Uncovering Theories in Software Engineering

Klaas-Jan Stol and Brian Fitzgerald
Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
klaas-jan.stol@lero.ie, bf@ul.ie

***Abstract*—There has been a growing interest in the role of theory within Software Engineering (SE) research. For several decades, researchers within the SE research community have argued that, to become a *real* engineering science, SE needs to develop stronger theoretical foundations. A few authors have proposed guidelines for constructing theories, building on insights from other disciplines. However, so far, much SE research is not guided by explicit theory, nor does it produce explicit theory. In this paper we argue that SE research does, in fact, show traces of theory, which we call theory *fragments*. We have adapted an analytical framework from the social sciences, named the Validity Network Schema (VNS), that we use to illustrate the role of theorizing in SE research. We illustrate the use of this framework by dissecting three well known research papers, each of which has had significant impact on their respective subdisciplines. We conclude this paper by outlining a number of implications for future SE research, and show how by increasing awareness and training, development of SE theories can be improved.**

***Index Terms*—Software engineering research, middle-range theory, theory fragment, theory building, empirical research**

I. Introduction

Where is the theory for software engineering? This is a question that Johnson et al. recently posed [1]. When discussing the topic of theory in Software Engineering (SE) research, many authors point to other, more mature and established disciplines, such as physics and the social sciences, and argue that SE research, too, needs to develop theories [1][2][3][4][5][6]. A related question that arises in this discussion is whether or not SE is a science; some argue that SE is a branch of *engineering* (and not a *science* [7][8]). This somewhat disregards the fact that the phrase "Software Engineering" was coined in a provocative way [9]. While SE as a practice is seeing its successes, the so-called "Software Crisis" is still better characterized, in the words of Parnas, as a chronic problem [10, p. 29].

One important development in SE research has been the rise of Evidence-Based Software Engineering (EBSE) that started to gain significant traction in the SE research community in the early 2000s [11]. This strong focus on empirical research is reflected by two dedicated conferences (ESEM, EASE) and a specialized journal (Empirical Software Engineering). In order to organize, aggregate and synthesize empirical evidence, Kitchenham et al. [11] proposed the Systematic Literature Review (SLR) in SE research, borrowing from the medical research domain where Evidence-Based Medicine has seen great success. In SE research, too, SLR is seeing widespread adoption; one of the SE field's prominent journals (Information and Software Technology (IST)) explicitly solicits SLR submissions. Of the 25 most-cited papers in IST per January 2013, more than half (13) were SLRs or mapping studies.

However, with this strong focus on empiricism, we argue that we cannot see the theoretical trees through the woods of evidence. The usefulness of theories is widely recognized by other disciplines; after all, "*nothing is so practical as a good theory*" [12]. Theories provide a vocabulary for different researchers, which helps to put research studies in context and converge towards more focused topics of research. An important function of theory is to make explanations and understandings of how the world works explicit [13]. This makes knowledge transferable. As Gregor pointed out, "*theories are practical because they allow knowledge to be accumulated in a systematic manner and this accumulated knowledge enlightens professional practice*" [14, p. 613].

In answering their question *Where is the theory in SE research?*, Johnson et al. presented three possible explanations. Firstly, that software engineering *doesn't need* theory; secondly, that software engineering *already has* its theory, and thirdly, that software engineering *can't have* a theory. Johnson et al. argued that none of these three arguments hold. It should be noted that Johnson et al. (see also [5]) are looking for a *General Theory for Software Engineering*. While this quest is an admirable one, in this paper we limit ourselves to so-called "middle-range" theories. Merton [15, p. 38] referred to these as those theories lying "*between the minor but necessary working hypotheses that evolve [...] in [...] day-to-day research and the all-inclusive systematic efforts to develop a unified theory*." As Bourgeois [16] wrote about behavioral theory, we believe that SE research is too immature for an all-inclusive unifying general theory. Development of middle-range theories is, however, an important step towards maturity of the SE discipline.

We agree with Johnson et al. that none of these three arguments hold. However, we would like to add a few observations.

Firstly, there is no common agreement on what theories look like in SE, and as a result, theories may be difficult to recognize. It is difficult to study something if you cannot recognize it. Researchers may not be familiar with theorizing, perhaps due to the fact that it was not a part of their research training. As a result, researchers may not have a good understanding of what constitutes theory, its role in research studies, and how to recognize it.

Secondly, researchers may not see the need for theorizing, and consider it a task for "philosophers."

Thirdly, we argue that, while there is a lack of explicit theory within SE research, many papers do contain a form of

conceptualization, or what Weick would call *theorizing* [17]. We adopt the term "theory fragment," [2] to refer to a partial theory that has not been completely developed yet. However, such partial theories may not always be clearly presented. Also, these theory fragments are rarely fully developed to mature theories.

The purpose of this paper is twofold. Firstly, we outline the different roles that theory can play in SE research. In order to increase awareness within the SE research community of the level of conceptualization and theorizing that is currently happening, we adapted an analytical framework originally proposed by Brinberg and McGrath [18], named the Validity Network Schema (VNS). The original purpose of the VNS is to illustrate the different meanings of *validity*, depending on what *research path* is chosen. We adapted the VNS to better fit the context of SE research, the result of which we named the Research Path Schema (RPS) so as to distinguish the resulting schema from the original VNS.

Secondly, we argue that *explicit* theory plays only a small role in SE research, but that there are many *theory fragments*, products of what Weick called *theorizing* [17]. We illustrate the use of the RPS with three well known papers.

The remainder of this paper proceeds as follows. Since the term theory can mean different things to different researchers [19][17], we discuss the nature, origins, and purpose of theory in Section II. Section III presents the Research Path Schema. Section IV presents the three case studies of the RPS, followed by a discussion of the implications for the practice of future research in Section V.

II. Background and Related Work

A. The Importance of Conceptualization

Software Engineering is a multi-disciplinary field, and as such, research studies are much more varied and heterogeneous than in, say, the natural sciences such as physics. Much of the research in physics is of a quantitative nature, with "standard" approaches to present research results. In SE, in contrast, research studies are much more heterogeneous, with a wide variety of research approaches, methods and techniques, with both quantitative and qualitative approaches. Various research methods, approaches and techniques have been imported from other disciplines, in particular the social sciences; some are more common (survey [20], case study [21], grounded theory [22]) than others (ethnography [23], repertory grid technique [24]).

As a result of this heterogeneity, assessing an SE paper's scientific contribution can be challenging. A paper may present interesting findings, but if these are not further interpreted or conceptualized, the scientific contribution may be insufficient. Dubin [25, p.16] expressed this sentiment as follows: "*The distinction [between reporting and 'doing science'] lies in whether the information is gathered for its own sake, or whether it is used to measure the values associated with 'things' [...], the relationships among two or more of which is the focus of attention. The first procedure we call description; the second we call research.*" In the same vein, Suddaby [26] (p. 636) observed that a common problem with Grounded Theory (a research method originating from the social sciences which sees increasing uptake in SE research) is a "*a failure to 'lift' data to a conceptual level.*" Nisbett [27, p. 4] described how the early Mesopotamian and Egyptian civilizations made systematic observations, but that only the Greeks made significant progress by explaining their observations in terms of the principles underpinning them. Hall et al. [28], citing Robson [29] wrote: "*without theory the research may be easier and quicker, but the outcome will often be of little value.*" A lack of conceptualization may also apply to *secondary* studies, such as Systematic Literature Reviews (SLR). In this context, conceptualization can be done through synthesis of findings of a set of so-called *primary* studies. Cruzes and Dybå [30] observed that synthesis of findings in SLRs is often poorly performed.

Figure 1 presents a continuum of conceptualization. The dotted line distinguishes "description" from "research"; merely reporting empirical data without conceptualization is "description" (using Dubin's terminology). Reports that present an empirical study with conceptualization, or a conceptual paper that presents concepts or theory only are called "research." There is, of course, no clear and hard boundary between the two, which is expressed by the dotted style of the line.

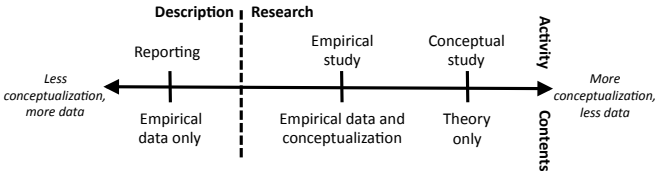

Fig. 1. Continuum of conceptualization.

Reynolds [31, p. 43] argued that, "*unless the 'conceptualization' is explicitly described other scientists cannot understand it and probably will not adopt it.*" Thus, we argue that conceptualization is an important aspect of good research studies. Conceptualization is closely related to *theorizing*, or the process of building theories [17]. We have observed an increasing level of attention for the role of theory within software engineering research [1][2][4]. However, conceptualization and theory building has not been recognized to be as important as empirical research within the SE research community. This lack of attention for theory building is somewhat surprising, given the various calls over the years. For instance, Basili and Zelkowitz [32] wrote: "*any future advances in the computing sciences require that the empiricism takes its place* ***alongside theory formation*** *and tool development*" (our emphasis). Broy argued that, "*engineering disciplines must be based on scientific practices and theory to justify their approaches and to give scientific evidence for why and where their methods work properly*" [33, p.19].

Figure 2 illustrates the role of theory in relation to empirical research, and as such it represents "the flip side" of the empirical "coin." (Similar diagrams representing the research process are offered by Bunge [35, p. 9], Shaw [36], Carroll and

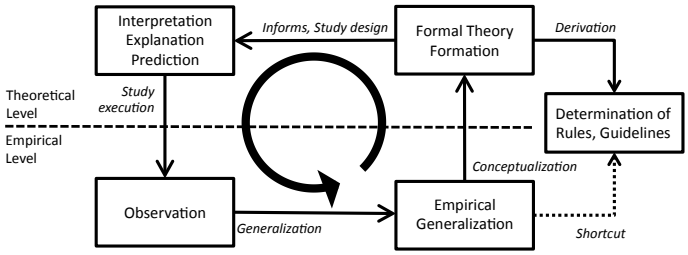

Fig. 2. Research cycle (adapted from Lehman and Ramil [34])

Swatman [37], Lynham [13], and Endres and Rombach [38]. Wieringa et al. [39] presented an *engineering cycle* for the requirements engineering subdiscipline.) Sutton and Staw cite Kaplan [40] in asserting that "*data describe* ***which*** *empirical patterns were observed, and theory explains* ***why*** *empirical patterns were observed or are expected to be observed*" [19, p. 374]. Theory not only *informs* the design and motivation of new research studies, but also forms the basis of "rules" and guidelines for practitioners. The formation of formal theory may be skipped, which is indicated by the "shortcut" in the figure. However, such guidelines do not provide the justifications that underpin them. Chalofsky expressed this well as: "*Professionalization comes from theory and research: the 'why' instead of the 'how to'*" [41]. To illustrate this, we use nutrition as an example. We now know that eating white bread or white pasta (as opposed to the wholewheat variants) can be a factor in obesity. Thus, one guideline that dietary consultants can provide to clients is to eat wholewheat variants only. However, such a guideline does not provide any insight as to *why* this is so. To explain this, knowledge of how carbohydrates affect a person's blood glucose levels is useful. The Glycemic Index (GI) is a measure to indicate this; the lower, the better. Digesting food with a high GI means that carbohydrates are converted into energy quickly–resulting in spikes in the glucose levels. The body may not need that much energy, and may store the superfluous energy as fat [42]. (We acknowledge that our brief discussion is a great simplification; however, it does support our point that an understanding of the theory that explains the mechanism is very valuable.) With this knowledge, clients may be better able to assess which foods will support weight loss—rather than merely following a guideline that states that "*eating white bread is unhealthy.*" Knowledge of the GI and how to measure it allows clients to assess any food. Furthermore, knowledge of glucose production and use allows clients to *contextualize* advice: foods with a high GI are, in fact, useful for marathon runners, and weight-lifters directly after their work-out.

One question that may arise is where to start in this cycle of theorizing and empirical investigation. Reynolds [31] describes two approaches: "research-then-theory" and "theory-then-research." The former refers to conducting empirical studies, based on which one develops a theory (also known as the *Baconian* strategy [31, Ch.7]); the latter starts with a theory that informs the design of a study that can subsequently be executed. Which approach to take depends, of course, on how much theory is available on a particular topic. Nascent research areas would typically take the research-then-theory approach, whereas more mature areas could rely on (and refine) existing theories to further advance the field.

B. What Theory Is and Is Not

The question *What is Theory?* has been topic of much discussion in other disciplines (see e.g., [43]). There is a wide variety of methods and approaches proposed for constructing theories [44][31][25][16][13][45][4]. Weick observed that many descriptions of theory building wrongfully suggest that it is a mechanistic process, "*with little appreciation of the often intuitive, blind, wasteful, serendipitous, creative quality of the process*" [46, p. 519]. Bourgeois argued that such steps are not discrete processes, but that the presentation of a theory may suggest a sequential ordering of thought [16] (see also [47]). We agree that constructing theories is not a linear, sequential process consisting of a number of steps, but that the various activities may occur in parallel. The research cycle depicted in Fig. 2 presents a graphical representation of this sentiment, where the activities of theorizing and gathering empirical evidence is an alternating process.

The aim of this section is not to present a final answer to the question *how* theory is constructed, but rather to present a brief overview of the key components that are widely accepted in other disciplines to be a part of theories. In what follows we briefly summarize the key components of a theory.

The main elements of a theory are its **constructs**. In SE research, two example constructs are *program size* and *software quality*. In order to *measure* the size of a software program, one needs to *operationalize* that construct, using a **measure**. This can be done through a variety of empirical indicators: lines of code (LOC), memory footprint (during runtime), number of classes (in object-oriented languages), and size of the compiled object code. To operationalize "software quality," one could choose to count the number of known defects, or select a quality attribute (e.g., performance) and operationalize that with performance metrics such as start-up and response time. Not all constructs are directly measurable; these *hypothetical constructs* [31] may still be useful to build a theory. Wieringa et al. [39] cited "gravity" and "organizational culture" as examples of hypothetical constructs.

A theory also defines the **relations** among constructs and how they interact with one another. These relations may be of different forms, of which causality is perhaps best known and arguably the most interesting. In a SE context, one relation that a researcher could suggest is between program size and software quality, such as, the bigger a software program is, the lower the quality. Theories typically have a limited scope, indicated by their **boundaries**. That is, theories are likely to be only valid under certain conditions. This is directly related to the concept of *generalizability*, or *external validity*.

A theory may have different **states**. Each state may have a different set of laws of interaction that apply only to that state. For instance, certain software development practices

(e.g., peer-review) may result in high-quality code in Open Source projects, but only in *popular* projects (in other words, less popular Open Source projects may not achieve the same level of code quality). A theory can transition from one state to another. Some state transitions may be invalid. Weber [45] illustrated this with an example of a theory about human life, which has two states: alive and dead. Whereas the transition from alive to dead is "lawful," the reverse transition is generally "unlawful."

Constructs, relations, boundaries and states are all elements of a theory that must be considered in the activity of building a theory. Once constructed, a theory is put to use. Reichenbach refers to these contexts as *discovery* and *justification*, respectively [48][49]. To that end, one would define a set of **propositions**; these are concerned with making predictions about a theory's constructs. While related to relations, propositions are logically derivable from a relation, whereas for the reverse one needs to make an "*inductive leap*" [25, p.170]. One example of a "theorizing" study that presented a number of propositions is reported in [50]. **Hypotheses** are to propositions what measures are to constructs. That is, hypotheses (empirical level) are instantiated propositions (theoretical/conceptual level), through the replacement of constructs within these propositions by measures. For instance, to further develop the example given above, a researcher could hypothesize that, as a software program grows in terms of lines of code (*size* construct), the number of defects (*software quality* construct) will increase in a linear fashion.

There is a fine line between what is a theory, and what is not. In particular, Sutton and Staw outline a number of elements that they, by themselves, do *not* consider to be a theory [19]:

- **References**; Sutton and Staw argue that references (to prior literature on a topic) are sometimes used as "*a smoke screen to hide the absence of theory.*"
- **Data**; while descriptions can be a source to build theories from, they do not constitute a theory [51]; this corresponds to Dubin's distinction between "description" and "research" discussed above, and the "continuum of conceptualization" in Figure 1.
- **Lists of variables or constructs**; a mere list of concepts and their definitions are what Homans [52, p. 957] described as "*a dictionary of a language that possesses no sentences.*"
- **Diagrams**; often consisting of "boxes and arrows" [43], they can be helpful in providing structure, but "*Some verbal explication is almost always necessary*" [19].
- **Hypotheses**; a mere set of hypotheses without further justification or clarification does not constitute a theory.

While these elements by themselves are not theories, they can be *parts* of a theory. As Weick, arguing that the focus should be on the *process* (of theorizing) rather than the *product*, wrote: "*What Theory is Not, Theorizing Is*" [17]. In this context, we use the term *theory fragment*, to refer to something that can develop into a theory. We argue that, while fully developed theories in SE research may be rare, the field has a lot of theory fragments. One of this paper's goals is to show how these can be found.

C. The Purpose of Science and Goals of Theory

Reynolds [31] discussed five purposes that science should serve: (i) to provide a method to *organize and categorize* things (typology); (ii) to *predict* future events; (iii) to *explain* past events; (iv) to provide an *understanding* of events, and (v) to potentially *control* events. Reynolds argued that predicting future events (iii) and explaining past events (iv) differ only in a temporal perspective (that is, past v. future) but are similar otherwise.

Once constructed, theories may have different goals, independent of the degree to which a theory has been validated. Gregor [14] presented a taxonomy of theory types that they observed in Information Systems (IS) research. Previously, this taxonomy was used in an analysis of theory use in SE research [2]. We discuss each type briefly below.

Analysis; says *what is*; provides a description, but no explanation of causality. There is generally disagreement over whether a typology can be labeled as "theory" [19]. Some would disqualify this as theory, arguing that the primary goal of theory is to answer *how*, *when*, and *why* questions, rather than *what* questions [51]. However, on the other hand, when using the term "theory" more freely, typologies are useful in communication and education. For instance, SE students could study the "theory" of software evolution, and learn the different types of maintenance activities as identified by Swanson [53].[1] This would qualify as providing a typology, which is one of the purposes of science [31]. Note that a typology differs from a "list of variables or constructs" in that the former links the various viable "values" (e.g, adaptive v. corrective maintenance), whereas the latter may consists contain an *unrelated* constructs.

Explanation; says *what is, how, why, when, where*. This type of theory provides explanation (insight) but has no predictive power. Though Reynolds used the word "explanation" in the context of explaining past events, what Gregor [14] means here is what Reynolds referred to as providing an understanding [31].

Prediction; says *what is, and what will be*. This type of theory provides predictions and testable propositions, but no explanatory power.

Explanation & Prediction; combination of (2) and (3).

Design and action; says *how to do something*. This type of theory provides prescriptions for constructing artifacts (such as methods and techniques).

D. Theory in Software Engineering

There is increasing agreement that Software Engineering is not merely a branch of Computer Science [8]. In fact, Offutt [7] wrote that "*Software Engineering is Engineering, Not Science*." However, we agree with Broy that: "*An engineering discipline without a theory cannot work*" [33]. As pointed

[1] Adaptive, corrective or perfective maintenance.

out by Offutt, mechanical engineering relies on physics (a traditional field with well-developed theories; a "normal" science as Kuhn would argue [54]). However, there does not seem to be a "primary" or fundamental research field with well-developed theories on which SE can depend. For instance, while mechanical engineers in designing and building structures such as buildings and bridges, can depend on well-defined theories and laws from physics, software engineers do not seem to be able to rely on such theories in SE. Interestingly, there is an increasing attention to social and human aspects in SE, so one potential fundamental field can be the social sciences that have studied team performance, for instance. However, clear laws, rules and theories about how to build reliable, resilient and high-performance software systems are not generally defined nor taught.

There have been a few reports of the use of theory in SE research; we summarize these next.

Hannay et al. [2] conducted an SLR on the *use* of theory in software engineering experiments. They found that of the 113 published experiments, only 24 studies used a total of 40 theories. A similar study to the one by Hannay et al. was conducted by Hall et al. [28], who investigated the use of theories in studies of software engineers' motivation. One of their findings was that many of the 92 studies they analyzed were not underpinned by the "classic" theories of motivation that originated in the social sciences.

Endres and Rombach [38] composed an extensive collection of empirical observations, laws and theories. For instance, one law is: "*Well-structured programs have fewer errors and are easier to maintain*" [38, p. 74]. While this law may have some predictive power, there is no justification or explanatory power, and as such practitioners may feel such statements are unsatisfactory.

Sjøberg et al. [4] presented a set of steps to construct theories for the domain of software engineering. In addition, they propose a template with four archetype concepts: (i) actor, (ii) technology, (iii) activity, and (iv) software system. Furthermore, they proposed a UML-based diagrammatic notation. Shull and Feldmann [55] discussed the construction of theories from multiple and different sources of evidence. This is particularly relevant to SE given the aforementioned heterogeneous nature of research in this field.

Both Sjøberg et al. [4] and Runeson [56] argued that theories must be relevant to practitioners. We tend to disagree with this position by adding that we believe that theory plays an important role in software engineering research, and as such, one purpose of theory is to guide and support further research. So, instead of *practical utility*, a theory may also have *scientific utility* [57]. The researcher's "tools" need not be directly relevant to practitioners. Even theoretical, or "conceptual" research may, in the long run, be useful and relevant to practitioners. It is of course difficult, if not impossible, to suggest when and how this will happen.

III. The Research Path Schema

This section presents the Research Path Schema (RPS), which is the result of our adaptation of the Validity Network Schema (VNS) proposed by Brinberg and McGrath [18]. The VNS, as the name suggests, was originally proposed to explain how the term "validity" has different meanings depending on the *research stage* and the *type* of research study. This term has been a topic of much discussion in the field of consumer research [58], where epistemological considerations have received much more attention than in SE research. The remainder of this section presents the RPS in more detail.

A. Domains of Research Elements

Brinberg and McGrath argued that "*research involves (a) some content that is of interest, (b) some ideas that give meaning to that content, and (c) some techniques or procedures by means of which those ideas and content can be studied*" [18, p.14]. These three aspects are then referred to as the *substantive*, *conceptual*, and *methodological* domains, respectively. We discuss them next.

Substantive domain. The substantive domain is the domain of phenomena and real-world systems that can be a topic of study. In more abstract terms, this is the content that a researcher may be interested in. In SE research, elements of the substantive domain are, for instance, open source software [59], developer motivation [60], and software architecture [61]. Each of these topics are phenomena as found in the real world, and are considered by researchers to be worthy of study.

Conceptual domain. Whereas the substantive domain deals with "subject matter," ("substance"), the conceptual domain deals with concepts, models and theories. Within this domain falls also any conceptual paradigm, which may underpin the conducted research. A conceptual paradigm is a set of paradigmatic assumptions and has an important impact on what a researcher may or may not discover. For instance, Pfleeger [62] pointed out that the model used by nineteenth-century physics was faulty; scientists in that time never considered light to be an electromagnetic wave, and as a result, they never observed light *particles*. In other words, following Kuhn [54], the conceptual paradigm defines what research problems are considered important to be studied, as well as any expectations with respect to the answer.

Methodological domain. The methodological domain of research refers to the methods and techniques to gather data about study topics (substantive domain) or theories (conceptual domain). Such methods may be "modes of treatment," comparison techniques, or other research methods well known in SE research, such as case studies, surveys, and controlled experiments. Also included in this domain are any research techniques or approaches that a researcher may be interested in, for instance to evaluate its use in a certain setting. For example, Edwards et al. [24] discussed how the Repertory Grid Technique can be used in software engineering research.

Brinberg and McGrath argued that, "*The research process is the identification, selection, combination, and use of elements*

and relations from the conceptual, methodological, and substantive domains" [18, p.16]. Thus, a research study consists of some phenomenon or topic of study, a research method or technique, and a set of concepts or theory. These three elements can be combined in different ways, as we outline next.

B. Research Paths

Scientific research studies may have different goals; some studies attempt to generate new theory (e.g., Grounded Theory), whereas others attempt to evaluate a set of hypotheses based on an existing theory (e.g., using a controlled experiment). Others still seek to demonstrate the value of a certain method or technique within a certain domain. As a result, a researcher designing a study will typically have a particular and primary interest in one of the three domains discussed above. The domain of the researcher's primary interest defines the type of research.

In other words, the order in which the elements are chosen defines which *research path* a researcher takes. Brinberg and McGrath identified three main *research paths*, which they labeled the *experimental*, the *theoretical*, and the *empirical* path, which reflect different goals of a study. We decided to rename these three paths as *study design path*, *hypothetical path* and the *observational path*, respectively, to better fit the more heterogeneous nature of SE research. We discuss these three research paths below.

Study Design path. The goal of the study design path is to build a study design, and use it on an element of the substantive domain. The study design is comprised of a set of concepts or a theory on the one hand, and a research method or technique developed by the researcher on the other hand. The last element to add to complete the study is a real-world system or phenomenon (i.e., the substantive domain).

Following this route, a researcher's primary interest may be the conceptual domain (concept-driven research), for instance, a conceptual framework or theory. Alternatively, the primary interest may be the methodological domain (method-driven research). A common scenario in SE research is that a researcher has a conceptual model or framework, and develops a technique (or tool) to implement or support this. In this scenario, the substantive domain has lowest "importance"; the implementation is the result, which serves as a validation of the researcher's proposed idea.

Observational path. The goal of the observational path is to collect a set of observations, and to explain them in terms of a set of meaningful concepts. In other words, a researcher starts with a topic of interest (substantive domain) and a research method (or technique). The result will be a *set of observations*. The next step is to *interpret* these observations. One goal may be to generate a set of concepts (or theory), using for instance a Grounded Theory approach [63]. Alternatively, the set of observations (resulting from a case study, for instance) may be interpreted using an existing set of concepts or theory that was developed prior to the study. However, the researcher's primary interest is in either the substantial domain (a phenomenon such as open source software) or a method or technique.

Brinberg and McGrath referred to this path as the *empirical* path, but this term is ambiguous, since controlled experiment (for instance) is also an empirical method, which could follow the hypothetical path that we discuss next.

Hypothetical path. The hypothetical path refers to research that seeks to *test* theory rather than build it. In particular, the RPS defines two sources of potential hypotheses; they can either originate from the substantive domain (substance-driven research) or from a theory (concept-driven research). In the case of evaluating a set of hypotheses to test a certain theory, a researcher's primary interest is, of course, the theory being tested (the conceptual domain). The researcher will then select an appropriate real-world situation, phenomenon, or system (from the substantive domain) to gather data. Finally, an appropriate research method or technique is selected; while the choice of a suitable method (methodological domain) is important to achieve valid results, the choice of method will be guided (restricted) by the research situation, and is therefore of "least" interest. In other words, a researcher typically will not start with a research method in mind – say, controlled experiment – when evaluating a set of hypotheses. Rather, the main interest is in investigating the hypotheses.

IV. Applying the Research Path Schema

In order to demonstrate how the RPS is useful to uncover theory and theory fragments, this section presents three case studies, of well known and highly cited papers, one for each of the three research paths. (This paper itself can be viewed through the RPS lens: following the Study Design path, the RPS itself constitutes an element of the conceptual domain, as it presents a conceptualization of the research process; we use (multiple) "case study" as our method of studying these papers; and the three papers that we examine here are elements of the substantial domain, since they are elements from the real world.) We selected one paper that was published in ICSE and selected as *Most Influential Paper*; the second paper presents a follow-up of the first, which was published in ACM Transactions on Software Engineering and Methodology, a top tier journal in the SE field. The third paper was published in IEEE Transactions on Software Engineering, also a top tier journal in the SE field, and was identified as the most cited paper published in the year 2000. All three papers have been very influential; using the RPS, we illustrate how theorizing takes place and identify theory fragments.

A. Case I: Observational Path

Our first paper by Mockus et al. [64] presents a case study of Open Source Software (OSS) development in the Apache web server. This is a highly cited paper (447 citations as measured by Google Scholar, January 2013) which has had a high impact on the OSS research field.

1) Substantial Domain: Mockus et al. argued for the need to gain an understanding of OSS development (a phenomenon;

1. What was the process used to develop Apache?
2. How many people wrote code for new Apache functionality? How many people reported problems? How many people repaired defects?
3. Were these functions carried out by distinct groups of people, i.e., did people primarily assume a single role? Did large numbers of people participate somewhat equally in these activities, or did a small number of people do most of the work?
4. Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?
5. What is the defect density of Apache code?
6. How long did it take to resolve problems? Were higher priority problems resolved faster than low priority problems? Has resolution interval decreased over time?

Fig. 3. Research Questions in the Apache study.

an element of the substantive domain), and decided to investigate the Apache web server, which is a very successful and well-known OSS project. Thus, the *substance* of this study is OSS development, and in particular the Apache web server. The research questions (RQs) that guided this study are listed in Figure 3.

2) Methodological Domain: To study this OSS project, Mockus et al. selected case study methodology, which represents the *methodological* domain. Both qualitative and quantitative data were collected. One of the authors of the study who was involved in the core development team of Apache wrote a detailed account of the development process used, which was checked by other members of the core team. RQ 1 was addressed using qualitative data based on this process description. Quantitative data were collected from the developers mailing list, the source code repository, and the issue tracker used in the project. RQs 2 to 6 were addressed using quantitative analyses of these data.

3) Conceptual Domain: The study by Mockus et al. resulted in a set of uninterpreted empirical observations, from which they distilled a number of concepts through a process of *theorizing*. In particular, some of their identified concepts include different *roles*, such as *core developers*, *bug fixers*, and *bug reporters*. These were later recognized as "layers" of the OSS *onion* model [65].

Mockus et al. further posed and justified a *set of hypotheses*, writing that "*Case studies such as this provide excellent fodder for hypothesis development*" [64, p. 271]. These hypotheses are listed in Figure 4. In Section II we argued (following Sutton and Staw [19]) that a mere set of hypotheses does not constitute theory. However, Mockus et al. provided clear justifications for each hypothesis (and used approximately one and a half pages for doing so).

While constructs and relations were identified, no explicit propositions were put forward. However, as Whetten argued, propositions are not always required [43].

4) Research Path and Theory: The research path that Mockus et al. followed is what we named the *observational path*. It is clear that in this study, the researchers' primary interest was in OSS development, and the Apache web server in particular. The next step by Mockus et al. was to select an appropriate research method to study this phenomenon; the paper does not suggest that the researchers based their study

1. Open source developments will have a core of developers who control the code base. This core will be no larger than 10-15 people, and will create approximately 80% or more of the new functionality.
2. For projects that are so large that 10-15 developers cannot write 80% of the code in a reasonable time frame, a strict code ownership policy will have to be adopted to separate the work of additional groups, creating, in effect, several related OSS projects.
3. In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems.
4. Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects in the released code.
5. Defect density in open source releases will generally be lower than commercial code that has only been feature-tested, i.e., received a comparable level of testing.
6. In successful open source developments, the developers will also be users of the software.
7. OSS developments exhibit very rapid responses to customer problems.

Fig. 4. Hypotheses resulting from the Apache study.

on an existing theory (which would suggest the researchers had taken the *hypothetical* path).

The collected data that resulted from combining the elements from the substantial domain (i.e. Apache web server) and the methodological domain (i.e. case study and data collection methods described above) formed a set of (uninterpreted) observations. Mockus et al. proceeded to interpret these observations and to *theorize* about OSS development; this resulted in the set of hypotheses (Figure 4).

Mockus et al. conducted a follow-up study to test these hypotheses in a case study of the Mozilla web browser [66]. This is an excellent example of a study following the *hypothetical path*, which is why we selected this study to illustrate this path. This is discussed in the next subsection.

B. Case II: Hypothetical Path

Our second case study of the RPS is an example of the Hypothetical Path. We selected the follow-up study by Mockus et al. that extended their study of the Apache web server discussed in the previous subsection. The follow-up study is a case study of the Mozilla web browser, and is published in ACM Transactions on Software Engineering and Methodology [66], and is a highly cited paper (1,129 citations according to Google Scholar, January 2013). Note that while this paper presents *both* case studies (the Apache and Mozilla studies), we focus only on the case study of the Mozilla web browser (Section 4 of [66]).

1) Conceptual Domain: We start the analysis of the Mozilla study by presenting the conceptual domain, which is the same set of hypotheses identified by Mockus et al. in their study of Apache (see Fig. 4). This set of hypotheses (including their rationale as outlined in their paper) is a theory fragment; in order to further develop this into a more complete and mature theory, these hypotheses should be tested. Therefore, Mockus et al. were interested in establishing the extent to which these hypotheses would hold when analyzing a different

OSS project. To that end, they selected another element of the substantial domain, discussed next.

2) Substantial Domain: For this study, the substantial domain is represented by the Mozilla web browser; this is the real-world system that Mockus et al. were interested in studying, given the set of hypotheses (in Fig. 4). Whereas Apache is a web *server*, the choice for Mozilla is interesting in that it is a web *browser*, and as such represents the other side of the communication over HTTP.

3) Methodological Domain: As more elements of a study are put together, the design of the study becomes more restricted. For instance, when Mockus et al. decided to study the Apache web server, a natural choice was case study methodology, in particular since the OSS research area was in its nascent phase at the time of that study. It would have been more difficult—though not impossible—to use, say, a survey (with questionnaires as data collection method). At the very least, this would have resulted in a different type of study with different findings. For this follow-up study, Mockus et al. also selected case study methodology.

4) Research Path and Theory: The study of Mozilla by Mockus et al. is a clear example of the *hypothetical* path, whereby the researchers started with a set of hypotheses (a combination of elements from the conceptual domain and the substantial domain), which were subsequently tested using an appropriate methodology. The findings of this study resulted in additional insights into OSS development, and based on this Mockus et al. revisited two of the seven hypotheses; these are listed in Fig. 5. This is a form of establishing the initial theory's *boundaries* (see Section II-B). Potentially, this theory fragment could be further developed through more studies on the one hand, and adding more parts of the theory, such as defining theory states, on the other hand. For instance, both Apache and Mozilla are examples of extremely successful OSS projects, but most OSS projects found on SourceForge.net do not succeed in attracting large numbers of contributors [67]. Different relations between this theory's constructs should be defined for less successful projects; it is likely that there are a number of state transitions as well, since new projects can *become* successful.

1a.	Open source developments will 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.
2a.	If a project is so large that more than 10 to 15 people are required to complete 80% of the code in the desired time frame, then other mechanisms, rather than just informal ad hoc arrangements, will be required in order to coordinate the work. These mechanisms may include one or more of the following: explicit development processes, individual or group code ownership, and required inspections.

Fig. 5. Revised Hypotheses 1 and 2 based on the Mozilla study.

C. Case III: Study Design Path

Our third case study is an example of the *Study Design Path*. We selected a study by Medvidović and Taylor [68], entitled "A Classification and Comparison Framework for Software Architecture Description Languages." This paper was identified by Elsevier's IST journal as the most cited journal paper published in the year 2000 (more than 2,000 citations according to Google Scholar in January 2013).

The paper presents a framework to classify and compare software architectural description languages (ADL). An ADL is a language to describe a software system's architecture, its components, and the connectors that link those components.

1) Conceptual Domain: Medvidović and Taylor observed that a wide variety of ADLs were available to practitioners. In order to achieve a better understanding of what is available, the authors developed a classification and comparison framework. This framework constitutes an element of the Conceptual Domain, as it is an analytical lens, a *conceptualization*, containing a number of dimensions ("themes") that are relevant to ADLs. Medvidović and Taylor speak of a *taxonomy*, which could be considered a theory with an "analysis" purpose only (see Section II-B), or a *typology* [31]. The framework consists of a number of elements (e.g., interfaces, types, semantics), organized in a number of categories (e.g., components, connectors, configurations). Space limitations prevent us from including the full framework in this paper.

2) Methodological Domain: The methodological domain is represented by the analysis performed by Medvidović and Taylor. One could speak of a "literature survey" of ADLs; in this case, the analysis is descriptive, using various "example ADL specifications" [68, p.78].

3) Substantive Domain: The substantive domain, then, consists of the elements chosen from the real-world that are studied; that is, a set of ADLs. Medvidović and Taylor selected a set of ten ADLs (e.g., ACME, Aesop, C2). The comparison analysis using the framework resulted in extensive descriptions and a set of detailed tables that outline the differences between the various ADLs.

4) Research Path and Theory: The study by Medvidović and Taylor is a clear example of the Study Design Path. That is, a study design was constructed by selecting (in this case, developing) an element from the conceptual domain (the comparison framework), and an element from the methodological domain (a comparative analysis/survey). This combination could have been applied on *any* set of ADLs. To complete the research study, Medvidović and Taylor selected ten ADLs from the real world that are elements from the substantial domain. The focus of this study was—judging from the title of the paper that starts with "A classification and comparison framework"—on the framework. The classification framework provides a typology that we consider a theory fragment, which can be used to inform the design of other studies; given the high citation count (over 2,000) this has clearly happened.

V. Conclusion and Future Work

In this paper we have presented an extensive discussion of what theory is, what it looks like, and its purpose. The Research Path Schema, based on the Validity Network Schema by Brinberg and McGrath [18], is a useful analytical tool to view SE research and to "uncover" theory (fragments).

Based on our presented discussions and presentation and demonstration of the RPS, we suggest a number of potential implications for future SE research:

Stronger focus on theorizing. While we strongly believe in the importance of theory as both a driver for, and a result from empirical research, we also admit that not each and every study should present new theory. However, one of our arguments is that theory is not a luxury, to be left to "philosophers," but that it is an essential part of software engineering research. Theory *should* inform the design of new studies, which will help to converge the research literature on a particular topic (or research question). This in turn will make the literature less scattered, but more focused on essential questions that SE research purports to address, namely those relating to building affordable, timely and high-quality software systems. We argue that with an increased awareness of the role of theory and the theorizing process, researchers may be able to design better research studies that are grounded in theory (or fragments). This will contribute to one of the purported benefits of theory-focused research, namely that of knowledge transfer.

Theorizing and conceptualization comes in different forms. The RPS, being based on the VNS, presumes that a research study always consists of elements of three domains: the substantial, the methodological, and the conceptual domain. It is important to emphasize that research papers may make other contributions than empirical studies. In particular, conceptual papers are important to bring the field as a whole forward, as such papers may introduce new and important perspectives on topics [49]. Conceptual contributions in empirical papers may take on a variety of forms. Not unimportant are meta-level studies that provide guidelines to other researchers.

Toward theory-focused software engineering research. The SE research field has a strong emphasis on Evidence-Based Software Engineering (EBSE) research, as advocated by Kitchenham et al. [11]. While we wholeheartedly support this advocacy, we suggest that EBSE is combined with a *theory-focused* research approach, so as to complete the cycle shown in Fig. 2. This may either follow a *research-then-theory* or a *theory-then-research* approach [31] as described above. The studies by Mockus et al. are good examples of this; their first study of the Apache web server resulted in a number of observations, based on which they hypothesized (theorized) about OSS project governance and development. This theory fragment was then used to inform their second study (of Mozilla).

Include theorizing in SE research training. Since current research in SE pays little attention to theories and theory building, PhD students get little—if any—exposure to, or training in building their own theories, or in using existing theories to guide and conduct their research. As argued above, in order for the SE research community to adopt a theory-focused approach to conducting research, new researchers (i.e., PhD students) need to receive training. Researchers in other fields, in particular the social sciences, provided guidance in theory building, such as Kaplan [40], Reynolds [31] and Dubin [25]. While some guidance has been provided, such as by Sjøberg et al. [4], no in-depth discussions of how to theorize in SE research are available. Leshem and Trafford [69] presented an analysis of how conceptual frameworks can be used in doctoral research.

SE researchers already theorize–they just don't know it yet. While theories have received limited attention, the SE research community is familiar with the use of *frameworks*. One of the purposes of developing or using a framework is to organize existing concepts from the literature, or to assist in the development and testing of a theory [70]. As such, frameworks can be seen as theory fragments with an analysis goal. Our third example application of the RPS, the study by Medvidović and Taylor, is a good example of this. Their framework provides a typology of ADLs, based on which researchers can design new studies.

We plan the following activities for future work. Firstly, as outlined in this paper there are different purposes and types of research; we plan to provide a further description of how this applies to SE research, and to provide a set of guidelines targeting the research community. Secondly, it is also informative to investigate the various types of theory fragments that are commonly contributed in SE research papers. Such insight will provide a reflection on how theorizing is done so far in SE research, and to provide suggestions as to how this could be improved. Thirdly, we will further develop, adapt, and illustrate use of the Research Path Schema, derived from Brinberg and McGrath's work [18]. While this paper provides an initial outline and example illustration, space restrictions prevent a more in-depth discussion of the implications.

Acknowledgments

This paper has been greatly improved based on extensive feedback from Howell Jordan and Pádraig O'Leary. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero–the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] P. Johnson, M. Ekstedt, and I. Jacobson, "Where's the theory for software engineering?" *IEEE Software*, pp. 94–96, 2012.

[2] J. Hannay, D. Sjøberg, and T. Dybå, "A systematic review of theory use in software engineering experiments," *IEEE Transactions on Software Engineering*, vol. 33, no. 2, pp. 87–107, 2007.

[3] D. Sjøberg, "Documenting theories: Working group results," in *Experimental Software Engineering Issues: Assessment and Future*, V. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, and R. Selby, Eds. Springer-Verlag, 2007, pp. 111–114, LNCS 4336.

[4] D.I.K. Sjøberg, T. Dybå, B.C.D. Anda and J.E. Hannay, "Building theories in software engineering," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008.

[5] "Software engineering method and theory (semat) website," `http://semat.org`.

[6] P. Ralph, P. Johnson, and H. Jordan, "Report on the First SEMAT Workshop on General Theory of Software Engineering (GTSE 2012)," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 2, 2013.

[7] J. Offutt, "Putting the engineering into software engineering education," *IEEE Software*, vol. 30, no. 1, pp. 94–96, 2013.

[8] L. Briand, "Embracing the engineering side of software engineering," *IEEE Software*, vol. 29, no. 4, pp. 93–96, 2012.

[9] P. Naur and B. Randell, Eds., *Report on a conference sponsored by the NATO SCIENCE COMMITTEE*, 1968.

[10] P. Ågerfalk and B. Fitzgerald, "Flexible and distributed software processes: Old petunias in new bowls?" *Commun. ACM*, vol. 49, no. 10, pp. 27–34, 2006.

[11] B. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *Int'l Conference on Software Engineering*, 2004.

[12] K. Lewin, "The research centre for group dynamics at massachusetts institute of technology," *Sociometry*, vol. 8, pp. 126–135, 1945.

[13] S. Lynham, "The general method of theory-building research in applied disciplines," *Advances in Developing Human Resources*, vol. 4, no. 3, pp. 221–241, 2002.

[14] S. Gregor, "The nature of theory in information systems," *MIS Quarterly*, vol. 30, no. 3, pp. 611–642, 2006.

[15] R. Merton, *Social Theory and Social Structure*, Enlarged ed. Free Press, 1968.

[16] L.J. Bourgeois, III, "Toward a method of middle-range theorizing," *Academy of Management Review*, vol. 4, no. 3, pp. 443–447, 1979.

[17] K. Weick, "What theory is not, theorizing is," *Administrative Science Quarterly*, vol. 40, no. 3, pp. 385–390, 1995.

[18] D. Brinberg and J. McGrath, *Validity and the Research Process*. SAGE Publications, 1985.

[19] R. Sutton and B. Staw, "What theory is not," *Administrative Science Quarterly*, vol. 40, 1995.

[20] S. Pfleeger and B. Kitchenham, "Principles of survey research: Part 1: Turning lemons into lemonade," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 6, pp. 16–18, 2001.

[21] P. Runeson, M. Höst, A. Rainer, and C. Wohlin, *Case Study Research in Software Engineering*. Wiley, 2012.

[22] G. Coleman and R. O'Connor, "Using grounded theory to understand software process improvement: A study of irish software product companies," *Information and Software Technology*, vol. 49, no. 6, 2007.

[23] H. Sharp and H. Robinson, "An ethnographic study of xp practice," *Empirical Software Engineering*, vol. 9, no. 4, pp. 353–375, 2004.

[24] H. Edwards, S. McDonald, and S. Young, "The repertory grid technique: Its place in empirical software engineering research," *Information and Software Technology*, vol. 51, pp. 785–798, 2009.

[25] R. Dubin, *Theory Building*, Revised ed. The Free Press, 1978.

[26] R. Suddaby, "From the editors: What grounded theory is not," *Academy of Management Review*, vol. 49, no. 4, pp. 633–642, 2006.

[27] R. Nisbett, *The Geography of Thought: How Asians and Westerners Think Differently and Why*. Nicholas Brealey Publishing, 2005.

[28] T. Hall, N. Baddoo, S. Beecham, H. Robinson, and H. Sharp, "A systematic review of theory use in studies investigating the motivations of software engineers," *ACM Transactions on Software Engineering and Methodology*, vol. 18, no. 3, 2009.

[29] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*, 2nd ed. Blackwell Publishers, 2002.

[30] D. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study," *Information and Software Technology*, vol. 53, no. 5, pp. 440–455, 2011.

[31] P.D. Reynolds, *A Primer in Theory Construction*. Macmillan Publishing Company, 1971.

[32] V. Basili and M. Zelkowitz, "Empirical studies to build a science of computer science," *Commun. ACM*, vol. 50, no. 11, pp. 33–37, 2007.

[33] M. Broy, "Can practitioners neglect theory and theoreticians neglect practice?" *IEEE Comput.*, vol. 44, no. 10, pp. 19–24, 2011.

[34] M. Lehman and J. Ramil, "An approach to a theory of software evolution," in *Int'l Workshop on Principles of Software Evolution*, 2001.

[35] M. Bunge, *Scientific Research I: The Search for System*. Springer-Verlag New York Inc., 1967.

[36] M. Shaw, "Prospects for an engineering discipline of software," *IEEE Software*, vol. 7, no. 6, pp. 15–24, 1990.

[37] J. Carroll and P. Swatman, "Structured-case: a methodological framework for building theory in information systems research," *European Journal of Information Systems*, vol. 9, no. 4, pp. 235–242, 2000.

[38] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Pearson Education Ltd., 2003.

[39] R. Wieringa, M. Daneva, and N. Condori-Fernandez, "The structure of design theories, and an analysis of their use in software engineering experiments," in *Empirical Software Engineering and Measurement*, 2011.

[40] A. Kaplan, *The Conduct of Inquiry: Methodology for Behavioral Science*. Crowell, 1964.

[41] N. Chalofsky, "Professionalization comes from theory and research: The "why" instead of the "how to"," in *New Directions for Adult and Continuing Education*, 1996, pp. 51–56.

[42] D. Benardot, *Advanced Sports Nutrition*, 2nd ed. Human Kinetics, 2012.

[43] D. Whetten, "What constitutes a theoretical contribution?" *Academy of Management Review*, vol. 14, no. 4, 1989.

[44] H.M. Blalock, Jr., *Theory Construction: From Verbal to Mathematical Formulations*. Prentice-Hall, Inc., 1969.

[45] R. Weber, "Theoretically speaking," *MIS Quarterly*, vol. 27, no. 3, pp. iii–xii, 2003.

[46] K. Weick, "Theory construction as disciplined imagination," *Academy of Management Review*, vol. 14, no. 4, pp. 516–531, 1989.

[47] P.B. Medawar, "Is the scientific paper a fraud?" *Saturday Review*, pp. 42–43, 1 August 1964.

[48] H. Reichenbach, *Experience and Prediction*. University of Chicago Press, 1938.

[49] M. Yadav, "The decline of conceptual articles and implications for knowledge development," *Journal of Marketing*, vol. 74, pp. 1–19, 2010.

[50] K. Crowston, H. Annabi, J. Howison, and C. Masango, "Effective work practices for software engineering: Free/libre open source software development," in *WISER*, 2004, pp. 18–26.

[51] S. Bacharach, "Organizational theories: Some criteria for evaluation," *Academy of Management Review*, vol. 14, no. 4, pp. 496–515, 1989.

[52] G. Homans, "Contemporary theory in sociology," in *Handbook of modern sociology*. Rand McNally, 1964, pp. 951–977.

[53] E. Swanson, "The dimensions of maintenance," in *2nd International Conference on Software Engineering*, 1976, pp. 492–497.

[54] T.S. Kuhn, *The Structure of Scientific Revolutions*, 4th ed. The University of Chicago Press, 2012.

[55] F. Shull and R. Feldmann, "Building theories from multiple evidence sources," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 337–364.

[56] P. Runeson, "Theory building attempts in software engineering," in *The Semat Workshop on a General Theory of Software Engineering*, 2012.

[57] K. Corley and D. Gioia, "Building theory about theory building: What constitutes a theoretical contribution?" *Academy of Management Review*, vol. 36, no. 1, pp. 12–32, 2011.

[58] J. McGrath and D. Brinberg, "External validity and the research process: A comment on the calder/lynch dialogue," *Journal of Consumer Research*, vol. 10, no. 1, pp. 115–124, 1983.

[59] J. Feller, B. Fitzgerald, and A. van der Hoek, Eds., *First International Workshop on Open Source Software Engineering (WOSSE)*, 2001.

[60] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Information and Software Technology*, vol. 50, pp. 860–878, 2008.

[61] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Software*, vol. 23, no. 2, pp. 31–39, 2006.

[62] S. Pfleeger, "Albert Einstein and Empirical Software Engineering," *IEEE Computer*, vol. 32, no. 10, pp. 32–38, 1999.

[63] B. Glaser and A. Strauss, *The discovery of grounded theory: strategies for qualitative research*. Aldine Transaction, 1967.

[64] A. Mockus, R. Fielding, and J. Herbsleb, "A case study of open source software development: The apache server," in *International Conference on Software Engineering*, 2000.

[65] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

[66] A. Mockus, R. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, 2002.

[67] C. M. Schweik, R. English, and S. Haire, "Factors leading to success or abandonment of open source commons: An empirical analysis of sourceforge.net projects," *South African Computer Journal*, vol. 43, 2009.

[68] N. Medvidović and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[69] S. Leshem and V. Trafford, "Overlooking the conceptual framework," *Innovations in Education and Teaching International*, vol. 44, no. 1, pp. 93–105, 2007.

[70] A. Schwarz, M. Mehta, N. Johnson, and W. Chin, "Understanding frameworks and reviews: A commentary to assist us in moving our field forward by analyzing our past," *The DATA BASE for Advances in Information Systems*, vol. 38, no. 3, pp. 29–50, 2007.