
Paul Ralph
Lancaster University
Lancaster, United Kingdom
paul@paulralph.name

Pontus Johnson
KTH Royal Institute of Technology
Stockholm, Sweden
pontusj@ics.kth.se

Howell Jordan
Lero – University College Dublin
Dublin, Ireland
howell.jordan@lero.ie

ABSTRACT
Many academic disciplines have general theories, which apply across the discipline and underlie much of its research. Examples include the Big Bang theory (cosmology), Maxwell’s equations (electrodynamics), the theories of the cell and evolution (biology), the theory of supply and demand (economics), and the general theory of crime (criminology). Software engineering, in contrast, has no widely-accepted general theory. Consequently, the SEMAT Initiative organized a workshop to encourage development of general theory in software engineering. Workshop participants reached broad consensus that software engineering would benefit from better theoretical foundations, which require diverse theoretical approaches, consensus on a primary dependent variable and better instrumentation and descriptive research.

Keywords

1. INTRODUCTION
Many academic disciplines including oncology, psychology and civil, electrical and aeronautical engineering emerged from practice without particular focus on underlying theories. However, as they matured, these disciplines developed advanced scientific theories. Mature academic disciplines generally emphasize the importance of their general theories [14].

Various fields including optics, circuit theory, psychology, organizational theory and international relations either rest on either a single, general theory or on a small number of alternative theories providing competing predictions about the discipline’s essence. A prime example of a single, general theory is Maxwell’s equations, which explain the behavior of electromagnetic fields. Similarly, psychology has been recently dominated by two competing general theories – cognitive and psychodynamic. Even though electrical engineering and psychology are very different, they are both highly concerned with their core theories. Meanwhile, less mature disciplines, such as Information Systems, may lack one or a few general theories but have intense debates about a larger set of “core” theories [1].

In contrast, few general or core theories of software engineering (SE) have been proposed, and none have yet achieved significant recognition. This is harmful to practice, research and education in at least three ways. First, lack of theoretical foundations inhibits developing a cumulative research tradition, in which theories act to preserve established findings and implicitly coordinate future empirical inquiry. “Theory provides explanations and understanding in terms of basic concepts and underlying mechanisms, which constitute an important counterpart to knowledge of passing trends” [9]. The expression of the SE discipline’s knowledge and education is therefore limited to rules-of-thumb or “best practices.”

Second, without theory, SE is often reduced to trial and error [11] and is vulnerable to fads and piecemeal empiricism. Innovations can therefore only be tested in vivo, which can be both expensive and painful, since the trial-and-error process inevitably produces high failure rates [6, 15, 20].

Third, the lack of theory facilitates proliferation of untested or empirically dubious SE concepts and practices. The rapid succession of fashions in the community serves as evidence of this state of affairs.

Consequently, the Software Engineering Method and Theory initiative (SEMAT) was founded on the belief that increased focus on general theories of SE would be deeply beneficial to SE research, practice and education. The SE community was invited to “refound” SE based on solid theory [19]. To date, SEMAT has submitted a proposed standard to the Object Management Group’s Foundation for the Agile Creation and Enactment of Software Engineering Methods [7]. To advance this goal, SEMAT hosted a workshop on “general theory of software engineering.” The next section describes the workshop and its structure, followed by a summary of the main themes that emerged. The paper concludes with suggestions for future research.

2. THE WORKSHOP
The SEMAT Workshop on a General Theory of Software Engineering (GTSE 2012) was held at KTH Royal Institute of Technology, Stockholm, Sweden, November 8-9, 2012. The aim of the workshop was to contribute to the promotion of the scientific process of proposing, debating, testing and revising general theories of SE, in the hope that the discipline will eventually reach a state similar to that of many other academic disciplines, with one or a few theories that are able to make important predictions about concepts central to SE.

The workshop called for an explorative discussion on questions including:
- How can a general theory of SE be of practical use?
- What are the objectives of such a theory?
- What questions should it address?
- What is a useful definition of theory?
- How foundational/universal should a general theory of SE be?
- What should its main concepts be?
- How formally or informally should it be expressed?

The workshop was organized into two main paper presentation sessions and two discussion sessions. In the first presentation session, participants offered desiderata for or considered aspects of general theories of SE. This was followed by a discussion session on the potential utilities of general theories of SE. In the second presentation session, participants proposed aspects of, content for or preliminary versions of general theories of SE. This was succeeded by a discussion on those concepts, theories and proto-theories. Presentations were recorded and most slides and videos are available via http://semat.org/?page_id=561.)

3. WORKSHOP THEMES
Prior to the workshop each presenter submitted a two-page extended abstract describing their intended talk [cf. 10]. During the workshop one of the authors took detailed minutes of both impromptu discussions within paper presentations and planned discussion during discussion sessions. We applied thematic synthesis [4] to identify patterns in the abstracts and notes. We then sought feedback on the resulting themes from workshop participants. Revising our findings resulted in five core areas in need of attention.
3.1 Theoretical Foundations for SE

Electrical and mechanical engineering are based on a sound theoretical foundation, commonly called “physics.” Chemical engineering is similarly grounded in chemistry, horticulture in biology, and management in psychology and sociology. Similarly, software engineers utilize various theoretical aspects of psychology (e.g., in graphical interface design), mathematics (e.g., in code optimization) and other fields (e.g., in managing software projects), and many theories related to SE have been proposed. However, SE does not rest upon a solid theoretical foundation in the same way as other engineering and professional sciences. Although SE researchers have developed theories, none have gained the widespread acceptance and use of core or general theory, i.e., they may be veracious and useful but are not foundational. This absence of theoretical foundations is deleterious to the SE discipline in several ways (§1).

Consequently, participants agreed that SE would greatly benefit from theoretical foundations, especially a general, unifying theory or paradigm, that is, one which applies across diverse SE contexts and integrates much existing theory, knowledge and empirical insight. Such a unifying theory should include both the product of SE (software artifacts) and the process(es) of software creation and modification.

3.2 Diverse Theoretical Approaches

However, jumping directly to a state where the theoretical foundations of SE are encompassed by a single general theory may be impractical. Instead, an intermediate phase may be necessary where SE’s theoretical foundations are distributed among a larger set of more specific core theories, which may later be integrated into a general theory. Therefore, multiple theory types and approaches may be needed to build up theoretical foundations. Theories may differ in their nature (e.g., process theories explaining how something happens and variance theories explaining why something happens [22]) or their purpose (analyze, explain, predict or design/take action [8]); and their approach to causality (probabilistic, regularity, counterfactual or teleological [12]). Consequently, participants agreed that exploring diverse approaches to theorizing about SE is desirable and call on reviewers and editors to be pragmatic and tolerant of diversity in this area.

Moreover, participants noted the importance of better understanding the common features of SE which may be important for general theory, including projects, actors, software documents, software code, models, practices, rationale, tests, explicit and tacit knowledge, development context and goals. It was felt that understanding these phenomena will require diverse theoretical lenses including state-transition, actor-network, entity-relationship, formal ontology and the theory of boundary objects [2]. More generally, progressing toward useful general theory will involve many parallel and sequential theoretical steps of varying precision, formality, scope and predictive power.

3.3 Consensus on a Primary Dependent Variable

Although developing general theory in SE may necessitate myriad approaches and intermediate concepts and theories, participants felt that some consensus on a key dependent variable would be beneficial. Such a variable may have several causally or temporally related components, as in the DeLoan and McLean Model of Information Systems Success [5], which relates information, system and service quality with use and user satisfaction to constitute the “net benefits” for stakeholders. In SE, this variable should comprise both what general theory seeks to explain and what SE tools and techniques seek to influence.

Workshop participants developed an initial model of SE success (Figure 1). At the most abstract level, SE’s primary dependent variable may be stated “Software Engineering Success,” however, this is too general to use easily or measure directly in many circumstances. Therefore, the primary dependent variable should be formulated as a multifarious construct, i.e., one having formative and causal relationships.

Figure 1: Initial Model of Software Engineering Success

The primary proximate antecedents of SE Success include the quality of design thinking and practice; i.e., design thinking quality and design practice quality would often mediate the effects of tools, methods and techniques intended to improve SE Success. Here, “design thinking” refers to the combination of problem framing, exploration, idea generation and decision making elucidated by [3]. Design practice refers to the aggregate of software engineering behavior including behavioral patterns, group interaction and the use of tools and techniques. The layer of antecedents prior to design thinking and practice quality may include SE research effectiveness, decision quality, and the efficacy of knowledge sharing, understanding and argumentation; e.g. one’s capacity to explain to a client with no clear requirements why a fixed-price/-schedule contract is risky.

Meanwhile, SE Success comprises at least five quasi-independent dimensions (formative constructs). The “Project Triangle” refers to three interacting constraints on outcome quality – schedule, cost and scope. “Artifact properties” refers to the many quality dimensions and attributes of different objects produced in SE projects; for example, participants felt that the predictability of software artifacts was a key concern. Time is included to convey that a seemingly failed project can later become successful and vice versa. Market performance refers to not only absolute profitability but also comparative victories, e.g., the Blu-ray media format’s victory over the HD DVD format in 2008. Finally, the same initiative may be perceived as successful by some stakeholders and unsuccessful by others. This formulation permits more complex understanding of success; e.g., an SE project may be seen as a success by its developers at time 1 because code quality is high but as a failure by management at time 2 because market performance failed to justify budget overruns.

3.4 Better Instrumentation and Metrics

Clear dependent variables, however, are of limited use without effective methods of measuring changes in those variables. Theoretical and empirical work in SE has been hampered by insufficiently robust instrumentation. For example, many SE methods, tools and project management practices are intended to improve developer productivity or code quality. However, no widely accepted, robust instrument for measuring either developer productivity or code quality is available. Similarly, software documents vary in purpose, scope, formality and detail; however, studies of document usefulness are hindered by the absence of robust instrumentation for each of these dimensions. Furthermore, much SE research rests on variable/metrics pairs with low construct or measurement validity, e.g., theorizing about usability as an objective software property but measuring it using perceptual survey scales.

Therefore, participants agreed that developing better metrics and instruments for key SE variables, including those suggested above, will facilitate not only developing, testing and exploiting SE general theory but also higher standards for empirical work in SE more broadly.
3.5 Better Descriptive Research

However, high quality instrumentation and metrics depend on the clarity of the discipline’s nomological network, that is, the precision with which core phenomena are defined. Furthermore, effective operational definitions depend on our basic understanding of the SE context. For example, if a researcher designs a code quality instrument assuming an object-oriented paradigm and unwittingly applies the instrument to a purely functional-paradigm codebase, the results would be questionable if not meaningless. Similarly, analyzing an amethodical development process [21] through the lens of SE methods may lead a well-meaning researcher to paradoxical conclusions.

These examples illustrate SE’s linguistic problem. Software engineers and researchers sometimes say “requirement” when they mean “something somebody said they wanted”, or “we use Scrum” when they mean “we use a homegrown process loosely based on Scrum”, or “system goals” when they mean “goals stated by the CEO that may not be shared by other stakeholders or even represent the CEO’s own value dimensions.” Even traditional divisions of SE activities into analysis, design, coding and testing may be misleading [16-17].

Consequently, participants felt that confusion over SE context and language should be addressed. Possible approaches include better descriptive research including more survey and case study [18] research. Without good descriptive research, new metrics and instruments may exhibit poor measurement validity, dependent variables may exhibit poor construct validity, and proposed core theories may have little relevance to practice.

4. CONCLUSIONS

Although there is today nothing close to a widely accepted general theory of SE, much is in place to support its development. SE is becoming mature. Over the past few decades, the SE community has developed better understanding of scientific methodology [13], better definitions of core concepts, and most importantly, numerous (at least implicit) theories. The workshop demonstrated two important things – first, that there are today many potential candidates for or embryos of a general theory of SE. The papers submitted to the workshop included several interesting proposals. Second, it seems possible to reach a reasonable consensus on quality criteria of general SE theories. As mentioned, significant agreement concerning the primary dependent variable was evident. Considerable agreement was also evident on other relevant criteria including universality, empirical validation and internal consistency.

Continued exploration of the field is planned for the 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE 2013), held in conjunction with ICSE 2013 in San Francisco on May 26, 2013. This event will feature full ten-page research papers as well as four-page position papers. We hope for as interesting discussions as experienced in November in Stockholm.

ACKNOWLEDGMENTS

We extend our thanks to the participants of the workshop:

- Ilia Bider, Stockholm University, Sweden
- Dines Bjørner, DTU, Denmark
- Markus Buschle, KTH, Sweden
- Mathias Ekstedt, KTH, Sweden
- Iaakov Exman, The Jerusalem College of Engineering, Israel
- Ivar Jacobson, IJI, Switzerland
- Ekkart Kindler, DTU, Denmark
- Harold Lawson, Lawson Consulting, Sweden
- Joakim Lilliesköld, KTH, Sweden
- Algitlas Pakstas, UK
- Tero Päivärinta, Luleå University of Technology, Sweden
- Per Runeson, Lund University, Sweden
- Mikko Tiusanen, Tampere University of Technology, Finland
- Pan Wei Ng, IJI, Singapore

REFERENCES
