showed that they do not constitute an actual threat to their experimental results: in these cases, for instance, if the use of design patterns has a beneficial effect on the maintainability of software programs (as proven by the Vokáč experiment), then a deeper knowledge of the artifact endowing the original designers or more experienced maintainers is very unlikely to make the situation worse. The authors "would expect the beneficial effects of patterns to be greater." [12] (p. 189). However, there are other issues that turn out to be more problematic, because the experimenters are not able to provide a solution to counter them. We have already mentioned that time and money budget restrictions force the experimenters to work with programs that are smaller than their real-world counterparts. Moreover, "real-world programs are sometimes less well documented, and changes may be larger and involve more than one [design] pattern. The effects of such differences are difficult to predict on a general basis. There are undoubtedly interaction effects that can occur between patterns, but would not be visible in such an experiment." [12] (p. 189).

The shift of from the lab to the real world (which means, in most of the cases we have considered, performing experiments in industrial contexts) presents several challenging issues from a methodological perspective. We have already argued in the past that taking into consideration how experimental methodologies are developed and used is a way of reflecting on the disciplinary status of computing that moves us away from the somehow too rigid dualism between science and engineering [1]. Here, we focus on how the conceptual categories of the traditional scientific method are challenged in the context of SE because of the social components that are involved. Keeping in mind the examples we have discussed so far, it is quite evident, for instance, that the notion of control as intended in the traditional experimental protocol (the experimentalist is not part of the experimental system and is able to intervene in and control the experimental system from the outside) does not fully apply. Not only is it problematic to draw a line between the experimental system and its environment, in particular if this environment is meant to include several subjects from different contexts, but since the experimental environment does not consist only of physical objects, also social components are to be taken into account.

We are aware that the social components at play in current SE experiments are only given by the human beings whose interactions with technical artifacts are under observation. However, we have also seen that other social factors exert a significant influence on the experimental practice in the context of SE: many, if not all, researchers are concerned about the external validity of their results, and the context they are referring to is the SE industry. The industrial dimension is not directly involved in SE experiments at the moment, but we believe that to move toward a more comprehensive account of future practices where experiments go out of the restricted environments of labs in universities and research centers, such a component should be taken into account, in terms of conditions, rules, organizations and companies.

Thus, with our work, we aim at showing that SE includes a very significant socio-technical dimension beside its long-studied technological dimension. The human factor brought in by the experimental subjects, as well as the whole organizational dimension given by the SE industry and its ramifications need to be taken into account if one wants to obtain a comprehensive analysis of the experiments in this discipline.

The work in [3] defines as socio-technical a system whose behavior is "significantly affected by [its] technical components but the functioning (...) as a whole depends as much on the functioning of these technical components as on the functioning of its social components (...) and it depends on the behavior of human actors." According to [64], a socio-technical system is a special sort of system, namely an entity that can be separated into parts, which are linked to each other in a specific way, including also components that are not tangible objects. In other words, a socio-technical system is a hybrid system formed by different components, some of which are described and analyzed using the natural sciences paradigm, while others must be described by drawing on the social sciences tools.

The example discussed at length in [64] is the world civil aviation system, composed of a great number of different technical artifacts (computers, airplanes, airports, baggage trolleys, *etc*.), people (cabin crew, personnel working at the check-in desks, air controllers, *etc*.), but also other components of a more abstract nature (air corridors, air companies, regulations having to be observed by pilots and airline companies, organizations enforcing such regulations, *etc*.). The articulation of these different components is what makes socio-technical systems different from even the most complex technical artifact, which, although incorporating characteristics of both physical and social objects, *i.e.*, its hardware and the function assigned to it by human designers and users, respectively, is a building block of systems of greater complexity.

The involvement of people, regulations and institutions beside technical artifacts is the reason why the natural scientific way of description and explanation represents only one way of formulating matters within the context of socio-technical systems and why a social-scientific approach is required. Although the hybrid character of socio-technical systems is what eminently characterizes them, their design, implementation and maintenance are still in the hands of engineers that have been mainly educated in the natural scientific way. However, the complexity of socio-technical systems consists precisely in the fact that they have many different users (unlike typical technical artifacts), and their functioning depends not only on the articulation of the technical components, but also on the coordination of the behavior of different users; the success of such coordination comes through habits, agreements, rules, laws, *etc*., that is entities studied in the social sciences and not in the natural sciences.

The main aim of SE is creating technical artifacts in the form of software, but, as clearly shown by the examples in this work, technology alone does not cover everything that happens in SE: software is created by people to meet other people's needs within the context of communities, companies and markets, which do not simply work as a passive background, but pose several constraints, for example in terms of resource limitations, business strategies and societal values that shape and guide the software development process.

The co-existence of technological and social aspects in the conception, development and deployment of software indicates that the context of SE includes socio-technical systems. As a consequence, experiments in SE are significantly characterized by socio-technical features. Thus, we envision a development of SE experimentation in which the conceptual framework sustaining the methodological practice has been expanded in a socio-technical direction. After all, every aspect of SE can be and, in our opinion, should be interpreted from such a perspective.

Let us take a closer look at the variety of activities that are included in SE as a discipline. The construction of an SE artifact consists of a number of operations, which are carried out at different stages in the life cycle of the software that is being created. The context in which such a life cycle unfolds is notably vast, as there are several degrees of freedom depending on the choices made with respect to the entities involved at each stage. The process starts with the requirements, which describe what is required of the software; such a description can be provided in a natural or a formal language and at different levels of detail. A programmer or, more commonly, a team of programmers is responsible for the creation of the software, by first conceiving of an algorithm and then turning it into a program. The programming paradigm is the view that underlies the way the software is going to be written: with a functional paradigm, for instance, all operations that a program is to perform are modeled as functions, whereas the object-oriented paradigm views such operations as actions taken by different objects, that is independent subparts that constitute the program. Given a paradigm, a programmer can have more than one programming language to choose from: for example, C++, Java and Python are all object-oriented languages. The program itself can be written in different ways in the same language, depending on the structure by which the included operations are organized. A testing process must be carried out once a program is written to ensure that it fulfills the requirements it was created for without faults. Finally, if more programs are available to provide the same service, a benchmarking technique can be used to assess which one achieves the goal with the least resource consumption in terms of computing time and computer memory.

As said before, SE research aims at helping make the best choice with respect to every stage mentioned above. The decision making process comes with a significant social component, in that every choice is strongly dependent not only on human actors with their personal backgrounds and subjective preferences, but also on organizations and institutions with the relevant conditions and rules. When it comes to picking the programming language to develop software with, the decision is *de facto* taken by the market the engineers are aiming for: for instance, if they intend to develop software running on an iPhone, then the most natural choice is to work with the most used language for this specific platform, namely Objective-C, for which full support from the developer community and the relevant literature is ensured [65]. Programming languages are themselves the product of institutional decisions: the rules of the Java language, for instance, are determined by the Java Community Process (JCP), which is the sole official mechanism for developing standard technical specifications for this technology. Anyone from the SE community can sign up to become a JCP member and then join expert groups and participate in reviewing processes and providing feedback [66]. Testing is also an activity that is significantly influenced by organizations and, more specifically, by their economic conditions:

the resources demanded by exhaustive program testing procedures are incompatible with any software company's economic strategy, so that a key task in testing is the selection of critical runs to detect as many software faults as possible to ensure that the final product reaches an acceptable quality level to be marketed [67]. Finally, even a highly technical practice, like benchmarking, is seen by SE researchers not only as a rigorous examination of contributions, but also a way to ensure greater communication and collaboration among researchers, leading to a stronger consensus in the community [68].

4. Conclusions

In this paper, we hope to have substantiated the importance of enlarging the context to socio-technical systems when analyzing SE experiments. The limits in applying the traditional experimental protocol to different fields of computing appears particularly significant in the case of SE experiments that we have systematically analyzed: not only the technical dimension, beside the scientific one, needs to be taken into account, but also the social issues, meaning both the human agents and the social institutions play a significant role in this framework. If SE experiments aim at leaving the limited contexts of the lab to move toward real industrial applications, it is important to recognize these issues, which we have explored here with a particular focus on the problem of external validity. Obviously, this social dimension has to be dealt with by the tools of the social sciences, as the functioning of complex socio-technical artifacts depends on the coordination amongst different users through rules, laws, habits and agreements, besides the proper working of the technical components (traditionally managed with the tools of natural sciences).

The acknowledgment of the social dimension of experiments within a socio-technical perspective calls for a paradigmatic shift that we have just initiated reflecting upon, considering SE experiments as a case study for the reasons we have discussed in the Introduction of this paper. We are confident, however, that an extension to other fields of computing, given the obvious differences, will show some very close similarities and strengthen the importance of extending the traditional notion of experiment in socio-technical systems. We believe, moreover, that this shift could have an impact on the philosophy of computing in general and in particular on those reflections aimed at clarifying the disciplinary status of computing from a methodological point of view.

Finally, to rethink experiments in a socio-technical perspective could represent also an important step for the methodological advancement of SE itself, where the attempts to reflect on experiments (see [5] for a comprehensive and influencing work on this topics) have generally been focused exclusively on the implementation of the traditional experimental categories, in particular the replication of experiments. Now, it is time to move beyond and construct the debate on experimental methods in SE upon the most recent developments in the philosophy of technology and in the philosophy of science to take into account a complex and articulated socio-technical practice that cannot be reduced to the categories of the traditional scientific disciplines.

Author Contributions

Both authors have equally contributed to the design of the study, the analysis of the scientific papers, the interpretation of the results and the preparation of the manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Schiaffonati, V.; Verdicchio, M. Computing and experiments. *Philos. Technol.* **2013**, *27*, 1–18.
2. Shull, F.J.; Carver, J.C.; Vegas, S.; Juristo, N. The role of replications in Empirical Software Engineering. *Empir. Softw. Eng.* **2008**, *13*, 211–218.
3. Kroes, P. Experiments on socio-technical systems: The problem of control. *Sci. Eng. Ethics* **2015**, 1–13, doi:10.1007/s11948-015-9634-4.
4. Eden, A.H. Three paradigms of computer science. *Minds Mach.* **2007**, *17*, 135–167.
5. Juristo, N.; Moreno, A. *Basics of Software Engineering Experimentation*; Kluwer: Dordrecht, Germany, 2001.
6. Basili, V.R., Briand, L.C., Eds. *Empirical Software Engineering*; Springer: Berlin, Germany, 1996.
7. Andersson, C., Runeson, P., Eds. First International symposium on empirical software engineering and measurement. In Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, Madrid, Spain, 20–21 September 2007.
8. Scopus. The Largest Abstract and Citation Database of Peer-reviewed Literature. 2014. Available online: http:// www.scopus.com (last accessed 10 February 2014).
9. De Lucia, A.; Oliveto, R.; Tortora, G. Assessing IR-based traceability recovery tools through controlled experiments. *Empir. Softw. Eng.* **2009**, *14*, 57–92.
10. Genero, M.; Manso, E.; Visaggio, A.; Canfora, G.; Piattini, M. Building measure-based prediction models for UML class diagram maintainability. *Empir. Softw. Eng.* **2007**, *12*, 517–549.
11. Karlsson, L.; Thelin, T.; Regnell, B.; Berander, P.; Wohlin, C. Pair-wise comparisons versus planning game partitioning–experiments on requirements prioritization techniques. *Empir. Softw. Eng.* **2006**, *12*, 3–33.
12. Vokáč, M.; Tichy, W.; Sjøberg, D.I.K.; Arisholm, E.; Aldrin, M. A controlled experiment comparing the maintainability of programs designed with and without design patterns—A replication in a real programming environment. *Empir. Softw. Eng.* **2004**, *9*, 149–195.
13. Fenton, N.; Neil, M.; Marsh, W.; Hearty, P.; Radliński, L.; Krause, P. On the effectiveness of early life cycle defect prediction with Bayesian Nets. *Empir. Softw. Eng.* **2008**, *13*, 499–537.
14. Reinhartz-Berger, I.; Dori, D. OPM *vs.* UML—Experimenting with comprehension and construction of web application models. *Empir. Softw. Eng.* **2005**, *10*, 57–79.
15. Shull, F.; Mendonc, M.G.; Basili, V.; Carver, J.; Maldonado, J.C.; Fabbri, S.; Travassos, G.H.; Ferreira, M.C. Knowledge-sharing issues in experimental software engineering. *Empir. Softw. Eng.* **2004**, *9*, 111–137.
16. D'Ambros, M.; Lanza, M.; Robbes, R. Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empir. Softw. Eng.* **2012**, *17*, 531–577.

17. Jiang, Y.; Cukic, B.; Ma, Y. Techniques for evaluating fault prediction models. *Empir. Softw. Eng.* **2008**, *13*, 561–595.
18. Pan, K.; Kim, S.; Whitehead, E.J., Jr. Toward an understanding of bug fix patterns. *Empir. Softw. Eng.* **2009**, *14*, 286–315.
19. Turhan, B.; Menzies, T.; Bener, A.B.; di Stefano, J. On the relative value of cross-company and within-company data for defect prediction. *Empir. Softw. Eng.* **2009**, *14*, 540–578.
20. Damian, D.; Chisan, J.; Vaidyanathasamy, L.; Pal, Y. Requirements engineering and downstream software development: Findings from a case study. *Empir. Softw. Eng.* **2005**, *10*, 255–283.
21. Gonzalez-Barahona, J.M.; Robles, G.; Michlmayr, M.; Amor, J.J.; German, D.M. Macro-level software evolution: A case study of a large software compilation. *Empir. Softw. Eng.* **2009**, *14*, 262–285.
22. Kapser, C.J.; Godfrey, M.W. "Cloning considered harmful" considered harmful: Patterns of cloning in software. *Empir. Softw. Eng.* **2008**, *13*, 645–692.
23. Karlstrŏm, D.; Runeson, P. Integrating agile software development into stage-gate managed product development. *Empir. Softw. Eng.* **2006**, *11*, 203–225.
24. Khoshgoftaar, T.M.; Seliya, N. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empir. Softw. Eng.* **2003**, *8*, 255–283.
25. Khoshgoftaar, T.M.; Seliya, N. Analogy-based practical classification rules for software quality estimation. *Empir. Softw. Eng.* **2003**, *8*, 325–350.
26. Khoshgoftaar, T.M.; Seliya, N. Comparative assessment of software quality classification techniques: An empirical case study. *Empir. Softw. Eng.* **2004**, *9*, 229–257.
27. Kommeren, R.; Parviainen, P. Philips experiences in global distributed software development. *Empir. Softw. Eng.* **2007**, *12*, 647–660.
28. Mendes, E.; Watson, I.; Triggs, C.; Mosley, N.; Counsell, S. A comparative study of cost estimation models for web hypermedia applications. *Empir. Softw. Eng.* **2003**, *8*, 163–196.
29. Pikkarainen, M.; Haikara, J.; Salo, O.; Abrahamsson, P.; Still, J. The impact of agile practices on communication in software development. *Empir. Softw. Eng.* **2008**, *13*, 303–337.
30. Poshyvanyk, D.; Marcus, A.; Ferenc, R.; Gyimóthy, T. Using information retrieval based coupling measures for impact analysis. *Empir. Softw. Eng.* **2009**, *14*, 5–32.
31. Segal, J. When software engineers met research scientists: A case study. *Empir. Softw. Eng.* **2005**, *10*, 517–536.
32. Adolph, S.; Hall, W.; Kruchten, P. Using grounded theory to study the experience of software development. *Empir. Softw. Eng.* **2011**, *16*, 487–513.
33. Carver, J.C.; Jaccheri, L.; Morasca, S.; Shull, F. A checklist for integrating student empirical studies with research and teaching goals. *Empir. Softw. Eng.* **2010**, *15*, 35–59.
34. Dieste, O.; Grimán, A.; Juristo, N. Developing search strategies for detecting relevant experiments. *Empir. Softw. Eng.* **2009**, *14*, 513–539.
35. Do, H.; Elbaum, S.; Rothermel, G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empir. Softw. Eng.* **2005**, *10*, 405–435.
36. Falessi, D.; Babar, M.A.; Cantone, G.; Kruchten, P. Applying Empirical Software Engineering to software architecture: Challenges and lessons learned. *Empir. Softw. Eng.* **2010**, *15*, 250–276.

37. Ivarsson, M.; Gorschek, T. A method for evaluating rigor and industrial relevance of technology evaluations. *Empir. Softw. Eng.* **2011**, *16*, 365–395.
38. Juristo, N.; Moreno, A.M.; Vegas, S. Reviewing 25 years of testing technique experiments. *Empir. Softw. Eng.* **2004**, *9*, 7–44.
39. Lethbridge, T.C.; Sim, S.E.; Singer, J. Studying software engineers: Data collection techniques for software field studies. *Empir. Softw. Eng.* **2005**, *10*, 311–341.
40. Mohagheghi, P.; Conradi, R. Quality, productivity and economic benefits of software reuse: A review of industrial studies. *Empir. Softw. Eng.* **2007**, *12*, 471–516.
41. Runeson, P.; Hŏst, M. Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* **2009**, *14*, 131–164.
42. Shull, F.J.; Carver, J.C.; Vegas, S.; Juristo, N. The role of replications in Empirical Software Engineering. *Empir. Softw. Eng.* **2008**, *13*, 211–218.
43. Beecham, S.; Hall, T.; Rainer, A. Software process improvement problems in twelve software companies: An empirical analysis. *Empir. Softw. Eng.* **2003**, *8*, 7–42.
44. Do, H.; Rothermel, G.; Kinneer, A. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis. *Empir. Softw. Eng.* **2006**, *11*, 33–70.
45. German, D.M. An empirical study of fine-grained software modifications. *Empir. Softw. Eng.* **2006**, *11*, 369–393.
46. Măntylă, M.V.; Lassenius, C. Subjective evaluation of software evolvability using code smells: An empirical study. *Empir. Softw. Eng.* **2006**, *11*, 395–431.
47. Melton, H.; Tempero, E. An empirical study of cycles among classes in Java. *Empir. Softw. Eng.* **2007**, *12*, 389–415.
48. Nagappan, N.; Maximilien, E.M.; Bhat, T.; Williams, L. Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empir. Softw. Eng.* **2008**, *13*, 289–302.
49. Schach, S.R.; Jin, B.; Yu, L.; Heller, G.Z.; Offutt, J. Determining the distribution of maintenance categories: Survey versus measurement. *Empir. Softw. Eng.* **2003**, *8*, 351–365.
50. Thummalapenta, S.; Cerulo, L.; Aversano, L.; Di Penta, M. An empirical study on the maintenance of source code clones. *Empir. Softw. Eng.* **2010**, *15*, 1–34.
51. Vegas, S.; Basili, V. A characterization schema for software testing techniques. *Empir. Softw. Eng.* **2005**, *10*, 437–466.
52. Weyuker, E.J.; Ostrand, T.J.; Bell, R.M. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empir. Softw. Eng.* **2008**, *13*, 539–559.
53. Zou, X.; Settimi, R.; Cleland-Huang, J. Improving automated requirements trace retrieval: A study of term-based enhancement methods. *Empir. Softw. Eng.* **2010**, *15*, 119–146.
54. Azzeh, M.; Neagu, D.; Cowling, P.I. Fuzzy grey relational analysis for software effort estimation. *Empir. Softw. Eng.* **2010**, *15*, 60–90.
55. Li, J.; Ruhe, G.; Al-Emran, A.; Richter, M.M. A flexible method for software effort estimation by analogy. *Empir. Softw. Eng.* **2007**, *12*, 65–106.
56. Li, Y.F.; Xie, M.; Goh, T.N. A study of the non-linear adjustment for analogy based software cost estimation. *Empir. Softw. Eng.* **2009**, *14*, 603–643.

57. Robillard, M.P.; DeLine, R. A field study of API learning obstacles. *Empir. Softw. Eng.* **2011**, *16*, 703–732.
58. Sharp, H.; Robinson, H. An ethnographic study of XP practice. *Empir. Softw. Eng.* **2004**, *9*, 353–375.
59. Sommerville, I.; Rodden, T.; Sawyer, P.; Bentley, R.; Twidale, M. Integrating ethnography into the requirements engineering process. In Proceedings of IEEE International Symposium on Requirements Engineering, San Diego, CA, USA, 4–6 January 1993; pp. 165–173.
60. Sutcliffe, A. *User-centred Requirements Engineering*; Springer: Berlin, Germany, 2002.
61. Naur, P. Understanding Turing's universal machine—Personal style in program description. *Comput. J.* **1993**, *36*, 351–372.
62. Harrison, R.; Smaraweera, L.G.; Dobie, M.R.; Lewis, P.H. Comparing programming paradigms: An evaluation of functional and object-oriented programs. *Softw. Eng. J.* **1996**, *11*, 247–254.
63. Tichy, W.F. Hints for reviewing empirical work in software engineering. *Empir. Softw. Eng.* **2000**, *5*, 309–312.
64. Vermaas, P.; Kroes, P.; van de Poel, I.; Franssen, M.; Houkes, W. *A Philosophy of Technology. From Technical Artefacts to Sociotechnical Systems*; Morgan and Claypool: San Rafael, CA, USA, 2011.
65. Skelton, G.W.; Jackson, J.; Dancer, C.F. Teaching software engineering through the use of mobile application development. *J. Comput. Sci. Coll.* **2013**, *28*, 39–44.
66. Java Community Process. Community Development of Java Technology SPecifications. 2014. Available online: http://www.jcp.org/ (last accessed 1 July 2014).
67. Myers, G.J.; Badgett, T.; Sandler, C. *The Art of Software Testing*, 3rd ed.; Wiley and Sons: Hoboken, NJ, USA, 2011.
68. Sim, S.E.; Easterbrook, S.; Holt, R.C. Using benchmarking to advance research: A challenge to software engineering. In Proceedings of the 25th International Conference on Software Engineering, Piscataway, NJ, USA, 3–10 May 2003; pp. 74–83.