If you want better empirical research, value your theory: On the importance of strong theories for progress in empirical software engineering research

Klaus Schmid

Software Systems Engineering, Institute of Computer Science,
University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Please cite this publication as follows:
Klaus Schmid. "If you want better empirical research, value your theory: On the importance of strong theories for progress in empirical software engineering research". In: *Evaluation and Assessment in Software Engineering*. EASE 2021. ACM, 2021. DOI: 10.1145/3463274.3463360.

The corresponding BibTeX-entry is:

```
@INPROCEEDINGS{Schmid21,
  author    = {Klaus Schmid},
  title     = {If you want better empirical research, value your theory: On the importance
               of strong theories for progress in empirical software engineering research},
  booktitle = {Evaluation and Assessment in Software Engineering},
  series    = {EASE 2021},
  year      = {2021},
  publisher = {ACM},
  numpages  = {6},
  keywords  = {Theory-empirism relation; Software engineering theories; Scientific inquiry
               cycle; Managing research knowledge; Theory building; Hidden concepts},
  doi       = {10.1145/3463274.3463360},
}
```

If you want better empirical research, value your theory

On the importance of strong theories for progress in empirical software engineering research

Klaus Schmid
Software Systems Engineering, Institute of Computer Science, University of Hildesheim
Hildesheim, Germany
schmid@sse.uni-hildesheim.de

ABSTRACT

Scientific progress comes from creating sound theories. However, current software engineering still mostly falls short of this goal, although its importance is widely accepted. Thus, in this paper, we discuss the importance of a successful interaction of empirical research with a strong theoretical basis and the ramifications this has. In particular, we will extensively discuss the implications on theory building and the empirical vs. theory interaction, etc. While not everything we will discuss is novel, we present a number of insights, which we at least did not see in software engineering literature. We strongly believe that a careful consideration of the insights discussed in this paper has the potential to lead to a significant improvement in software engineering research.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; *Surveys and overviews.*

KEYWORDS

Theory-empirism relation, Software engineering theories, Scientific inquiry cycle, Managing research knowledge, Theory building, Hidden concepts

ACM Reference Format:
Klaus Schmid. 2021. If you want better empirical research, value your theory: On the importance of strong theories for progress in empirical software engineering research. In *Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3463274.3463360

1 INTRODUCTION

In this paper, we aim at providing some inputs to the ongoing discussion of how to better perform empirical research in of software engineering (SE) [4, 12, 16–19]. While we value the existing significant contributions and the strong impact that empirical software engineering research has, we believe there are certain aspects of software engineering research that should be more intensively discussed than they were in the past. The common theme of our discussion is the interaction between theory and empirical research and some consequences we propose for empirical research.

In order to scope our discussion, we will mainly focus on the part of software engineering research that addresses human activities, i.e., as opposed to purely technical research, which is very different in terms of repeatability and number of possible replications, etc. Such human-oriented research may well have a technical artifact or topic as part of its object of research, e.g., how productive are humans with different programming languages is still human-oriented. Also, the broad area of research addressing which methods bring which benefits in practical software engineering is typically human-oriented as the application by humans is part of the research object. Our restriction to human-oriented research is not fundamental in the sense that we believe the discussed aspects would not be relevant in technology-oriented research. To the contrary, we believe, they apply there mostly as well. Rather, we do not regard research practice just as problematic there. At the same time the typical approaches and issues are also different, hence, our choice of focus.

The thoughts, assumptions, and focus of this paper partially stem from my personal background in software engineering, which is intimately linked with the cooperation with companies. Very often the question was: what could be the best possible advice that could be given to a company to improve its results in software engineering, given the specific contexts and pressures (e.g., organizational, market-oriented) the company faces? While I was already then deep into empirical software engineering[1], I found it very difficult to provide truly scientifically grounded guidance as the contexts of existing research was invariable significantly different from the current case at hand and there were only few general rules (e.g., [6]). Thus, this always involved a significant amount of experience. Since then I cherish the vision that, if we would have a deep understanding of software engineering (methods), we should be able to predict within certain boundaries what would be the outcome of applying a specific software engineering method in a particular context, respectively, what would be the result of a particular intervention.[2] While the goal is obviously to engineer methods, it is ultimately a scientific goal as it requires a sufficient understanding of the conditions and results of software engineering methods. My assumption here is that this is also a reasonable way of framing the goal of software engineering research as a whole. This is in line with statements that *usefulness* is/should be a fundamental goal of software engineering research [16].

Throughout this paper, we will repeatedly refer to Physics (and other sciences) as a reference to discuss the properties of good research and in particular theories. The reason is not that we want to take a stance on aspects like whether software engineering is a science or an engineering discipline. We actually believe that this

EASE 2021, June 21–23, 2021, Trondheim, Norway

ACM ISBN 978-1-4503-9053-8/21/06…$15.00
https://doi.org/10.1145/3463274.3463360

[1] Back then I worked at Fraunhofer IESE one of the very early strongholds of empirical research in Germany.
[2] I should note that I never saw this as a fully achievable goal, but rather as a guiding light, similar to the grand unified theory in physics.

distinction is mostly unimportant for the specific aspects discussed here. We also do not want to claim that observations made about Physics are directly transferable to other disciplines. Rather, Physics is widely considered as a gold standard in terms of applying the scientific method, e.g., compared to areas like Psychology or social sciences, which largely suffer from a reproducibility crisis [1]. This is also the reason why Physics was very inspirational in the philosophy of science in general. General concepts of scientific progress like Popper's *falsification principle* [15] or the notion of a *paradigm shift* [9] were strongly influenced by observations in Physics, just like the philosophy of science. Also, we hope it makes our examples more accessible to the reader. However, we will also refer to theories in other disciplines like psychology, but not in a detailed way because we do not assume many readers will have a background in this.

In the next section, we discuss the relation between empirical work and theory. We will use this to draw initial conclusions on what makes good empirical work from the broader perspective of the scientific process. In Section 3 we will then discuss how scientific theories are created and improved by empirical work and will draw further conclusions on "good" empirical work and compare this with some examples from actual SE research. A common theme throughout our discussion is how to aggregate empirical evidence into scientific theories. We will discuss this in Section 4. Again, this will lead to certain proposals for scientific work in SE. Finally, Section 5 will summarize our findings. Throughout Sections 3–4 we will move from things already discussed in software engineering (e.g., [17]) towards aspects that are rarely discussed, but we believe are very fundamental.

2 EMPIRISM VS. THEORY

In software engineering the fundamental relation between theory and empirical work is widely accepted to be one where theory leads (or should lead) to hypotheses, which are then evaluated by the use of empirical methods [12, 18]. This leads to an understanding of the correctness of the underlying hypothesis and either a confirmation of the theory or an improvement of the theory, in case the empirical result does not fit the current state of the theory. This is depicted in Figure 1. The revision of theories based on outcomes and in particular the need that a true scientific theory creates hypotheses that are falsifiable was strongly argued by Popper with the *falsification principle* [15]. Kuhn added the observation that a theory is usually not revised when first contradictory observations appear, but only once significant evidence accumulates that it is inconsistent to current observations [9]. While the term theory is sometimes only used for strongly validated and accepted knowledge, for simplicity we will useit here in a nondiscriminatory way and only discuss different levels of supporting evidence. Another perspective, which is also included in Figure 1 is that in case there is no theory, empirical inquiry can also be used to create initial hypotheses and consolidate them into theories. One such approach is Grounded theory [19]. However, core to scientific work is the incremental refinement of theories through this cycle, which is sometimes also referred to as the scientific inquiry cycle. The idea to make this the core of software engineering was already very early proposed, e.g., in the *Experience Factory* approach [2], albeit with a strong focus on theories for a specific development organization, e.g., a company.

From this very simple understanding we can derive some initial constraints on properties good scientific theories should have:

Proposition 1. *A scientific theory should allow to make predictions for new situations.*

While this may sound simple, it has widely ranging ramifications. The first is that:

Proposition 2. *A theory should provide a rich set of concepts sufficient to characterize all relevant phenomena to which it applies.*

Thus, it needs to provide a standardized vocabulary (terminology) that is sufficiently expressive to describe the various phenomena in the domain like the relevant observable variables, any constraints, any laws that apply in the context, etc. In physics such a vocabulary could be "mass", "resistance", "speed", etc. But not only natural sciences provide this. For example, psychological theories also come (among other things) with fundamental vocabularies to organize observation. Actually, different theoretical frameworks differ significantly in their basic vocabulary [13]. In order to provide clear semantics to specific variables, there should also be standards for them, like it is clear how to measure one meter, the speed of light, etc. It is interesting to compare this requirement with the confusion in software engineering to measure s.th. as fundamental as the size of a software artifact. If there is no standardized meaning, then it is also extremely difficult to integrate different empirical results.

Another important property is that such a language should be more expressive than is needed to describe an individual case. While it does not need to be as expressive as the language of classical mechanics or thermodynamics, especially initially, it still should be sufficient to describe all situations in which the theory will be applied. Again, we can compare this with SE research, where many papers start by defining their specific terminology and this terminology is often not applicable outside the specific case.

A second corollary is:

Proposition 3. *A scientific theory needs to define ways to apply its content to new situations.*

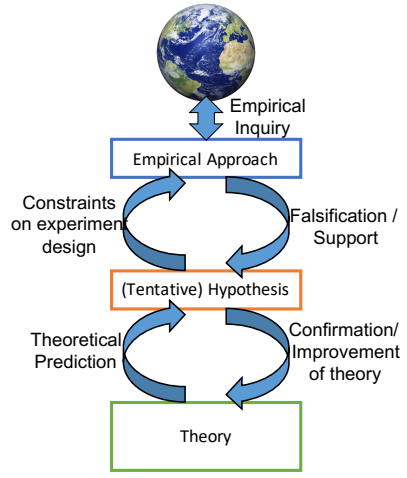

Figure 1: The relation between empirical work and theory. (Inspired by [12, Fig. 1])

This builds and extends the previous proposition. While this may sound trivial, it is not as a scientific theory is not necessarily a solitary statement, but can consist of numerous individual statements that may have complex interdependencies. Hence, the case to which we would like to apply the theory might be different from any situation that was involved in constructing the theory. Just think about s.th. like predicting the position of an accelerated body over time. This requires various non-trivial pieces of theory and corresponding computations. In case of multiple interacting bodies, we will have to even resort to simulations as there is no easy way of determining the position otherwise. Of course, some may say, nobody expects to handle such complex cases in SE, but then, the question is: would we not expect that a successful SE theory would be able to predict outcomes when a software engineering process would be modified? And would we expect this to be simpler than dealing with a moving body? We will return to the combination of multiple elements of a theory in Section 4.

A further property that we expect of a scientific theory is:

Proposition 4. *A scientific theory should have a clearly defined range of applicability.*

We see this clearly in the context of physics. We have, for example, Newtonian mechanics and Einsteinian mechanics. While they fundamentally deal with the same phenomena, they provide different answers for certain conditions. Thus, we know Newtonian mechanics is incorrect. However, we also know it is easier to handle and in a large range of conditions (non-relativistic speeds) it provides correct answers. Thus, it is still used today. This clear understanding, of when a theory is applicable, is very important as it can only be used to create hypotheses within this range and it is not meaningful to revise a theory based on empirical evidence outside its range of applicability (except, in order to explicitly extend the theory to a larger range of situations). Again, we can compare this with the state in Software Engineering. Often results that are derived from empirical work strive to be implicitly all-quantified. This lack of understanding of the limits is hugely detrimental for creating scientific theories.

3 THEORY BUILDING AS LEARNING

Creating and improving theories is the fundamental learning process in science. Nevertheless, sometimes the idea is brought up that theories themselves could be a problem by focusing energy in the wrong direction or creating preconceived assumptions that lead to inadequate interpretations. As an example of this sometimes the geocentric worldview is seen, which did rule for centuries and required a paradigm shift to be resolved [9]. However, this view misses two aspects (a) that opposed to most trivializing renditions of the theory various extensions allowed for a broad range of very precise predictions, and (b) the question whether without such a model sufficient systematic evidence would have been gathered to introduce a well-founded heliocentric world view. Thus, we propose:

Proposition 5. *Any theory is better than no theory as it provides a crystallization point for insights.*

In sciences like Physics (but, to my knowledge in many other scientific disciplines as well) it is the rule that a theoretical analysis should be done prior to any empirical analyses. Even though there are exceptions like purely exploratory settings, I would argue, even then a detailed analysis guided by "best knowledge" is useful to determine the relevant phenomena that should be taken into account. When I received my Physics training in University, for lab courses, it was a precondition that a full theoretical analysis of the experiment was done. This included, in particular, predictions of the outcome and definitions of how this would be evaluated. If this was found insufficient, then one could not progress to execution. This is also not really new in Software Engineering, for example, GQM guidelines state that prior to performing measurements (quantified) hypothesis should be identified [8]. However, while those cases focus on identifying expectations from target groups, we emphasize the need to create hypotheses based on an initial theory. In GQM (and in general) a major reason is that even if the prediction is wrong, it helps to understand what phenomena to look out for and what values to measure.

One should not confuse this with the fact that many empirical studies define statistical hypotheses. A statistical hypothesis has little relation with the notion of a scientific hypothesis in a scientific theory. There may be some overlaps, depending on the specific validation methodology, but mostly, these are different terms.

This leads to the following statement on how a theory should provide input for empirical work:

Proposition 6. *An empirical study (except for pure theory finding) should be started based on a hypothesis, which should be derived from predictions based on an initial theory.*

We exclude the starting point of explicit theory finding as sometimes no initial theory is available. However, we believe, in most cases, where this is done and methods like grounded theory are invoked this is not really the case and it would be better to introduce even very simple theories. One should also note that beyond the predictions themselves the mere process of trying to derive them typically highlights implicit assumptions, constraints that are relevant to the study, etc. By not attempting this, a natural outcome is that these issues may easily go unnoticed and may lead to biases. Validation of scientific theories should happen using experiments that are significantly different to those that were used to initially derive them. One may want to compare this to software engineering were often a one-to-one replication is considered as a goal one should aspire to. A thorough upfront analysis also clarifies restrictions on the relevant parameters of a study such as the expected statistical effect size, etc.

Predictions are also critical to updating a theory. Ideally, the observation is in line with the prediction. In this case no update of the theory is necessary, but it is still important to note that corroborating evidence exists, e.g., for novel combinations. The amount of evidence supporting a theory make a huge difference from a scientific point of view. In case there is a significant divergence between the prediction and the evidence (i.e., the hypothesis is falsified), it is important to update the theory. However, this is not just assuming the opposite of the hypothesis. Rather, the new evidence needs to be reconciled with the existing theory, while still ensuring predictions that were confirmed in other empirical research based on the same theory. The richer and more complex a theory, the more difficult is this step.

If this is taken seriously, it leads to more complex theories that propose interactions among different laws and are able to create predictions and integrate evidence from many individual studies. Only such theories are also useful to apply them to novel situations. The complexity of such interconnected theories is also a major reason why an unexpected observation does not directly lead to overturning

an existing theory, but first to attempts to "fix" the theory and only if evidence becomes overwhelming a theory is finally overturned. In our view, the impact on the larger theory should also be made explicit as part of an empirical study. However, this requires, of course, that a corresponding theoretical context is provided to begin with. We call this the output relation from empirical work to theory. This leads to:

Proposition 7. *Empirical work should always clarify the input and the output relation relative to an underlying theory. On the input side, it should clarify how it uses existing theories, e.g., to derive predictions, while on the output side, it should highlight the need for revisions of the theory (in an as limited way as possible).*

We call empirical work that only defines its input-relation *input-only*, while we call work that is output-only *ouput-only*. One should note that, based on our arguments above, a mere statistical test that may be confirmed or refuted and then leads to statements on the original statistical hypothesis, is not using input, if the statistical hypothesis is not supported by a larger theory and is not providing output, if it is not providing information on a larger theory. We call such empirical work *singleton theories* as they provide only singular statements. Examples of this are the typical empirical papers, which only provide some ad-hoc research question without larger context and answers it than empirically. In our view, singleton theories do not provide significant benefit to the progress of software engineering as a discipline. We would expect that good scientific work is grounded both on the input side and the output side firmly in the notion of a theory.

Given the lack of strong SE theories one should recognize that this is certainly a very strong requirement. However, we do not want to lament the situation, but propose it as a *goal for the future*.

While, we believe, this is how it should be, it is our impression that most empirical work in SE does not conform to these observations. However, in order to not only rely on impressions, we decided to have a look at the current situation. Of course, a detailed analysis of existing research in the area was beyond our available resources. Hence, as this workshop is co-located with the International Conference on Evaluation and Assessment in Software Engineering (EASE), which is an A-ranked conference, according to CORE-rankings, we decided to take a look at the research track of last year's edition. We went through all those papers and categorized them as follows:

(1) All papers were classified based on the SE aspects they dealt with as human-based, technology, or meta-research. Meta-research papers dealt with general aspects of how to do scientific analysis. Technology papers did not involve humans-in-the-loop. This included, for example, ML-approaches even if they dealt with aspects of human-artifacts (e.g., language analysis) if the focus was mostly on improving the ML-approach.
(2) We only looked systematically into those papers that dealt with human-oriented aspects, like SE methods.
(3) We always read introduction and conclusions in order to understand the overall direction of the paper. We further checked the background and related work sections as well as the part of the text that aimed to identify an hypothesis (if at all) to determine whether any form of theory was referenced why a particular hypothesis was assumed (input relation). Papers that did mention a clear argumentation for their hypothesis or did provide a reasonable background theory in order to derive it were taken as "has theory input".

Table 1: Analysis of the EASE-2020 research track papers

Category	# of cases
Total papers	26
Papers in category: human-based	12
Papers in category: technology	11
Papers in category: meta-research	3
Papers with input	2
Papers with output	2(0)
Singleton (neither input nor output)	9 (11)

(4) We checked the analysis and any form of result / discussion section to understand whether a theory-relevant result beyond a mere statistical hypothesis test was identified. In particular, whether ramifications on existing theories or the proposal of new ones were given. Papers that had such a connection identified were taken as "has theory output".

Some limitations apply to this study. First, this was performed individually, i.e., without a cross-check. Also as the goal was a rough estimate and not to get hard data no full reading was done. Nevertheless, the results are telling. They are summarized in Table 1.

Further observations include: it was sometimes difficult to differentiate between a theory that is used in general as a background of a study vs. a theory that is really used as an input to the empirical study. Obviously, authors did not have such a distinction in mind.

Out of the total of 26 papers, we categorized 12 as dealing with human-aspects. This was sometimes rather difficult. For example, one paper aimed at identifying human decision making behavior from text using NLP. This we classified as a human-based paper as the hypothesis foused on human behavior. However, some other papers simply tried to improve precision and recall of some ML-approaches on NLP input. In these cases we categorized them as technical.

Out of those 12 we could only identify 2, which had some form of explicitly stated theory input. However, one of those gave SEMAT as its input [14]. Hence, this mostly dealt with the design artifact under study and less with the hypotheses. We still put it in this category to be on the safe side. The paper by McChesney and Bond clearly had a focus on how humans read program code [11]. They derived their hypotheses from general theories from psychology on reading behavior. We regard this as the clearest case of theory as input within the whole batch. Papers that only have abstract relations to other work (e.g., in related work sections) were not counted as "having input".

Regarding the output, the situation was a bit different as this conference focuses on evaluation and assessment. So, all papers had s.th. to say with respect to empirical results. However, we could observe that these were again mostly lists of observations or such. Sometimes, these were not even easily visible, but somewhat hidden in the text. Even, if they were clearly visible, there was usually no relation expressed among them (e.g., in the sense of the combination of theories as we will discuss in the next section). In particular, only in few instances relations to other observations were discussed, but it was never lifted to an impact on existing theories, probably, because usually no theory-relation was discussed in the first place. The only exceptions in our view are: The paper by Freire et al. provided a systematic map of advantages and disadvantages of paying back technical debt [7]. While, we wouldn't call this a theory, at least the

intention was made clear that this should be used later on in a systematic way to predict whether to address technical debt. On the other hand McChesney and Bond discussed the theoretical implications in some depth [11] by pointing out where things were consistent with the prior theory and where they were not. However, they did not discuss ramifications for the underlying theory in greater depth. If we would be very stringent, we would need to say that truly no paper provided a clear theory impact as an output. However, with all given caveats we rate those two as output (hence: 2 (0) in Figure 1). Thus, we can say (if we are not too stringet) that there are 9–11 (out of 12) singletons, i.e., papers that are basically isolated from any deeper theory.

Overall, we regard the situation as rather poor. Even on a top conference most (nearly all) papers are not really integrated into theory formation, which would be a fundamental characteristic of science. Unfortunately, this situation was expected and was already described by others as problematic [18]. This leads us to:

Proposition 8. *We need to stop doing* theoretical singletons *and move towards larger, integrated theories.*

4 OPERATIONS ON THEORIES

In a few places throughout this paper, we commented on the need to make predictions using a theory, on the need to update theories in order to incorporate new evidence, while preserving other results, and so forth. In this section, we will ask how can these needs can be addressed and try to provide some tentative answers.

Again, a quick look at other scientific fields is useful. If we look at Physics, it is as successful as it is, because it casts its theories in mathematical terms (often calculus). As the laws of Physics have a predictive interpretation, they can be directly used with mathematics to derive predictions about reality. In more complex situations simulations are performed, which, however, are based on the same foundation. An important concept here is the notion of a mathematical derivation of the prediction. The problem of updating a theory in case of falsification is then reduced to perform a root-case analysis and try to fix the problems in a way that corrects the current result, while still allowing for the other correct results to remain predicted. One could regard this as similar to the problem of truth-maintenance [5], only not on a value level, but including the underlying rules as well.

However, in software engineering, especially if we talk about human-in-the-loop, we apparently can rarely use a nice mathematical theory as Physics does, or can we? What could we use as a replacement for operations on theories similar to the way other disciplines use calculus?

As we need to address complex situations, where many rules would be interlinked, we need to assume that such a *SE calculus* would need to be able to deal with complex interdependent pieces of knowledge and combine them, e.g., to create predictions. While this would be common to all calculi, another question is what values to deal with. Sometimes, this could be just boolean values, allowing to easily apply causal chains, where some general directions could be identified. Causal modelling is an approach that supports such reasoning [10].

In other cases there might be quantities, but a detailed computation is out of the question. However, this would still allow for some general reasoning. A framework that aims to support such kind of analysis capabilities is qualitative reasoning [3].

Additional approaches that can be used and would also allow for using numerical values were used in SE research as well. For example, two very common approaches are *discrete-event simulation* and *system dynamics* [20]. While they have mostly been used to understand the situation in individual projects, we believe they can provide a conceptual basis for such a calculus as well.

Basically, we use the term SE calculus to denote any systematic approach for deriving predictions for specific cases from a (partial) theory. Hence, the list above is nowhere a complete overview, rather it should just illustrate the breadth of approaches already available today, ranging all the way from only loosely formal approaches like causal chains to rather formal approaches like system dynamics. Many more are certainly around. Thus one cannot blame the absence of such foundational work on a lack of options for a SE calculus. For creation of any specific theory, a choice of such an underlying SE calculus would need to be made.

Proposition 9. *Any theory should define a relevant form of underlying software engineering calculus.*

A side effect of using such a combination approach for theories is that it addresses another problem that is often mentioned in SE research: the many (potential) influential factors. It is unrealistic to create empirical studies that observe and perhaps even control all variables that may potentially be relevant in any given situation.

However, this is also not the case in other areas of science, where it might even in principle be possible. Let's Physics again use as an example. You may know that a free falling body is accelerating over time. You may also know that in an atmosphere it will ultimately reach a maximum speed, which depends on its aerodynamics, the mass, the atmospheric pressure, etc. Would anyone study this in Physics, by varying all these parameters and creating all parameter combinations? Of course not! One studies free fall in a vacuum and builds a partial theory around this. Then, one would create a partial theory of aerodynamics that describes how an atmosphere at a particular speed leads to a force on the body, independent of falling. In order to predict the result of a body falling in an atmosphere both partial theories are combined. Both are linked by "force" as an intermediate concept. In this way, the potential of rich theories that enable operations on a theory allow for creating partial theories that are each much easier to validate. We thus conclude that

Proposition 10. *Having an underlying SE calculus helps to address the many-variable-problem in SE validation.*

We would like to emphasize now that we just connected the theories in the falling body case through the concept of a *force*. However, no one has truly seen a force directly, similarly energy cannot be observed directly, actually very many extremely important concepts in well-founded theories are unobservable. We refer to them also as *hidden concepts*. Today, they are widely accepted, because they make the theories work. Some are even fundamental to our understanding of the physical world, e.g., energy conservation. Physics needed a significant amount of time to get them right. The evidence was always indirect and built on predictions in which they played a role. We believe that such hidden concepts are of fundamental importance in any scientific theory. Their postulation and validation requires, however, many interlinked empirical studies. They also require an SE calculus with the properties described above. And it

requires a shift of mindset that such unobservables can be critical to the scientific progress in our discipline.

Proposition 11. *Introducing hidden concepts that help to organize the theory and connects subtheories is central to creating powerful scientific frameworks.*

A final note on theory-building is that we assume that fundamentally any theory needs to be built in a piecemeal fashion as otherwise the complexity is just too high. This incremental approach could rely on a domain-oriented decomposition like we see in the case of our physics example. Here different classes of phenomena and relations among them were identified and each basic situation was intensively studied and led to a partial theory.

Another approach to theory-building would be to create it incrementally based on different contexts (e.g., different companies). Probably also a combination of both could be useful, i.e., understanding a sub-domain within a certain class of contexts.

Proposition 12. *Theory construction needs to focus on creating partial theories interlinked through a common SE calculus.*

5 CONCLUSION

Empirical research has created a strong-hold in software engineering over the last few decades. While it is by now standard to have empirical validations in software engineering papers, the larger picture of the empiricism vs. theory interaction is rarely taken into account, leading to what we call theory singletons.

Part of the problem underlying this is that in our view there are no great scientific theories that provide a basis for software engineering research to date. Part of the reason seems to be that current SE research hardly aims for them. In this paper, it is our goal to help to resolve this problem, by introducing a number of important properties that scientific theories for software engineering should have as described and derived in this paper. We believe that focusing more intensively on *theory-driven empirical* research will lead to better research

Throughout the paper we highlighted a number of important properties that good theories should have. These are:

- A scientific theory defines its range of applicability.
- It should support making predictions for empirical studies.
- It should provide concepts beyond individual experiments that can be applied to novel situations.
- It must support its application to novel situations.

We also found that this has consequences for how empirical research should be done. For example, we identified that:

- Predictions should be a prerequisite for empirical studies.
- Empirical studies should always clarify their input and output relation wrt. a theory.
- A bad theory is better than no theory for an empirical study, except in the very initial stages.
- We need to stop theoretical singletons and should move towards theory-driven empirical studies.

We also identified that theories are not just statements, rather they need some kind of operations on theories, i.e., an SE calculus.

- Any theory should identify such a calculus as a basis.
- A SE calculus helps to resolve the many-variable-problem in empirical studies.
- The introduction of hidden concepts can provide important connection points for integrating partial theories.
- Hidden concepts can help to organize large theories and lead to powerful scientific frameworks.

Finally, we identified the need for striving towards relevant partial theories of SE based on domains or contexts or a combination of both.

The basic idea of a stronger relation between theory and empirical work in SE is not new. It was already proposed several decades ago. However, we believe, we could augment this basic idea in this paper with a number of interesting observations on how such theoretical work and its relation to empirical research should look like. We hope, this provides useful impulses for improving SE research in the future.

ACKNOWLEDGMENTS

The author would like to thank the anonymous reviewers for their very constructive and insightful comments.

REFERENCES

[1] M. Baker. 2015. Over half of psychology studies fail reproducibility test. News article in Nature. https://www.nature.com/news/over-half-of-psychology-studies-fail-reproducibility-test-1.18248 Last visited: 02.05.2021.
[2] V. R. Basili. 1993. The Experience Factory and its relationship to other improvement paradigms. In *Proceedings of the 4th European Software Engineering Conference*. 68–83.
[3] B. Bredeweg and P. Struss. 2003. Current Topics in Qualitative Reasoning. *AI Magazine* 24, 4 (2003), 13–16.
[4] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh. 2017. The case for context-driven software engineering research: Generalizability is overrated. *IEEE Software* 34, 5 (2017), 72–75.
[5] J. Doyle. 1979. A truth maintenance system. *Artificial Intelligence* 12, 3 (1979), 231–272.
[6] A. Endres and D. Rombach. 2003. *Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories.* Addison-Wesley Longmands.
[7] S. Freire, N. Rios, B. Gutierrez, D. Torres, M. Mendonça, C. Izurieta, C. Seaman, and R. Spínola. 2020. Surveying Software Practitioners on Technical Debt Payment Practices and Reasons for not Paying off Debt Items. In *International Conference on Evaluation and Assessment in Software Engineering (EASE'20)*. 210–219.
[8] C. Gresse, B. Hoisl, and J. Wüst. 1995. *A Process Model for GQM-Based Measurement.* Software Technology Transfer Initiative (STTI) STTI-95-04-E. University of Kaiserslautern. https://www.researchgate.net/publication/2694323
[9] T. Kuhn. 1996. *Die Struktur wissenschaftlicher Revolutionen.* Suhrkamp. German.
[10] A. Lamersdorf and J. Münch. 2010. Studying the Impact of Global Software Development Characteristics on Project Goals: A Causal Model. *Open Software Engineering* 4, 2 (2010), 2–13.
[11] I. McChesney and R. Bond. 2020. Observations on the Linear Order of Program Code Reading Patterns in Programmers with Dyslexia. In *International Conference on Evaluation and Assessment in Software Engineering (EASE'20)*. 81–89.
[12] D. Méndez and J.-H. Passoth. 2019. Empirical software engineering: From discipline to interdiscipline. *Journal of Systems and Software* 148 (2019), 170–179. Also as: https://arxiv.org/abs/1805.08302.
[13] A. Neel. 1975. *Handbuch der psychologischen Theorien.* Verlag für die Verbillgung des Studiums. German translation.
[14] A N.guyen-Duc, I. Sundbø, E. Nascimento, T. Conte, I. Ahmed, and P. Abrahamsson. 2020. A Multiple Case Study of Artificial Intelligent System Development in Industry. In *International Conference on Evaluation and Assessment in Software Engineering (EASE'20)*. 1–10.
[15] K. Popper. 2002. *The Logic of Scientific Discovery* (2nd ed.). Routledge.
[16] L. Prechelt. 2021. SW Engineering empirical research: Are we doing the right thing? (mental yoga). https://se-2021.tu-bs.de/wp-content/uploads/sites/4/2021/02/2021-02-are-we-doing-the-right-thing.pdf Keynote at SE 2021.
[17] D. Sjøberg, T. Dybå, B. Anda, and J. Hannay. 2008. Building Theories in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer, 312–336.
[18] K.-J. Stol and B. Fitzgerald. 2013. Uncovering Theories in Software Engineering. In *2nd SEMAT Workshop on a General Theory of Software Engineering*. 5–14.
[19] K.-J. Stol, P. Ralph, and B . Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 120–131.
[20] H. Zhang, B. Kitchenham, and D. Pfahl. 2008. Reflections on 10 years of software process simulation modeling: A systematic review. In *International Conference on Software Process (ICSP)*. Springer, 345–356.