The Role and Relevance of Experimentation in Informatics [*]

Carlos Andujar[†] Viola Schiaffonati[‡] Fabio A. Schreiber[§] Letizia Tanca[¶] Matti Tedre[‖] Kees van Hee[**] Jan van Leeuwen[††]

Summary

Informatics is a relatively young field within science and engineering. Its research and development methodologies build on the scientific and design methodologies in the classical areas, often with new elements to it. We take an in-depth look at one of the less well-understood methodologies in informatics, namely *experimentation*.

What does it mean to do experiments in informatics? Does it make sense to 'import' traditional principles of experimentation from classical disciplines into the field of computing and information processing? How should experiments be documented? These are some of the questions that are treated.

The report argues for the key role of empirical research and experimentation in contemporary Informatics. Many IT systems, large and small, can only be designed sensibly with the help of experiments. We recommend that professionals and students alike are well-educated in the principles of sound experimentation in Informatics. We also recommend that experimentation protocols are used and standardized as part of the experimental method in Informatics.

Contents

1. INTRODUCTION

(*Letizia Tanca* and *Jan van Leeuwen*)

> *The ability to design and support experiments is a vital but still little appreciated part of Computer science* (K. Keahey *et al.*, 2011)

Experimentation in Informatics? Does it apply, as in other sciences? Do we understand how experimental methods work in the field? How should they be documented? Despite the wide use of empirical studies and experiments, computer scientists do not seem to agree on the role and relevance of experiments in their field.

In this paper we argue that experimentation should be better understood and appreciated as a key methodology in Informatics. The basic ideas of the 'experimental method' should be included in Informatics curricula like in all other science curricula. We also argue that experimentation requires

[*]This paper is based on the contributions to the Workshop on *the Role and Relevance of Experimentation in Informatics*, held prior to the 8th European Computer Science Summit (ECSS 2012) of *Informatics Europe*, November 19th 2012, Barcelona.

[†]Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona (ES), email: andujar@lsi.upc.edu.

[‡]**Corresponding author**, Dip. Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano (I), email: viola.schiaffonati@polimi.it.

[§]Dip. Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano (I), email: fabio.schreiber@polimi.it.

[¶]Dip. Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano (I), email: letizia.tanca@polimi.it.

[‖]Dept. of Computer and Systems Sciences, Stockholm University, Stockholm (S), email: matti.tedre@acm.org.

[**]Dept. of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven (NL), email: k.m.v.hee@TUE.nl.

[††]Dept. of Information and Computing Sciences, Utrecht University, Utrecht (NL), email: J.vanLeeuwen1@uu.nl.

detailed protocolisation. The conclusions are based on the results of the Workshop on ‘Experimentation’ held prior to the ECSS’12 summit meeting of Informatics Europe in Barcelona (2012).

1.1 A View of Informatics

Informatics emerged in the 20th century as the field that underlies all uses of computing and information technology. Owing to its many origins, Informatics is often described both as a science and as an engineering discipline. Due to its far-reaching implications for the understanding of natural phenomena and systems, Informatics is sometimes even called a *natural science*. Due to its emphasis on the construction of computational artifacts (e.g. software), Informatics is sometimes called a *technical design science* as well. As a consequence, the scientific methodologies of Informatics have varied origins as well and are not always uniquely defined or understood.

Empirical methods and experimentation have been part of Informatics for many years. In fact already in 1967, one of the founding fathers of the discipline, George Forsythe, pointed out that ‘Computer Science’ is to be valued both as a theoretical and as an experimental science. Using the analogy with physics, he wrote:

> Whether one is primarily interested in understanding or in design, the conclusion is that [...] good experimental work in computer science must be rated very high indeed, ([21], p. 4)

but, with great foresight, he also noted that:

> [...] computer science is in part a young deductive science, in part a young experimental science, and in part a new field of engineering design. Because of this broad sweep of activity, the nature of computer science is usually misunderstood and deprecated by those with an acquaintance, however close, to only one aspect of computing. ([21], p. 4)

In a nutshell this explains the complexity of the methodological landscape in the field of Informatics. Many different concerns developed simultaneously and, in-as-much as the development of the field has been influenced by sociological factors [47], the interest for experimentation has varied greatly over the years and per sub-discipline [30]. With experimentation currently gaining recognition in the field at large, the principles of experimentation should be well-understood in the discipline.

In this paper we highlight both the philosophy and some concrete ramifications of the experimental method in Computer Science. The paper leads to several conclusions which will become clear in the subsequent sections.

Our first, main thesis is:

> A: *Experimentation is a key methodology in Informatics, and its principles should be fully recognized in the field’s philosophy. Informatics curricula should give an adequate background in the understanding and practical use of experimental methods.*

In particular Sections 2, 3, and 6 will provide further background to thesis A.

1.2 Experimentation

Experimentation is well-understood in the philosophy of science. In his classic on *The principles of science*, W.S. Jevons [26] already noted in the nineteenth century that (observations and) experiments are the primary source of experience (and thus knowledge, in his view). He proceeded to explain how experiments are intended to give insight in the causal effects of different (and ideally independent) parameters on a phenomenon or system, when they are varied in a controlled setting. They reveal unknown properties, confirm or refute hypotheses, help in debugging theories or prototypes, and set the stage for alternate explanations or designs.

One can only guess why experimentation has developed so unevenly in the various subareas of Informatics (or: Computer Science). There is no shortage of experimental methods in the field. Some were developed to great sophistication, like visualisation, case studies, simulation and parameter tuning. Sometimes statistical methods are employed, as e.g. in the analysis of optimization methods [7]. In areas like Information Systems there is a lot of experience with experimentation e.g. on the interplay between human behaviour and design as well (see e.g. [29]). An often confirmed experience is that *good experimentation is hard*.

It seems that the more formalized and theory-driven subareas of Informatics have long suppressed the possible role of experimentation. Milner [35] made the case that ‘theory’ and ‘experiment’ can well go together. For example, in a classical field like algorithm design it has long beenl common practice to assess algorithms by their asymptotic properties only. However, their actual behaviour in practice is now increasingly being studied as well, even though the models for the latter are typically still theoretical. With Informatics becoming ever more interactive, large-scale, and distributed, experimentation is increasingly called for in all areas, to understand the unknown behaviours of a design or system, in any realistic setting.

The essential role of experimentation in Informatics was explicated before, notably by Denning [12], and later by Tichy [48] when he wrote:

> To understand the nature of information processes, computer scientists must observe phenomena, formulate explanations and theories, and test them. ([48], p. 33)

Interestingly, in those areas where experimentation did not come naturally, the approach to science through experimentation has led to a kind of rediscovery of the experimental methodology. It has occasionally led to separate fields like experimental algorithmics [33] and empirical software engineering c.q. software engineering experimentation [8, 27, 52], which isn't an simple way to get experimentation integrated in a field. In other cases, the 'experimental methodology' is explicitly brought under attention as a possible way to publishable results provided suitable standards are followed, as done e.g. by the IEEE journal *Pervasive Computing* [31].

1.3 Protocols

Despite this positive, albeit gradual, development we note that there seems to be hardly any effort in Informatics to standardize its experimental methodologies. Fletcher[19] observed already in 1995 that:

> When we *do* have occasion to do experiments, we should adopt the same rigorous protocols as practiced in the natural and social sciences. ([19], p. 163)

Fletcher's advice is commendable, but does not seem to have been implemented widely in the discipline (with notable exceptions again e.g. in the area of Information Systems). Good experiments require that they have sound designs and that their outcome can be validated. This is why many areas require explicit *experimentation protocols*. We believe that Informatics should do the same.

Experimentation protocols depend on the theory, behaviour or properties one is trying to assess by experiment. A protocol typically documents the approach, the parameters and hypotheses involved, the ways of measurement and analysis (with the reasoning behind it), and the way outcomes are recorded and stored for later inspection. As experiments must be sound and verifiable, experimentation protocols are indispensable. Clearly, different subareas of Informatics may require different types of protocols.

Our second main thesis is therefore:

> B: *Experimentation in Informatics requires sound and verifiable experimentation protocols. In the case of multidisciplinary research, these protocols should follow standard practices that are recognizable across discipline boundaries.*

We will see examples of thesis B in Sections 4 and 5 of this paper. Section 6 reviews where we stand overall.

1.4 Workshop

The discussion of experimentation in Informatics requires a good understanding of the issues. With this in mind, we organized a small workshop on *the Role and Relevance of Experimentation in Computer Science* and a subsequent panel discussion on the same issue within the 8th European Computer Science Summit (ECSS'12). This paper contains the written text of the contributions.

Among the questions that were discussed, were the following:

- What does it mean to make experiments in Informatics. Is it possible to ask this question in general terms, or does it have to be asked separately for each subfield?
- Is it possible to single out some common features that characterise 'good experiments' in Informatics?
- Does it make sense to 'import' traditional experimental principles from the classical scientific disciplines into Informatics?

We believe that the discussion on experiments in Informatics (or: computer science and engineering) can contribute to a better understanding of the scientific approach in the discipline. We also believe that it may contribute to the debate on the relationship between Informatics and science.

1.5 This Paper

The paper is organized as follows. In Section 2 (by V. Schiaffonati) the philosophical premises of experimentation in Informatics are presented. In section 3 (by M. Tedre) the analysis is continued with an assessment of the different types of experiments that computer scientists have been using so far. Sections 4 (by K. van Hee) and 5 (by C. Andujar) show how experiments and experimentation protocols can be designed to great advantage in algorithmic and user studies, respectively. Finally, section 6 (by F. Schreiber) assesses whether the methodological landscape, and experimentation in particular, is really understood sufficiently across the discipline. The section also includes a summary of the comments and suggestions from the discussions at the Workshop and in the panel session at ECSS'12. In Section 7 we offer some final conclusions.

2. ROLE AND RELEVANCE OF EXPERIMENTATION IN INFORMATICS

(*Viola Schiaffonati*)

Experiments play a fundamental role in empirical sciences, both in acquiring new knowledge and in testing it, and modern science has its roots in the very idea of experiment, as devised by Galileo Galilei during the so called Scientific Revolution in the XVII century. One of the main achievements of this revolution has been to recognize that pure observations are not enough for the comprehension of the natural world, and that to properly 'interrogate' Nature, interference with the natural world itself by means of experiments is necessary. An experimental procedure aims, thus, at *controlling* some of the characteristics of a phenomenon under investigation with the purpose of testing the behavior of the same phenomenon under some controlled circumstances. Accordingly, an experiment can be defined as a *controlled experience*, namely as a set of observations and actions, performed in a controlled context, to support a given hypothesis.

A trend has recently emerged toward making experimental scientific method take center stage in Informatics as well. Given that Informatics possesses an empirical component, even without considering here the debate about the prevalence of the empirical versus the theoretical one [46], it is natural to ask about the role and relevance of experimentation in it. Thus experimental scientific method has recently been under attention in computer science and engineering both at a general level [23, 36] and in specific disciplines (see for example the case of empirical software engineering [27]). Experiments are recognized as relevant in this field as they help build a reliable base of knowledge, lead to useful and unexpected insights, and accelerate progress [48].

Unfortunately computer scientists do not seem to agree on how experimental methods are supposed to impact their theory and practice. This section is a first attempt to start reflecting on the role of experiments in Informatics with the help of some tools developed in the philosophy of science and the philosophy of technology. The peculiarity of Informatics in between science and engineering should be taken as a starting point. Therefore, if on the one side inspiration can be drawn from how experiments are conducted in pure science, on the other side this perspective should be enlarged to include the analysis of how experiments are conducted in engineering disciplines, a topic so far almost completely neglected by the traditional philosophy of science.

2.1 Taking inspiration from science

It is commonly recognized that experimental methodologies in Informatics have not yet reached the level of maturity of other traditional scientific disciplines. A possible way to start investigating the role of experiments in this field is to analyze how experiments are performed in traditional science in order to evaluate whether some useful lessons can be learned. This is a way to obtain, at least, a better terminological and conceptual clarity in using terms that are historically and conceptually loaded. For example, in Informatics there still exists confusion between generic *empirical methods* (only requiring to be based on the aggregation of naturally occurring data) and *experimental methods* (that must adhere to strict rules as assessed in the course of the history of experimental science). Equally misleading is the idea that the replication of experiments is sufficient to guarantee the requirements of a serious experimental approach.

Generally speaking, an experiment is a controlled experience, a set of observations and actions, performed in a controlled context, to test a given hypothesis. Accordingly, the phenomenon under investigation must be treated as an isolated object; it is assumed that other factors, which are not under investigation, do not influence the investigated object. Despite the controversies about the scientific experimental method and its role, experiments possess some general features that are universally acknowledged and often are not even made explicit. These are: *comparison*, *repeatability* and *reproducibility*, *justification* and *explanation*.

- *Comparison.* Comparison means to know what has been already done in the past, both for avoiding to repeat uninteresting experiments and for getting suggestions on what the interesting hypotheses could be.
- *Reproducibility* and *repeatability.* These features are related to the very general idea that scientific results should be severely criticized in order to be confirmed. Reproducibility, in particular, is the possibility for independent researchers to verify the results of a given experiment by repeating it with the same ini-

tial conditions. Repeatability, instead, is the property of an experiment that yields the same outcome from a number of trials, performed at different times and in different places.

- *Justification* and *explanation*. These features deal with the possibility of drawing well justified conclusions based on the information collected during an experiment. It is not sufficient to have as many precise data as possible, but it is also necessary to look for an explanation of them, namely all the experimental data should be interpreted in order to derive the correct implications leading to the conclusions.

Is it possible to decline these general principles in a computer science and engineering experimental context? And, if the answer is positive, is there any utility in doing this? To try to decline these questions within specific areas of research offers some preliminary, but promising, answers. The case of autonomous mobile robotics, involving robots with the ability to maintain a sense of position and to navigate without human intervention, offers some useful insights. In response to a quite recent interest of this field in experimental methodologies, some research projects funded by the European Commission, journal special issues, series of workshops, and single papers have been produced with the aim of assessing rigorous evaluation methods for empirical results. In particular [1] and [2] have declined the three aforementioned principles for the case of mobile autonomous robotics, leading to the conclusion that, even if some works in this field are addressing in a more and more convincing way these experimental principles thus bringing autonomous mobile robotics closer to the standards of rigor of more mature scientific disciplines, its engineering component cannot be put aside and needs to be included in the discussion. Thus, importing these principles from physics to autonomous robotics is only the first step of a serious investigation on the relevance and role of experiments in a computer engineering field.

2.2 From science to engineering

Engineering can be defined as an activity that produces technology, and technology is a practice focused on the creation of artifacts and artifact-based services [22]. Hence, to move from a purely scientific context to an engineering one means not only to address different objects (*natural objects* versus *technical artifacts*), but also to consider the different purposes for which experiments are performed. If in science the goal of experimentation is *understanding* a natural phenomenon (or a set of phenomena), in engineering the goal is *testing* an artifact.

Technical artifacts are material objects deliberately produced by humans in order to fulfill some practical functions. They can be defined in accordance with the three following questions.

- What is the technical artifact for? Namely its *technical function*.
- What does it consists of? Namely its *physical composition*.
- How must it be used? Namely its *instructions for use*.

Informatics products are technical artifacts, as they are physical objects with a technical function and a use plan deliberately designed and made by human beings [50].

The notion of technical artifact plays an important role in analyzing the nature of experiments in engineering disciplines: experiments evaluate technical artifacts according to whether and to what extent the function for which they have been designed and built is fulfilled. This is the reason why normative claims are introduced in engineering experiments. An artifact, such as an airplane, can be 'good' or 'bad' (with respect to a given reference function), whereas a natural object, such as an electron, whether existing in nature or produced in a laboratory, can be neither 'good' nor 'bad'; in fact it is analyzed without any reference to its function and use plan and so is free from any normative constraints regarding its functioning[1].

Is the reference to the notion of technical artifact enough to analyze the role and relevance of experimentation in Informatics? Of course not. As the inspiration from science represents just one facet of this analysis, the attention to the way technical artifacts are evaluated in engineering experiments is also just only another partial facet. While discussing experiments and their role, the dual nature of Informatics at the intersection between science and engineering strongly emerges. Experiments are performed to *test* how well an artifact works with respect to a reference model and a metric (think for example of a robot or a program); but, at the same time, experiments are performed to *understand* how complex artifacts, whose behavior is hardly predictable, work and interact with the environment (in different degree if you think of the example of the robot or the program).

[1]It is worth noticing, however, that although an electron cannot be 'good' or 'bad' *per se*, a theory pertaining electrons can be 'good' or 'bad'.

Surely, the relevance of a better (in terms of rigor) experimental approach in computer science and engineering is an important first step in the methodological maturation of Informatics. Moreover, a deeper and a wider analysis is required to understand the peculiar role of experimentation in this discipline. Informatics is composed of very heterogeneous subfields and whether it is possible to single out some common features characterizing "good experiments" across these subfields is still under discussion.

An interesting side effect of this discussion is to promote a reflection on the disciplinary status of Informatics based on its methodological stances, and not only on the nature of its objects. This, again, evidences the peculiarity of Informatics also from a methodological point of view. As said, it makes sense to "import" traditional experimental principles (comparison, reproducibility and repeatability, justification and explanation) from traditional scientific disciplines into computer science and engineering, thus promoting a more rigorous approach to experimentation. This would avoid all those cases in which published results are considered as validated just by a single experiment that is impossible to reproduce and/or repeat because the experimental conditions are only vaguely described [6]. But it is worth remembering that these general principles are valid for disciplines (such as physics and biology) that aim at understanding and explaining natural phenomena, whereas computer science and engineering realize artifacts. This awareness can represent a first step in the direction of the development of a philosophy of engineering that should have, among its goals, the analysis of the features that characterize experiments in engineering disciplines.

3. COMPETING VIEWPOINTS ON EXPERIMENTATION IN COMPUTING

(*Matti Tedre*)

The discipline of computing was born at the conjunction of a number of research traditions that came together around the 1930s and the 1940s. The first research tradition came from mathematical logic and from the attempts to formalize human patterns of rational thought; its pioneers were mathematicians and logicians like George Boole, Gottlob Frege, and Alan Turing [11]. The second tradition was that of electrical engineering; its pioneers included engineers like Konrad Zuse, Claude Shannon, and John Atanasoff [51]. The third tradition came from the mechanization of numerical computation for the purposes of science and applied mathematics, with pioneers like Charles Babbage, Herman Hollerith, and Vannevar Bush [10].

The research traditions that formed modern computing continued to flourish within the discipline as three intertwined, yet partially autonomous lines of research, each with its own research agenda [14]. The theoretical tradition developed an identity autonomous of mathematics, and it established the groundwork for the development of theory and practice in computing. The engineering tradition drove the unprecedented development of machinery (and later software). The scientific tradition intertwined with other disciplines, giving birth to countless variations of computational science as well as new insights into computing itself.

The rapid growth of knowledge in computing fields was fueled by the interplay of the three intertwined traditions. However, the three very different intellectual traditions also caused continuous friction within the discipline of computing. The lack of rigor in some branches of engineering work gave rise to criticism from the science and theory camps of computing. The abstract orientation of theoretical computer science was accused of alienation from the real problems in computing. The empirical side of computing was criticized for improperly presenting what computing really is about.

3.1 Experimental Computer Science Movement

Science has always been a central part of computing as a discipline. Although its roots are in office machinery, the modern (fully electronic, Turing complete, digital) computer was born in universities, and the first modern computers were used for applied sciences and numerical calculation. The term 'computer science' was adopted into computing parlance in the late 1950s. As the discipline matured, the discussions about the disciplinary nature of computing frequently saw glimpses of experiment-based empirical research. But it was only at the turn of the 1980s when the role of experimentation in computing became a popular topic in the field's disciplinary discussions.

'Experimental computer science' was brought to limelight at the turn of the 1980s by a strong campaign for 'rejuvenating experimental computer science'. The campaign was initiated by a report to the National Science Foundation [18] and ACM Executive Committee's position paper on the "crisis in experimental computer science" [32]. The initial papers were followed by a number of position papers about what experimental computer science actually is (see e.g. [13, 38]).

However, despite the attempts to clarify experimental computer science terminology, it was never clear what exactly was meant by 'experimental' in computer science. The practitioners and pioneers from different traditions understood the term 'experimental' very differently. One sense of the word refers to exploratory work on new, untested techniques or ideas—the thesaurus gives words like 'trial', 'test', and 'pilot' as synonyms to 'experimental'. Another, more specialized, sense of the word refers to the use of experiments to test hypotheses. The original 'rejuvenating' report [18] teetered between the two meanings of the word, never defining what exactly was meant by 'experimental' computer science. Rooted in the different disciplinary mindsets in the three traditions of computing, the 'rejuvenating' report was followed by several decades of polemics where discussants talked about experimental computer science but meant different things.

3.2 Five Views on Experimentation

After the initial papers on experimental computer science, the topic became a popular topic for workshops, conferences, and journal articles. In those arenas experimentation terminology was used in various, often conflicting ways. One can easily find in the debates numerous implicit and explicit meanings of terms like 'experiment', 'experimenting,' 'experimental,' and 'experimentation'. Of the various meanings of 'experiment,' five are relatively common and easily distinguishable: the demonstration experiment, the trial experiment, the field experiment, the comparison experiment, and the controlled experiment.

3.2.1 The Demonstration "Experiment"

The first common use for the term 'experiment' can be found in reports on new tools and techniques. In those texts, it is not known if a task can be automated efficiently, reliably, feasibly, or by meeting some other simple criterion. A demonstration of 'experimental' technology shows that it can indeed be done (e.g., [25]). Although this view was criticized already in some of the first texts on experimental computer science, it is still a relatively common use of the term—often defended and often criticized.

3.2.2 The Trial Experiment

The second common use of the term 'experiment' can be found in reports that evaluate the performance, usability, or other aspects of a system against some previously defined specifications or variables. In those texts, it is not known how well a newly developed system meets its requirement specifications or how well it performs. A trial experiment is set up to evaluate the system. Varieties of trial experiments include, for instance, emulation, benchmarking, and simulation [24].

3.2.3 The Field Experiment

The third common use of the term 'experiment' can be found in reports that evaluate systems in their intended use environment. In those texts, it is not known how well the system works in the full richness of the live environment. In a field experiment [39], or '*in-situ*' experiment [24], the system's qualities are tested in its intended sociotechnical context of use, and evaluated against a pre-defined set of criteria.

3.2.4 The Comparison Experiment

The fourth common use of the term 'experiment' can be found in reports that compare two or more competing solutions for the same problem. In those texts, it is not known whether the author's proposed solution performs better than the previous solutions with some data set, parameters, and set of criteria. An experiment is set up to compare the solutions, and to show that the author's solution is in some ways better than the other candidates [19]. Oftentimes objectivity is improved by not involving the author's own solutions, and in many research fields the test data and parameters are standardized.

3.2.5 The Controlled Experiment

The fifth common use of the term 'experiment' can be found in reports that test models or hypotheses under a controlled environment, where the effects of extraneous and confounding variables can be controlled [17]. The controlled experiment comes in various types for various purposes, but often controlled experiments are used in situations where it is not known whether two or more variables are associated, or if one thing causes another. The controlled experiment is the approach of choice if the results should be generalizable.

4. BEYOND BENCHMARKING: STATISTICALLY SOUND EXPERIMENTATION

(*Kees van Hee*)

As noticed already computer science has two founding fathers: mathematics and electrical engineering. Mathematicians live in a world of models they create themselves, therefore mathematics is a pure theoretical discipline without an empirical component. The empirical law of large numbers seems to be the only empirical phenomenon that is mentioned in mathematical statistics. This law was the motivation for the derivation of the theoretical law of large numbers. Electrical engineering is based on empirical phenomena, such as the empirical laws of electricity. But experiments play a little role in the research domain of electrical engineering. Of course systems are built as a proof-of-concept, which is in fact an empirical existence proof. So the founding fathers of computer science had no experience and probably little interest in experimentation, which most likely caused of the lack of an experimental component in computer science.

Today model-driven engineering is a successful approach in almost all engineering disciplines and in particular in software engineering. There are several reasons to build a mathematical model of a system, such as: (1) to document or explain a system, (2) to specify or construct a system, (3) to optimize or control a system and (4) to analyze or predict the behavior of a system. In all these cases the model should have so much similarity with system that properties derived for the model should hold for the system and vice versa. Although this sounds obvious, it is far from easy to establish such a similarity relationship. Since a system is a physical object the only way to do this, is by experimentation. So one has to make a bijection between structural elements of the model and to the system. And one has to observe the behavior of the system and compare it with the modeled behavior. In most cases the behavior is an infinite set of sequences of events or activities. Hence it is often impossible to establish the similarity completely by experimentation. Then statistical methods should offer a solution. If we have enough evidence that a model is a good description of a system then one may extrapolate, by claiming that properties verified for the model also will hold for the system. This is in particular important if we want to analyze the behavior of a system under extreme circumstances that are difficult or impossible to realize in a experiment, such as the behavior of an aircraft during a crash.

In software engineering it was long time believed that we would reach a stage where we should be able to build a model from requirements and afterwards the program code from a model and that we should be able to verify formally that the model satisfies the requirements and the program code is conform to the model. Today model checkers can deal with models of systems of a realistic size. So part of the dream has come true. However we know that this is only a partial solution, since: (1) the program code is executed on a physical device, (2) requirements are never complete and often informal and (3) the model of a system is always a simplification in which we abstracted from certain details. So experimentation will be necessary to establish the similarity between a system and its model. This form of experimentation is called *software testing*. Although it is essential from an engineering point of view, the topic is far from mainstream research in computer science.

Software testing is only one form of experimentation in computer science. There are other questions that can only be answered by experimentation e.g.:

- effectiveness and efficiency of algorithms;
- quality of software engineering methods;
- usability of human-machine interfaces;
- visualization of very large data sets.

In the rest of this section we will focus on the first topic, quality of algorithms. Of course there are simple algorithms that can be analyzed formally: for example for a sorting algorithm we can prove that it delivers a right answer and we can compute the complexity. Note that complexity gives an answer for the worst case, while we often want an answer for something as the 'average case'. There are many complex problems in practice for which there are no best solutions known and where we develop heuristic algorithms. For example for combinatorial problems like the traveling salesman or timetabling for schools, we only have algorithms for which we do not know if it produces always the best possible answer (effectiveness) and the efficiency may vary strongly over the problem instances. In principle an algorithm is a function that can be applied to an argument and it delivers a result. The argument is a *structured data set*, which we will call a *model*. (Note that such a data set represents a part of the real world.) The set of all possible models is the *model class*. So a model is an instance of a model class. These concepts may also be called *problem instance* and *problem type*. The effectiveness of an algorithm is informally defined as: the 'fraction' of models where the algorithm gives the right answer, and the efficiency as: the 'average' computing time it takes to answer to produce the result. However, in most cases, the model class has infinitely many instances. So the 'fraction' and the 'average' are not even defined! If we have two or more algorithms for the same model class then we would like to compare them and then we run into the same problem. This is a similar problem as in software testing, where we may have to test an infinite set of behavior.

In order to solve these problems in practice, we seek resort in *benchmarking*. The idea is that some group of experts has chosen a finite sample from the model class and that all algorithms are applied to models of this benchmark set. Although it is a feasible solution it does not really answer the original questions. In order to do this in a more sophisticated way, we have to consider the model class. If we could define a *probability distribution* over this class, then we could speak about the probability of producing the right result and the mean computing time. So the solution to the problem is to find probability distributions for model classes!

This is what we will consider in the next subsection. We will restrict us to the case where models are represented by graphs and we will show how we can define probability distributions on the class of graphs. We also will discuss how we could obtain this distribution in an empirical way. Finally we describe a case study of this approach.

4.1 Probabilistic model classes

In most cases an algorithm is applicable to an infinite model class. The 'effectiveness' should be the 'fraction' of models for which the algorithm gives the right answer and the efficiency the 'average' computing time. However on an infinite set the 'fraction' and the 'average' are not defined. If we have a probability distribution on the model class, then the effectiveness and efficiency are at least *defined*. The question is then how to *compute* them. So there are four questions to be answered:

1. How to define a probability distribution for a model class?
2. How to sample models from a model class?
3. How to identify the parameters of a probability distribution from empirical data?
4. How to compute the quality measures if the probability distribution is known?

We start with the last question. Since in most cases we should really apply the algorithm on the models to determine the quality. However this would require infinitely many runs. Since we have a probability distribution we can approximate effectiveness and efficiency by computing it for a finite subset of the model class, say a subset with probability q, e.g $q = 0.99$. This subset is a *sample* from the model class. The effectiveness can be computed for the sample, say p, which means that the algorithm gives the right result in fraction p of the sample. Then we know that the probability $\tilde{p}$ of the model class satisfies: $q.p \leq \tilde{p} \leq q.p + (1-q)$, i.e. $0.99.p \leq \tilde{p} \leq 0.99.p + 0.01$. For the computing time t we can use the law of large numbers and the central limit theorem derive a confidence interval for t. Actually the sample can be considered as a benchmark! However in this case we know how much the benchmark covers the whole model class and we can generate for each experiment a different benchmark.

The first and second question can be answered simultaneously. Note that an infinite model class, where each model has the same probability, does not exists (they would all have probability zero and their sum should be one). Observe that systems in nature and man-made systems are not coming out of the blue, but they are grown or constructed in a systematic way. For instance live tissues are grown by cell division and cell differentiation. Software systems are normally built in a top down or bottom up process. In the first case a simple program is stepwise refined to a more complex one and in the second case several small components are glued together to construct a bigger one. Each intermediate step is also a system of the class. We will use this idea to construct model classes together with a probability distribution over it.

From now on we will assume that each model is represented as a *graph*. In computer science graphs are probably the most used data structure (see [5]). A *graph* can be defined as a quintuple $(Node, Edge, \sigma, \tau, \lambda)$ where $Node$ is a set of nodes, $Edge$ a set of edges (connecting the nodes) and $\sigma : Edge \rightarrow Node$ is called the *source* function, $\tau : Edge \rightarrow Node$ is called the *target* function and λ is a *labeling* function with $Edge$ as domain. We will define graphs by *graph transformations* that can be applied to a given graph in the class in order to obtain a new one in the class. Graph transformations are defined by *production rules*. As in [5] we follow [15] in a simplified way. A *production rule* is a triple (L, K, R) where L, K and R are all graphs and K is common subgraph of L and R. (Often K has only nodes, i.e. the set of edges is empty). The first step in the application of a production rule is that in a graph G the graph L is *detected*, *i.e.* a subgraph of G is found that is isomorphic with L. The second step is that the graph G is transformed into $\tilde{G} = G\backslash(L\backslash K) + (R\backslash K)$. The graph K is the *interface* between G and L and R, *i.e.* $G\backslash L$ and $R\backslash K$ are only connected via K. (Here we use $\backslash$ for graph subtraction and $+$ for graph addition, i.e. nodes and edges). There are some "sanity" requirements for these operations such that all intermediate results are proper graphs. A *graph transformation system* is a set of production rules and a *graph transforma-*

tion system together with an initial graph is called a *graph grammar*. With a graph grammar we may define a class of graphs: all graphs that can be derived by applying the production rules in a finite number of steps. The production rules may enforce that we only produce graphs of a certain type. We may distinguish transformation rules that expand a graph, called *expansion rules* and transformation rules that reduce a graph, called *reduction rules*. In fact every expansion rule may be applied in the reversed order to become a reduction rule and vice versa. (If (L, K, R) is an expansion rule, the corresponding (R, K, L) is a reduction rule). So if we have one or more initial graphs and a set of expansion rules, we have defined implicitly a, possibly infinity, set of graphs. To add a probability distribution we will endow the production rules with a *non-negative weight function*. Given a graph G a new graph G' is generated by applying an expansion rule, with probability equal to the weight of the rule divided by the sum of the weights of all rules that could be applied. Note that the same rule can be applied maybe several times on graph G. Now we are almost done: we only need a *stopping rule* for this generation process. This can be done in several ways, e.g. by sampling the number of generations from distribution before we start the generation process, or 'flipping a coin' during the generation process to determine to stop or to continue. In both cases the number of expansion steps is stochastically independent of the generated graph. A more sophisticated stopping rule could be based on a Markov chain where the state is the last generated graph, but we will not consider this here. So, by repeating the generation process, we have a method to determine an arbitrary large sample from a graph class with a specified total probability.

The third question is about the application of the approach in practice. For comparison of the quality of algorithms a team of experts could decide on the production rules, their weights and a stopping rule. Then every developer could evaluate an algorithm by creating a large sample and test its algorithm. This is much better than the existing benchmarking practice since we have arbitrary large samples and we know what the evaluation means for this whole class.

However we can apply the approach in an even better way, since we often have already an *empirical sample* of the class observed in practice, e.g. a sample of problem instances of the traveling salesman or timetabling problems. So it would be interesting to see if we can derive the stopping rule as well as the weights of the expansion rules from this empirical sample.

Sometimes this can be done in a quite straightforward way by applying the expansion rules as reduction rules to obtain one of the initial graphs, while counting the number of applications of each rule. So we obtain an estimate for the weights of the rules and since we count the number of steps as well, we have a sample of the stopping time distribution, from which we may estimate the parameters of the stopping rule distribution. In case there are more reduction paths we have to be a little more careful, but a similar approach seems feasible. Given these parameters we can generate a large sample, much larger than the empirical sample, and use this sample to evaluate the algorithm. As shown in [49] this gives much better results than using the original sample as benchmark. Although this sounds as 'magic', it is a phenomenon similar to the bootstrapping technique of statistics (see [44]). A simple explanation is that the graphs in the empirical sample contain more information than we use when we just use it as benchmark. Even in a small empirical sample we may have a lot of information over the weights of the production rules.

4.2 Case study: Petri nets

This case study is based on [49]. Here we will illustrate the approach for *Petri nets*, which are bipartite graphs with two kind of nodes, called *places* and *transitions* and they are only connected with directed arcs to nodes of the other kind. Petri nets have *markings*, which are distributions of objects, called *tokens*, over the places. When a transition has in each of its input places a token, the marking may change by consuming these tokens and producing for each output place of the transition a new token. So a Petri net determines a transition system. We consider a special class of Petri nets called *workflow nets*. These nets have one input place, one output place (called initial and final place respectively) and all other nodes are on a directed path from the input place to the output places. As initial marking they have only one token in the initial place. They are used frequently in practice to model business processes or procedures. We used 9 very simple expansion rules, see Figure 1. The initial graph consists of just one place. All 9 rules preserve the property that the generated nets are workflow nets. The first 6 rules also preserve a behavioral property, the *weak termination property* which says that if a marking is reachable from the initial marking, then the marking with only one token in the final place is reachable from here.

We applied the approach to a collection of 200

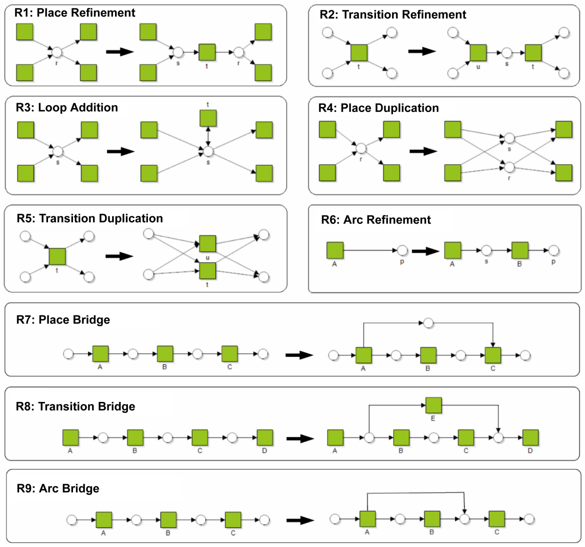

Figure 1: The generation rules for workflow nets

workflow nets representing business processes from a manufacturing company. We consider this set as original model class, which is finite. In order to validate our approach, we took 10 samples of 5 models each from the given 200 models. And with each of these samples we derived the parameters and we generated 200 workflow nets. Instead of testing algorithms with these generated sets, we computed characteristics of the graphs themselves. In particular we considered two characteristics: the *length of the shortest path* (LSP) from the initial place to the final place, which is a structural property of the nets and we determined if the net has the *weak termination property* (WTP) or not.

The results are as follows. First we consider the LSP. In the original population the mean and standard deviation of the LSP are: 7.14 and 0.85. In the first sample of five models it was: 6.0 and 2.9 respectively, which falls outside a 95 percent confidence limit of the original mean. For the collection of 200 generated from the first sample the values are: 7.34 and 0.95 respectively and it fits in the interval of the original mean, in fact the intervals are almost the same. This was the case with the first sample. We repeated this for the other 9 samples and we computed the average and standard deviation of the mean LSP values over these 10: 7.29 and 1.46 respectively.

For the weak termination property (WTP) we followed the same procedure. All models in the original collection had the WTP and so all the models in the 10 samples of five as well. The reduction process preferred the first 6 rules that preserve soundness, however the original models were not generated with our rules (as far as we know!) and so we also had to apply not WTP preserving rules. It turned out that in only 3 of the 10 generated collections there were models not satisfying the WTP. They had a probability of WTP of 0.96, 0.89 an 0.85. They average probability was 0.97 with a standard deviation of 0.05.

4.3 Summary

We have seen that for the comparison of algorithms empirical methods are essential and that there is a better approach than classical benchmarking. The key is that we need a probability distribution over infinite sets of models. We sketched a method to construct such a probability distribution and to generate a large sample:

1. observe an empirical sample of models;
2. define a suitable model class by production rules and an initial model;
3. define a stopping rule;
4. identify the parameters: estimate the weights for the production rules and the stopping rule;
5. generate a large sample using the production rules and the parameters;
6. run the algorithm for each model of the generated sample.

In a case study for a special class of Petri nets we showed how the method works and that it is much better to use a large amount of generated models using the parameters obtained from a small empirical sample, than using this sample directly. This case study encourages us to apply the method for other model classes.

5. ROLE OF USER STUDIES IN COMPUTER GRAPHICS AND VIRTUAL REALITY

(*Carlos Andujar*)

We now take a closer look at a kind of experimentation which also plays a major role in several of computer science fields: experimentation involving human subjects. Conducting experiments involving humans is a challenging task and poses a number of problems not found in other types of experiments. We will focus on user studies in three user-centric fields, namely Computer Graphics (CG), Virtual Reality (VR), and 3D User Interfaces (3DUI).

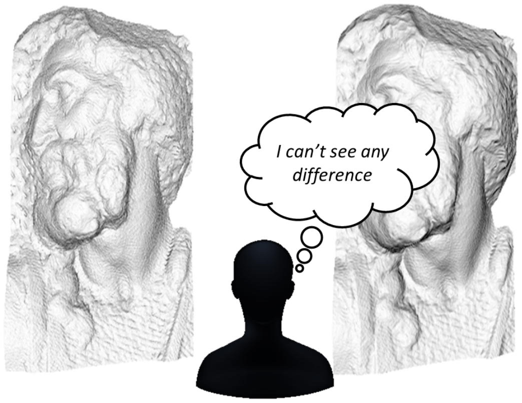

Figure 2: Visual equivalence problem

5.1 Sample problems

Let us introduce some problems that will serve to illustrate the need for user studies in CG-related areas and to exemplify some major challenges. Our first example refers to the visual equivalence problem. Consider for example the two images in Figure 2. These images look nearly identical despite the image on the right has been rendered with a level-of-detail algorithm which is much faster but less accurate. This represents a speed/quality tradeoff, and the most appropriate algorithm will depend on the different conditions (such as viewing distance, saliency of the image differences) that determine to which extent users will perceive the two images as the same. This kind of question requires a user study and probably a psychophysical experiment [20]. Although several image metrics have been proposed to compare pairs of images, and some of them take into account key features of human perception, there is some evidence that these metrics often fail to predict the human visual system response [41]. Indeed, there is an increasing interest of the CG community to get a deep understanding of the Human Visual System, as evidenced by some top journals (e.g. ACM Transactions on Applied Perception) aiming to broaden the synergy between computer science and psychology.

The second example is the evaluation of presence in VR systems, that is, to which extent users feel and behave as if physically present in a virtual world. Users of immersive systems might forget about the real environment and the virtual environment can become the dominant reality. The evaluation of presence is very important in many VR applications, from phobia therapies to psychological experiments through pain relief for patients with serious injuries. Many presence evaluation studies report surprising findings when analysing the human behaviour when presented a stress situation in a virtual environment [34, 43]. Nowadays, the common practice is to evaluate presence by observing users's behavior and measuring their physiological response (as captured by specialized devices such as heart rate monitors and galvanic skin sensors).

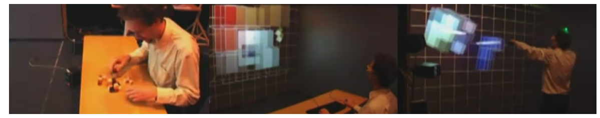

Figure 3: Different interfaces for a puzzle-solving problem

The last example is about the comparison of 3D UIs in terms of their usability. Figure 3 shows three different user interfaces to solve a 3D puzzle [42], using either a physical puzzle, a classic keyboard-and-mouse interface, or a Wii controller. For this task, users must be able to select pieces, manipulate them, and explore the model from different viewpoints. The key problem thus is to determine which UI is better in terms of usability, and the only solution nowadays is to conduct a user study comparing these techniques. Despite some models have been proposed to predict human performance for some tasks (the well known Fitts' law is the most notable example), typical spatial tasks are just too complex to be predicted by such simple models.

5.2 Major challenges in user studies for CG and VR

The following list (adapted from [45]) shows the typical steps involved in empirical methods involving human subjects.

1. Formulate a hypothesis;
2. Make the hypothesis testable;
3. Design an experiment;
4. Get approval by ethics committee;
5. Recruit participants;
6. Conduct the experiment and collect data;
7. Pay participants;
8. Analyze the data;
9. Accept or refute the hypothesis;
10. Explain the results;
11. If worthy, communicate your findings.

The list above will serve to guide the discussion about some major challenges in user studies not found in other types of experiments.

5.3 Hypothesis formulation

A first step is to formulate a hypothesis and make it testable. Using the 3D puzzle problem as an example, a general hypothesis might be formulated as follows: *using the Wii controller will make people more effective when doing manipulation tasks.* A possible testable version of it could be formulated as: *we measured the time it takes for users to solve a particular 3D puzzle, using either Wii or mouse; we hypothesize users will be faster using the Wii.* Here we find a major problem: to make the hypothesis testable, we had to choose a particular task which we take as representative (besides fixing some other important variables). Unfortunately, many problems have a task space so large and heterogeneous that it can be really difficult to find a small set of representative tasks. Here we wrote *task* for the 3D puzzle example, but we could have written 3D model, image, movie, stimulus and whatever other elements are fixed to make the hypothesis testable.

Complex tasks also depend on a number of variables. For example, the completion time for a 3D puzzle might depend on the interaction device (Wii, mouse), viewing conditions (stereoscopic, mono), mapping between user actions and application changes, and quality of the graphics [9]. The more variables are controlled, the more general the findings will be, but the more difficult will be the data collection. Indeed, data collection is a further issue in typical VR applications. Compare the measurement of simple dependent variables such as task completion times and error counts, with hard-to-collect data such as user physiological response, heart rate or galvanic skin response.

5.4 Experiment design

Independent variables can vary in two ways: within-subjects (each participant sees all conditions) and between-subjects (each participant sees only one condition). The decision on which design to use is often controversial. Within-subject designs are more time consuming for the participants, and require the experimenter to counterbalance for learning effects and fatigue effects. But between-subject designs are not free from limitations, as more participants need to be recruited and we are likely to loose statistical power (thus less chances to proof our hypothesis).

5.5 Ethics

Many usability guides address in depth all the ethical issues related with user studies [3, 16]. Most organizations require experimenters to get the approval by an ethics committee before running the experiment. After the approval, it is often a hard task to recruit participants and get their informed consent, in particular when participants should be chosen from a specific target user group (such as physicians).

Experiments involving immersive VR systems often need a detailed informed consent. Researchers should never deceive participants about aspects that would affect their willingness to participate, such as risks (VR users might bump into walls, trip over cables), discomfort (many 3D interaction techniques for spatial tasks are physically demanding) and unpleasant experiences (some VR systems cause motion sickness). Frustration handling is also important when measuring user performance. In case of failure to complete a task, experimenters should make it clear that the responsible is the technology. Usability tests should not be perceived as tests of the participant's abilities [16].

5.6 Experimenter issues

Experimenters should be careful to avoid manipulating the experiment. Besides the well known placebo effect, there are other experimenter issues that often hinder data collection. The Hawthorne effect occurs when increased attention from superiors or colleagues increases user performance. The performance of a participant might change if somebody else, e.g. the previous participant, is observing. Observer-expectancy effect occurs when the researcher unconsciously manipulates the experiment, using for example body language. Experiments should be double-blind, but researchers in non-life critical fields often disregard these issues.

5.7 Data analysis

Proper data analysis is absolutely required to accept or refute the hypothesis and to provide statistical evidence of the findings. Unfortunately, a part of the CG community seems to lack enough background on experimental design and statistical analysis to conduct the user studies required to evaluate their own research. This is evidenced by the large number of submitted and even published papers with serious evaluation errors related with the statistical analysis of the results. Not surprisingly, some leading CG groups around the world count on psychologists' contributions.

5.8 Summary

In the last decades some computer science disciplines are experiencing a shift of focus from *implementing the technology* to *using the technology*, and empirical validation through user studies is becoming critical. In this section we have discussed some major issues of such validation experiments: lack of background on experimentation, psychology and psychophysics, time-consuming and resource-consuming nature of user studies, and the difficulties to fulfill all requirements (double-blind experiments, informed consent, representative users, representative data sets/models/tasks).

The user performance during typical computer-related task depends on a number of domain-specific factors as well as hardware-related factors. Considering all these factors simultaneously as independent variables in controlled experiments is clearly not practical. This fact limits the validity of the findings reported in the CG and VR literature to a specific domain and a particular setup. The lack of *de-facto* standard data sets for testing purposes (more common in other scientific communities) along with the plethora of hardware setups makes it difficult to make fair comparisons. Furthermore, many techniques are still proposed and evaluated in isolation, whereas in the real world user tasks are mixed with other tasks. These are issues that must still be addressed.

6. EXPERIMENTS IN COMPUTER SCIENCE: ARE TRADITIONAL EXPERIMENTAL PRINCIPLES ENOUGH?

(*Fabio A. Schreiber*)

Francis Bacon, in his *Novum Organum*, observes that "... simple experience; which, if taken as it comes, is called *accident*, if sought for, *experiment* ..."[4]. This fundamental observation calls for the establishment of a methodology in experimentation, which cannot be based on the observation of casual events and their simplistic interpretation, but must rely on accurately designed and rigorously performed experiments. On the other hand, Louis Pasteur accepted some degree of casualty in discovery, but stated that: "In the field of observation, chance favors only the *prepared mind* ..." [40], so admitting that only a precise framework can produce meaningful research results.

Figure 4 shows how, in natural sciences, observational studies are performed: from the observation of natural phenomena some hypotheses are formulated which lead to experiments in order to formulate theories which, in turn, suggest further experimentation until a well established theoretical framework is reached (a); sometimes, however, observations are not casual, but they are induced by autonomously formulated conjectures which must be experimentally proved (b).

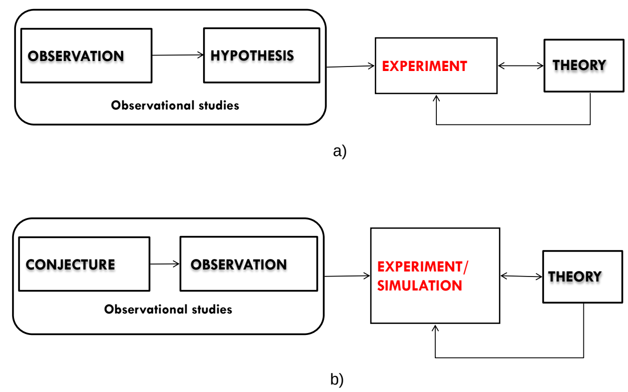

Figure 4: Observational studies

On this basis, the final panel [2] of ECCS 2012 Summit tried to answer some questions the first of which is: *Is this model applicable to Computer Science/Engineering?*

So, is Informatics a "natural science"? One of those disciplines that are blessed with the legacy of Galileo? If not so, would that be somehow desirable? Would that even make any sense? And, further: whether or not experiments are (or ought to be) strictly related to the way computer science advances, have they any other role in it?

6.1 Experimentation Goals and Properties

Matti Tedre, in *Section 2*, mentioned many reasons for experimenting and here we recall four of them:

- to discover the unknown - in this case we want to answer questions like: "What does it happens if I mix Oxygen and Hydrogen?";
- to test a hypothesis - "From a stoichiometric computation, it comes out that if I mix two molecules of Hydrogen and one molecule of Oxygen I get some energy and a molecule of water; is it true?";
- to determine the value of some physical variable - "What is the speed of light in vacuum?";
- to compare a set of different "objects" to determine their relative merits (benchmarking) - "Which, among a set of cars has the best energetic performance?"

[2]The panelists were Francesco Bruschi, Natalia Juristo, Antoine Petit, Matti Tedre moderated by Fabio A. Schreiber

All of these classes must satisfy at least three methodological conditions for an experience to be considered an experiment:

- Repeatability at different times and in different places to check the universality of results;
- Reproducibility by other scientists to confirm that results are independent of the details of the specific experiment;
- Comparison of the results of different instances of the same experiment;

Then a second question is: *How such goals and properties apply to Computer Science/Engineering theories and artifacts?*

Francesco Bruschi put the focus on the fact that hard experimentation is already present in the day to day practice of applied informatics. Not only: some of the requirements for an experience to be considered an experiment have both a positive and a normative role in the engineering fields related to computer science.

Considering the cited methodological conditions for an experience to be considered an experiment - repeatability, reproducibility, and comparability of the results - Francesco argued that there are two ways in which these requirements have a role in the practice of Informatics: they have *positive value* in describing how some tasks are carried out and accomplished, and they have *normative value* with respect to the definition of some tools and methodologies. Francesco proposed an example for each role.

Let us start with the positive role, considering the task of modifying a software system in order to correct a behavior not compliant with the specifications (practice commonly referred to as *debugging*). The cornerstone of debugging is the definition of at least one test case able to highlight the aberration from the expected behavior. A test case is but the description of a setup (a set of boundary conditions) such that: i) it must be possible to impose it on a running instance of the system an arbitrary number of times (it must be *repeatable*); ii) it must be possible to impose it for any other instance of the computing architecture on which the software is intended to be executable, everywhere else (it must be *reproducible*); iii) the execution of the system with the given conditions must highlight, in an unambiguous way, the deviation from the expected behavior (results of the test running after changing the code must be clearly *comparable*). The interesting thing is that, even though at various levels of awareness, these requirements soak the engineering practice at any level.

As far as the normative role of the experimental requirements is concerned, let us now consider the interest recently gained, in the software engineering field, by purely functional languages such as Haskell. It is advocated that these languages are superior at developing robust, maintainable, complex software systems. If we look closer at the features that define functional purity, we find that the core ones can be stated this way:

i) a function, whenever invoked with the parameters, must always produce the same result (i.e., the invocation must be *repeatable*);

ii) the invocation of a function contains all the information on the boundary conditions for the execution; this means, for instance, that there is no hidden state, which makes it *much easier to reproduce* a given particular execution;

iii) all the possible side effects (i.e.: I/O, state modification) produced by a function invocation are explicitly inferable from the function call (i.e.: the set of effects of two different functions calls are *comparable*). The remarkable thing here is that the features which in the scientific domain are definitional for experiments, somehow act as desiderata in the field, noteworthily central to computer science, of programming language design.

So, rigorous experimentation already has a role, widespread albeit implicit, in at least two domains of computer science, and it would be interesting to deepen the possible epistemological and didactic consequences of this fact.

Anyhow, Natalia Juristo pointed out that software engineering (SE) experimental paradigm is still immature: i) SE experiments are mostly exploratory and they lack mechanisms to explain, by means of inference steps, the meaning of the observations results; ii) SE experiments have flaws since they lack thoroughly thought-out designs to rule out extraneous variables in each experiment, and proper analysis techniques are not always used.

Natalia further observed that it is not enough just to apply experimental design and statistical data analysis to take experimental research in SE forward, since a discipline's specific experimental methodology cannot be imported directly from others.

6.2 Design and Simulation

A point on which all the speakers agreed is the need of adopting a well defined language to give rigor and precision to experimental data and of using rigorous measurement methods and tools to quantitatively describe the phenomena under investigation. The role of Statistics in both the design of an experiment and in the interpretation of its results emerged unanimously.

Antoine Petit also pointed out how software, besides being the object of some experimental activity, as shown in 6.1, is also a tool for making experiments in a similar way a telescope is the object of research and experimentation in the field of optics and a tool for astronomical research and discovery. Computer scientists need to have their own research infrastructures as physicists have their lasers or the Large Hadron Collider, astronomers their space telescopes, or biologists their animal houses. Some of our research need such similar huge infrastructures, in particular to produce results at a right scale to be transferred to industry. We can think for instance to immersion systems, cloud computing, smart dust, robots, but these infrastructures can be useful only if there is enough technical staff besides the researchers.

A discussion followed about the usage of simulation models and frameworks to make scientific experimentation cheaper and faster than in real-life. While simulation, since longtime, is successfully used to predict the performance of computing systems, its application to disciplines other than Informatics (Natural Sciences, Physics, Economy, etc.) requires a careful interaction between people with different cultures and languages in order to avoid misunderstandings leading to erroneous results which often are not the fault of "... that damn computer".

6.3 Other Issues

New experimental activities emerged in the recent years in Information Management; two among them have been explicitly mentioned in the discussion: i) Data Mining for knowledge discovery in large amount of operational data and ii) Pervasive Systems support to sensing real-life physical data to be used as input to application programs which compute the experiments' output. These applications are very far from each other, but *how do they compare to the classical notion of "experiment"? Do we need any new vision?*

Contrary to the position of Natalia Juristo, Antoine Petit argued that there is no specificity of Informatics with respect to other sciences and that its experimental dimension amounts to the experimental dimension of software. We use software as astronomers use telescopes, but to study and "construct " software is also part of our job, whereas astronomers do not study or construct telescopes. This induces that the work done by researchers to study and construct software has to be evaluated, as classical publications are. A difficulty comes from the few number of conferences or journals devoted to software; an organized Software Self-Assessment could be an interesting option for such an evaluation.

In the discussion which followed, a last important question emerged: *"Are CS/CE curricula suitable for giving our students an experimental awareness?"*. At a first sight, it seems that, generally, engineering curricula include some separate, and sometimes optional, courses about Measurement Theory and Statistics, while these are not always found in Computer Science curricula, but, in any case there is no general treatment of the principles of experimentation. *This could be the topic for a deeper survey on the state of art and a subsequent proposal for the inclusion of an ad-hoc course in CS/CE curricula.*

7. CONCLUSIONS

(*Viola Schiaffonati* and *Jan van Leeuwen*)

The *experimental method* is clearly gaining ground in computer science. It is now widely recognized that experimentation is needed as a key step in the design of all complex IT applications, from the design of the processes (Empirical Software Engineering) to the design of the systems (as in Robotics). The quest for experimentation manifests a renewed attention for rigorous methodologies. Although many experiments have been put to good use over the years, more recent efforts emphasize the *systematic* adoption of the experimental approach. However, it is less clear that the precise use of experimentation is always well-understood. Can we simply adopt the experimental method from classical disciplines or engineering? If not, in what ways should it be modified to serve as a sound scientific methodology for the computing and information sciences?

In this paper we first focused on the origins and the potential uses of the experimental method in the field of Informatics (Section 1). We highlighted the specific context of the design of artefactual systems and their requirements (Section 2), and the different views on experimentation that can be encountered in the field (Section 3). We showed subsequently that experimentation in Informatics can be aimed at a variety of different properties not found in other sciences (Section 4), and might involve e.g. the use of human subjects as well (Section 5).

In a final appraisal, we observe that the experimental method is well-recognized in various areas within (applied) Informatics, but that its rigorous application needs to be investigated further (Section 6). Design and simulation methods need to be carefully described and documented, so it is clear to everyone when they count as instances of experimentation and when they do not.

We recommend that modern Informatics curricula offer an adequate background in both the philosophy and the potential uses of the experimental method. Furthermore, experiments in Informatics should follow sound, replicable, and verifiable protocols, as is in all other sciences and in engineering (Section 1). Experimentation should be recognized in all branches of Informatics as a crucial methodology, with well-defined rules and practices.

This paper resulted from a collaborative effort of the authors after the Workshop on *the Role and Relevance of Experimentation in Informatics*, held in Barcelona in 2012 prior to the 8th European Computer Science Summit of Informatics Europe. The paper also includes the conclusions from the subsequent panel on experimentation at the Summit (Section 6).

Both the workshop and the panel raised many interesting questions about the experimental method as it is to be used in Informatics. In this paper we have only started to discuss some of the questions, as one of our aims was rather to show how many relevant issues are still open and need to be dealt with, from a variety of different perspectives. When properly applied, experimentation is a powerful component of the methodological basis of Informatics as a fully fledged science.

Acknowledgements

We thank *Informatics Europe* for the opportunity to organize our Workshop in conjunction with its ECSS 2012 summit meeting. We also thank the Department of LSI of the Universitat Politècnica de Catalunya (UPC) in Barcelona for the excellent local arrangements. Finally, we greatly acknowledge the contribution of Francesco Bruschi and the comments of Antoine Petit to Section 6.

8. REFERENCES

[1] F. Amigoni, M. Reggiani, and V. Schiaffonati. An insightful comparison between experiments in mobile robotics and in science. *Autonomous Robots*, 27:4 (2009) 313–325.

[2] F. Amigoni, V. Schiaffonati. Good experimental methodologies and simulation in autonomous mobile robotics. In: L. Magnani, W. Carnielli, and C. Pizzi (Eds.), *Model-Based Reasoning in Science and Technology*, Springer, Berlin, 315-322, 2010.

[3] American Psychological Association. Ethical principles of psychologist and code of conduct. *American Psychologist*, 47 (1992) 1957–1611.

[4] F. Bacon. *Novum Organum*. 1620.

[5] J.C.M. Baeten, K.M. van Hee. The role of graphs in computer science. *Report CSR 07-20*, Eindhoven University of Technology, 2007.

[6] M. Barni, F. Perez-Gonzalez. Pushing science into signal processing. *IEEE Signal Processing Magazine*, 120 (2005) 119–120.

[7] Th. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, Springer, Berlin, 2010.

[8] V.R. Basili, The role of experimentation in software engineering: past, current and future, *Proc. 18th Int. Conf. Software Engineering* (ICSE'96), IEEE Press, 1996, pp. 442-449.

[9] D. Bowman, J. Gabbard, and D. Hix. A survey of usability evaluation in virtual environments: classification and comparison of methods. *Presence: Teleoperators and Virtual Environments*, 11:4 (2002) 404–424.

[10] M. Campbell-Kelly, W. Aspray. *Computer: A History of the Information Machine*, Westview Press, Oxford (UK), 2nd edition, 2004.

[11] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*, CRC Press, Boca Raton, FL, 2012.

[12] P. J. Denning. What is experimental computer science. *Communications of the ACM*, 23:10 (1980) 543–544.

[13] P. J. Denning. ACM president's letter: On folk theorems, and folk myths. *Communications of the ACM*, 23:9 (1980) 493–494.

[14] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young. Computing as a discipline. *Communications of the ACM*, 32:1 (1989) 9–23.

[15] H. Ehrig, K. Ehrig, U. Prange and G. Taentzer. Fundamentals of Algebraic Graph Transformations. *EATCS Monographs on Theoretical Computer Science*, Springer, 2006.

[16] European Telecommunications Standards Institute (ETSI). Usability evaluation for the design of telecommunication systems, services and terminals. ETSI Guide EG 201472. Sophia Antipolis, 2000.

[17] D. G. Feitelson. Experimental computer

science. *Communications of the ACM*, 50:11 (2007) 24–26.

[18] J. A. Feldman, W. R. Sutherland. Rejuvenating experimental computer science: A report to the National Science Foundation and others. *Communications of the ACM*, 22:9 (1979) 497–502.

[19] P. Fletcher. The role of experiments in computer science. *Journal of Systems and Software*, 30:1–2 (1995) 161–163.

[20] G. Gescheider. *Psychophysics: The Fundamentals*, 3rd Edition. Lawrence Erlbaum Associates, Mahwah, NJ, 1997.

[21] G.F. Forsythe, A university's educational program in Computer Science, *Communications of the ACM*, 10 (1967) 3–11.

[22] M. Franssen. G. Lokhorst, and I. van de Poel. Philosophy of technology. In: E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy*(Spring 2010 Edition), 2010.

[23] P. Freeman. Back to experimentation. *Communications of the ACM*, 51:1 (2008) 21–22.

[24] J. Gustedt, E. Jeannot, and M. Quinson. Experimental methodologies for large-scale systems: A survey. *Parallel Processing Letters*, 19:3 (2009) 399–418.

[25] J. Hartmanis, H. Lin. What is computer science and engineering? In J. Hartmanis and H. Lin (Eds.) *Computing the Future: A Broader Agenda for Computer Science and Engineering*, 163–216. National Academy Press, Washington, D.C., 1992.

[26] W.S. Jevons, *The principles of science : a treatise on logic and scientific method*, MacMillan and Co., London, 1874.

[27] N. Juristo, A.M. Moreno, *Basics of Software Engineering Experimentation*, Kluwer Academic Publishers (Springer), Dordrecht, 2001.

[28] K. Keahey, F. Desprez (Eds.), *Supporting experimental computer science*, ANL MCS Technical Memo 326, Argonne National Laboratories, Argonne, IL, 2012.

[29] G. Leroy, *Designing User Studies in Informatics*, Springer, London, 2011.

[30] P. Lukowicz, E.A. Heinz, L. Prechelt, and W.F. Tichy. Experimental evaluation in computer science: A quantitative study. *J. Systems Software*, 28:1 (1995) 9–18.

[31] P. Lukowicz, S. Intille, Experimental methodology in pervasive computing, *IEEE Pervasive Computing*, 10:2 (2011) 94–96.

[32] D. D. McCracken, P. J. Denning, and D. H. Brandin. An ACM executive committee position on the crisis in experimental computer science. *Communications of the ACM*, 22:9 (1979) 503–504.

[33] C.C. McGeoch, *A Guide to Experimental Algorithmics*, Cambridge University Press, Cambridge (UK), 2012.

[34] M. Meehan, B. Insko, M. Whitton, and F. P. Brooks. Physiological measures of presence in virtual environments. *ACM Transactions on Graphics*, 21:3 (2002) 645–652.

[35] R. Milner. Is computing an experimental science? *Report* ECS-LFCS-86-1, Laboratory for Foundations of Computer Science, University of Edinburgh, 1986.

[36] C. Morrison, R. Snodgrass. Computer science can use more science. *Communications of the ACM*, 54:6 (2011) 38–43.

[37] A. Newell, H.A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19:3 (1976) 113–126.

[38] J. K. Ousterhout. More on experimental computer science and funding. *Communications of the ACM*, 24:8 (1981) 546–549.

[39] P. Palvia, E. Mao, A. F. Salam, and K. S. Soliman. Management information systems research: What's there in a methodology? *Communications of the Association for Information Systems*, 11(16):1–32, 2003.

[40] L. Pasteur. *Lecture*, Université de Lille, Dec. 7th, 1854.

[41] G. Ramanarayanan, J. Ferwerda, B. Walter, and K. Bala. Visual equivalence: towards a new standard for image fidelity. *ACM Transactions on Graphics*, 26:3 (2007) 1–11.

[42] Reality-based User Interface System (RUIS). IEEE 3DUI contest entry. Department of Media Technology, Aalto University.

[43] M. Slater, A. Antley, A. Davison, D. Swapp, G. Guger, C. Barker, N. Pistrang and M. Sanchez-Vives. A virtual reprise of the Stanley Milgram obedience experiments, *PLoS ONE* 1:1 (2006): e39.

[44] M.J. Shervish. *Theory of Statistics*, Springer Verlag, New York, 1995.

[45] J. Swan, S. Ellis and B. Adelstein. Conducting human-subject experiments with virtual and augmented reality. In *Proc. of IEEE Virtual Reality 2007 Tutorials*, Charlotte, NC, March 10–14, 2007.

[46] M. Tedre. Computing as a science: A survey of competing viewpoints. *Minds & Machines*,

21 (2011) 361–387.

[47] M. Tedre, *The Development of Computer Science: A Sociocultural Perspective*, Doctoral thesis, University of Joensuu, 2006.

[48] W. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31:5 (1998) 32–40.

[49] K.M. van Hee, M. La Rosa, Z. Liu and N. Sidirova. Discovering characteristics of stochastic collections of process models. In: S. Rinderle-Ma, F. Toumani, K Wolf (Eds.) *Business Process Management 2011*, LNCS vol. 6896, Springer, 298–312, 2011.

[50] P. Vermaas, P. Kroes, I. van de Poel, M. Franssen, and W. Houkes. *A Philosophy of Technology. From Technical Artefacts to Sociotechnical Systems*, Morgan and Claypool, 2011.

[51] M. R. Williams. *A History of Computing Technology*. IEEE Computer Society Press, Los Alamitos, CA, 2nd edition, 1997.

[52] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén. *Experimentation in Software Engineering*, Revised Edition, Springer, Heidelberg, 2012.