Six strategies for generalizing software engineering theories

Roel Wieringa *, Maya Daneva

University of Twente, The Netherlands

Abstract

General theories of software engineering must balance between providing full understanding of a single case and providing partial understanding of many cases. In this paper we argue that for theories to be useful in practice, they should give sufficient understanding of a sufficiently large class of cases, without having to be universal or complete. We provide six strategies for developing such theories of the middle range. In lab-to-lab strategies, theories of laboratory phenomena are developed and generalized to other laboratory phenomena. This is a characteristic strategy for basic science. In lab-to-field strategies, theories are developed of artifacts that first operate under idealized laboratory conditions, which are then scaled up until they can operate under uncontrolled field conditions. This is the characteristic strategy for the engineering sciences. In case-based strategies, we generalize about components of real-world cases, that are supposed to exhibit less variation than the cases as a whole. In sample-based strategies, we generalize about the aggregate behavior of samples of cases, which can exhibit patterns not visible at the case level. We discuss three examples of sample-based strategies. Throughout the paper, we use examples of theories and generalization strategies from software engineering to illustrate our analysis. The paper concludes with a discussion of related work and implications for empirical software engineering research.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

This paper aims to show two things: First, we aim to show that it is not worthwhile to develop general theory of software engineering, but that it is very useful to develop incompletely specified, partial theories that can be applied to practice. Second, we identify four classes of strategies to build theories, namely lab-to-lab and lab-to-field strategies, each of which can be about individual cases or about samples of cases. Each of these four combinations has a different way of dealing with the variability of the real world. We also give examples of such theories and generalization strategies from the field of software engineering (SE).

These two aims are an operationalization, for software engineering, of a view about scientific theories that has been expressed well by the philosopher of science Nancy Cartwright:

The laws that describe this world are a patchwork, not a pyramid. They do not take after the simple, elegant and abstract structure of a system of axioms and theorems. ...The dappled world is what, for the most part, comes naturally; regimented behaviour results from good engineering. [1, p. 1]

* Corresponding author.
E-mail addresses: r.j.wieringa@utwente.nl (R. Wieringa), m.daneva@utwente.nl (M. Daneva).

http://dx.doi.org/10.1016/j.scico.2014.11.013

0167-6423/© 2014 Elsevier B.V. All rights reserved.
The basic tension to be resolved by all scientists, including scientists in software engineering, is the tension between the idealization required to build neat, general theories, and the variability of the real world that allows only partial, incomplete generalizations. In basic science, the first horn of this dilemma is chosen. In engineering science, the second horn must be chosen.

To achieve our first aim, in Section 2 we discuss the structure and function of scientific theories, and identify the special features of design theories used in the engineering sciences. We argue that design theories necessarily deal with the variability of the real world, and that this implies that we will never have general design theories. Our discussion is applicable to all engineering sciences, and to show that it is applicable to software engineering as well, we give examples of theories from software engineering that illustrate our points.

To achieve our second aim, in Section 3 we discuss four strategies to develop the generalizations that are needed for theories. Each strategy has to deal with the variability of the real world, but approaches this in a different way.

- In lab-to-lab generalization, we require the target of generalization to be controlled so that the generalization applies to it. This strategy achieves generality at the price of idealization.
- In lab-to-field generalization, we refine the generalization by dropping idealizing assumptions. This achieves realistic generalizations, at the price of a limited, less-than-universal scope.

Each of these strategies can be performed in two ways:

- In case-based generalization, we study individual cases, and generalize about components and mechanisms found in a case, by similarity. The assumption is that components are less varied than the cases they occur in.
- In sample-based generalization, we study samples of cases, and generalize about statistical properties of these samples.

The assumption is that individual variety cancels out in sample statistics.

In Section 4 we will summarize our main contributions, discuss related work and draw some further implications for software engineering research.

2. Theories

There is no agreement among philosophers about what a theory is. One of the briefest definitions is that a theory is a belief that there is a pattern in phenomena [2, p. 55]. This includes all kinds of theories, including conspiracy theories about the causes of the credit crisis, economic theories about the causes of the same crisis, astrology, the theory of classical mechanics, and string theory. What makes a theory scientific?

This question has been analyzed by philosophers in various ways, of which the only conclusion seems to be, again, that there is no criterion agreed on by all philosophers that settles the matter [3]. Here we will be pragmatic and consider a theory as scientific if it has been submitted to, and survived, two kinds of tests [4,5]:

- **Empirical tests.** The theory has been submitted to, and survived, tests against experience. A theory can be tested against experience in observational research or in experimental research.
- **Justification to a critical peer group.** The theory has been submitted to, and survived, criticism by competent and critical peers. Part of the justification to critical peers is that empirical tests do not depend on the person of the researcher, and hence are repeatable: Critical peers must be able to repeat the empirical tests.

Surviving criticism and empirical testing is never final. Even for a theory that survived testing and criticism for a long time, it is always possible that someone will find a flaw in the argument or that a test will falsify part of the theory. Scientific theories are fallible. We should always consider them to be improvable.

The absence of absolute certainty about a theory does not imply that we should give up searching for theories, let alone abolish the search for theories. To quote Gordon [6, p. 76]:

That these ideals cannot be attained is not a reason for disregarding them. Perfect cleanliness is also impossible, but it does not serve as a warrant for not washing, much less for rolling in a manure pile.

2.1. Structure of theories

There has been an evolution in views on the structure of theories from the classical one that scientific theories are sets of propositions with a deductive inference system [7,8], to current views that scientific theories are abstract models of phenomena [1,2,9–11]. Our purpose here is not to summarize these views nor to defend a viewpoint on what we think the definite structure of scientific theories is, but to point out three elements that are present in most scientific theories, according to many philosophers (Fig. 1).
Conceptual framework The one element that is part of a scientific theory according to all views is a conceptual framework by which to describe phenomena. A conceptual framework is a set of definitions of concepts, used, for example, to ask research questions, describe and analyze phenomena, state generalizations about phenomena, specify models of mechanisms, etc. For example, a theory of effort estimation may contain a definition of the concepts of effort and size, and a theory of program comprehension may contain concepts like chunking, short-term memory, and long-term memory. These concepts are used, among others, to describe and analyze phenomena.

Some theories only consist of a conceptual framework. For example, Li et al. [12] defined a conceptual model of multiple-component defects, containing definitions of concepts such as defect, multiple-component defect, architectural hotspot, and repair dependency. Using this framework, phenomena could be described that exhibit regularities. For example, in the investigated case a relation was observed between cost of maintenance, number of multiple-component defects, and persistence of defects.

Note that there is a difference between the generality of a conceptual framework and the generality of descriptions made using the conceptual framework. Since a conceptual framework is a set of definitions, it cannot be true or false, but it can be applicable or not. A conceptual framework is general if it can be applied to many phenomena. The more general a framework, the larger the set of phenomena to which it is applicable. The definitions in the framework of Li et al. are generally applicable to large software systems.

By contrast, descriptions can be true or false. A description is general if it is true of many phenomena. In the case studied by Li et al., 20% of the components contained 80% of the multiple-component defects. This is a true description of their case. It may well be false in other cases.

Generalizations The second element usually found in theories is a collection of generalizations about phenomena. Generalizations may be formal or informal, expressed in words or diagrams, may be known to be true often but false sometimes, and may not all be connected deductively.

For example, a theory that could be proposed based on the research by Li et al. [12] mentioned above, is that in most large software systems, about 20% of the components contain 80% of the multiple-component defects. This theory is probably false in some cases, but it is possible that it is true in many cases. It would acquire support if in a series of case studies of large software systems performed independently, each time about 20% of the components turn out to contain about 80% of the multiple-component defects. It would acquire even stronger support if in a random sample of 30 or more large software systems, in each system the set of components that jointly contain 80% of the multiple-component defects, contains on the average about 20% of the components of the system.

Models The more recent notions of a scientific theory agree that many theories provide abstract models of phenomena. Models may be defined in text or diagrams, and the definitions may be formal or informal and are often incomplete. Models represent a phenomenon as a system of interacting components. Craver [2], Machamer et al. [13] and Bechtel and Abrahamsen [14] give examples of model-based theories from biology, specified by means of diagrams that show interacting components, such as the heart, lungs, and tissue that explain some of the phenomena of blood flow and respiration, or the biochemical substances and their interactions that explain part of the metabolic process. Glennan [15] gives some examples of theories about physical technical structures, such as the transistor and resistors that make up a voltage switch.

The interactions among components that produce interesting system-level phenomena are called mechanisms, and often models are primarily described by their mechanisms. An example of a neuropsychological mechanism given by Bunge [16] is the extinction of aversive memories by the action of cannabinoids on neuronal processes in the amygdala, and an example of an economic mechanism is the use of a stabilization fund by a central bank to stabilize government revenue in the face of major commodity price fluctuations. Hedström and Swedberg [17] list a number of social mechanisms, such as the reference group mechanism identified by Merton and Kitt [18], Thagard [19] gives a range of examples across the basic and applied sciences.

These examples illustrate that the components studied by researchers can be physical, biological, psychological, or social. If we extend our view to software engineering, then we encounter software components, hardware components, components of the cognitive processes of program comprehension, components of software engineering projects, etc. This corresponds well with the diversity of examples of software engineering theories found by Hannay et al. [20]. Concrete examples of models used in software engineering theories will be given later.
2.2. The scope of scientific theories

The scope of a theory is the set of phenomena to which it is applicable. The scope of a generalization is the set of phenomena for which it is true, and the scope of a model is the set of phenomena to which it can be applied. For example, we may propose a 20/80 theory based on the case study of Li et al. discussed above, where we vaguely characterize the scope to be all complex, large-scale, commercial software systems developed over a period of at least a dozen years [12, p. 676]. As another example, Mayrhofer et al. [21] propose a model of the cognitive processes by which programmers understand code. The scope is claimed to be, precisely, all programmers.

We should reiterate here that theories are fallible, and therefore claims about the scope of a theory are fallible as well. Our research should be aimed at improving the accuracy of our scope claims. At any point in time, a scope claim is our best bet, given the arguments and evidence so far.

Idealization and generality Scope claims in basic science are the opposite of scope claims in engineering science. Basic scientific research makes idealizing assumptions that are known to false in practice but that make patterns of behavior visible [22,23]. Concepts like that of point mass, frictionless surface, perfectly elastic body, absolute vacuum, rational actor and Turing machine do not exist in the real world, but allow researchers to analyze patterns in phenomena conceptually or even mathematically. This approach to knowledge has been called Galilean idealization [24,25]. As a consequence of Galilean idealization, models proposed in basic research are idealized structures that abstract and simplify structures found in the real world, in the interest of conceptual and computational tractability. Cartwright [1] calls them nomological machines. Basic laws of nature are true for nomological machines but false for the uncontrolled real world [26,9]. In other words, the laws of basic science cannot be applied in practice [27,28,9]. The universality of basic laws of nature is obtained at the price of idealization.

In many sciences, middle-range theories are more useful than universal theories. A theory is middle-range if its generalizations do not have universal scope. The concept of middle-range theory was developed for the social sciences [29], but is also applicable to special sciences such as geology, meteorology and political science, which all have to deal with a variety of uncontrolled conditions of practice (Fig. 2). Engineering sciences produce middle-range theories as well [23]. We can view theories like the COCOMO model, some of the software engineering principles of Davis [30] and some of the theories listed by Endres and Rombach [31] as middle-range theories.

Sciences of the middle-range, which are sciences that produce middle-range theories, are shown in the middle of Fig. 2. Researchers in these sciences try to avoid unrealistic assumptions and aim for generalizations that have less-than-universal scope. As a consequence, practitioners who want to apply these middle-range theories to their particular case should assess whether the middle-range theory is true for their case, or perhaps needs to be adapted. In a way, a practitioner should build a theory of his or her particular case, based on more general, middle-range knowledge produced by researchers [32].

2.3. Functions of theories

Scientific theories can be used to explore, frame, describe, analyze, explain, predict, specify, design, control, and organize phenomena [2]. Here we describe only a few of these functions, namely description, analysis, explanation, prediction and design.

Description and analysis The conceptual framework of a theory can be used to describe and analyze phenomena. For example, the conceptual framework of the 20/80 multiple-component defect theory discussed earlier [12], defines concepts like defect, multiple-component defect, and architectural hotspot. These concepts can be used to analyze data about a product development project, to describe some of the phenomena in this project, and to analyze the relations between these phenomena.
We need not restrict our descriptions to observed phenomena. We can also generalize descriptively beyond observed phenomena. For example, Huynh and Miller found in a sample of web application that roughly 70% of the vulnerabilities were due to insecure coding practice, which they call implementation vulnerabilities \[33, p. 565\]. In their conclusion they generalize this finding by saying that the majority of vulnerabilities in web applications is implementation vulnerabilities \[33, p. 574\]. This is a descriptive generalization.

**Explanation** The generalizations or models of a theory can sometimes be used to explain phenomena. We will distinguish two kinds of explanations \[34,23\]. There are other kinds of explanations in the philosophy of science, but for the purpose of this paper we will focus on these two.

- A theory explains a phenomenon causally if it has identified an earlier phenomenon that caused it. We call this a causal explanation. For example, we may explain a project failure by requirements creep. Causation is a complex concept that defies non-circular definition yet is central in scientific explanation \[35–38\]. At least we can say that causal explanations refer to variables, and to causal relationships between variables. Here we follow Woodward \[39\] in saying that \(X\) influenced variable \(Y\) causally, if \(Y\) changed because earlier, \(X\) changed in a particular way.

- A theory explains a phenomenon architecturally if it identifies components of a system that by their interaction produced the phenomenon. We call this an architectural explanation. For example, Mayrhoaser et al. \[21\] propose components of the cognitive process of program comprehension, such as short-term and long-term memory, and explain how interactions among these components produce program comprehension phenomena.

Architectural explanations refer to components of systems, and interactions among these components that produce system-level phenomena. The systems can be physical, social, psychological, digital, etc. Components are characterized by their capability to interact with their environment in certain ways. As indicated earlier, interactions by which components of a system produce system-level phenomena are called mechanisms \[15,40,13,41,16,42\].

We can illustrate these two definitions with a metaphorical story: Pushing a button on a coffee machine causes the machine to dispense a cup of coffee. After learning about this cause-effect relation, you will explain the phenomenon that a machine has dispensed coffee with the explanation that someone pushed a button. This is your causal theory of coffee-dispensing phenomena by coffee machines.

In contrast, an architectural theory would explain the coffee-dispensing phenomenon by means of the coffee machine components. The machine contains a coffee reservoir and a water supply, connected by mechanisms that ensure that if you push a button, it dispenses coffee. This is an architectural explanation, which is generalizable to other coffee machines with a similar architecture. Coffee machine engineers use this theory. Different machines that satisfy the same causal theory (coffee-dispensing is caused by pushing a button) may satisfy different architectural theories (they have different architectures that realize this cause-effect relation).

Your coffee machine theory would consist of the three elements listed earlier (Fig. 1). It would have a conceptual framework in which concepts like button, dispenser and coffee reservoir are defined, contain generalizations about the effect of events, and contain a model of the architecture of a typical coffee machine. The theory would be nondeterministic, meaning that there some cases where it is false — e.g. when a physical machine behaves erratically and spontaneously dispenses coffee.

An example of the use of both kinds of explanations in one case is given by the case study of Damian and Chisian \[43\]. They studied a software development organization in which requirements engineering was introduced. After introduction of requirements engineering, there was less requirements creep. The causal explanation of Damian and Chisian is that this was caused partially by the introduction of project tracking and partially by the introduction of change management \[43, p. 436\]. Like all causal explanations, these explanations explain a change in some variable (requirements creep) by particular changes in other variables, that happened earlier. The explanations are nondeterministic, meaning that the earlier changes usually contribute to, but do not deterministically determine, the later changes.

The paper provides information that we can use to provide an additional, architectural explanation. Part of the introduction of requirements engineering was the creation of a change management board through which all customer change requests had to pass. The previous practice that customers could call developers directly, an almost sure mechanism for requirements creep, was replaced by the mechanism involving the change control board. Another part of the introduction of requirements engineering was the extension of the project management function with project tracking \[43, p. 450\]. These architectural explanations refer to components (change control board, project manager) with capabilities to interact with other components. Some components (change control board) have been added to the situation, and other components (project managers) changed their capability, as the result of which some mechanisms disappeared and new mechanisms were created, producing different phenomena than before.

Causal explanations require a conceptual framework that defines the relevant variables. Variables are the machine language of science, but the real world contains a lot more structure than just variables and relationships. Architectural explanations assume a richer structure of the world, consisting of systems, components, capabilities and mechanisms. Many philosophers who take a model-based view of theories also allow architectural explanations \[2,13,14,44,15\]. The model is then a nomological machine that shows how phenomena are produced \[1\].
Prediction A major function of theories in the engineering sciences is prediction. For example, we can use the results of Huynh and Miller mentioned earlier [33] to predict that in other web applications too, the majority of vulnerabilities will be implementation vulnerabilities. This example also illustrates that we do not need to be able to explain a phenomenon in order to predict it. Conversely, we may be able to explain a phenomenon but not to predict it. For example, we may observe that all obvious causes of failure are absent in a project, but still not be able to predict reliably whether the project will succeed. Our knowledge of project failure and success may be too incomplete for that. Knowledge of social phenomena is often too incomplete to allow deterministic prediction [45, p. 348].

Design Engineers are interested in the interactions between an artifact and its context. Artifacts in software engineering are algorithms, notations, techniques, methods, etc. The context in which they are used is a software engineering project, project personnel, customers, software, hardware, organizations, etc. A typical architecture for software engineering projects is that an actor applies technologies to perform activities on a software system [46]. Software engineering researchers are interested in the interactions among these elements.

Theories produced by software engineering researchers may help practicing software engineers in the design and improvement of artifacts. For example, if the majority of vulnerabilities in web applications is implementation vulnerabilities, a company that develops web applications may decide to improve the competence of its programmers to avoid insecure coding practice.

If a theory provides explanations about why the interaction between an artifact and its context produces certain phenomena, then we may be able to use it to improve the artifact or to choose the best context for it. A classical example is the steam machine, which had been operational for over 100 years before Sadit Carnot explained how it worked [47]. This in turn provided knowledge that could be used to make the design of steam machines more efficient.

If a theory allows the prediction of phenomena, even without explanation, then it can still be used to choose a design. For example, a software engineer may do performance measurements of the execution time of some algorithm in different contexts. If the performance measurements have been shown to be repeatable, then this can be used to predict what the performance of new implementations of this algorithm will be, even if the exact performance numbers cannot be explained to the last digit. Practical engineering contains many of these empirically developed practically usable predictions [48].

2.4. Summary

Scientific theories consist of a conceptual framework, and usually contain generalizations and/or models of phenomena. They can be used, among others, to describe, analyze, explain, and predict phenomena. Explanations may be causal or architectural. A causal explanation explains a change in a variable by an earlier change in another variable. An architectural explanation explains a phenomenon in terms of the interactions among components that produced it.

Theories are useful for engineers because they can allow them to describe, analyze, explain and predict the behavior of artifacts in a context. Not every theory may be usable in all of these ways at the same time.

Software engineering researchers must balance between the extremes of idealization and practice. By making too many idealizations their theories would lose relevance for practice; by including too many conditions of practice they would lose the ability to generalize [49]. In the next section we look at different strategies to choose a balance between the two extremes.

3. Strategies for generalization

To discuss the strategies for generalization, we need to distinguish objects of study, samples, and populations, as indicated in Fig. 3. The object of study is the object from which measurements are taken, such as for example a software engineering project, a software engineer, a software program, etc. We will distinguish case-based research, in which single objects of study are investigated, from sample-based research, in which samples of objects of study are investigated. For example, we may investigate a single software engineering project in-depth, or we may survey a sample of projects statistically.

In sample-based research, sample data are used to generalize statistically to a well-defined population, called the study population. The sample is a subset of the study population. We may extend the generalization further from the study
population to a so-called theoretical population, of which the study population is a subset. The theoretical population may be less well-defined than the study population.

For example, we may survey a sample of projects selected from a list of projects in a large company. The projects on this list form the study population. After generalizing statistically from the surveyed sample to the study population, we may generalize to the larger set of all projects in the company, which is ill-defined because we do not have a list of them. We may even generalize to the set of all software engineering projects in all similar companies, which is even more ill-defined if the required similarity has not been specified completely.

In case-based research, we may attempt generalization from the object of study to the theoretical population immediately. For example, from the investigation of a single project, we may tentatively hypothesize a generalization about all similar software engineering projects in similar companies.

We will see that all generalizations to a theoretical populations are based on similarity. An important research question is then what kind of similarity is sufficient to warrant generalization to a theoretical population. For different generalizations, we may need different concepts of similarity.

The distinction between sample-based and case-based research is idealized, because in practice there are mixed forms of research, in which for example we investigate a sample of projects one by one, in a case-based way, or in which we investigate a sample first statistically, and follow this up with a case study of one of them. The theoretical population may not be a superset of the study population, but a population similar to it in some respects, etc. But the picture suffices as a guide for the discussion of generalization strategies.

Validity The inference steps by which we produce explanations and generalizations are fallible, which means that they can lead to incorrect conclusions from correct premises. The degree of support for a conclusion of a fallible inference is called its validity [50, p. 513]. The validity of a statistical inference from a sample to a study population is called its conclusion validity.

Internal validity is defined by Shadish et al. as the degree of support for the claim that a relation between two variables is causal [50, pp. 53, 508]. Because we recognize causal as well as architectural explanations, we generalize the definition by Shadish et al. and define internal validity here as the degree of support for a causal or architectural explanation of a phenomenon.

External validity is defined by Shadish et al. [50, pp. 83, 507] as the extent to which a causal relationship also holds over variations in Units, Treatments, Outcomes and Settings. In our more general interpretation of internal validity, and in terms of Fig. 3, we here define external validity as the degree of support for the generalization of a causal or architectural explanation to a theoretical population. The source of this generalization can be an explanation of phenomena in a single object of study or in a study population. The target is always a theoretical population. The problem of the external validity is the problem of the variability of the real world. Each generalization strategies has a way to deal with this problem without solving it completely.

3.1. Lab-to-lab generalization

The first strategy to generalize to a theoretical population is case-based, and deals with the problem of external validity by requiring uniformity in the theoretical population. The research goal is to achieve theoretical understanding of a phenomenon, and the strategy is to achieve this by doing laboratory experiments under controlled conditions (Fig. 4).

This is the classical approach in basic science, where the major challenge is to create the conditions in the laboratory that are ideal enough for the phenomenon to be produced [51, p. 92]. For example, in 1820 Oersted showed by experiment that an electric current deflected a nearby magnetic needle. In terms of Fig. 4, step one was to observe the deflection of the needle in the experimental setup. Step two was the identification of the presence of the electric current as the cause. Step three was to confirm that this was repeatable by Oersted and by all other researchers who tried replication.

Interesting in this example is that the experimental manipulation showed that there was a repeatable causal relation by which the electric current influenced the needle, but that there was yet no mathematical description or architectural explanation of it. So it was just a phenomenon that experimenters knew how to produce. A few years later the French mathematician Ampère built a mathematical theory of this phenomenon that described the phenomenon exactly. Architectural understanding was achieved much later, when Maxwell developed his field theory relating magnetism and electricity. Generalization, causal explanation, description, and architectural explanation are not always found in a logical order.

Characteristic of laboratory research is that external validity is claimed only for the theoretical population of laboratory experiments. Experimental researchers in basic science spend their research budget on creating the ideal conditions required for the generalization to hold. This seems irrelevant for engineering researchers, who want to create effects in the real world.
In the real world, effects created in the laboratory may be swamped by a multitude of uncontrolled causes and be interfered with by other mechanisms.

However, this does not make laboratory research irrelevant for engineering science. First, creating theoretical understanding is useful in software engineering research as well as in any other science [52].

Second, laboratory theories can be used to predict and explain other laboratory phenomena beyond those they were originally developed for. For example, the theory of cognitive processing developed from laboratory experiments by Gellenbeck and Cook [53] in 1991 was used 10 years later by Prechelt et al. [54] to justify the design of a laboratory experiment with pattern comment lines (PCLs). A PCL is a comment line that describes the use of software patterns where applicable. Prechelt et al. wanted to test the effect of PCLs on maintainability of programs, and used the beacon theory of program comprehension proposed by Gellenbeck and Cook to predict and explain what can be observed in the lab. This is an example of lab-to-lab generalization.

Third, ongoing testing of a theory developed for lab phenomena may very well show that the theoretical prediction and explanation still hold in the field, even though this is not a goal of lab-to-lab generalization. And there may be reasons to expect this in advance. For example, Prechelt et al. use extreme case reasoning to speculate that if PCLs are already effective for the relatively small and well-commented programs in the laboratory, they may be effective in an environment of large ill-documented programs [54, p. 604]. This provides a reason for further testing this prediction in the field.

Fourth, in the absence of further investigations in the field, and assuming that the laboratory theory is internally valid, a laboratory theory can be used to suggest what regimentation we should impose on the context, or what "protective covering" we should put around the artifact, if we want to reproduce the phenomena in the real world as they have been produced in the lab [1, p. 86]. For example, the theory of Prechelt et al. may be generalizable to real-world projects in which programs are kept small and well-documented.

Lab-to-lab generalization is one of the core generalization strategies in basic science, where the goal is to understand and create phenomena in isolation. In engineering science we find another core strategy, that we call lab-to-field generalization.

3.2. Lab-to-field generalization

The object of study in engineering sciences is an artifact in a context of use [34,23]. Engineering researchers iterate between (re)designing artifacts for use in a class of contexts, and investigating artifacts that interact with contexts of this class. For example, we may design a new software engineering notation for use in a particular kind of software engineering project, investigate its properties in the classroom or in the field, redesign it, investigate some more, etc. In this strategy, researchers start their investigations under ideal conditions in the lab and finish them under realistic conditions in the field. At the start of this process, when only lab research is done, the goal is to support lab-to-lab generalizations. At the end of this process, when field research has been done, the goal is to support field-to-field generalizations. During the process, artifacts are scaled up to practice and generalizations are increasingly targeted at field conditions. We will abbreviate this process as "lab-to-field generalization". It is a characteristic generalization strategy in the engineering sciences [55,48,49].

In this process, artifacts are scaled up to practice [56]. For example, the turbojet was designed and built first as a prototype in the lab, and investigated under ideal circumstances that did not yet resemble practical conditions of a flying aircraft [55]. During its development, it was scaled up by making it more robust to conditions of practice. At each step, an increasingly robust prototype was investigated under more realistic conditions of practice, until a realistic prototype was investigated when it was used to propel an aircraft.

The knowledge that is built up during this process has a continuously changing subject: a relatively simple prototype operating under ideal conditions at first, and a sophisticated prototype operating under realistic conditions of practice at last. At each stage, the technology is investigated, until satisfactory support for a generalization about the artifact in similar conditions can be given. When sufficient support for such a generalization has been acquired, the next step in scaling up takes place. Ideally at each stage, the behavior of the artifact prototype can be explained architecturally, i.e. in terms of components of which the interactions produce the behavior of the prototype artifact in context (Fig. 5).

Lab-to-field generalization is a form of technology validation [57,58]. Elsewhere we discuss research methods that can be used in technology validity, such as simulation, technical action research, and statistical difference-making experiments in the lab or in the field [56].

In software engineering research, we can see quite a few examples in the past twenty-five years where software engineering researchers have scaled up technology from laboratory to practice, or generalized their knowledge of new technology. For example, in the 1980s the CleanRoom methodology was tested in a series of experiments that scaled up from the lab-
ory (using students as subjects) to small real-world projects to large real-world projects, very similar to the way new drugs are tested [59,60]. These tests started out as simulations in the laboratory and ended up as pilot projects in practice.

In the early 1990s Lubars et al. [61,62] and Potts [63] investigated object-oriented analysis methods in three experiments that started from simulation, in which the authors built an object-oriented requirements model of a relatively simple ATM in the lab, and ended with an action research project in which they developed a specification of a cellular telephone protocol in practice. The methods investigated were already on the market, but generalizable knowledge about them was still lacking.

Scaling up technology from the laboratory to the field is common in industrial research, while developing new technology for a market. However, scaling up must be distinguished from technology transfer. Transferring new technology to the field can happen without any systematic process of scaling up and before scientific investigation of its performance in practice has been done — the UML is a case in point. Conversely, scaling up technology to field conditions can happen, without it ever being actually transferred to practical use in the market — 4K resolution screens could be an example. Technology transfer includes activities like mass production, marketing and distribution, which are absent from the activity of scaling up technology.

Lab-to-field generalization is a process in which lab-to-lab generalization evolves into field-to-field generalization. We may also distinguish field-to-lab generalization, in which a phenomenon found in the field is reproduced and investigated in the lab. For example, the dependence of productivity of pair programmers on personality characteristics of the programmers, found in the field, may be reproduced and investigated in the lab [64,65]. We do not discuss these strategies further and now turn to another classification of generalization strategies, based not on the source and target of generalization, but on the object that is generalized about: A case or a sample. In the following two sections we discuss sample-based and case-based strategies. In the discussion and in the examples it will become clear that these strategies can be combined with any of the strategies discussed so far.

3.3. Sample-based generalization strategies

Sample-based generalizations reduce the variability of the real world by aggregating individual phenomena over samples, so that some individual variation cancels out. This idea arose in the early 19th century in connection with the governance of states [66–68]. Mathematicians and politicians discovered that even where at the individual level, no repeatable pattern could be discerned, at the population level, there were nearly stable patterns. The number of births in a city like Paris was nearly the same every year, as were the number of marriages, the number of deaths, and even the number of letters that got lost in the Parisian mail system every year. In today’s language we would say that the variance of these sample statistics was surprisingly small from year to year, and we understand it as a consequence of the law of large numbers.

The early development of statistics as a discipline was closely related to the development of sociology as a science, with its own domain of study: mass phenomena in populations. Later in the 19th century, statistics as a discipline was applied to other topics, such as mass phenomena in gasses [66].

We will discuss three strategies for sample-based generalization:

- Randomized difference-making experiments: Generalize a causal theory about statistical population phenomena.
- Quasi-experiments: Generalize a causal theory about sample phenomena.
- Statistical learning: Generalize a statistical description of sample phenomena.

Randomized difference-making experiments The first kind of statistical inference from samples to populations was developed early in the 20th century by Gosset and Fisher [69,70]. The goal is to infer a property of the probability distribution of one or more variables over a study population, from observations of a random sample selected from the population. There are two forms of statistical inference. In statistical hypothesis testing, sample observations are compared to a hypothesis about the population distribution of the variable(s) of interest. The result of the comparison is then interpreted as support for or against the hypothesis, or as inconclusive. Which of these options is available depends on the kind of inference the researcher wants to use, the one proposed by Fisher or the one proposed by Neyman and Pearson, or a mix of the two. In confidence interval estimation, sample data are used to estimate an interval in which a distribution parameter can be confidently assumed to fall, that is with a relatively low rate of making the wrong estimation. Both classes of techniques use the Central-Limit Theorem, which assumes random sampling. They are discussed in any textbook on statistical inference [71, 72]. In terms of Fig. 3, statistical inference takes one from sample observations to properties of distributions of the study population. Details about the methodological differences between them are given by Hacking [73] and Wieringa [23].

Statistical inference is used in for example randomized controlled experiments to compare the effect of a randomly allocated treatment with non-treatment, or treatment by a placebo. Here we discuss the logic of statistical and causal inference for a slightly generalized kind of experiment, that we will call a randomized difference-making experiment (Fig. 6), in which we compare arbitrary treatments. The reason for this name is that in this kind of experiment we take a difference-making view of causality, as we also do in this paper. Rephrasing our earlier definition of causality, we can say that if a difference in X makes a difference to Y, then X causally influences Y. If taking an aspirin makes a difference to my headache, then taking an aspirin causally influences my headache. This justifies the structure of difference-making experiments, where we compare the difference in outcome variables of two treatment groups [74].
The idea is to provide evidence for the claim that two treatments, \( A \) and \( B \), have a different effect in a population. We define a dummy variable \( X \) with two levels \( A \) and \( B \), select a random sample from the study population, and randomly allocate \( A \) or \( B \) to the sample elements. Suppose for simplicity that we do this so that we get two subsamples of equal size, one treated by \( A \) and one treated by \( B \). These samples can be viewed as being selected randomly from two virtual populations, the population treated by \( A \) and the population treated by \( B \). We are interested in providing evidence that there is a statistically discernable difference between the averages of a variable \( Y \) in the population treated by \( A \) and in the population treated by \( B \). In addition, we want to provide evidence that this difference is caused by the difference between \( A \) and \( B \).

In a randomized difference-making experiment, this evidence is provided as follows (Fig. 6). Step one is to apply to the treatments, and to compute the averages \( \mu_A \) and \( \mu_B \) of the values of \( Y \) on the two samples. Based on this we use a statistical inference technique to infer whether there is a statistically discernable non-zero difference between the averages of \( Y \) in the two populations, i.e. whether \( \mu_Y,A \) and \( \mu_Y,B \) are different. The data can provide strong, weak, or inconclusive evidence for a non-zero difference. If we conclude that there is a difference, then the degree of support provided by the evidence for this conclusion is the statistical conclusion validity of the inference. Random sampling and allocation plays a crucial role in providing this support [71,75].

If there is satisfactory evidence of a non-zero difference, then step two is to conclude that because sampling and allocation have been random, in the long run, the only possible causal explanation of the difference between \( \mu_Y,A \) and \( \mu_Y,B \) is the difference between \( A \) and \( B \). This is a long-run explanation: we assume that the statistical difference will occur again in most replications, and therefore is a stable phenomenon. In the long run, this can only be caused by the difference between \( A \) and \( B \). The degree of support provided for this causal inference is the internal validity of the inference.

Step three is to generalize this to similar populations, e.g. the theoretical population. What is a “similar” population? This depends on the mechanism that produces the causal influence. A theoretical population of which the elements all contain the mechanism that is responsible for the effect of \( X \) on \( Y \), is likely to exhibit the same causal relationship. In pharmacology this is called a mechanism of action. For example, caffeine has several mechanisms of action, two of which are that it antagonizes a biochemical compound (adenosine) that inhibits neurotransmitters, and that it increases the activity of neurotransmitters such as dopamine [76]. These mechanisms create a causal influence of, for example, drinking coffee on staying awake. We can generalize this explanation to all organisms in which caffeine can trigger this mechanism of action.

In engineering the mechanism of action is called a principle of operation [48]. For example, the principle of operation of an airplane is that by the shape of its wings, air above the wing flows faster relative to the wing than air below it, which according to Bernoulli’s principle produces upward lift. This is the mechanism by which forward speed causes upward lift. All bodies with a similar shape will experience this. The theory “forward speed causes upward lift” is externally valid in all bodies with a shape that justifies the application of Bernoulli’s principle.

What the relevant similarity in step three is, depends on the architectural explanation of the causal relationship postulated in step two. If no architectural explanation has been found, then we should be cautious and generalize to objects of study on which the variables used in the causal explanation have “similar” values. For example, we may tentatively generalize a causal relation between variables to projects of a similar size, cost and lead time. Without architectural understanding of the mechanism that produces this relation, this generalization has weaker support than with such an understanding.

The above strategy, which is strategy 3, contains generalization in two steps. In step one, there is a statistical generalization from a sample to the study population. In step three, there is generalization by similarity between the theoretical population and the study population. Our sketch of this strategy is idealized. Regarding step one, when sampling a study population, this is usually done without replacement, so that after each selection of a population element, the remaining population to select from is smaller and the probability of population elements to be selected is not equal. If the population is large compared to the sample, the difference with sampling with replacement is not noticeable, but if the population is relatively small, this requires a correction factor to be applied to sample-based inferences [71]. In addition, sampling is often not random but is based on self-selection, which may create an unknown systematic bias by which it is not known from which population you are sampling randomly. According to Freedman [71], the few domains where statistics has a satisfactory applications are measurement (e.g. precision measurements in astronomy) and genetics.

Random allocation can often be achieved, but because subjects know which task they have been asked to perform, in step two this knowledge must be included in the factors that may have influenced the outcome [77].

Regarding step three, the specification of relevant similarity between populations is currently mainly based on similarity in the values of variables and less on similarity of architectures or repeatability of mechanisms [78]. As pointed out above, architectural similarity gives a stronger basis for generalization by similarity.
To give an example of strategy 3 in software engineering, we use the study by Hannay et al. [65]. They tested a number of hypotheses about the effect of personality, expertise, task complexity and country of residence on pair programming in a large-scale field experiment in which subjects were professionals and settings were natural, but treatments were artificial. Step one showed statistically discernible differences in performance among samples of pairs with different expertise, that are unlikely to be accidental. Step two identified some independent variables as plausible causes for these differences. Expertise, extraversion, and task complexity were among the variables that could cause the observed differences. Step three considered the similarity and dissimilarity between the studied samples and real-world programming pairs. The use of professional software engineers is the major similarity between the experimental setting and real-world settings in the field. However, low task complexity and controlled group dynamics reduce the generalizability of this field experiment to real-world pair programming [65, p. 75].

**Quasi-experiments** Random sampling and allocation is often not achievable, and is rarely done in experimental software engineering [79]. Since the early 1960s, Campbell and colleagues developed quasi-experiments, that do not assume random sampling or allocation and by which causal inferences could be supported [80]. Generalization to the theoretical population is still by analogy. The strategy is now to collect sufficient data in step one to provide sufficient evidence for a causal theory in step two (Fig. 7). This is harder than in the randomized approach of the previous strategy, where randomization allowed us to have a simple theory of the experiment: in the long run, all causes other than the difference in treatments are excluded. Importantly, in randomized experiments we do not have to know what these causes are. In quasi-experiments, developing a causal explanation of sample phenomena is considerably harder, because we must know identify all relevant causes [81,82,50]. In step three, we generalize the theory of the sample to the theoretical population by similarity as we did before.

It is hard to find examples of strategy 4 in software engineering quasi-experiments, because many publications discuss mostly step one, showing a difference, and spend little or no attention to steps two and three, explanation and generalization [83]. To illustrate strategy 4, we use the article by Laitenberger et al. [84] on three replicated quasi-experiments to compare perspective-based reading with checklist-based reading. The three experiments happened in three editions of a course for professional software engineers, so this was a set of three quasi-experiments. The authors use descriptive and inferential statistics to argue that checklist-based reading is more costly and less effective than perspective-based reading. This is a description of a statistical phenomenon, that corresponds to step one of strategy 4 (Fig. 7).

Some explanations for these results are provided [84, p. 388]: Checklists are often based on past information, contain too many questions, do not require inspectors to document their analysis, and require inspectors to check all information in the document for possible defects. These factors could reduce the effectiveness of checklist-based inspections compared to other inspection techniques. On the other hand, part of the effectiveness of an inspection may not be attributable to a reading technique at all, but to the competence of the inspector. By reading the specification and code, the inspector may find defects regardless the reading techniques used [84, p. 408]. All of this contributes to a theory of the experiment, that explains the observations. This corresponds to step two of strategy 4.

As an illustration of step three, we point out that the theory of the experiment only refers to those elements of an inspection that are present in other checklist-based inspections too, both in the classroom as well as in the field. This similarity supports generalization to other classrooms and to the field. However, in real-world inspections, other mechanisms may work against this generalization: Inspector competence makes inspections more effective and less costly, even checklist-based inspections. Furthermore, the experimental setting used individual inspections, whereas in industrial practice, team inspections are common [84, p. 408]. There are thus some similarity-based arguments in favor of generalizability, and other dissimilarity-based arguments that limit generalizability.

**Statistical learning** In the past decades, computer-intensive methods have been developed in machine learning, pattern recognition, data mining and process mining by which statistical patterns in large samples can be discovered [85,86]. Descriptions of these statistical sample phenomena can be used to predict similar phenomena in new samples, without necessarily being able to explain why these mass phenomena occur, and without necessarily being able, to state in advance for which class of phenomena these regularities occur. The goal is not to generalize to a population, but to generalize to the next few cases.

For example, researchers may develop an effort estimation formula that describes the relation between effort and complexity that has been observed in a large sample of past projects. A practitioner may choose to use this formula if she assesses her own case similar to the cases that the researcher sampled from. See Fig. 8 for the steps in this strategy.
We note that similarity between the training sample, in which the phenomenon has been identified, and the application sample, where the description is to be applied, is judged by the user of the description, and does not have to be specified by the producer of the description. The producer of the description is a researcher, and the user may be another researcher or a practitioner. Of course, it helps if the producer gives a clear description of features of the training sample. Over time, the application samples may become less similar to the training sample. To remain usable, the description should then be recalibrated on a new training sample.

An example from software engineering is the COCOMO effort estimation model. Boehm et al. [87] suggest that an estimating specialist takes historical data of at least ten projects in an organization in order to analyze organization-specific patterns and use this information to calibrate COCOMO equations in an organization-specific way. This model is claimed to hold in the organization as long as it does not change its practices drastically. As the specialist keeps collecting new information from new projects in the organization, the accuracy of the model can be usually improved over time, again assuming there is no drastic change in company practices.

Cost estimation researchers also use cross-company data made available by public databases (e.g. through the PROMISE conference [88]) or proprietary databases (such as that of the International Software Benchmarking Standards Group) as data sets to analyze regularities and propose improvements of cost estimation models. Statistical regularities have been also used to compare the performance of models developed by using cross-company data and within-company data [89,90].

### 3.4. Case-based generalization

In sample-based studies, the variability of the real world is reduced by taking sample statistics as the object of study, such as the sum, mean, or standard deviation of a variable in a sample. This may reveal large-scale stable patterns of behavior. In case-based research, variability is reduced by decomposing a single case into components with interactions, such as for example people and roles in a project. These components and mechanisms may be recurrent across a large set of different cases, and are hence interesting subjects of generalization. Fig. 9 shows the steps in this strategy.

A case study cannot give support for causal explanations, because evidence for causality requires observations of differences [74]. To show that difference in $X$ makes a difference to $Y$, we should be able to observe two values of $X$. But a single case study will show us only one value of $X$. This is the reason we have two samples in a difference-making experiment.

However, it is possible to take a causal theory from another source and use it to explain what we observed in a case. For example, Sahberwal [91] uses agency theory to explain coordination in outsourcing as the effect of opportunistic behavior by the vendor. He in addition explains why which mechanisms the client in an outsourcing contract is more vulnerable to opportunistic behavior of the vendor than the other way around. This illustrates that case phenomena can be explained causally and architecturally.

In step three of strategy 6, we generalize to other cases architecturally. As stated before, similarity of the values of variables is a weak basis for generalization by analogy. For example, in the case studied by Damian and Chisan [43], we may argue that similar improvements will be observed in other cases where similar components (change control board and a cross-functional team) with similar capabilities are introduced.

As in strategy 5, users of this generalization must assess if their own case is relevantly similar to the case studied by Damian and Chisan to apply this generalization to their own case. Other cases may contain additional mechanisms, such as organizational dynamics caused by budget cuts, organizational mergers, political tensions, etc. that may prevent the effects of the introduction of requirements engineering to occur.

Architectural generalizations can be made more robust against real-world variation and interference by a process of analytical induction. This was introduced as a case-based generalization strategy in sociology in the 1930s by the sociologist Znaniecki [92]. In the social sciences, analytical induction consists of a series of case studies, where all cases have a similar architecture, but also differ from each other [93–95]. In each case study, architectural explanations are sought that explain phenomena in all cases studied so far. So, for a researcher to confirm that an explanation constructed for one case is valid

---

1. Collect sample data.
2. Describe statistical sample phenomenon.
3. Generalize the description to similar cases.

**Fig. 8.** Steps in strategy 5: statistical learning.

1. Observe case phenomena.
2. Explain the phenomena architecturally.
3. Generalize the theory to architecturally similar cases.

**Fig. 9.** Steps in strategy 6: building architectural theory of case phenomena in the field.
in another case too, he or she may choose the next case to be as similar as possible. On the other hand, in order to test the robustness of the explanation under different circumstances, a researcher may choose the next case similar enough to make the occurrence of the phenomenon possible, but dissimilar enough to make it plausible that it may occur by different mechanisms, or not at all.

The case studies by Mockus et al. [96] are an example of analytical induction across two cases. They analyzed development and maintenance of the Apache and Mozilla open source projects. In the Apache case, they observed that the project has a core of about 10–15 developers who controlled the code base and created approximately 80% or more of new functionality. This is step one of strategy 6. They explained this architecturally by the following mechanism [96, p. 9]:

- (Apache mechanism): “The core developers must work closely together, each with fairly detailed knowledge of what other core members are doing. Without such knowledge they would frequently make incompatible changes to the code. Since they form essentially a single team, they can be overwhelmed by communication and coordination overhead issues that typically limit the size of effective teams to 10–15 people.”

The Mozilla project is architecturally similar, yet different. This project had a core of 22 to 36 developers who coordinated their work according to a concretely defined process and used a strict inspection policy, and who each had control of a module and created approximately 80% or more of new functionality. The authors therefore refined their explanation [97, p. 340]:

- (Apache and Mozilla mechanism): Open source developments have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal ad hoc means of coordinating their work, the group will be no larger than 10 to 15 people.

Note that this refined explanation addresses both cases.

3.5. Summary

To summarize, generalization strategies can be classified according to their source and target, and according to whether they are about sample phenomena or about case phenomena. Field-to-lab generalizations are possible too, but we have not discussed them. Basic science usually generalizes from lab to lab, requiring the researcher to eliminate in the laboratory much of the variability of the real world, to approximate the idealizations required by fundamental laws of nature. The engineering sciences usually generalize from lab to field. They sacrifice universality by incorporating conditions of practice in their generalizations.

In each of these strategies, we can study samples or cases. Sample-based research deals with the variability of the real world by studying aggregate phenomena of samples, in which individual random variation cancels out. There are various ways to do that, using randomization, quasi-experimentation, or computer-intensive methods. In case-based research, the variability of the real world is reduced by decomposing a case into components, that can produce case phenomena by their interactions. Generalization should be based on architectural similarity, and generalizations can be made more robust by analytical induction.

4. Conclusions, related work and implications

4.1. Conclusions

This paper has three contributions. Our first contribution is to emphasize the utility of middle-range theories that balance generality with practicality (Fig. 2).

Our second contribution is that we recognize models as theories. This opens the road to accepting architectural models as theories. Architectural models supplement causal explanations of phenomena by showing how a causal influence is produced by an underlying mechanism. Architectural models are also useful to define the relevant similarity relationship when generalizing from a case to a theoretical population, or from a study population to a theoretical population.

Third, we have identified six generalization strategies useful for theories of the middle-range. The characteristic strategy of basic science is lab-to-lab generalization, and the characteristic strategy for engineering sciences is lab-to-field generalization. Each of these strategies can be done in a case-based and a sample-based way. For case-based generalization we have argued that architectural models allow better support for judging similarity between cases than variable-based theories do. Finally, we have reviewed three sample-based strategies, namely randomized difference-making experiments, quasi-experiments, and statistical learning strategies.

4.2. Related work

Our view of theories is similar to that of Sjoberg et al. [46], who present a theory of UML-based development consisting of (1) a conceptual framework, (2) propositions, (3) explanations, and (4) an indication of scope. Their propositions and explanations seem to be similar to our causal and architectural explanations, respectively.
Comparison of our analysis with Gregor’s [98] analysis of theories in information systems research (Fig. 10) reveals that we agree on the importance of constructs, generalizations and scope. However, Gregor ignores the role of models in theories and does not recognize architectural explanations. We do not regard notations to be part of a theory, and we do not require a theory to give causal explanations. Testable propositions may follow from a theory in particular cases, but we do not regard them as part of a theory. And we do not think that prescriptive theories can exist. Theories can be used to justify design decisions, but do not prescribe them. We can use a scientific theory to predict the effect of a decision, but not to tell us what we must do.

Gregor and Jones [99] outline a structure for design theories, in which testable propositions seem to have taken the place of generalizations (Fig. 11). The elements in the lower part of the table are not required to be part of a theory in our approach. In the table they are briefly explained.

Seddon and Scheepers [100] provide a roadmap for generalizations in information systems research, that is relevant for software engineering research too. They identify analytical induction (called “analytical generalization” by them, just as Yin [95] does) as one of the important routes to theoretical generalization from samples or from cases, as we do too. They emphasize combination of research findings with prior knowledge, with which we agree.

Seddon and Scheepers provide a more elaborate analysis of different kinds of statistical generalization than we do. They emphasize that statistical inference based on the Central Limit Theorem, using e.g. p-values or confidence intervals, should satisfy the assumption of that theorem, namely random samples. They provide a useful analysis of strategies to follow some kind of statistical inference for non-random sampling, that complements ours. In particular they discuss the possibility to use Bayesian statistics, which we do not discuss. Seddon and Scheepers do not consider the possibility of architectural explanations and do not emphasize the need for middle-range theories as much as we do.

Lee and Baskerville [101] identify four kinds of generalization, which they call EE, TT, ET and TE. Their type EE is, in our terms, generalizing from case to case. Their type TT is, in our terms, the extension of a conceptual framework with a causal explanation. Their type ET is, in our terms, the causal explanation of a phenomenon, and their type TE is the application of a prior causal theory to a phenomenon.

4.3 Implications

The implications of this paper for empirical software engineering are simple: Since software engineering is an engineering science, all observations and conclusions of this paper apply to software engineering too. The examples that we have given of all the points made in this paper illustrate this.

The generalization strategies discussed in this paper are strategies to acquire generalizable knowledge about software engineering artifacts in context, where these artifacts may be novel or may have been used in practice for a long time. As we stated before, scaling up to practice is not a model for technology transfer but a way to establish knowledge in the lab and then generalize it to the field. The sample-based strategies allow researchers to establish knowledge about statistical regularities and, using causal reasoning, to establish causal explanations of these regularities. The case-based strategies allow researchers to establish architectural knowledge about the mechanisms that produce phenomena. In particular, they can establish knowledge about the mechanisms that can produce causal relationships in the lab or in the field.

To establish theoretical knowledge, we need to generalize to a theoretical population, and to do that we need adequate knowledge of the relevant architectural similarity relation that defines the theoretical population. Different generalizations will probably need different architectural similarities. Architectural similarity is a worthy research topic that can only be performed in cooperation with practitioners [102,103]. There are some published results about the (dis)similarity between
software engineering students and software engineering professionals [104–106], but more needs to be done. There are interesting new developments such as the International Workshop on Replication in Empirical Software Engineering Research (RESER) [107]. The implication of this paper for replication is that the definition of replication requires the definition of an architectural similarity relationship. The real subject of replications should therefore be the architectural explanations of phenomena. Replications should be theory-based.

The books by Davis [30] and Endres and Rombach [31] present a patchwork of principles, observations, laws and theories that all call for more empirical research to validate and elaborate them. Empirical research into these claims is more than replication of earlier results. It is the investigation of the mechanisms that could explain the effects described by the theories in this patchwork, and the investigation of the limits of the scope of these theories, using the generalization strategies discussed in this paper.

References


