Developing and Evaluating Software Engineering Process Theories

Paul Ralph
Department of Computer Science
University of Auckland
Auckland, New Zealand
paul@paulralph.name

***Abstract* - A process theory is an explanation of how an entity changes and develops. While software engineering is fundamentally concerned with how software artifacts change and develop, little research explicitly builds and empirically evaluates software engineering process theories. This lack of theory obstructs scientific consensus by focusing the academic community on methods. Methods inevitably oversimplify and over-rationalize reality, obfuscating crucial phenomena including uncertainty, problem framing and illusory requirements. Better process theories are therefore needed to ground software engineering in empirical reality. However, poor understanding of process theory issues impedes research and publication. This paper therefore attempts to clarify the nature and types of process theories, explore their development and provide specific guidance for their empirically evaluation.**

Index Terms—Methodology, process theory, reviewing, checklist

I. What is a Process Theory?

The purpose of this paper is to adapt existing guidance on developing and evaluating process theory research to the software engineering (SE) context. A process theory is a system of ideas intended to explain (and possibly to describe, to predict or to analyze) how an entity changes and develops (cf. [1], [2]). For example, the theory that organisms evolve through mutation and natural selection (Fig. I) is a process theory because it explains how species change and develop over time. It explains a higher level of process (evolution) by dividing it into several lower-level component processes (mutation, selection and reproduction).

Process theories may be contrasted with variance theories, e.g., The Unified Theory of the Adoption and Use of Technology [3] (Fig. II). A variance theory explains and predicts the variance in a dependent variable (e.g. Use Behavior) using independent variables (e.g. Performance Expectancy), mediating variables (e.g. Behavioral Intention) and moderating variables (e.g. Gender). While a variance theory comprises variables and causal relationships, a process theory may include activities, actors, phases, steps, subprocesses and diverse non-causal interconnections.

Furthermore, process theories are not methods. A method(ology) prescribes while a process theory explains. Scrum [4], Lean [5] and the V-model [6] are methods because they prescribe specific practices, techniques, tools or sequences, which are ostensibly better than their alternatives. A process theory, contrastingly explains how something actually occurs, regardless of effectiveness. While a method claims "this way will work;" a process theory claims "it happens this way." The theory of natural selection posits that evolution follows a specific pattern, not that this pattern is particularly efficient. Scrum similarly claims that organizing development into time-boxed sprints is wise – not that everyone does so [4].

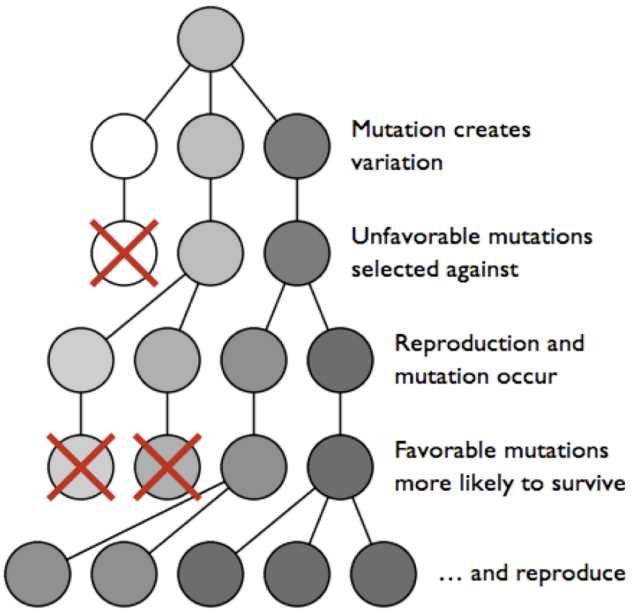

FIGURE I. Evolution through Mutation and Selection

Note: by Elembis - Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons.

Moreover, *process theory* is not a synonym for *process model*. While all theories are models in a sense, process theory research differs significantly from process model research. "A process model is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model" [8, p. 76]. A process model *describes* a *specific* process while a process theory *explains* a process in *general*.

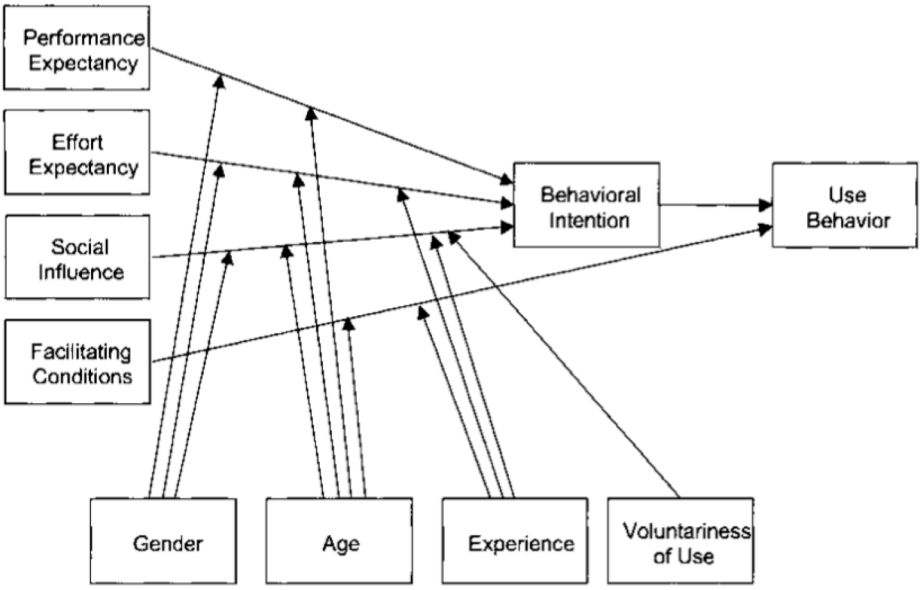

FIGURE II. THE UNIFIED THEORY OF THE ADOPTION AND USE OF TECHNOLOGY (FROM [3])

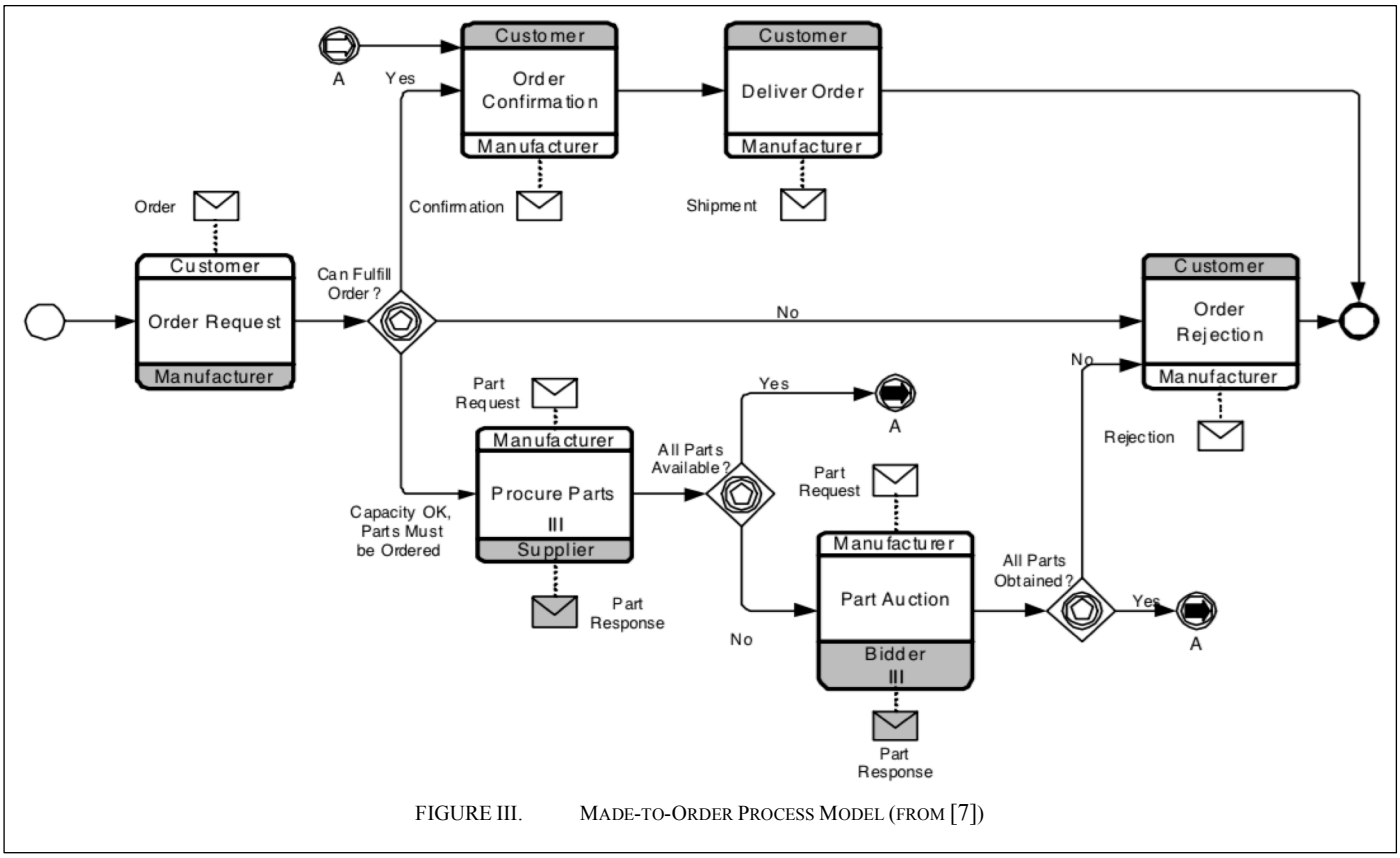

FIGURE III. MADE-TO-ORDER PROCESS MODEL (FROM [7])

For example, Fig. III shows how a manufacturer might build custom components. It does not claim that *all* manufacturers act this way or that a manufacturer acts this way all of the time. The theory of natural selection, in contrast, posits that *all* complex life arises through mutation, selection and reproduction. Process models (like methods) often describe one or a few sequences while process theories seek to encompass all of the ways an entity can change.

Finally, a process theory is *not* a set of steps. While software engineers often think of a process as a set of steps in a

specific order (an algorithm) process theories do not necessarily posit sequential orders or divide processes into steps. Thinking of selection, retention and reproduction as evolutionary steps is deeply misleading. These activities occur chaotically and in parallel across organism populations rather than in a disciplined cycle (see Section V).

The paper proceeds by clarifying why process theories are needed specifically in SE, defining the four types of process theories and providing specific guidance for developing and testing process theories and, by extension, for writing and reviewing process theory papers. The paper also specifically addresses problems with lifecycle process theories in SE.

II. Why do we Need Process Theories?

Despite numerous calls for more theory in Software Engineering (e.g. [9]-[14]), theories of SE processes remain elusive. SE needs process theories for three main reasons.

1) Without process theories, SE academics conceptualize their research using concepts from methods, which inevitably oversimplify and over-rationalize reality [15], [16]. Researchers indoctrinated in methods may therefore struggle to recognize phenomena that are observable in real projects but downplayed in the methods literature, e.g. problem framing [17], illusory requirements [18] and coevolution [19], [20]. Furthermore, without process theories, researchers base surveys and qualitative coding schemes on the concepts and assumptions of methods (e.g. [21]). This may introduce insidious, poorly-understood biases; for instance, a survey question might ask, *during which phase of the project were the requirements agreed – analysis, design, coding, testing, implementation or maintenance?* This presumes: 1) all projects have the same phases; 2) project phases map to those listed; 3) all projects have requirements; 4) requirements are always agreed. In summary, process theories, which encapsulate empirical, explanatory knowledge, counterbalance methods, which provide prescriptions.

2) Absence of communicable theories of SE processes hinders SE education. Educators need some basis for describing SE to students. Without process theories, they continue to base curricula on methods [15], [22], [23]. This gives students an idealistic, oversimplified, incorrect view of the great diversity of SE approaches and projects.

3) Lack of testable process theories permit pseudoscientific nonsense to infest both the academic and popular SE literature. For example, the Gartner Hype Cycle posits that information technologies exhibit a common pattern of visibility as they mature. Simply comparing the hype cycle in subsequent years reveals that many technologies do not follow the posited path. However, without a rigorously evaluated theory of technology visibility to displace Gartner's nonsense, the Hype Cycle continues to enjoy substantial media coverage and perceived credibility.

SE-related processes that may benefit from a process theory approach include the following.

- The process of forming SE teams
- The process through which agility arises
- The process of evaluating the coverage of a test suite
- The process of integrating a new developer into an existing team
- The process of communicating a software design to a new developer
- The process of selecting an architectural pattern (e.g. tiered, service-oriented, model-view controller)
- The process by which open source projects gain momentum, contributors and market share
- The process of finding a bug
- The process of finding and eliminating security vulnerabilities
- The process of problem/solution coevolution
- The software development process in general

III. Types of Process Theories

At least four types of process theories are evident in the literature – teleological, dialectic, evolutionary and lifecycle [2]

In a teleological process theory, an agent "constructs an envisioned end state, takes action to reach it and monitors the progress" [2, p. 518]. In other words, teleological theories explain the behavior of agents taking actions to reach goals, but the agent chooses its own sequence of actions. Teleological process theories are very common in the social sciences [2].

Sensemaking-Coevolution-Implementation Theory (SCI) is a teleological theory that explains how a cohesive development team builds complex software systems (Fig. IV). It posits that development teams engage in three basic activities: 1) making sense of an ambiguous, problematic context (*Sensemaking*); 2) rapidly oscillating between ideas about the context and ideas about the space of possible design artifacts (*Coevolution)* and building the system (*Implementation*) [19], [24]. These activities may occur serially (in any order) or in parallel.

In a dialectic process theory, "stability and change are explained by reference to the balance of power between opposing entities" [2, p. 517]. Possible entities include people, teams, organizations , activities and social forces. Dialectic process theories therefore posit two or more conflicting entities and model change in inter-entity power. These theories are rooted in the argumentative methods of classical philosophy.

For example, Allison and Merali [25] proposed a dialectic theory of software process improvement (Fig. V). It posits a dialectic interplay between software development and software process improvement, where each informs the other. Both process and product metamorphose over time, changing and being changed by their surrounding context.

In an evolutionary process theory, a population undergoes structural changes through *variation* (producing new entities through chance occurrences), *selection* (the preservation of entities with higher fitness and elimination of those with lower fitness) and retention (forces (including inertia and persistence) … perpetuate and maintain certain organizational forms" (p. 518)) [2].

For example, the Problem-Design Exploration Model [20] is an evolutionary process theory that explains how genetic algorithms may be used to design systems (Fig. VI). The problem space (constraints) and solution space (designs) are modeled as coevolving populations – each affecting the other.

A lifecycle theory "is a unitary sequence (it follows a single sequence of stages or phases), which is cumulative (characteristics acquired in earlier stages are retained in later stages) and conjunctive (the stages are related such that they derive from a common underlying process)" [2, p. 515]. This progression occurs because "the trajectory to the final end state is prefigured and requires a particular historical sequence of events" [2, p. 515]. Lifecycle theories have their roots in biological life cycles. Despite their name, lifecycle theories need not contain loops.

If it were a theory, the Waterfall Model [26] – either the linear version Royce condemned or the more iterative version he proposed – would be a lifecycle theory. Outside of SE, the nitrogen cycle and continental drift are lifecycle theories.

While process theories do not posit causal relationships between constructs, they may adopt a particular approach to causality [27]. Evolutionary process theories, for instance, adopt a probabilistic approach to causality in that entities with higher fitness have a greater probability of surviving and reproducing. Teleological process theories adopt teleological causality, i.e., actions are caused by the choices of agents with free will.

IV. Developing Process Theories

When developing a process theory, the following four questions may be useful.

1. *What entities are changing?* Process theories explain how an entity changes, so what is the entity whose change process we want to explain? Example entities include software artifacts, tests, documentation, projects, teams, individuals, and organizations.
2. *Is there a pattern at all?* Some processes are so chaotic, lawless or context-dependent that they lack sufficient common structure on which to base a process theory. Information systems development may even be such a process [28]. It is crucial to recognize our propensity for apophenia – seeing patterns in randomness.
3. *If there is a pattern, what type is it?* If we find a person or cohesive team pursuing one or a few goals through numerous activities, we may adopt a teleological approach. If we find a small number of things engaged in a power struggle, or several conflicting forces, a dialectic approach may help. If we are trying to understand how the structure of a large set of entities changes over time, an evolutionary approach my be preferred. If we discover a common sequence of phases, we may take a lifecycle approach. Finally, if we find two or more of the above, we may adopt a hybrid approach.
4. *What are the specific concepts and relationships of the theory?* For a teleological theory, what are the agent, its goals and activities? For a dialectic theory, what are the entities, what is the nature of their power structure and how do they conflict? For an evolutionary theory, what is the population, how do entities exhibit variation, selection and retention, and what is the fitness function? For a lifecycle theory, what are the stages or phases and what is the sequence. For a hybrid theory, which types are we hybridizing and how do the elements from the different types relate to each other?

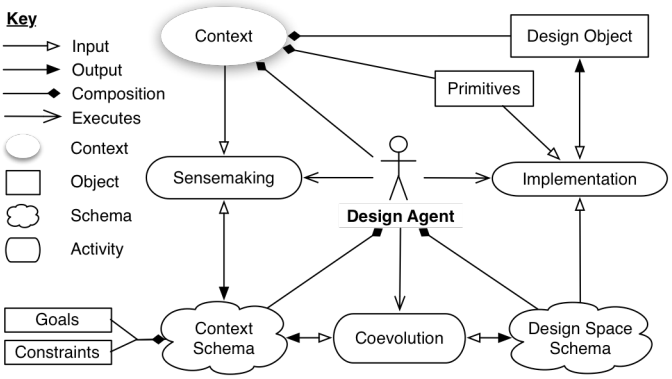

FIGURE IV. Sensemaking-Coevolution-Implementation Theory (adapted from [24])

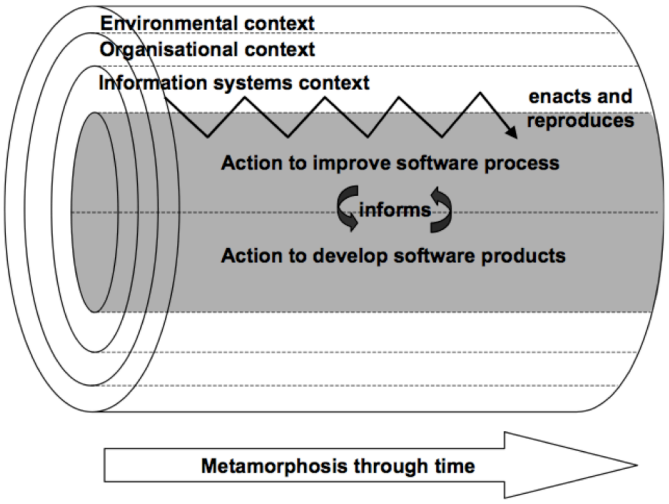

FIGURE V. An Emergent View of Software Process Improvement (from [25])

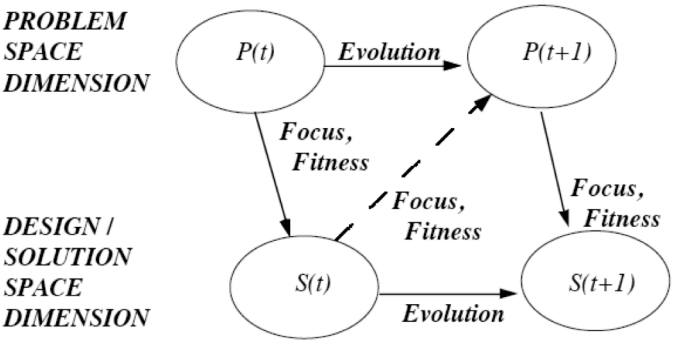

FIGURE VI. Problem-Design Exploration Model (from [20])

To investigate these questions, we can apply at least three theory development strategies, as follows.

1. *Synthesize a process theory using existing theories or models from SE or reference disciplines*. Some SE processes are special cases of more general processes. For example, making software design decisions is a special case of decision making. Therefore, it may be possible to synthesize a process theory of software design decision making by extending or adapting a more general theory of decision making (e.g. [29]). General theories of negotiation, group mind, contracts, learning, communication, creativity, problem structuring and crime may be similarly applied to SE special cases. SCI, for example, was developed by refining an older model of the "self-conscious" design process [19], [30]. Benefits of the synthesizing approach include speed (compared to months or years of field work), natural linkages to existing theories and possibly greater scientific rigor (in that, working with rigorous theories may encourage us to produce more rigorous theories). Source theories however may be incorrect in general or not translate well to SE.
2. *Develop theory based on field studies.* Another approach is to develop a theory from extensive field work, e.g., ethnography or interpretivist case studies. As many SE processes are observable, the researcher can observe the process, consult with its participants and develop a theory in much the same way we develop conceptual models. Process coding [31] may be especially helpful here. The primary advantage of this approach is that it encourages deep immersion in the data, which may protect researchers from oversimplifying or overrationalizing the process. However, this approach is time-consuming and interpretivist methods focus on participants' perceptions of reality, which may be systematically biased [32]-[35] or simply wrong. Moreover, over-reliance on interviews and focus groups can be detrimental (see below).
3. *Use Grounded Theory*. Grounded theory refers to a collection of similar methodologies originally developed by Glaser, Strauss and Corbin [36] for generating theory from mostly qualitative data. Grounded theory is not a synonym for qualitative research and differs substantially from ethnography and case studies. Researchers who adopt grounded theory should read at least one seminal book on grounded theory and clearly indicate whether they are adopting Glaser's approach, Strauss and Corbin's approach or constructivist grounded theory [37]. Grounded theory benefits from a rigorous and systematic analytical approach and high potential for immersion; however, like ethnography, it is very time consuming.

Furthermore, some approaches are highly unsuited to developing process theories, including the following.

1. *Do not base process theories on methods.* A method is a system of prescriptions for doing something effectively. A process theory is a system of ideas for explaining how something happens. Methods are inappropriate foundations for process theories since process theories explain both effective and ineffective behavior [38], [39]. Adapting a method into a theory is therefore likely to overrationalize reality. To overrationalize reality means to present it as we think it should be rather than as it is. For example, presenting something messy as clean, presenting something chaotic as lawful, presenting something unreasonable as reasonable. When economists model humans as rational agents, they are over-rationalizing reality.
2. *Do not base process theories predominately on expert opinion*. An expert is one who is skilled in a specific domain. However, expertise does not inoculate people against cognitive and perceptual biases [40], [41]. For example, experts are susceptible to system justification – the tendency to irrationally defend the status quo [42], [43]. Furthermore, skill in software development or managing software projects, for example, does not imply expertise in theorizing software processes. An expert programmer probably spends much of his or her time programming, not reflecting on the nature of programming processes or systematically recording process variations between different kinds of programmers in different kinds of projects. Unless the experts in question derive their expertise from lifetimes of systematic, rigorous research and meta-analysis, expert opinion is intrinsically anecdotal and unreliable. That said, experts may be an excellent source of data; they are simply not substitutes for observation.

A related pitfall in process theory development is the tendency to blend explanation and description, or combine a theory with a method. Simultaneously modeling how something should happen and how it actually happens is difficult and confusing. Therefore, avoid adding any prescriptive elements to a process theory.

In summary, three ways of developing a process theory are to synthesize it from existing theories, to base it on field studies or to use grounded theory. Process theories should not be based on methods or include prescriptions. While expert opinion may be helpful, it is no substitute for empirical data. During theory development, researchers should ask *what is changing, is there really a pattern, and if so, what kind of pattern*? before determining the theory's concepts and relationships.

V. The Lifecycle Problem

At this juncture, a slight tangent is needed to address SE's lifecycle problem. Many SE researchers, consultants and practitioners emphasize lifecycle models including the Waterfall Model [26], Systems Development Lifecycle [44], V-Model [6], and Spiral Model [45]. The proliferation of lifecycle models superficially suggests adopting a lifecycle approach for theorizing SE processes. However, lifecycle models are intrinsically inappropriate for theorizing most human behavior.

Lifecycle theories posit that the configuration of a process is immutable. SE processes, meanwhile, involve human agents who can choose to change their goals, change their activity sequence, invent new activities, give up part way through a

process and generally take unexpected actions. Modeling the process of fixing nitrogen as a lifecycle is useful partially because nitrogen will not spontaneously refuse to interact with plants or invent a new method of bonding directly to the soil. Human actors, in contrast, *can* spontaneously change their processes. Theorizing software development, for example, as a defined sequence of phases is problematic because software is built by people who can decide to take different actions in a different order. This possibility of spontaneous variation is incommensurate with a lifecycle approach.

Put another way, mathematician George Polya modeled problem solving as a four-stage process: 1) understand the problem, 2) devise a plan, 3) carry out the plan, 4) check the result [46]. His work is not only a parsimonious lifecycle for mathematical problem-solving but also one of the clearest expositions of the planning model of human action [47]. However, Polya's model would make a terrible foundation for an SE process theory because, in many projects, people act without ever agreeing on a clear problem [48], expert designers do not separate thinking from doing [17], and most human action is better understood as improvisation rather than enacting a plan [47]. In short, lifecycle theories and improvisation are incompatible. Nitrogen does not improvise.

Moreover, the rise of Agile methods – specifically the principle of "responding to change over following a plan" [49] – reflects a growing understanding of the limitations of lifecycles. Unlike previous frameworks including the Rational Unified Process [50], Agile methods including Scrum [4] and Lean [5] present sets of principles, values and practices rather than steps, stages and phases.

Contrastingly, teleological, evolutionary and dialectic approaches do not make assumptions so misaligned with SE processes. Teleological theories posit agents (e.g. SE teams) who form goals (e.g. eliminate bugs). Evolutionary theories posit populations of competing entities (e.g. mobile applications, open source projects, design concepts). Dialectic theories posit conflict between groups with differing levels of power (e.g. developers vs. managers, producers vs. consumers). Consequently, of the four process theory approaches, lifecycle appears least suited to theorizing SE processes.

In summary, while lifecycle theories may explain deterministic processes (e.g. how compilers work), lifecycle theories of social processes are fundamentally flawed. No matter what phases and sequence we hypothesize, particular developers can choose to take different actions in a different order.

VI. How to Evaluate Process Theories

A. Compare Rival Theories

This section extends and clarifies previous guidance for empirically evaluating process theories, which emphasizes questionnaires and field studies [51], [52]. More recent work has explored process theory evaluation in communication [53] and decision-making [54].

Whether an evaluation method is appropriate for a theory depends on the kind of pattern the theory posits. For example, since the hypothesis "all swans are white" is existential rather than causal, experimental methods would be inappropriate. Process and variance theories claim different kinds of patterns. Therefore, they need different kinds of evaluation – both empirically and conceptually.

Randomized controlled trials are not appropriate for testing process theories. As process theories do not have independent or dependent variables, an experiment where the investigator manipulates the independent variable(s) and observes the dependent variable(s) while controlling everything else to establish a causal relationship does not make sense.

More generally, null hypothesis testing, where a hypothesis (e.g. X causes Y) is tested against a null hypothesis (e.g. X and Y are unrelated), is inappropriate for process theory testing. Null hypothesis testing is problematic in at least three ways.

1. *Null hypothesis testing is susceptible to confirmation bias.* "Confirmation bias means that information is searched for, interpreted, and remembered in such a way that it systematically impedes the possibility that the hypothesis could be rejected" [55, p. 79]. Confirmation bias is related to the tendency in questionnaire-based research, to unconsciously exploit response bias for positive results.
2. *The null hypothesis is often a straw man.* In this context, a straw man is a weak proposition presented only to be refuted. For example, the Technology Adoption Model hypothesizes that the *perceived usefulness* of a technology leads to *intention to use* it [56]. The null hypothesis – that perceived usefulness and intention to use are unrelated – is a straw man.
3. *Null hypothesis testing answers the wrong question.* In field studies, one will usually make some observations consistent with the theory (unless it is a straw man) and some observations incongruous with the theory. The key question is not "is the theory correct?" but "are we on the right track?" Null hypothesis testing does not tell us whether we are on the right track.

Consequently, Platt proposes the strong inference model, which simply emphasizes comparing several plausible alternative hypotheses, and argues that it leads to faster scientific progress [57]. Comparing rival theories is now commonly recommended for case study research [58] and some have even argued that theory testing is essentially comparative [59].

SCI, for example, posits that designers engage in sensemaking, i.e., giving meaning to an ambiguous situation. The null hypothesis, "situations are never ambiguous or designers do not make sense of them", is a straw man. Cherry-picking observations to support sensemaking would be straightforward. Moreover, a field study is likely to produce observations of both clear situations participants take for granted and ambiguous situations participants have to make sense of. The relevant question is not whether we can view some design activities as sensemaking, but whether sensemaking has more explanatory power than an alternative concept, e.g., analyzing. SCI was therefore tested against a rival theory, the Function-Behavior-Structure Framework [24], [60].

Selecting an appropriate rival may be challenging. If several potential rivals are evident, the choice is inherently subjective and reviewers may complain that the rival is arbitrary. While authors should attempt to justify their choice of rival theory; reviewers should not criticize this choice unless they can name and cite a different rival and give definitive reasons for its superior appropriateness. Imperfect rivals are often better than no rivals.

Comparative evaluation may also be augmented by team-based research. Having one team member seek evidence for Theory A while another seeks evidence for Theory B may ameliorate confirmation bias. The process of reconciling conflicting interpretations through discussion may furthermore spur more nuanced analysis and reflection.

In summary, comparing rival process theories is simply more tractable and defensible than null hypothesis testing. Both field students and questionnaire studies support comparing rival theories.

B. Use Observational Studies

Observational studies may provide extensive evidence with which to discriminate between rival theories. To be clear, while *interpretivist* field studies were recommended for theory *building*, *positivist* observational studies are recommended for theory *testing*.

Data collection depends on the specific approach. In a field study, data may include field notes, interview transcripts, copies of relevant documents, diagrams and email, segments of source code, code commit logs, screen shots, photographs of the physical environment and video of meetings or activities cf. [61]. In a lab-based simulation, data may include video of the simulation, copies of artifacts produced and the written or oral reflections of participants. In a think-aloud protocol study (where participants are asked to think aloud during a task) data would include the protocol transcripts (cf. [62], [63]). In a retrospective analysis, data would consist of previous case data (e.g. transcripts, documents) or case narratives (i.e. descriptions of cases written by researchers). In a discourse analysis, the data would consist of a corpus of relevant texts (e.g. journal articles, project plans). In all cases, observational studies may produce both quantitative data (e.g. bug counts, number of unit tests, number of code commits, time spent in meetings, word frequencies) and qualitative data (e.g. field notes, interview transcripts, software documentation).

For data analysis, researchers may choose between two basic strategies, or employ both serially or in parallel.

In one strategy, the researcher develops separate coding schemes for each theory (Table I). Each coding scheme lists all of the components (e.g. phases, activities, actors) and relationships (e.g. sequence, dependency) of the theory. The coding scheme provides space for evidence both for and against each component and relationship. It may be useful to brainstorm examples of each evidence category before data collection begins. Again, evidence may be qualitative or quantitative. The researcher organizes the data into this coding scheme and then weighs the evidence to reach a conclusion about which theory is better supported. While the evidence may better support the proposed theory or the rival theory, other conclusions are possible, e.g., that both theories are deficient or both are partly correct and can be merged. This strategy promotes thorough consideration of the data but is messy and time-consuming.

TABLE I. Coding Scheme Template (First Strategy)

Theory A		
Theory Component	***Evidence For***	***Evidence Against***
Concept 1		
Concept 2		
Relationship 1		
Relationship 2		
Theory B		
Theory Component	***Evidence For***	***Evidence Against***
Concept 1		
Concept 2		
Relationship 1		
Relationship 2		

TABLE II. Coding Scheme Template (Second Strategy)

Proposition	**Evidence for Theory A**	**Evidence for Theory B**
1		
2		
3		
4		

In the other strategy, the researcher first lists as many propositions as possible for both theories. The researcher then identifies the conflicting propositions, i.e., those where the two theories make contrasting predictions. Next, the researcher creates a coding scheme around the conflicting propositions (Table II). Again, brainstorming examples for each category before data collection begins may help. The researcher organizes the data into this coding scheme. For each proposition, the researcher decides which theory is favored by the evidence. Ideally, one theory is favored on most or all propositions and is therefore supported. This strategy relinquishes some degree of interpretive depth in favor of a cleaner, simpler analysis.

It is also possible to apply both strategies simultaneously. This may force the researcher to consider observations more deeply and from different angles. However, I personally found it challenging to conceptualize observations in terms of two different coding schemes, especially when events unfolded rapidly in situ.

In any case, it is both normal and desirable to adjust coding schemes as data collection progresses. As the researcher is

immersed in the observational context and begins classifying data, new insights may emerge, motivating refinements to the coding schemes.

C. Use Questionnaire Studies

Questionnaires may also be used to discriminate between rival theories.

Concerning data collection, sampling is hindered by the fact that SE researchers have no comprehensive lists of software developers, projects or companies. Two strategies are evident: 1) randomly sample from a specific community that does have a population list (e.g. Source Forge); 2) try to maximize the diversity of a convenience sample. Neither approach supports statistical generalization *to the population of interest.* Both approaches have benefits and drawbacks; reviewers should accept both as no superior option is available without a defensible population list. A more ambitious approach might utilize respondent-driven sampling [64] – a method of addressing bias in referral-chain (snowball) samples.

Two instrument development strategies (analogous to the two data analysis strategies above) are evident.

In one strategy, the questionnaire comprises both open-ended questions about participant's perceptions of the process and closed-ended questions about specific propositions of the theories. Examples of open-ended questions include "How are programming tasks distributed among team members in your current project?" and "please list all of the work activities (e.g. writing code, reading documentation) that you remember doing on your last day of work." Examples of closed-ended questions include "On a scale of 1 (strongly disagree) to 9 (strongly agree), to what extent do you agree with the following: I) our project goals were clear from the beginning; II) programming tasks sometimes require subject judgments; III) we often face impractical deadlines." Open-ended questions are designed to elicit predominately qualitative data, which can be classified using a coding scheme as in the first observational strategy / Table I. Closed-ended are designed to directly test specific theory propositions. For example, a median response of 7 (agree) to "Our project goals were clear from the beginning" would be evidence against the sensemaking proposition of SCI. Adopting this strategy may produce comparatively more data per respondent; however, the length of the instrument may reduce response rates.

In the other strategy, the researcher again identifies conflicting propositions. The researcher then generates numerous questions that reflect these differences. Specifically, one can generate bipolar scales (including Likert and semantic differential scales) where one pole is consistent with Theory A and the other pole is consistent with Theory B, e.g., *The goals of my current project were...*

a) ... completely provided by the client (Rival Theory)
b) … mostly provided by the client
c) ... determined equally by the client and development team
d) ... mostly determined by the development team
e) *... completely determined by the development team (SCI)*

Validating these kinds of questionnaires fundamentally differs from validating a variance theory testing questionnaire. In social science, variance theories usually posit causal relationships between constructs. A construct is a postulated (often psychometric) variable that cannot be measured directly, e.g., trust, extroversion, team cohesion, software quality. A variable is a quantity that may have different values. Therefore, the researcher generates several questionnaire *items* (questions) intended to *reflect* each construct. To analyze validity, the researcher pilots the questionnaire and analyzes the correlations between the items. Items that reflect the same construct should be highly correlated with each other (convergent validity) and less correlated to items reflecting other constructs (discriminate validity) [65].

Running the survey will produce a response distribution for each question; e.g., a) 5; b) 10; c) 20; d) 30; e) 15. In this example, visual inspection suggest the distribution favors SCI. To determine statistical significance we can use non-parametric tests including Chi-Square Test and Kolmogorov-Smirnov. These tests require an *expected distribution,* three options for which are:

1) uniform distribution, e.g., 20, 20, 20, 20, 20
2) pseudo-normal distribution, e.g., 6, 19, 30, 9, 6
3) reflected distribution, e.g., 15, 30, 20, 10, 5

Using a uniform distribution makes little sense as it does not answer the question at hand. Using the pseudo-normal distribution addresses the question, *is the extent of skew in the observed distribution significant?* However, treating discontinuous, ordinal data as normally distributed is statistically problematic. The reflected distribution is the most defensible as it addresses the question, *is the observed distribution significantly different from an equally compelling distribution supporting the rival theory?* and reflecting a distribution does not obviously violate statistical norms (cf. [24] for a detailed example of this approach).

Process theories do not necessarily have constructs. Sensemaking and coevolution [19] are sub-processes. The problem space and the solution space [20] are populations of ideas. "Action to improve software process" and "action to develop software products" [25] are categories of activities. None of these process theory elements are constructs because none of them can take on quantities. Similarly, rival propositions, e.g., *goals are given by stakeholders / constructed by developers*, are not constructs. When researchers devise several questions about a process theory proposition, the answers are variables but these variables are not reflective indicators of the same underlying construct. Without *a priori* reasons to believe that items associated with one theory difference will be more closely related to each other than to items associated with other differences, convergent/discriminate validity analysis does not apply. Instead, process theory testing questionnaires may be validated using pilot studies with both experts and members of the target population. Researchers should request question-by-question feedback from pilot participants

– some familiar with the purpose of the study, others not (see [24], [66] for a detailed example).

D. Multi-Methodological and Team Approaches

Process theories may also be evaluated using a multi-methodological approach. Researchers can combine questionnaire studies with case studies or other observational approaches to achieve good breadth and depth simultaneously. One way to approach the data analysis is to analyze the two theories separately in the observational study (which promotes deeper analysis) and focus on conflicting propositions in the questionnaire-based study (to keep the instrument short and simple). While other configurations may be preferable in some contexts, this one appears to make the most of each study type – see [66] for a detailed example.

Combining case studies and questionnaire studies in this way overcomes many of the limitations of each individually. Observational studies benefit from extensive data collection and triangulation; however, the number of organizations studied is limited. Questionnaire studies, in contrast, produce more shallow data but can reach a large number or organizations. Observational studies also produce real-time observations, which are not subject to the participant memory effects that threaten questionnaire studies. While both questionnaire and observational approaches are valid on their own, combining one or more observational studies with a questionnaire study facilitates greater data triangulation, mitigates mono-method bias and generally encourages more nuanced reflection.

Improved rigor notwithstanding, multi-methodological research is much more difficult to present. Presenting each individual case, the cross case analysis, the questionnaire results and the case study / questionnaire triangulation will consume many pages. It may be better to omit individual case analyses and provide only the cross case analysis in a question-and-answer format [61]. Reviewers should accept this as a necessary abbreviation. Reviewers should more generally give credit for the added robustness of multimethodological research.

E. Conceptual Evaluation

Process theories may also be evaluated conceptually on numerous dimensions including testability, usefulness, simplicity and communicability. Regarding testability, researchers do not categorically prove or disprove theories [59]. However, if we cannot imagine any specific observation that would constitute evidence against a theory, it may be tautological. While a process theory may not be directly useful to software developers in general, it may be useful to those who develop SE tools and practices or to researchers studying SE phenomena. Similarly, while a process theory should not *over*simplify it should not be so complicated that few people can understand it – a charge sometimes leveled against the Situated Function-Behavior-Structure Framework [67] for example. Finally, process theories obviously should be communicable. Authors can demonstrate communicability by providing clear definitions of their theory's elements, possibly in tabular form. Giving examples of the theory's applications and discussing its implications also helps to demonstrate communicability.

Moreover, some process theories may be evaluated by analogy to similar kinds of artifacts with well-defined evaluation criteria. For example, a teleological theory posits an agent, pursuing goals through activities. However, the "activities" posited by a teleological theory may actually be categories of activities, making the theory in part a categorization system. A good category is one that supports specific inferences about its members [68]. Consequently, it may be instructive to apply the supports-inferences criteria to evaluate a teleological theory.

VII. Discussion and Conclusion

Section III listed some topics that might benefit from a process theory approach. The following discussion suggests how some of these process may be theorized.

For example, we know that designers simultaneously reframe problems and design artifacts to address them – a process called coevolution [69]. However, we do not know exactly how the coevolution process unfolds in SE. Because coevolution involves the interaction between entities (e.g. the developers, solution conjectures, problem frames), it a dialectical process theory may be appropriate.

Similarly, while many believe that agility is important, we lack a clear understanding what agility is and how the agility, the team property, differs from ostensibly Agile methods (cf. [70]). Since agility involves the interaction between the need for change and responses for change, a dialectical process theory of how developers experience change might help explain the role of agility in SE projects.

Open source projects, meanwhile depend on networks of contributors to implement features, improve quality and gain market share [71]. A process theory may help explain how some projects succeed while others stagnate. Because so many open source projects compete for attention among developers and users, an evolutionary approach may be appropriate.

Meanwhile, my own research predominately concerns process theories of software engineering in general [19], [24], [66]. As most software is developed by human agents, a teleological approach has been helpful.

From my own limited experience, the greatest barriers to process theory research in SE are 1) many SE academics are unfamiliar with process theory; 2) little SE-specific methodological guidance is available; 3) reviewers incorrectly evaluate process theory research using the criteria and norms of variance theory research; 4) reviewers evaluate process theory papers based on tone and argumentum ad hominem rather than methodological rigor.

This paper therefore describes process theories and synthesizes the methodological concepts I have acquired and developed through working with process theories. Its core contribution is its exploration of observational, questionnaire and multimethodological empirical studies as they pertain to process theories. I am not aware of any previous research, in SE or

elsewhere, which describe the coding-scheme / instrument development approaches for comparing rival process theories contained here.

Despite their many benefits, however, process theories are intrinsically limited in several ways. Lifecycle theories encourage us to oversimplify and over-rationalize complex human behaviors. Teleological and dialectical theories help us understand but rarely produce useful predictions. Evolutionary theories may predict (probabilistically) success and failure; however, quantifying the fitness function may be difficult.

This paper also has several limitations. It does not present a taxonomy of quality dimensions for process theory studies, such as internal validity, external validity, reliability and objectivity [72] or credibility, transferability, dependability and confirmability [73]. This is an intentional choice following Rolfe's [74] convincing argument that research quality cannot be reliably ascertained from a pre-ordained set of criteria. Furthermore, this paper is intended to share a conceptualization of process theory and ideas that seemed useful to the author in his own research. Evaluating such guidance requires other experienced researchers to apply it in diverse contexts. Future work may therefore include reflections, refinements or refutations from other researchers who use or disregard the above guidance in process theory studies. Moreover, my own research primarily concerns teleological and lifecycle theories, and the tension between the two. The discussion of evolutionary and dialectic theories is therefore less developed and future work may include better treatment of these theory types. Future work should moreover address SE's sampling problem, i.e., the lack of population lists from which to randomly sample and the tendency for reviewers to capriciously reject some papers for convenience sampling despite the absence of any strongly defensible representative samples in our literature.

These limitations notwithstanding, the Software Engineering academic discipline is fundamentally concerned (among other things) with the process of developing software. A process theory explains how a process unfolds. Consequently, process theories may be useful for capturing and reasoning about scientific accounts of software processes. SE needs explanatory process theories to counterbalance more prescriptive methods. The lack of SE process theory research obstructs scientific consensus by focusing the academic community on methods, which inevitably oversimplify and over-rationalize complex and unpredictable human phenomena [15], [16]. Specifically, SE needs more evolutionary, dialectic and teleological process theories to overcome the tendency to oversimplify and over-rationalize human processes into lifecycles.

Finally, this work may be particularly useful for developing general theories of software engineering. As such theories should include process theory elements [10], [11], [13], the lack of guidance for and acceptance of process theory research has perhaps hindered their development.

ACKNOWLEDGEMENT

Thanks are due to Marshall Scott Pool and the participants of the 2014 SEMAT Workshop on General Theory of Software Engineering for their constructive feedback on an early version of this paper.

REFERENCES

[1] A. H. Van de Ven, *Engaged scholarship: a guide for organizational and social research*. Oxford, UK: Oxford University Press, 2007.

[2] A. H. Van de Ven and M. S. Poole, "Explaining development and change in organizations," *The Academy of Management Review*, vol. 20, no. 3, pp. 510–540, 1995.

[3] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quarterly*, vol. 27, no. 3, pp. 425–478, 2003.

[4] K. Schwaber and J. Sutherland, "The Scrum Guide," 2010. [Online]. Available: http://www.scrum.org/scrumguides/. [Accessed: 02-Jul-2012].

[5] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003.

[6] K. Forsberg and H. Mooz, "The Relationship of System Engineering to the Project Cycle," *Engineering Management Journal,*, vol. 4, no. 3, pp. 36–43, 1992.

[7] T. Allweyer, BPMN 2.0- Business Process Model and Notation. OMG Standard formal/2011-01-03, 2011.

[8] B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75–90, 1992.

[9] P. Johnson, M. Ekstedt, and I. Jacobson, "Where's the Theory for Software Engineering?," *IEEE Software*, vol. 29, no. 5, pp. 94–96, 2012.

[10] P. Ralph, P. Johnson, and H. Jordan, "Report on the First SEMAT Workshop on a General Theory of Software Engineering (GTSE 2012)," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 2, pp. 26–28, 2013.

[11] P. Johnson, P. Ralph, M. Goedicke, P.-W. Ng, K.-J. Stol, K. Smolander, I. Exman, and D. E. Perry, "Report on the Second SEMAT Workshop on General Theory of Software Engineering (GTSE 2013)," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 47–50, 2013.

[12] P. Ralph, "Possible core theories for software engineering," in *Proceedings of the 2nd SEMAT Workshop on a General Theory of Software Engineering*, San Francisco, CA, USA, IEEE, 2013, pp. 35–38.

[13] P. Ralph, I. Exman, P.-W. Ng, P. Johnson, M. Goedicke, A. T. Kocatas, and K. L. Yan, "How to Develop a General Theory of Software Engineering: Report on the GTSE 2014 Workshop," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 6, pp. 23–25, 2014.

[14] J. Hannay, D. I. K. I. K. Sjøberg, and T. Dyba, "A Systematic Review of Theory Use in Software Engineering Experiments," *IEEE Transaction on Software Engineering*, vol. 33, no. 2, pp. 87–107, Feb. 2007.

[15] F. P. Brooks, *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley Professional, 2010.

[16] M. W. Lewis, "Exploring Paradox: Toward a More Comprehensive Guide," *The Academy of Management Review*, vol. 25, no. 4, pp. 760–776, Jan. 2000.

[17] D. A. Schön, The reflective practitioner: how professionals think in action. USA: Basic Books, 1983.

[18] P. Ralph, "The Illusion of Requirements in Software Development," *Requirements Engineering*, vol. 18, no. 3, pp. 293–296, 2013.

[19] P. Ralph, "The Sensemaking-Coevolution-Implementation Theory of Software Design," *Science of Computer Programming*, vol. 101, pp. 21-41, 2015.

[20] M. Maher, J. Poon, and S. Boulanger, "Formalising design exploration as co-evolution: A combined gene approach," *Preprints of the Second IFIP WG5.2 Workshop on Advances in Formal Design Methods for CAD*, pp. 1–28, 1995.

[21] P. Palvia and J. Nosek, "An empirical evaluation of system development methodologies," *Information Resource Management Journal*, vol. 3, no. 3, pp. 23–32, 1990.

[22] P. Ralph, "Introducing an Empirical Model of Design," in *Proceedings of The 6th Mediterranean Conference on Information Systems*, Limassol, Cyprus: AIS, 2011.

[23] P. Ralph, "Improving coverage of design in information systems education," in *Proceedings of the 2012 International Conference on Information Systems*, Orlando, FL, USA: AIS, 2012.

[24] P. Ralph, "Comparing Two Software Design Process Theories," in *Global Perspectives on Design Science Research: Proceedings of the 5th International Conference, DESRIST 2010*, LNCS 6105, Springer, 2010, pp. 139–153.

[25] I. Allison and Y. Merali, "Software process improvement as emergent change: A structurational analysis," *Information and Software Technology*, vol. 49, no. 6, pp. 668–681, Jun. 2007.

[26] W. Royce, "Managing the development of large software systems," in *Proceedings of WESCON*, Los Angeles, USA: IEEE, 1970, pp. 1–9.

[27] J. Kim, "Causation," in *The Cambridge Dictionary of Philosophy*, 2nd ed., R. Audi, Ed. Cambridge, UK: Cambridge University Press, 1999, pp. 125–127.

[28] D. P. Truex, R. Baskerville, and J. Travis, "Amethodical systems development: the deferred meaning of systems development methods," *Accounting, Management and Information Technologies*, vol. 10, no. 1, pp. 53–79, 2000.

[29] J. S. Dyer, P. C. Fishburn, R. E. Steuer, J. Wallenius, and S. Zionts, "Multiple Criteria Decision Making, Multiattribute Utility Theory: The Next Ten Years," *Management Science*, vol. 38, no. 5, pp. p645–654, May 1992.

[30] C. W. Alexander, *Notes on the synthesis of form*. Harvard University Press, 1964.

[31] J. Saldana, *The Coding Manual for Qualitative Researchers*. SAGE Publications, 2012.

[32] W. Stacy and J. MacMillan, "Cognitive bias in software engineering," *Communications of the ACM*, vol. 38, no. 6, pp. 57–63, 1995.

[33] P. Ralph, "Toward a Theory of Debiasing Software Development," in *Research in Systems Analysis and Design: Models and Methods: 4th SIGSAND/PLAIS EuroSymposium 2011*, no. 93, S. Wrycza, Ed. Gdansk, Poland: Springer, 2011, pp. 92–105.

[34] D. Arnott, "Cognitive biases and decision support systems development: a design science approach," *Information Systems Journal*, vol. 16, no. 1, pp. 55–78, 2006.

[35] J. Parsons and C. Saunders, "Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse," *IEEE Transaction on Software Engineering*, vol. 30, pp. 873–888, 2004.

[36] B. G. Glaser and A. L. Strauss, *The Discovery of grounded theory: Strategies for qualitative research*. Chicago: Aldine Pub. Co, 1967.

[37] J. Mills, A. Bonner, and K. France, "The Development of Constructivist Grounded Theory," *International Journal of Qualitative Methods*, vol. 5, no. 1, 2006.

[38] P. Ralph, *Fundamentals of Software Design Science*, PhD Dissertation, University of British Columbia, Vancouver, Canada, 2010.

[39] P. E. Vermaas and K. Dorst, "On the Conceptual Framework of John Gero's FBS-Model and the Prescriptive Aims of Design Methodology," *Design Studies*, vol. 28, no. 2, pp. 133–157, 2007.

[40] B. Fischoff, "Debiasing," in *Judgment Under Uncertainty: Heuristics and Biases*, no. 31, D. Kahneman, P. Slovic, and A. Tversky, Eds. Cambridge, USA: Cambridge University Press, 1982.

[41] R. F. Pohl, Ed., *Cognitive Illusions*. East Sussex, UK: Psychology Press, 2004.

[42] J. T. Jost and R. Andrews, "System Justification Theory," in *The Encyclopedia of Peace Psychology*, Blackwell Publishing Ltd, 2011.

[43] J. T. Jost, M. R. Banaji, and B. A. Nosek, "A Decade of System Justification Theory: Accumulated Evidence of Conscious and Unconscious Bolstering of the Status Quo," *Political Psychology*, vol. 25, no. 6, pp. 881–919, 2004.

[44] M. Kennaley, *SDLC 3.0: Beyond a Tacit Understanding of Agile*, vol. 1. Fourth Medium Press, 2010.

[45] B. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, May 1988.

[46] G. Polya, *How to Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, New Jersey, USA: Princeton University Press, 1957.

[47] L. Suchman, *Human-machine reconfigurations: Plans and situated actions*. Cambridge University Press, 2007.

[48] P. Checkland, *Systems Thinking, Systems Practice*. Chichester: Wiley, 1999.

[49] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001. [Online]. Available: http://www.agilemanifesto.org/. [Accessed: 02-Jul-2012].

[50] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley Professional, 2003.

[51] R. A. Wolfe, "Organizational innovation: review, critique and suggested research directions," *Journal of Management Studies*, vol. 31, no. 3, pp. 405–431, 1994.

[52] M. Poole, A. H. Van de Ven, K. Dooley, and M. E. Holmes, *Organizational change and innovation processes theory and methods for research*. New York, NY, USA: Oxford University Press, 2000.

[53] M. S. Poole, "On the Study of Process in Communication Research," in *Communication Yearbook 36*, C. T. Salmon, Ed. New York, USA: Routledge, 2012, pp. 371–409.

[54] M. S. Poole and A. H. Van de Ven, "Empirical Methods for Research on Organizational Decision-Making Processes," in *The Blackwell Handbook of Decision Making*, Nutt, P. C. and D. Wilson, Eds. Oxford: Blackwell, 2010, pp. 543–580.

[55] M. E. Oswald and S. Grosjean, "Confirmation Bias," in *Cognitive Illusions: A Handbook on Fallacies and Biases in Thinking, Judgement and Memory*, no. 4, R. F. Pohl, Ed. Hove, UK: Psychology Press, 2004, pp. 79–96.

[56] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, Sep. 1989.

[57] J. R. Platt, "Strong inference," *Science*, vol. 146, no. 3642, 1964.

[58] L. Dube and G. Pare, "Rigor in information systems positivist case research: Current practices, trends and recommendations," *MIS Quarterly*, vol. 27, no. 4, pp. 597–635, Dec. 2003.

[59] E. Sober, "Testability," Proceedings and Addresses of the American Philosophical Association, vol. 73, no. 2, pp. 47–76, 1999.

[60] J. S. Gero, "Design prototypes: A knowledge representation schema for design," *AI Magazine*, vol. 11, no. 4, pp. 26–36, 1990.

[61] R. K. Yin, *Case study research: Design and methods*, 4 ed. California, USA: Sage, 2008.

[62] M. E. Fonteyn, B. Kuipers, and S. J. Grobe, "A Description of Think Aloud Method and Protocol Analysis," *Qualitative Health Research*, vol. 3, no. 4, pp. 430–441, Nov. 1993.

[63] K. Dorst and J. Dijkhuis, "Comparing Paradigms for Describing Design Activity," *Design Studies*, vol. 16, no. 2, pp. 261–274, 1995.

[64] D. D. Heckathorn, "Respondent-Driven Sampling: A New Approach to the Study of Hidden Populations," *Social Problems*, vol. 44, no. 2, pp. 174–199, 1997.

[65] D. T. Campbell and D. W. Fiske, "Convergent and discriminant validation by the multitrait-multimethod matrix," *Psychological Bulletin*, vol. 56, no. 2, pp. 81–105, 1959.

[66] P. Ralph, "Software Engineering Process Theory: A Multi-Method Comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory," *arXiv [cs.SE]*: 1307.1019.

[67] J. S. Gero and U. Kannengiesser, "The Situated Function-Behaviour-Structure Framework," *Design Studies*, vol. 25, no. 4, pp. 373–391, 2004.

[68] J. Parsons and Y. Wand, "A question of class," *Nature*, vol. 455, no. 7216, pp. 1040–1041, Oct. 2008.

[69] K. Dorst and N. Cross, "Creativity in the design process: co-evolution of problem–solution," *Design Studies*, vol. 22, no. 5, pp. 425–437, Sep. 2001.

[70] K. Conboy, "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Information Systems Research*, vol. 20, no. 3, pp. 329–354, 2009.

[71] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 95–105, 2006.

[72] W. Trochim, *Research Methods Knowledge Base*. Cincinnati, OH, USA: Atomic Dog Publishing, 2001.

[73] Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry*. SAGE Publications, 1985.

[74] G. Rolfe, "Validity, trustworthiness and rigour: quality and the idea of qualitative research," *J Adv Nurs*, vol. 53, no. 3, pp. 304–310, Feb. 2006.