Software engineering process theory:
A multi-method comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory

Paul Ralph[a]

[a]*Department of Computer Science, University of Auckland, Auckland, 1142, New Zealand; paul@paulralph.name*

A B S T R A C T

Context: Software engineering has experienced increased calls for attention to theory, including process theory and general theory. However, few process theories or potential general theories have been proposed and little empirical evaluation has been attempted.
Objective: The purpose of this paper is to empirically evaluate two previously untested software development process theories – Sensemaking-Coevolution-Implementation Theory (SCI) and The Function-Behavior-Structure Framework (FBS).
Method: A survey of more than 1300 software developers is combined with four longitudinal, positivist case studies to achieve a simultaneously broad and deep empirical evaluation. Instrument development, statistical analysis of questionnaire data, case data analysis using a closed-ended, a priori coding scheme and data triangulation are described.
Results: Case data analysis strongly supports SCI, as does analysis of questionnaire response distributions ($p<0.001$; chi-square goodness of fit test). Furthermore, case-questionnaire triangulation found no evidence that support for SCI varied by participants' gender, education, experience, nationality or the size or nature of their projects.
Conclusions: SCI is supported. No evidence of an FBS subculture was found. This suggests that instead of iterating between weakly-coupled phases (analysis, design, coding, testing), it is more accurate and useful to conceptualize development as ad hoc oscillation between making sense of the project context (Sensemaking), simultaneously improving mental representations of the context and design space (Coevolution) and constructing, debugging and deploying software artifacts (Implementation).

Keywords: Process Theory, Software Process, Case Study, Questionnaire

1. Introduction

The Software Engineering (SE) field has witnessed increasing calls to theorize about its core concepts and processes (e.g. [1-7]). However, SE remains preoccupied with normative research on software development methods, methodologies and process models [8] and characterized by "a lack of interest in theories aimed at *understanding* and *explaining* the how and why of the observed design activities" in favor of "a rush from observation and description to prescriptive modelling and the construction of design tools" [9].

Building and empirically evaluating SE theories has many benefits. Theories synthesize, preserve and communicate empirical knowledge, thereby implicitly coordinating future inquiry. Unlike method and tool knowledge, theories withstand fashions and fads. Adopting a theoretical mindset furthermore implicitly refocuses researchers on fundamental rather than superficial features of SE.

A theory is simply a collection of interconnected concepts. Theories have differing purposes including *to describe*, *to explain*, *to analyze* and *to predict* [10] and units of analysis including *individual*, *group*, *process*, *organization* and *industry* [11]. Middle range theories apply to specific

empirical phenomena while general theories apply to a broad class of phenomena. Variance theories focus on *why* events occur while process theories focus on *how* events occur [12]. Variance theories employ different approaches to causation including *regularity* (Y always follows X), *counterfactual* (Y cannot occur without X), *probabilistic* (Y is more likely given X), and *teleological* (X, an agent with free will, choses to do Y) [13]. Meanwhile, process theories may approximate one of several "ideal types" – *lifecycle* (a sequence of phases), *evolution* (a population of competing, reproducing elements), *dialectic* (struggle between several entities with varying power) and *teleological* (goal-oriented, self-directed actions of an autonomous agent) [14]. Given the diversity of possible theoretical approaches, deeply understanding sociotechnical phenomena including software development necessitates numerous theoretical perspectives [15,16].

Following Brooks' [17] insightful elucidation of the fundamental confusion surrounding the software development process, this paper focuses on software development process theory. Specifically, it aims to empirically evaluate Sensemaking-Coevolution-Implementation Theory (SCI), which diverges from traditional engineering thinking to explain more accurately how software is developed in practice [18]. SCI is evaluated against a rival theory, The Function-Behavior-Structure Framework (FBS), which expresses a more traditional view of the development process [19,20]. The paper presents an extensive, multi-method, empirical initiative to evaluate these two theories, driven by the following research question.

> ***Research Question:*** *Does Sensemaking-Coevolution-Implementation Theory better explain how teams develop complex software systems in practice than The Function-Behavior-Structure Framework?*

Here a *complex* system is a collection of interconnected elements that exhibits behaviors not predictable from those elements [21]. Focusing on complex systems excludes routine re-implementation of well-understood artifacts. Meanwhile, software development here "encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" [22]. This paper furthermore focuses on development by individuals or coordinated teams predominately working together, rather than projects involving mass-collaboration, hostile teams working at cross purposes or multiple autonomous teams. Additionally, it primarily concerns direct actions of development teams, rather than indirect actions and related concepts including project management, politics, power and time.

Section two discusses process theory in SE, including detailed presentations of FBS and SCI. Section three presents the multi-methodological research design. Section four summarizes the results and section five discusses the study's limitations and implications. Related research is discussed throughout.

2. Software Engineering Process Theories

While a comprehensive review of theories used in SE is beyond the scope of this paper, Hannay et al. [4] identified 40 theories that were experimentally evaluated in studies published between 1993 and 2002. However, only two of these were used in more than one article: 1) the Theory of Cognitive Fit, which posits that the alignment between a task and the presentation of information needed for the task affects task performance [23,24], and 2) the theory that reading techniques affect software inspection effectiveness [25-27].

In the following decade, empirical research continued gaining prominence in SE, with, for example, the ISESE and METRICS symposia, followed by the Empirical Software Engineering and Measurement conference. However, most empirical work in SE continues either to evaluate specific

tools and techniques (e.g. bug prediction approaches [28]) or to investigate specific SE phenomena (e.g. source code clone maintenance [29]). Similarly, most SE theories concern specific SE activities (e.g. search-based testing [30]; visual notation [31]). Meanwhile, little theoretical and empirical work investigates the software development process holistically. Software process research is predominately prescriptive and method-focused [8]. This has produced more than one thousand software development methods [32], some of which (e.g. the Waterfall Model [33], Spiral Model [34], Axiomatic Design [35]) are sporadically treated as theories. For example, statements like "in conventional software development, the development lifecycle in its most generic form comprises four broad phases: planning, analysis, design, and implementation" [36], treat Waterfall as a theory.

However, methods are not appropriate foundations for process theories [9]. Methods prescribe ostensibly good approaches to an activity. Process theories, in contrast, encompass both good and bad approaches by explaining the fundamental properties of an activity. Therefore, this section focuses on process theories, not methods or prescriptions.

A recent review [37] found no software development process theories other than SCI. However, it found four process theories which *could* apply to software – The Basic Design Cycle [38], The Problem-Design Exploration Model [39], The Self-Conscious Process [40] and The Function-Behavior-Structure Framework (FBS) [19]. Adopting one of these as a rival theory facilitates a more rigorous evaluation (see below). The Problem-Design Exploration Model is a poor choice as it is intended to explain design using genetic algorithms while SCI is intended to explain how human teams develop software. As SCI is partially based on The Selfconscious Process, the latter would provide too weak a rival. The Basic Design Cycle, meanwhile, is a special case of FBS, so FBS is a stronger rival due to its greater explanatory power. Moreover, FBS has been conceptually – although not empirically – applied to SE [41,42]. Consequently, FBS is the best choice for rival theory. This section therefore reviews and conceptually evaluates SCI and FBS.

2.1. Sensemaking-Coevolution-Implementation Theory

SCI (Figure 1; Table 1) posits that complex software systems are produced by an agent (individual or team) that alternates between *sensemaking*, *coevolution* and *implementation* in a self-determined sequence [18].

Sensemaking refers to perceiving, assigning meaning to and organizing beliefs about a phenomenon or experience [43-45]. In SE, then, it may include interviewing stakeholders, writing notes, organizing notes, reading about the domain, reading about technologies that could be used in the project and sharing insights among team members. Sensemaking also includes problem framing [46] and the kinds of testing that are closely related to understanding the context (e.g. acceptance testing, usability studies).

Coevolution refers to a situation where two or more interconnected objects develop and change over time such that changes in one trigger changes in the other and vice versa. Meanwhile, a *schema* is "a mental representation of some aspect of experience, based on prior experience and memory, structured in such a way as to facilitate (and sometimes to distort) perception, cognition, the drawing of inferences, or the interpretation of new information in terms of existing knowledge" [47]. In SCI, and the interdisciplinary design literature more generally, Coevolution refers to mutually exploring and refining the design agent's schemas of the project context and design space [48-50]. Coevolution may occur in planning meetings and design meetings, following breakdowns or during an individual's internal reflection. It is especially evident when a team stands around a whiteboard drawing informal models and discussing how to proceed.

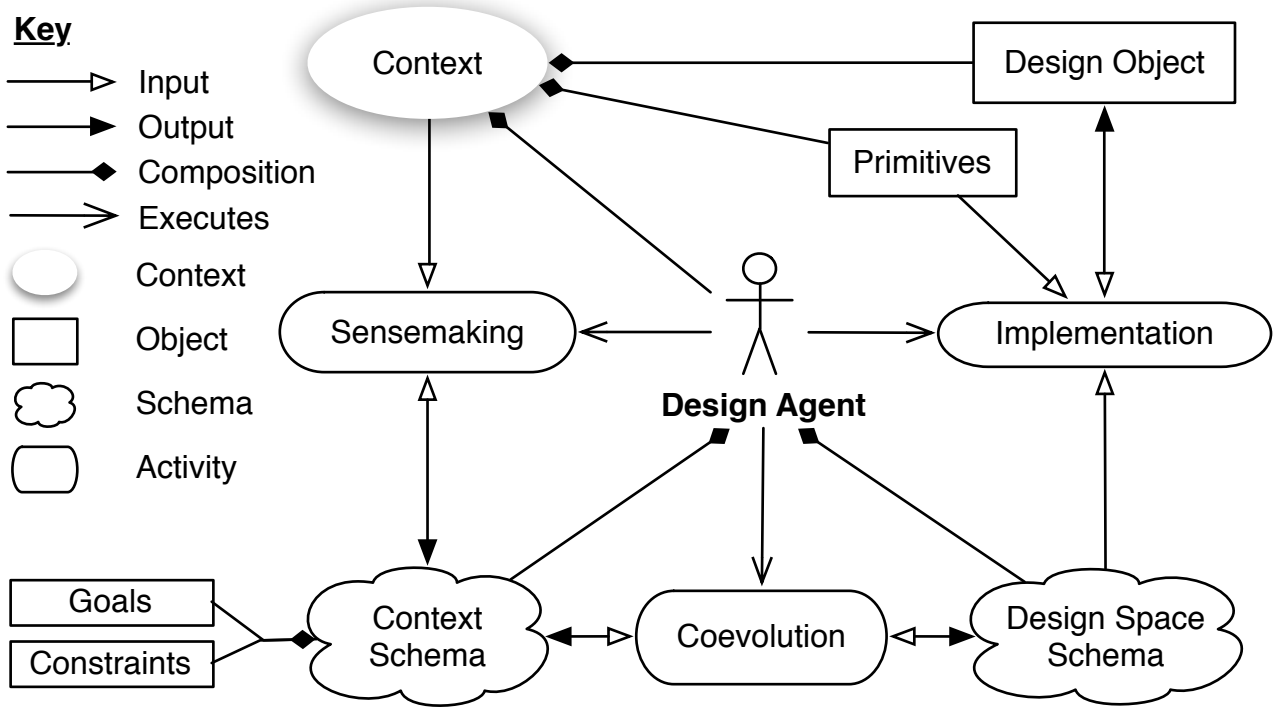

Fig. 1. Sensemaking-Coevolution-Implementation Theory (adapted from [18])

Table 1
SCI concepts defined (adapted from [18]).

Concept	Meaning
Coevolution	the process where the design agent simultaneously refines its design space and context schemas
Constraints	the designers beliefs about the restrictions on the design object's properties
Context	the totality of the surroundings of the design object and agent, including the object's intended domain of deployment
Context Schema	the collection of all of the design agent's beliefs about the context; the design agent's mental picture of the context.
Design Agent	an entity or group of entities capable of forming intentions and goals and taking actions to achieve those goals and that specifies the structural properties of the design object
Design Object	the thing being designed; also any partial or complete manifestation thereof
Design Space Schema	the collection of all of the design agent's beliefs about the diversity of possible design artifacts, including design decisions and candidate solutions; the design agent's mental picture of the design object and its alternatives
Goals	optative statements about the effects the design object should have on its environment
Implementation	the process where the design agent generates or updates the design object based on its design space schema
Primitives	the set of entities from which the design object may be composed
Sensemaking	the process where the design agent organizes and assigns meaning to its perception of the context, creating and refining context schema

For example, suppose a professor gives regular quizzes to evaluate students (design schema), but students complain that feedback on quizzes is taking too long (context schema). The designer imagines giving online quizzes with automated marking and instant feedback (design schema). However, the professor complains that online quizzes are too difficult to police for formal evaluation. They quickly realize that online quizzes are better for encouraging preparation (context schema). This suggests numerous changes to quiz structure, such as giving students multiple attempts (design space), that will lead to high quiz marks. This triggers another reframing – automated quizzes can be used to replace participation marks (context schema). In this way, changes to the context schema trigger new design directions (e.g. automated quizzes, multiple attempts) and exploring the design space triggers reconceptualization of the context schema (e.g. encouraging instead of evaluating, quiz marks as participation marks).

Implementation (i.e. building and deploying the software) may include coding, managing the codebase, writing documentation, refactoring, integration and deployment. Implementation also includes the kinds of testing that are closely related to coding; for instance, debugging compilation errors, static code analysis, unit testing, integration testing and system testing. In SCI, Implementation involves realizing aspects of the design space schema by creating or modifying artifacts.

SCI posits that the development team (design agent) faces an evolving, ambiguous context comprising stakeholders who may have different values, may disagree on project aims or may be unable to articulate objectives and constraints clearly. The team has to make sense of this problematic context (generating a context schema). Since the project environment continuously evolves, Sensemaking is an ongoing process. The team uses its evolving context schema to explore the design space, make decisions and generate and conceptually evaluate alternative solution concepts (the design space schema). Exploring the design space triggers reconceptualization of the context (changing the context schema). Meanwhile, the team creates and modifies software and related artifacts to realize aspects of the design space schema. A design agent can switch between Sensemaking, Coevolution and Implementation at will. When the design agent is a team, different team members may engage in different activities simultaneously. However, the three activities are intended to be mutually exclusive, so one person should not be able to engage in, for example, Sensemaking and Implementation, simultaneously.

Whereas much previous work suggests that design is intrinsically iterative (e.g. [51]), SCI differentiates between two concentric iterative loops (Figure 2). The inner loop, Coevolution, denotes cognitive oscillation between ideas or iterating between framing the problem and modifying solution conjectures. The outer loop involves making sense of the context, coevolution and modifying software artifacts, which alter the context and trigger more Sensemaking, etc. The inner loop typically takes minutes or hours while the outer loop typically takes weeks or months [18].

2.2. The Function-Behavior-Structure Framework

FBS (Figure 3; Tables 2 and 3) posits that engineers design systems by manipulating three kinds of models (i.e. abstract descriptions): function models (F) describe system goals, behavior models describe system requirements (B_e) and predicted behavior (B_S) while structure models (S) describe system components and their connections. The designer aims to transforms the function (goal) model into a design description (D), which is sufficiently detailed to permit construction of the system. Gero and Kannengiesser [20] extended FBS to elaborate how each model may have different versions in different "worlds", for example, the goals that the system would ideally achieve vs. the goals it is realistically expected to achieve. While FBS was intended initially to explain engineering design, it has also been applied to software development (cf. [41,52]).

FBS posits that the designer is given a well-defined problem with clear, agreed aims and objectives (F). F could take many forms including a contract, a goal model or a project mandate document. The designer first checks whether there exists a ready-made solution (e.g. commercial-off-the-shelf software) that would solve the problem. If not, the designer *formulates* a requirements model (B_e) from F. Again the requirements model could have different forms including a set of use cases or an IEEE-830 style list of "The system shall…" statements. The designer then combines B_e with knowledge and experience to *synthesize* a structure (S) as, for instance, a UML class diagram. Sometimes obvious problems with S trigger immediate structural reformulation. The designer then *analyzes* S by visual inspection or mathematical simulation to predict the behavior of the proposed software (B_s) and *evaluates* B_s against the requirements (B_e).

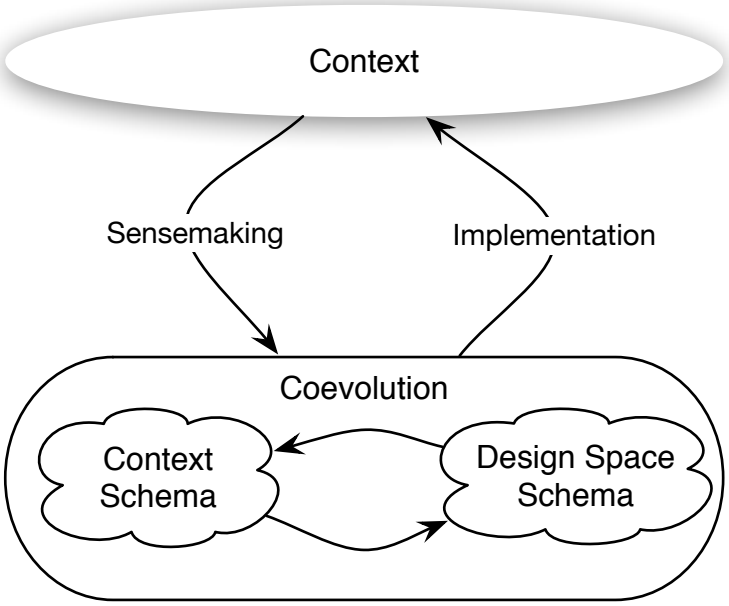

Fig. 2. Concentric Iterative Loops

If predicted behavior is not close enough to required behavior, the designer has several options. One option is to close the gap between predicted and required behavior by iterating between synthesis, analysis and evaluation. Alternatively, the designer may determine that the requirements or goals are unfeasible, and choose to *reformulate* (revise) one or both artifacts. Eventually, when the structure appears satisfactory, the designer fleshes out its details and passes on the complete design documentation (D) to programmers.

All of the artifacts in FBS are symbolic representations [9,53]. S is a structural model (e.g. a class diagram) while D is a design document. Neither S nor D is source code. Moreover, FBS does not include actually interactions between the project environment and design process (other than a one-time passing of the problem-specification, F) or between the design process and coding (other than a one-time passing of the design documentation, D).

2.3. Conceptual Evaluation of FBS and SCI

FBS and SCI are theories, i.e., collections of interconnected concepts. More specifically, they are theories for explanation [10] in that they offer an account of a more abstract phenomenon (software development) in terms of several less abstract phenomena (sensemaking, formulation, etc.). They apply to the *process* unit of analysis. They are process theories since they explain how software is developed rather than why it is successful. More specifically, they are teleological process theories since they posit an agent choosing its own activities rather than a predefined sequence. They could furthermore contribute to a general theory of software engineering in that such a theory should explain not only why software succeeds but also how software is created [7].

Neither FBS nor SCI are methods. Software development methods, such as the Rational Unified Process [54], prescribe ostensibly good ways of creating software. They claim that their constiuent practices are effective. SCI and FBS posit activities and transformations, respectively, which are necessary for and universal to software development. For example, Scrum claims that timeboxing meetings improves productivity [55]. It does not claim that all meetings are timeboxed. In contrast, SCI claims that all complex software development involves Coevolution, not that Coevolution improves productivity.

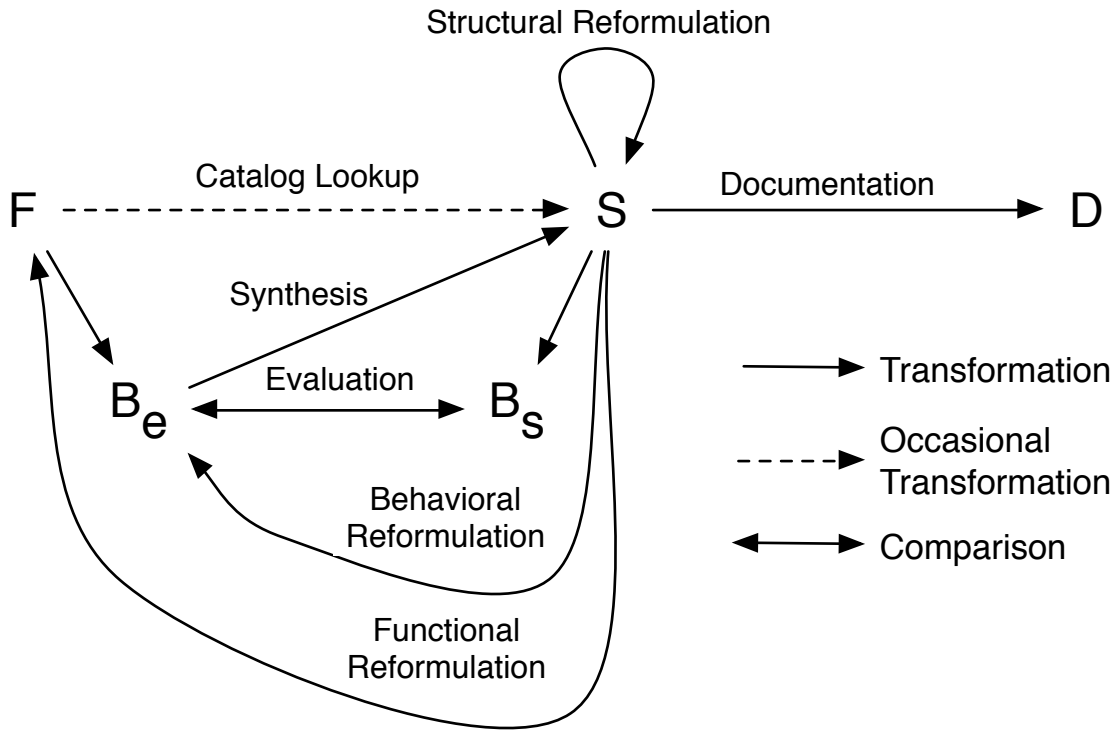

Fig. 3. The Function-Behavior-Structure Framework [adapted from 41]

Table 2
FBS artifact classes (adapted from [19]).

Symbol	Meaning (software)	Definition
Be	requirement model	expected (i.e. desired) behavior of the structure
Bs	simulation results	"the predicted behavior of the structure" (p. 3)
D	blueprints	a graphical, numerical or textual model that transfers "sufficient information about the designed artifact so that it can be manufactured, fabricated or constructed" (p. 2)
F	goal model	"the expectations of the purposes of the resulting artifact" (p. 2)
S	design model	"the artifact's elements and their relationships" (p. 2)

Table 3
FBS activity types (adapted from [19]).

Activtiy type	Meaning
Formulation	deriving expected (i.e. desired) behaviors from the set of functions
Synthesis	"expected behavior is used in the selection and combination of structure based on a knowledge of the behaviors produced by that structure" (p. 3)
Analysis	the process of predicting the behavior of a structure
Evaluation	comparing predicted behavior to expected behavior and determining whether the structure is capable of producing the functions
Documentation	transforming the structure into a design description that is suitable for manufacturing
Structural Reformulation	modifying the structure based on the structure and its predicted behaviors
Behavioral Reformulation	modifying the expected behaviors based on the structure and its predicted behaviors
Functional Reformulation	modifying the set of functions based on the structure and its predicted behaviors

Consequently, SCI and FBS are not simply analogues of Agile and Plan-Driven approaches. Plan-Driven approaches assume that better project planning and more upfront analysis positively impact success. Contrastingly, FBS makes no claims about project planning or upfront analysis beyond the assumption that system goals are known or given. FBS has no phases. Meanwhile, Agile approaches assume that greater success will result from focusing on individuals, software, customer collaboration and responsiveness rather than processes, documentation, contraction negotiation and planning.

Contrastingly, SCI makes no predictions concerning success antecedents and simply posits that developers oscillate between three core activities. While Extreme Programming, for example, prescribes rapid analyze-design-build-test cycles [56], SCI suggests that "analysis", "design", "coding" and "testing" are fundamentally misleading categories for development activities. One purpose of studying process theories including FBS and SCI is to reveal fundamental concerns obscured by the Agile/Plan-Driven debate (cf. [57]).

FBS and SCI are good rivals for empirical testing for several reasons, as follows.

1. Neither theory has received significant empirical testing. The study described in this paper is the first empirical evaluation of SCI. A previous paper reported preliminary results of this study [58]. I am aware of no empirical research examining the veracity of FBS in the SE domain.
2. FBS and SCI are similar enough for meaningful comparison. They are both teleological process theories – explanations of how and why an entity changes wherein change is manifested by a goal-seeking agent that engages in activities in a self-determined sequence [14,59,60]. They both explain design by organizing observable phenomena into several high-level classes (e.g. Synthesis, Sensemaking). They both involve iteration.
3. Both theories are credible. SCI originated in SE to explaining SE process phenomena [18]. It synthesizes influential previous research including Reflection-in-Action [46], Alexander's Selfconscious Process [40] and extensive research on Coevolution (cf. [48,49]). Meanwhile FBS is itself widely cited, and has spawned a stream of research extending beyond its creator, (e.g. [9,19,20,41,52,53,61-63]). Several papers apply FBS to software development (e.g. [41,42]). Finally, while FBS and SCI may not be perfect rivals, methodological guidance strongly suggests that an imperfect rival theory is still better than no rival theory [64-66].
4. Testing SCI against FBS may provide insights into several related models and theories. Since the Waterfall Model [33] and Basic Design Cycle [38] are special cases of FBS (cf. [37,41]), evaluating SCI and FBS may also reveal useful insights into Waterfall, the Basic Design Cycle and the tools and practices based on them.
5. FBS and SCI have many divergent propositions (Table 4). These differences provide a basis for generating empirically testable propositions.

3. Methodology

Process theory testing differs from variance theory testing in several ways. As process theories are concerned with explaining a contemporary phenomenon rather than a causal relationship, they have neither independent nor dependent variables and therefore cannot be tested experimentally [67]. Instead, process theories are best tested using questionnaires or longitudinal field studies [65,67-69]. Combining the two increases rigor [70]. Moreover, a causal theory positing that independent variable A causes dependent variable B may be clearly supported by experimentally manipulating A. However, when a process theory is evaluated in the field, at least some observations will support it unless it is absurd and at least some observations will contradict it unless it was overfit to the domain. Consequently, process theories are best evaluated against rival theories [64,66]. This "strong inference" model of testing produces more robust research and faster scientific progress [71].

Consequently, this section describes a multi-methodological approach to evaluating SCI against FBS. The approach combines a multiple-case study to enhance depth and a questionnaire study to enhance breadth within a primarily positivist epistemology. Here "multiple-case study" refers to a longitudinal empirical inquiry of a contemporary phenomenon that triangulates across multiple locations and types of evidence [66]. The methodology design was informed by common guidelines for questionnaire studies (e.g. [72-74]), positivist case studies (e.g. [66,75]) and multi-methodological

research (e.g. [70]). The unit of analysis is process and all team members are assumed capable of informing on their process.

Table 4
Key differences between SCI and FBS.

Dimension	SCI	FBS
Problem	The designer faces an ambiguous situation in which stakeholders disagree about aims or cannot clearly articulate goals and constraints.	The designer faces a given problem, i.e., one that is known, clear and agreed.
Coupling	Framing the problem, devising a solution and building the solution are tightly coupled.	Framing the problem, devising a solution and building the solution are loosely coupled.
Focus	Software design is primarily focused on creating and manipulating sourcecode – models are secondary.	Software design is prodominately focused on creating and manipulating symbolic representations (models) – sourcecode is secondary.
Order	Goals and structure are simultaneously co-created.	Software structure is synthesized from requirements, which are derived from goals.
Evaluation	Designers primarily evaluate their designs by implementing them and observing their effects.	Designers primarily evaluate their designs by predicting their behavior from design models.

3.1. Hypotheses

Hypothesis H_1: *SCI more accurately represents software development practice than FBS.*

Hypothesis H_2: *FBS more accurately represents software development practice than SCI.*

Since this study involves evaluating a theory against a rival, there is no null hypothesis. Moreover, when testing process theories it is better to state the hypotheses at the theory level rather than the proposition level because process theory propositions are more complex and interconnected than variance theory propositions [79]. This will be more clear in light of the following analysis.

3.2. Instrument development

A case study interview guide (Appendix A) was developed prior to the first case and refined throughout the study. A case study coding scheme (Appendix B) was developed such that each concept and relationship in SCI and FBS was given two columns – *evidence for* and *evidence against*. Soliciting feedback on the coding scheme from a colleague familiar with both theories resulted in minor changes.

A pilot was conducted to evaluate the interview guide and coding scheme. After minor improvements both were considered sound and the pilot demonstrated that the relevant phenomena were practically observable. The pilot results are included in the cross-case analysis as no significant methodological differences from subsequent cases were evident (Case C1, below).

The pilot also informed development of the questionnaire-study. The questionnaire was created according to the following eight-step process. This process resulted in an instrument comprising 13 items (questions), formulated using five-point bipolar scales (Appendix C).

1. The author identified differences between the two theories based on their formal descriptions and manifestations in the pilot case.
2. A colleague with expert knowledge of software design reviewed these differences, finding no omissions, biases or unwarranted differences.
3. The author generated approximately 80 items concerning these differences.
4. Items were reviewed by two colleagues with experience in questionnaire-based research and design practice.

5. Items were revised and a draft questionnaire was created.
6. A pilot was conducted with three professional developers and seven PhD students to get research-oriented feedback. Items were revised to enhance validity.
7. A second pilot was conducted with 12 professional developers. Items were revised to enhance clarity and brevity. Making the questionnaire as brief as possible was broadly considered critical to both response rate and response quality.
8. A third pilot with 10 professional developers was conducted. Their feedback was predominately positive and no further substantial changes were made.

Consistent with methodological guidance for process theory testing, items examine differences between SCI and FBS rather than specific propositions of either theory. To limit length and increase response rates, the questionnaire focused on three core differences: 1) whether system goals are given or constructed by the designer (5 items), 2) whether designing and coding are separate or entangled (5 items); 3) whether designing is model-focused or code-focused (3 items). Following methodological guidance for process theory testing [65,67], each bipolar item had one pole indicating agreement with FBS and another indicating agreement with SCI on one of these differences. The question order was randomized and the scales were reversed for items 5, 6 and 10 (i.e. the SCI pole was on the left instead of the right). Demographic and project-related questions were also included (see below).

In summary, an interview guide (Appendix A) and coding scheme (Appendix B) were developed for the case studies based on a pilot case. The case study coding scheme considers *individual propositions* of SCI and FBS separately. Meanwhile, a 13-item scale (Appendix C) was developed and validated for the questionnaire study using three pilot studies. The questionnaire items consider *specific differences* between propositions of SCI and FBS.

3.3. Sampling

Case site selection followed a literal replication strategy [66] with purposive sampling [76]. The specific sites were simply the organizations that were willing to participate. Case studies are not statistically generalizable research [77]; consequently, the four selected cases are not a representative sample. Their purpose is to explore in-depth manifestations of FBS and SCI elements in practice, not to provide statistically generalizable results.

For the questionnaire study, the population of interest includes all members (e.g. managers, analysts, etc. – not just programmers) of all software development teams worldwide. Obviously, obtaining a representative sample from this population is not possible because there is no population list from which to randomly select a sample. Consequently, two approaches to sampling were considered:

1) Randomly sample from a surrogate population, e.g., developers registered on SourceForge. This would not produce a representative sample because members of the SourceForge community may systematically differ from developers in general.
2) Non-randomly sample from the target population. This also would not produce a representative sample as population members have unequal probabilities of selection.

Both approaches are reasonable. The second approach was adopted to pursue the broadest possible sample of developers. While breadth does not guarantee representativeness in convenience sampling, a broader sample is more likely to reflect the vast diversity of SE.

For practical purposes, the population was limited to English speakers. The questionnaire was distributed through Twitter, blogs and online social networks including Facebook and LinkedIn to maximize responses through viral invitation. Link tokens were used to record the origin of respondents.

3.4. Case Context Summary

Four cases were conducted. They vary on several dimensions (Table 5).

Table 5
Case Diversity.

Case	Country	Sector	Method	Arrangement	Product	Project size
1	Canada	eBusiness	Scrum (agile)	commercial-of-the-shelf	novel product	5 participants / multi-year
2	UK	eCommerce	ad hoc (amethodical)	outsourced	novel product	5 participants / multi-year
3	UK	education	PRINCE2 (plan-driven)	in-house	legacy system replacement	25-100 participants / multi-year
4	UK	education	Scrum-like (agile)	internal entrepreneurship	novel component	7 participants / 8 months

Case One (C1) is a mid-sized software services and development company in Vancouver, Canada, which includes several distinct teams. The studied team has five members – two professional web developers, an intern developer, a product owner and a quality assurance analyst. It builds and maintains an online application that helps businesses manage relationships with their partner organizations. Its process was heavily influenced by Scrum (cf. [55]). Originally conceived as a pilot case, the data, analysis and results of C1 were reviewed extensively by a colleague with expertise in case research.

Case Two (C2) is a web development and online marketing agency of between 40 and 50 employees in England. Rather than discrete project teams, the company operates as a hub-and-spokes network where each project is lead by a manager (hub) who assign tasks to whoever has the necessary expertise (spokes) such that each developer's time is split between several simultaneous projects. The case focused on three developers, a graphics designer and the account manager who collaborated on a specific consumer e-commerce website. The project employed an evolving, ad hoc approach.

Case Three (C3) is a medium-sized English university developing and deploying a Moodle-based virtual learning environment. As in C2, participants split their time among many projects. The team was governed by a complicated management structure based on PRINCE2 [78]. In addition to several layers of governance, the project involved three core developers, a technology strategist, a project manager, and dozens of minor contributors.

Case Four (C4) is a team of part-time developers assembled within a university context to complete a series of small projects, including a mobile application to report facility faults (e.g. broken windows). The team employed a Scrum-like approach and consisted of four developers, a business analyst, a Scrum Master and a Product Owner. Team members were a mixture of BSc and MSc students with up to three years of industry experience.

3.5. Data Collection

Between December 2, 2009 and January 11, 2010, 1384 participants from 65 countries responded to the survey. As the sample size is undefined, the response rate cannot be calculated. However, of 4410 individual visitors, 1384 completed the survey (31%), 1118 partially completed it and 1908 bounced. Meanwhile, Cases 1 and 2 involved intensive on-site data collection over periods of two and six weeks respectively, followed by intermittent contact with informants to ask clarifying questions and validate conclusions. C3 involved intermittent data collection over approximately one year. C4 involved intensive data collection 1-2 days per week for eight months, of which five months were spent on the fault reporting project considered here. Although all cases involved semi-structured interviews,

the primary mode of data collection for Cases 1, 2 and 4 was direct observation of participants, which produced extensive field notes. Data collection also involved unstructured interviews, copying relevant documents, observing and recording meetings, photographing working conditions and collecting relevant email (Table 6). Each case had its own data collection protocol; all collected data was digitized (if necessary) and held in a single case database.

3.6. Data Analysis

For each case, data analysis began shortly after collection to facilitate adjusting interview questions for unexpected phenomena. Video and audio recordings were transcribed by either the researcher or a professional transcriber. The analysis then proceeded in roughly five phases beginning in April, 2008 and completing in April, 2013.

1. Questionnaire data was statistically analyzed.
2. All of the case evidence (i.e. transcripts, documents, emails and field notes) was reviewed.
3. The author coded the case study data using an a priori coding scheme (Appendix B) based on the two theories. Briefly, this involves scouring the data for excerpts supporting or contradicting propositions of either theory, and copying these excerpts into a table. The same item may simultaneously support or refute several propositions. In practice, the table is spread across several documents because it is so large; for example, the Case 1/SCI/For/Coevolution cell contains 1120 words.
4. Cross-case analysis or cross-case synthesis refers to comparing and contrasting patterns across cases [66]. While several approaches exist, here cross-case analysis involves comparing and contrasting the coded data for each proposition of each theory. Similarities and differences between the patterns observable in each case were noted and inspected to produce a set of aggregate patterns – one for each proposition. These patterns are often represented as descriptive narratives.
5. Case-questionnaire triangulation here refers to comparing and contrasting the results of the questionnaire with the results of the cross-case analysis. Again, multiple approaches are possible; however, in this study the researcher analyzed not only how the case data reinforces or contradicts the questionnaire results and vice versa, but also how the strengths and weaknesses of the two research methods interact.

The analysis of Case 1 was carefully reviewed by a colleague with extensive knowledge of case study research and software engineering. The results and interpretations of each case were, furthermore, discussed with case participants. Their feedback resulted in many minor changes but no substantive changes.

4. Results

This section summarizes the cross-case analysis, questionnaire analysis and case-questionnaire triangulation. Case-by-case analysis is omitted for brevity, as is common in multimethod research.

4.1. Cross-Case Analysis

Numerous propositions may be derived from FBS and SCI. Some propositions may be derived from both theories; for instance, both FBS and SCI are consistent with using diagrams. FBS's *structure* could be represented using a UML class diagram, while SCI's context schema could be represented using a Rich Picture diagram. However, comparative analysis focuses on dimensions where the rival theories conflict [65]. This section consequently evaluates a selection of conflicting propositions associated with each theory.

Table 6.
Data collection by case

Case	Interviews	Documents	Observation	Meetings	Photos	Emails
1	✔	✔	✔	✔	✔	
2	✔	✔	✔		✔	✔
3	✔	✔		✔		
4	✔	✔	✔	✔	✔	✔

SCI_1: Designers engage in Sensemaking. In SCI, *Sensemaking* includes investigating the context, organizing one's understanding of the context and obtaining feedback on artifacts. All four cases involved investigations, including calling stakeholders with specific questions (C1), face-to-face meetings with clients (C2, C4) and extensive user consultation (C3). One designer explained, "one of the most important parts of my job is to be talking with people on a regular basis, whether they're existing customers or potential customers or just people in the market in general" (C1). In all four cases, participants organized their understanding of the domain, for example, writing call reports summarizing stakeholder comments (C1), writing notes on stakeholder meetings (C2, C3, C4) and researching project infrastructure options (C4). One analyst explained, "I then compiled my notes into much more of a narrative where I've grouped things under themes" (C3). Similarly, all four cases involved feedback on artifacts, e.g., "I would bring the [alpha release] out with me when I'm talking to people and show them" (C1); "Fault report widget delivery meeting ... the facilities and [information systems] guys were reasonably pleased with the interface" (C3 field notes; 24 July 2012); "we get the clients to do some testing" (C2). Therefore, SCI_1 is supported.

SCI_2: Designers coevolve context schemas and design space schemas. In SCI, *Coevolution* specifically refers to a cognitive oscillation between one's understanding of the context and one's understanding of the design space where changes to the former trigger changes to the latter and vice versa. Coevolution was directly observed in C1, C2 and C4. For example, the context in C4 was originally framed as helping students report faults in their dorm rooms. Consequently, initial mock-ups did not include the fault location. This appeared counterintuitive to the developers and triggered contextual reframing such that students could submit faults anywhere on campus. This triggered design space reframing manifested by adding a location question to the mock-ups. Figure 4 provides a more nuanced example. Coevolution was difficult to observe in C3 as direct observation of developers was not possible. Therefore, SCI_2 is mostly supported.

SCI_3: *Designers implement the design object*. In SCI, *Implementation* includes coding, white-box testing and deployment. Coding and white-box testing were directly observed in C1, C2 and C4. While not directly observed, coding in C3 was evident from the regular arrival of new code. The nature of technical testing varied substantially. In C2 and C4, testing was quite ad hoc – "We do test ourselves but not really in a systematic way, there is never time for that" (C2). Unit testing was used extensively in C1 and proposed but abandoned in C4 because the proprietary mobile application framework had poor tool support. All four cases employed visual inspection wherein developers would observe the effects of their changes, clicking buttons and entering text to see if the software performed as expected. This type of testing was tightly coupled with coding, e.g. "D. makes changes to a copy of the website running on her own development machine. She regularly switches between the code and the website to see what effect her changes have had" (C1 field notes; 16 Apr 2008). Concerning deployment, C1 and C2 had both deployed commercial versions by the end of the data collection period. C3 and C4 both deployed prototypes to limited user groups during the data collection period and (at the time of writing) have both gone on to release full versions to their entire user bases. SCI_3 is therefore supported.

D. draws a diagram of the relationships between the concepts (channel/partner/etc) and the IT artifacts involved, i.e. the online forms, fields and values. They discuss how one would change the site so that we could have different forms, for different kinds of partners. Both hold markers, both edit the diagram, using it to guide their discussion and communicate. This is clearly a reflective, dialectic design process. The diagram mixes the problem and solution, form and context. They seem to be running rapid mental simulations (i.e. what-if analysis). D. draws out the possible combinations of partners/channels/etc. A. checks with M. on the business rules. D. and A. use M. as their "context." D. goes through all possible cases and how they will fit into the proposed design model. More what-if analysis. A. projects into the future: what will the product be like eventually, and how will this effect our design? Their discussion is grounded in their sketching: lots of "this is related to this" with pointing. Very visual. A. proposes new concepts (he calls "classes") to possibly solve the design issue. (C1 field notes; 15 Apr 2008)

Fig. 4. Coevolution in situ

FBS_1: Designers engage in formulation ($F \Rightarrow B_e$). In FBS, *formulation* refers to deriving a behavioral requirements model from a goal model. No artifact – textual or diagrammatic – approximating a goal model was observed in C1 or C4. Case 3 included at least 200 documents including a *project mandate*, which ostensibly clarifies the project's goals and scope. However, the project mandate did not actually contain any clear goals. Rather, it contained vacuous rhetoric such as "The aim is to provide "sector leading" provision for the 2012 intake of students". Participants were unable to articulate meaningful goals, instead making statements including "One of the fundamental things from my point of view is that it must work". Similarly, participants in C2 wrote a *project brief*, ostensibly to capture project goals. However, the brief focused on product features rather than goals, as explained by its writer: "normally I spend a few hours, like 3-4 hours, just looking at the features and the solution". Another participant admitted "I think sometimes we missed the core aim of what the project was trying to achieve". No other documents in C2 or C3 contained meaningful goal statements. Therefore, FBS_1 was not supported.

FBS_2: Designers engage in synthesis ($B_e \Rightarrow S$). In FBS, *synthesis* refers to devising a structural model of an artifact intended to satisfy a requirements model. No artifact – textual or diagrammatic – approximating a requirements model was observed in C4. A "requirements" document was observed in C1 and C3 and the "technical specification" in C2 ostensibly served the same purpose. However, at least one C1 team member was unaware of any requirements document and another explained "we think stories instead of requirements". A story, such as "partner application creation is necessary or not for channel creation?" was understood as "a promise to have a conversation" rather than a requirement. Similarly, in C2, although the technical specification was intended to drive the design process, participants explained that it contained insufficient detail – "we don't really write everything down" said one, while another complained "the problem with [the project] was that ... the specification for that was very, very brief" and a third admitted that the technical specification "quickly went out of the window because of the volume of changes". Likewise, although C3 included a substantial requirements elicitation and modeling process, it occurred circa spring 2011, several months after major design decisions including the platform and plug-ins were made. Rather than driving design modeling, the "consultation" process was used to justify a priori decisions, leading one participant to malign it as a "pseudo-consultation". Consequently, FBS_2 was not supported.

FBS_3: Designers engage in analysis and evaluation ($S \Rightarrow B_s \leftrightarrow B_e$). In FBS, *analysis* refers to predicting the behavior of the proposed structure and *evaluation* refers to comparing the predicted behavior model against the requirements model. Nothing approximating predicting behavior from design models was observed in any of the cases. All four cases included design models, especially website wireframes (C4) and renderings / visual mockups (C1, C2, C3). However, nothing like a model of predicted behavior was evident and no observations suggested behavior prediction. Contrastingly in C1, for example, participants evaluated the website (not a design model) by observing (not predicting) its behaviors (cf. Figure 4). FBS_3 is therefore not supported.

SCI_4 (FBS_4): Problem framing, design and artifact construction are strongly (weakly) coupled. SCI posits that problem framing, solution generation, coding and deployment are all tightly coupled with design while FBS assumes that design clearly begins with given system goals and ends when detailed design documentation is passed to developers for coding. No clear distinction between problem framing, solution generation, coding and deployment was evident in any of the cases. The same people simultaneously tried to determine what the system should do, explored what it should be like, built it and deployed it. In C1, C2 and C3 at least, the goals were still unclear even after deployment.

C1 and C4 developers built the software directly from their design space schemas, only occasionally referring to user stories, wireframes and mockups; C2 developers worked from an admittedly brief and vague technical specification. Meanwhile, the labyrinth of documents produced by C3 participants may have constituted detailed documentation; however, most of the key design decisions were made *before* the documentation was produced. Instead, participants appear to develop their ideas of project context and design artifact simultaneously (SCI_2, above). Moreover, in C1, C2 and C3, product deployment was an ongoing activity with updates including new features every few weeks and minor fixes and tweaks even more frequently. No separate deployment or transition phase was evident; for example, one participant explained that the "project is more about continuous development and improvement" (C2). In C3, deploying the prototype was a mechanism for understanding the project context – "The VLE pilot phase ... will allow us to learn as much as we can about the issues we will come up against in the larger project" (Project Mandate). In C4, however, the artifact was not deployed during the observation period.

As discussed above, none of the cases exhibited the type of goal models FBS posits. Moreover, no evidence of detailed design documentation (other than code commenting) was observed. Therefore, SCI_4 is mostly supported and FBS_4 is mostly unsupported.

In summary, none of the four FBS propositions are supported; SCI's Sensemaking and Implementation activities are strongly supported; Coevolution and the strong coupling proposition are mostly supported (Table 7). In the interests of brevity, the above analysis omits assumptions shared by both theories, including the existence of a design agent and teleological causation, and uncontroversial propositions including FBS's proposition that designers reformulate (i.e. edit) design models and SCI's proposition that the context includes constraints.

4.2. Questionnaire Data Analysis

Respondents varied substantially across demographic, project and company variables (Figures 5-9) but were overwhelmingly male (1241 men vs. 56 women). Most respondents came from the United States (549), Canada (176), the United Kingdom (118) and Australia (73). The most common roles were developer (1325), analyst (569), quality assurance (533), manager (266) and graphics (195). When asked "is your project more 'social' (like a website) or 'technical' (like a device driver)?", 34% of participants said more social, 29% said more technical and 36% said in between. Based on employers it can be inferred that at least the following sectors are represented: aerospace, applications development, digital media, eCommerce, education, finance, IT consulting, journalism, marketing, networking, operating systems, research, security, sports, telecommunications and tourism.

Table 7
Support for selected FBS and SCI propositions.

Theory	Proposition	Case 1	Case 2	Case 3	Case 4
FBS1	Formulation	✘	✘	✘	✘
FBS2	Synthesis	✘	✘	✘	✘
FBS3	Analysis/Evaluation	✘	✘	✘	✘
FBS4	Loose Coupling	✘	✘	✘	?
SCI1	Sensemaking	✔	✔	✔	✔
SCI2	Coevolution	✔	?	✔	✔
SCI3	Implementation	✔	✔	✔	✔
SCI4	Tight Coupling	✔	✔	✔	?

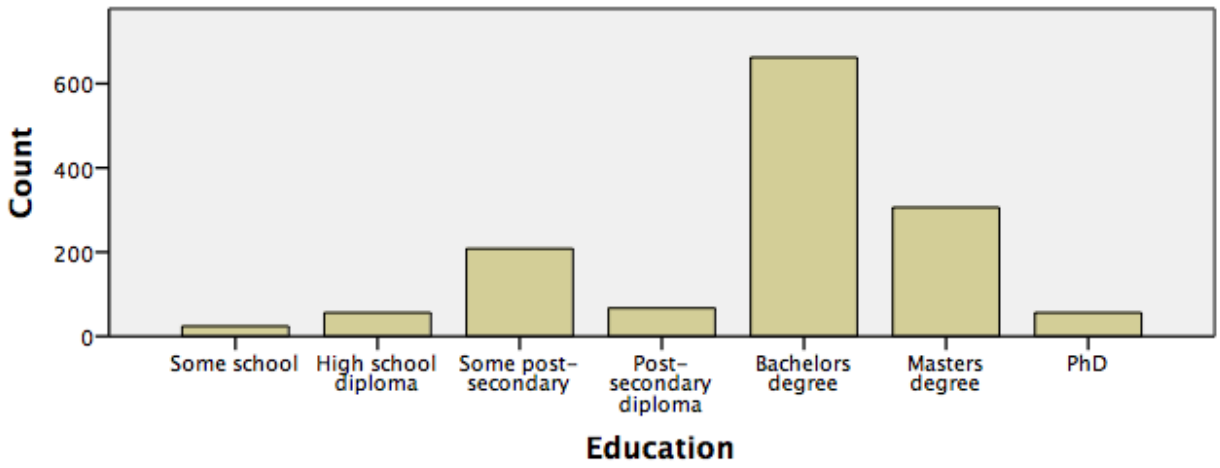

Fig. 5. Distribution of participant education levels

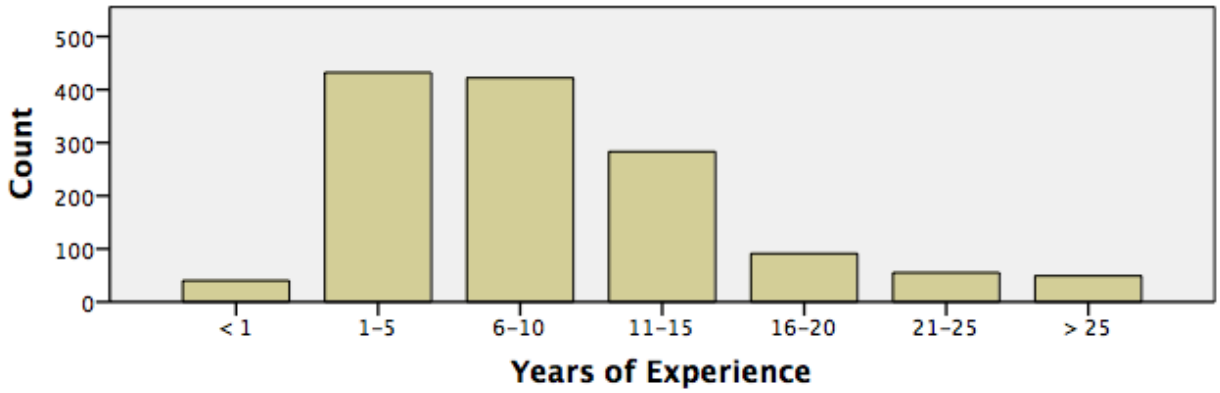

Fig. 6. Distribution of participant experience

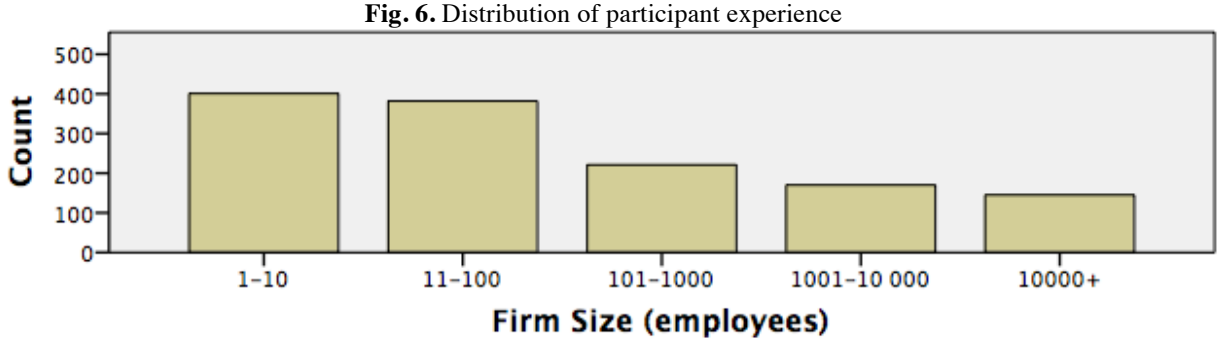

Fig. 7. Distribution of company sizes

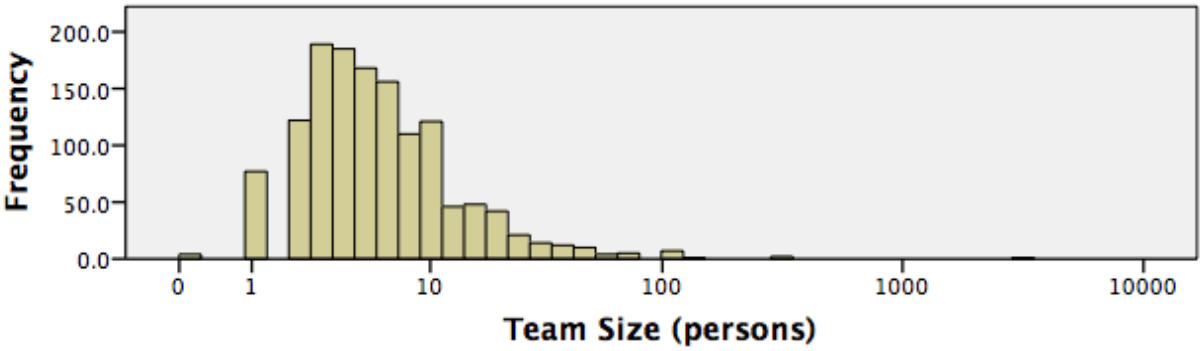

Fig. 8. Distribution of team sizes

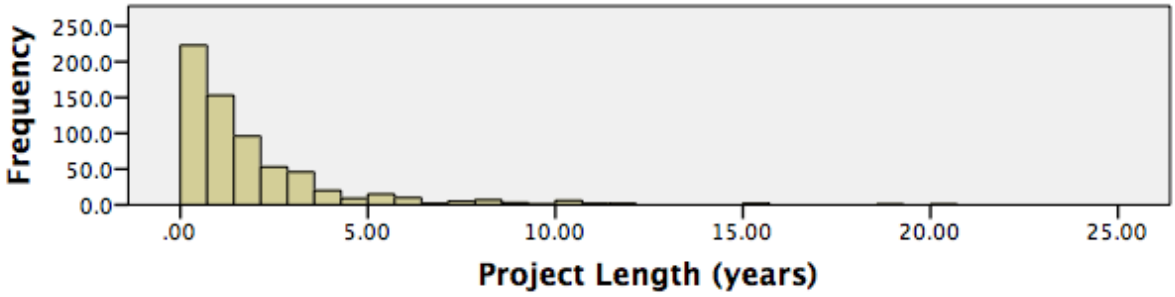

Fig. 9. Distribution of project durations

Participants responded to 13 items concerning differences between SCI and FBS. Assuming responses are coded from 1 (strong support for FBS) to 5 (strong support for SCI), five meaningful results patterns are possible:

1) A positively-skewed distribution (median 1 or 2) indicates that FBS is more accurate;
2) A negatively-skewed distribution (median 4 or 5) indicates that SCI is more accurate;
3) A symmetric distribution (median 3) suggests that SCI and FBS represent extreme positions with most development falling somewhere in between;
4) A bimodal distribution (e.g. modes of 2 and 4) suggests multiple development subcultures, i.e., some developers act as FBS predicts while others act as SCI predicts;
5) A variety of symmetric, positively and negatively skewed items indicates a problem with the survey instrument.

The overall distribution is negatively skewed (Table 8; Figure 4), favoring SCI. The negative skew is significant ($p<0.001$; χ^2 test) for all items (see Appendix D for details). The practical significance of the observed distribution should be evident from visual inspection of Figure 4. In summary, Hypothesis H_1 is supported; H_2 is not supported.

4.3. Case-Questionnaire Triangulation

Both questionnaire and case study results support SCI over FBS. Moreover, combining questionnaire and case study results leads to more nuanced findings.

Table 8
Questionnaire results by item.

		Item													Respondents	
		1	2	3	4	5	6	7	8	9	10	11	12	13	Median	Mode
Strong FBS		7	13	14	20	62	22	22	13	17	58	23	13	9	0	2
FBS		38	66	42	76	161	61	97	39	63	168	173	174	67	4	21
Neutral		72	162	109	120	195	78	113	55	122	148	320	299	303	43	32
SCI		597	662	576	572	572	398	539	452	539	492	671	623	562	932	717
Strong SCI		656	446	628	575	349	819	592	796	620	505	173	155	425	406	613
Item	Median	4	4	4	4	4	5	4	5	4	4	4	4	4		
	Mode	5	4	5	5	4	5	5	5	5	5	4	4	4		

Note: columns do not total 1384 as each question had a "N/A" option

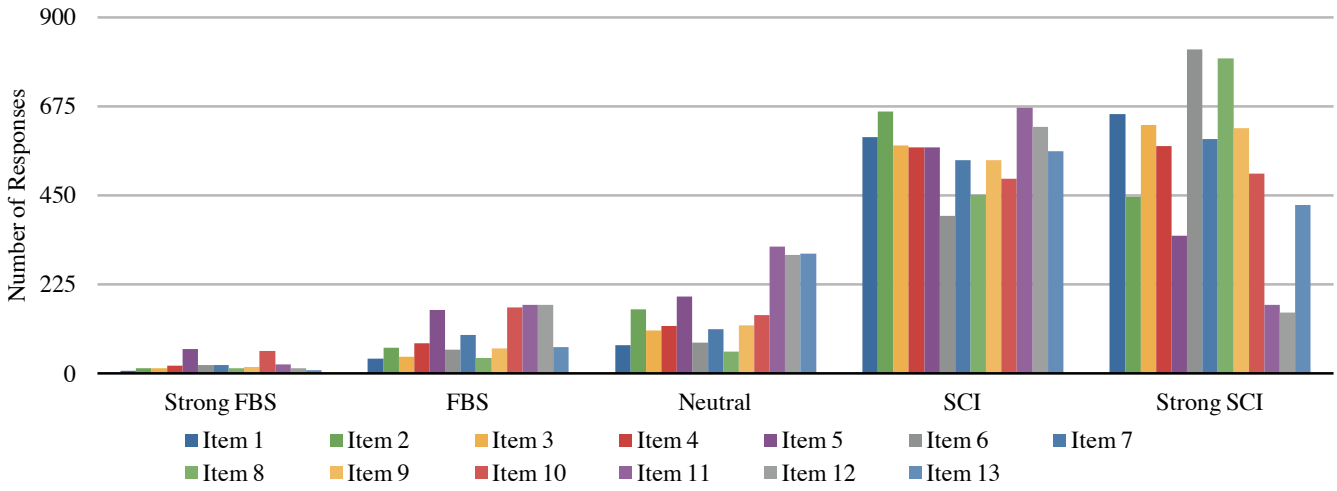

Fig. 4. FBS/SCI inter-item agreement and distribution

The questionnaire specifically investigates three core differences between SCI and FBS: 1) whether problem framing and design are tightly or loosely coupled; 2) whether design and coding are tightly or loosely coupled; 3) whether respondents focus on models or code. Both questionnaire and case data indicate that problem framing, design and coding are tightly coupled. Moreover, case data suggests that deployment of both prototypes and commercial products may be closely interconnected with design in some contexts. While the questionnaire data indicates that respondents principally manipulate source code rather than models, case data further suggests that goal models may be absent. Case data further indicates major design decisions may be made independently of requirements models. More generally, while the questionnaire was limited to core theoretical conflicts, the case studies facilitated examining individual propositions of each theory.

However, the survey could still suffer from sampling bias. Specifically, a development subculture where FBS is more accurate may exist; for instance, large firms with formal development processes. While such a subculture cannot be definitively established or refuted by this study, we can explore this possibility by examining whether responses vary on some demographic or project dimensions. This requires a measure of an individual's overall orientation toward the frameworks. For the following analyses, an individual's process theory agreement score (or simply "score") is an informal measure of how well a respondent's beliefs correspond to the assumptions of FBS or SCI on a scale of 1 (strong-FBS) to 5 (strong-SCI). This score, defined as follows, is used to simplify presentation of the exploratory analysis and should not be generalized to other meanings or purposes.

Process Theory Agreement Score: *a bipolar measure of the extent to which the individual's beliefs about his or her work practices conform to either FBS or SCI, operationalized as the respondent's median response across the 13 items.*

Larger teams and longer projects were *not* more FBS-like, in fact, participants having median score of 4 or 5 had a *higher* mean team size and longer mean duration (Table 9), although these differences were not statistically significant. Firm size and the nature of the project similarly had no effect on this score (Table 10). Likewise, median score did not vary by respondent gender, education level, years of experience (Table 11), country of residence (Table 12), occupation or project role (Table 13). Scores do not vary by self-reported method (Table 14) except that respondents who reported using Lean have a median score of 5 rather than 4. Moreover, scores do not vary by sampling origin, i.e., which advertisement attracted the respondent. In summary, there is no evidence of an FBS-supporting subculture in the survey data.

It is still possible than an FBS-supporting subculture exists. However, it seems highly unlikely that the sample would include large and small teams of men and women building everything from online games to device drivers for large and small companies in at least 16 different industries in over 60 countries, identifying with at least nine different development methods, but still somehow exclude a large FBS-supporting subculture. Therefore, although SCI may not better represent software practice in every possible case, it appears superior in at least a preponderance of cases.

Table 9
Independent samples t-test for team size and project length.

Variable	Median Score	N	Mean	Std. Deviation	Std. Error Mean	t	p
Team Size	≥ 4.00	1290	11.10	84.85	2.36	0.382	0.703
	< 4.00	54	6.69	6.92	0.94		
Project Length	≥ 4.00	638	1.94	2.38	0.09	0.540	0.589
	< 4.00	18	1.63	1.71	0.40		

Table 10
Median score by project-specific dimensions.

Dimension	Category	N	Median Score
Firm Size	1 - 10	393	4
(employees)	11 - 100	374	4
	101 - 1000	219	4
	1001 - 10 000	170	4
	> 10 000	144	4
Nature of Project	More Social	444	4
	Somewhere in between	475	4
	More Technical	372	4
	NA	9	4

Table 11
Median score by demographic dimension.

Dimension	Categry	N	Median Score
Gender	Male	1241	4
	Female	56	4
	Other	3	4
Education	Some school	18	4
	High school diploma	53	4
	Some post-secondary	198	4
	Post-secondary diploma	62	4
	Bachelors degree	626	4
	Masters degree	289	4
	Doctorate degree	54	4
Experience	less than 1	31	4
(years)	1 to 5	399	4
	6 to 10	405	4
	11 to 15	274	4
	16 to 20	89	4
	21 to 25	54	4
	more than 25	48	4

Table 12
Median score by respondent country.

Country	N	Median Score
United States	549	4
Canada	176	4
other	174	4
United Kingdom	118	4
Australia	73	4
Germany	44	4
France	40	4
Netherlands	33	4
India	23	4
Italy	19	4
Sweden	19	4
Beligium	17	4
Poland	17	4
New Zealand	16	4
South Africa	16	4
Romania	14	4
Brazil	13	4
Spain	12	4
Israel	11	4

Table 13
Median score by role in project and occupation.

Role			Occupation		
Response	N	Median Score	Response	N	Median Score
developer	1325	4	developer	1211	4
analyst	569	4	team lead	440	4
QA	533	4	manager	269	4
manager	266	4	analyst	197	4
graphics	195	4	QA	165	4
			graphics	59	4

Table 14
Median score by method.

Method	N	Median Score
Scrum	323	4
Extreme Programming	130	4
Agile	115	4
Test Driven Development	89	4
Service Oriented Architechture	79	4
Other	62	4
Waterfall	47	4
Kanban	17	4
Lean	16	5
Rational Unified Process	14	4
“Cowboy Coding”/“Seat-of-the-Pants”	11	4

Notes: 1) Methods mentioned by fewer than ten respondents are grouped as “other”. 2) Some of the “methods” listed by respondents are not technically methods, e.g., Service Oriented Architecture is an architectural design pattern. 3) “Median Score” refers to the median of the scores of all respondents indicating the influence of the corresponding method.

5. Discussion and Conclusion

SE theory is crucial to preserve and communicate empirical knowledge and to protect the field against piecemeal empiricism, fads and overemphasis on prescriptive knowledge. This paper consequently examines two dissimilar theories of the software development process. The results suggest the following.

1) SCI better explains how developers create software than FBS in many circumstances. That is, developers engage in three broad categories of activities – organizing their perceptions of the project context (Sensemaking), simultaneously improving their context and design space schemas by oscillating between them (Coevolution), and constructing, debugging and deploying software artifacts (Implementation).
2) Problem framing and design are tightly-coupled in practice. Conceptualizing them as separate phases or activities is misleading.
3) Design and coding are tightly-coupled in practice. Conceptualizing them as separate phases or activities is misleading.
4) Modeling goals and requirements is not necessary to develop software. Many projects proceed without requirements or ignore requirements (cf. [79-83]). Developers focus more on code artifacts than on conceptual models or design models.
5) A thorough analysis of a software project may reveal that documents and practices, which superficially appear to support FBS or at least suggest a rational and methodical development process *do not actually fulfill their ostensible roles*. For example, a project mandate may ostensibly lay out a project's aims and objectives but actually lack clear, specific, agreed goals.

These results should be interpreted in light of four main limitations.

1) Many observed phenomena are not obviously covered by either theory, including the use of informal models, quality, success, management and politics.
2) This study neither "proves" SCI nor "falsifies" FBS[1] for software development in general or any specific subfield thereof. The results simply favor SCI over FBS. Another interpretation of the results is that it is more common for software projects to meet SCI's assumptions than FBS's assumptions.
3) Case data is not statistically generalizable. Like most other surveys of software developers, the above survey results are also not statistically generalizable because there is no population list from which to randomly sample. SE includes diverse communities of practice and specialized domains that definitely are not represented by the case studies and may not be represented in the survey sample. A more FBS-like subculture may exist; however, post-hoc analysis revealed no evidence of a subculture.

[1] Verificationism and Falsificationism are defunct epistemologies (cf. [84-86]). Popper [87] refuted Verificationism by arguing that a universal hypotheses (e.g. all swans are white) could not be proven without observing every item in the population. However, Falsificationism may be analogously refuted by pointing out that an existential hypothesis (there exists a black swan) cannot be disproven without, again, checking every item in the population. Meanwhile, social and applied sciences are predominately concerned with probabilistic hypotheses. The hypothesis that smoking causes lung cancer means that smoking increases the probability of developing lung cancer. This hypothesis is neither proven by observing smokers with cancer nor disproven by observing smokers without cancer. From a Bayesian perspective, observations only change the a posteriori probability of the theory's correctness. Furthermore, when an observation appears to contradict a prediction, we do not know whether 1) the theory is incorrect, 2) the observation is incorrect, 3) the math connecting the theory and observation is incorrect or 4) the prediction was incorrectly derived from the theory [85].

4) The results imply nothing about the benefits or drawbacks of attempting to follow a more SCI-like or FBS-like process. FBS and SCI are theories, not methods. Trying to follow one or the other may or may not promote or hinder success.
5) The questionnaire results may suffer from response bias (where question phrasing distorts responses or sampling bias (where the sample is not representative of the target population). The sample may include participants in non-cohesive teams, which would violate the teleological underpinnings of both theories.

With these caveats in mind, the results of this study have numerous implications for research, industry and education.

For researchers, SCI provides a basis for coding observations in field studies, case studies, protocol studies, simulations and critical discourse analysis. Without SCI, researchers would have to invent their own categories or use categories based on methods (e.g. analysis, design, coding, testing), which lead to oversimplified and over-rationalized accounts of practice. This research furthermore illuminates fundamental questions often obscured in the methods literature. For example, what is the role of requirements engineering if design is driven by Coevolution rather than requirements specification? What tools and practices improve Coevolution performance? How should we manage public procurement where problem framing is entangled with problem solving? Moreover, this analysis suggests that asking when to use Lean or Scrum or the Unified Process or Waterfall may be the wrong question as actual processes may systematically differ from any specific method [8]. As the post-methodology era [88] (where developers reject methods in principle) solidifies, research may shift focus from methods to individual practices and psychology- or sociology-informed antecedents of success, including motivation [89] and cognitive bias [90,91]. Better measures of software engineering success are also needed [92].

For industry, it should be stressed, again, that SCI is not a method that can or should be adopted and used. Practically speaking this research indicates that instead of iteration between weakly-coupled phases (analysis, design, coding, testing), it is more accurate and useful to conceptualize development as ad hoc oscillation between organizing perceptions of the project context (Sensemaking), simultaneously improving mental representations of the context and design space (Coevolution) and constructing, debugging and deploying software artifacts (Implementation). This calls into question all industry practices and technologies based on dividing schedules, budgets, roles, people, teams and models into Waterfall-like categories. For example, since Coevolution is the primary mechanism for forming and revising design concepts, and low-level design decisions are inextricable from coding, dividing a project team into “analysts”, “system architects” and “developers” makes no sense. Similarly, since acceptance testing is fundamentally different from unit, integration and system testing, assigning all of these to the same “testers” makes no sense. Since problem framing is inextricable from design, the use of fixed-price contracts, government procurement or public tendering (where bids are solicited from firms based on pre-set requirements) and any attempt to divide work into phases approximating “requirements”, “analysis”, “design”, “coding” and “testing” similarly makes no sense. Additionally, as problem framing and solving are simultaneous, interconnected activities, expecting project participants to accurately estimate the time, budget or effort prior to development is simply unrealistic, which may explain the prevalence of inaccurate effort estimation [93]. This suggests that fixed-price/schedule contracts will increase overall project risk [17]. SCI specifically provides a language for explaining to stakeholders the problems with Waterfall-like categories.

For educators, recognizing the centrality of Coevolution in software development motivates major shifts in software engineering curricula, which largely ignores Coevolution [94]. Programs should cover SCI instead of the Waterfall Model as a basic description of development. Programs should cover specific techniques for Sensemaking (e.g. persona analysis [95]) and Coevolution (e.g. creativity techniques [96], sketching [97]) in addition to Implementation.

Future work may include both further testing and extensions of SCI. No single empirical evaluation can cover the enormous diversity of people, projects and conditions found under the SE umbrella. Consequently, future evaluations of SCI in edge cases such as model-driven engineering, reverse engineering and hardware-software coevolution could be enlightening. Meanwhile, possible extensions to SCI could include design by multiple agents, and breaking down Sensemaking, Coevolution and Implementation into meaningful sub-activities. SCI may also be integrated with other foundational theories [98] including The Theory of Boundary Objects [99], Transactive Memory Theory [100], the Theory of Cognitive Biases (cf. [91]) and the Theory of Distances [101].

In conclusion, this study presents the most comprehensive, if not the first, empirical analysis of either FBS or SCI in the domain of software development. Its core contribution is the finding that SCI provides the more accurate account of how most complex software is developed in practice. This conclusion rests on the responses of more than 1300 programmers, analysts, testers and managers from over 60 countries and approximately two years of field research including hundreds of hours of interviews and direct observation. Since a general theory of software engineering should explain not only *why* software is successful but also *how* software is developed, SCI may provide a useful foundation for a general theory of software engineering.

Acknowledgements

This research was funded by the National Sciences and Engineering Research Council of Canada, The University of British Columbia and Lancaster University. Special thanks are due to Brett Cannon, Ekkart Kindler, Pericles Loucopoulus, Pontus Johnson, Tero Päivärinta, Jeffrey Parsons and Olga Volkoff for their encouragement and advice. I would also like to thanks all of those who participated in and helped recruit participants for this study.

References

[1] T.B. Bollinger, The interplay of art and science in software, Computer. 30 (1997) 125–128. doi:10.1109/2.625346.
[2] E.M. Gray, W.L. Smith, On the limitations of software process assessment and the recognition of a required re-orientation for global process improvement, Software Quality Journal. 7 (1998) 21–34.
[3] M.M. Lehman, J.F. Ramil, An approach to a theory of software evolution, in: Proceedings of the 4th International Workshop on Principles of Software Evolution, ACM, 2001: pp. 70–74.
[4] J. Hannay, D.I.K.I.K. Sjøberg, T. Dyba, A Systematic Review of Theory Use in Software Engineering Experiments, IEEE Transaction on Software Engineering. 33 (2007) 87–107. doi:10.1109/TSE.2007.12.
[5] I. Jacobson, B. Meyer, Methods Need Theory, DrDobbs.com. (2009).
[6] P. Johnson, M. Ekstedt, I. Jacobson, Where's the Theory for Software Engineering? IEEE Software. 29 (2012) 94–96. doi:10.1109/MS.2012.127.
[7] P. Ralph, P. Johnson, H. Jordan, Report on the First SEMAT Workshop on a General Theory of Software Engineering (GTSE 2012), ACM SIGSOFT Software Engineering Notes. 38 (2013) 26–28.
[8] D.P. Truex, R. Baskerville, J. Travis, Amethodical systems development: the deferred meaning of systems development methods, Accounting, Management and Information Technologies. 10 (2000) 53–79.
[9] P.E. Vermaas, K. Dorst, On the Conceptual Framework of John Gero's FBS-Model and the Prescriptive Aims of Design Methodology, Design Studies. 28 (2007) 133–157.
[10] S. Gregor, The Nature of Theory in Information Systems, MIS Quarterly. 30 (2006) 611–642.
[11] R.L. Glass, I. Vessey, V. Ramesh, Research in software engineering: an analysis of the literature, Information and Software Technology. 44 (2002) 491–506. doi: 10.1016/S0950-5849(02)00049-6.
[12] A.H. Van de Ven, Engaged scholarship: a guide for organizational and social research, Oxford University Press, Oxford, UK, 2007.
[13] J. Kim, Causation, in: R. Audi (Ed.), The Cambridge Dictionary of Philosophy, 2nd ed., Cambridge University Press, Cambridge, UK, 1999: pp. 125–127.
[14] A.H. Van de Ven, M.S. Poole, Explaining development and change in organizations, The Academy of Management Review. 20 (1995) 510–540.
[15] P. Johnson, P. Ralph, M. Goedicke, P.-W. Ng, K.-J. Stol, K. Smolander, et al., Report on the Second SEMAT Workshop on General Theory of Software Engineering (GTSE 2013), ACM SIGSOFT Software Engineering Notes. 38 (2013) 47–50.
[16] P. Ralph, I. Exman, P.-W. Ng, P. Johnson, M. Goedicke, A.T. Kocatas, et al., How to Develop a General Theory of Software Engineering: Report on the GTSE 2014 Workshop, ACM SIGSOFT Software Engineering Notes. 39 (2014) 23–25. http://dl.acm.org/citation.cfm?id=2674647.

[17] F.P. Brooks, The Design of Design: Essays from a Computer Scientist, Addison-Wesley Professional, 2010.
[18] P. Ralph, The Sensemaking-Coevolution-Implementation Theory of Software Design, Science of Computer Programming. 101 (2015) 21–41. doi:10.1016/j.scico.2014.11.007.
[19] J.S. Gero, Design prototypes: A knowledge representation schema for design, AI Magazine. 11 (1990) 26–36.
[20] J.S. Gero, U. Kannengiesser, The Situated Function-Behaviour-Structure Framework, Design Studies. 25 (2004) 373–391.
[21] P. Anderson, Complexity Theory and Organization Science, Organization Science. 10 (1999) 216–232.
[22] P. Freeman, D. Hart, A Science of design for software-intensive systems, Communications of the ACM. 47 (2004) 19–21.
[23] R. Agarwal, P. De, A.P. Sinha, Comprehending object and process models: An empirical study, IEEE Transaction on Software Engineering. 25 (1999) 541–556. doi:10.1109/TSMCA.2010.2087017.
[24] G.S. Howard, T. Bodnovich, T. Janicki, J. Liegle, S. Klein, P. Albert, et al., The efficacy of matching information systems development methodologies with application characteristics–an empirical study, Journal of Systems and Software. 45 (1999) 177–195.
[25] B. Freimut, O. Laitenberger, S. Biffl, Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques, in: Proceedings of the Seventh International Software Metrics Symposium, IEEE, 2001: pp. 51–62.
[26] O. Laitenberger, C. Atkinson, M. Schlich, K. El Emam, An experimental comparison of reading techniques for defect detection in UML design documents, Journal of Systems and Software. 53 (2000) 183–204.
[27] O. Laitenberger, K. El Emam, T.G. Harbich, An internally replicated quasi-experimental comparison of checklist and perspective based reading of code documents, IEEE Transaction on Software Engineering. 27 (2001) 387–421.
[28] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empir Software Eng. 17 (2011) 531–577. doi:10.1007/s10664-011-9173-9.
[29] S. Thummalapenta, L. Cerulo, L. Aversano, M. Di Penta, An empirical study on the maintenance of source code clones, Empir Software Eng. 15 (2009) 1–34. doi:10.1007/s10664-009-9108-x.
[30] M. Harman, P. McMinn, A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search, IEEE Transaction on Software Engineering. 36 (2010) 226–247. doi:10.1109/TSE.2009.71.
[31] D.L. Moody, The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering, IEEE Transaction on Software Engineering. 35 (2009) 756–779. doi:10.1109/TSE.2009.67.
[32] N. Jayaratna, Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework, McGraw-Hill, Maidenhead, UK, 1994.
[33] W. Royce, Managing the development of large software systems, in: Proceedings of WESCON, IEEE, Los Angeles, USA, 1970: pp. 1–9.
[34] B. Boehm, A spiral model of software development and enhancement, IEEE Computer. 21 (1988) 61–72.
[35] N. Suh, Axiomatic Design: Advances and Applications, Oxford University Press, New York, NY, USA, 2001.
[36] B. Fitzgerald, The Transformation of Open Source Software, MIS Quarterly. 30 (2006) 587–598.
[37] P. Ralph, Fundamentals of Software Design Science, University of British Columbia, 2010.
[38] J. Eekels, On the Fundamentals of Engineering Design Science: The Geography of Engineering Design Science. Part 1, Journal of Engineering Design. 11 (2000) 377–397.
[39] M. Maher, J. Poon, 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. (1995) 1–28.
[40] C.W. Alexander, Notes on the synthesis of form, Harvard University Press, 1964.
[41] P. Kruchten, Casting software design in the Function-Behavior-Structure Framework, IEEE Software. 22 (2005) 52–58.
[42] J.S. Gero, U. Kannengiesser, An ontological model of emergent design in software engineering, in: Proceedings of the 16th International Conference on Engineering Design, Paris, France, 2007.
[43] B. Dervin, Sense-making theory and practice: an overview of user interests in knowledge seeking and use, Journal of Knowledge Management. 2 (1998) 36–46. doi:10.1108/13673279810249369.
[44] D.M. Russell, M.J. Stefik, P. Pirolli, S.K. Card, The cost structure of sensemaking, in: Proceedings of the INTERACT and CHI Conference on Human Factors in Computing Systems, ACM, 1993: pp. 269–276. doi:10.1145/169059.169209.
[45] K.E. Weick, K.M. Sutcliffe, D. Obstfeld, Organizing and the Process of Sensemaking, Organization Science. 16 (2005) 409–421.
[46] D.A. Schön, The reflective practitioner: how professionals think in action, Basic Books, USA, 1983.
[47] A. Colman, A Dictionary of Psychology, 3rd ed., Oxford University Press, 2008.
[48] K. Dorst, N. Cross, Creativity in the design process: co-evolution of problem–solution, Design Studies. 22 (2001) 425–437. doi:10.1016/S0142-694X(01)00009-6.
[49] N. Cross, K. Dorst, N. Roozenburg, Research in design thinking, Delft University Press, 1992.
[50] G. Goldschmidt, The dialectics of sketching, Creativity Research Journal. 4 (1991) 123–143.
[51] N. Berente, K. Lyytinen, The Iterating Artifact as a Fundamental Construct for Information System Design, in: Proceedings of the First International Conference on Design Science in Information Systems and Technology, Claremont, California, USA, 2006.
[52] J.S. Gero, U. Kannengiesser, A Function-Behavior-Structure Ontology of Processes, Artif. Intell. Eng. Des. Anal. Manuf. 21 (2007) 379–391. doi:10.1017/S0890060407000340.
[53] P. Galle, The Ontology of Gero's FBS Model of Designing, Design Studies. 30 (2009) 321–339.
[54] P. Kruchten, The Rational Unified Process: An Introduction, 3rd ed., Addison-Wesley Professional, 2004.
[55] K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004.
[56] K. Beck, Extreme programming eXplained: Embrace change, 2nd ed., Addison Wesley, Boston, MA, USA, 2005.
[57] V. Rajlich, Changing the paradigm of software engineering, Communications of the ACM. 49 (2006) 67–70. doi:10.1145/1145287.1145289.
[58] P. Ralph, Comparing Two Software Design Process Theories, in: R. Winter, J.L. Zhao, S. Aier (Eds.), Proceedings of the International Conference on Design Science Research in Information Systems and Technology, Springer, St. Gallen, Switzerland,

2010: pp. 139–153.
[59] C.W. Churchman, The design of inquiring systems: Basic concepts of systems and organization, Basic Books, New York, 1971.
[60] E.A. Singer, Experience and Reflection, University of Pennsylvania Press, 1959.
[61] L. Qian, J.S. Gero, Function-behavior-structure paths and their role in analogy-based design, Artif. Intell. Eng. Des. Anal. Manuf. 10 (1996) 289–312.
[62] R. Sosa, J. Gero, Social models of creativity: Integrating the DIFI and FBS frameworks to study creative design, in: J. Gero, M. Maher (Eds.), Computational and Cognitive Models of Creative Design VI, Key Centre of Design Computing and Cognition, University of Sydney, 2005: pp. 19–44.
[63] J.S. Gero, U. Kannengiesser, A function-behaviour-structure ontology of processes, in: Proceedings of Design Computing and Cognition, Springer, 2006: pp. 407–422.
[64] L. Dube, G. Pare, Rigor in information systems positivist case research: Current practices, trends and recommendations, MIS Quarterly. 27 (2003) 597–635.
[65] P. Ralph, Developing and evaluating software engineering process theories, in: Proceedings of the International Conference on Software Engineering, Florence, Italy, 2015.
[66] R.K. Yin, Case study research: Design and methods, 4 ed., Sage, California, USA, 2008.
[67] M.S. Poole, A.H. Van de Ven, Empirical Methods for Research on Organizational Decision-Making Processes, in: Nutt, P. C., D. Wilson (Eds.), The Blackwell Handbook of Decision Making, Blackwell, Oxford, 2010: pp. 543–580.
[68] M. Poole, A.H. Van de Ven, K. Dooley, M.E. Holmes, Organizational change and innovation processes theory and methods for research, Oxford University Press, New York, NY, USA, 2000.
[69] R.A. Wolfe, Organizational innovation: review, critique and suggested research directions, Journal of Management Studies. 31 (1994) 405–431. doi:10.1111/j.1467-6486.1994.tb00624.x.
[70] J. Brewer, A. Hunter, Foundations of Multimethod Research: Synthesizing Styles, 2nd ed., SAGE Publications, 2006.
[71] J.R. Platt, Strong inference, Science. 146 (1964).
[72] R.F. DeVellis, Scale Development: Theory and Applications, 3rd ed., Sage, Thousand Oaks, CA, USA, 2011.
[73] F.J. Fowler, Improving survey questions: Design and evaluation, Sage, Thousand Oaks, CA, USA, 1995.
[74] D.W. Straub, Validating instruments in MIS research, MIS Quarterly. 13 (1989) 147–169.
[75] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir Software Eng. 14 (2009) 131–164. doi:10.1007/s10664-008-9102-8.
[76] W. Trochim, Research Methods Knowledge Base, Atomic Dog Publishing, Cincinnati, OH, USA, 2001.
[77] A.S. Lee, R.L. Baskerville, Generalizing generalizability in information systems research, Information Systems Research. 14 (2003) 221–243.
[78] Great Britain Office of Government Commerce, Managing successful projects with PRINCE2, Stationery Office Books, 2009.
[79] P. Ralph, R. Mohanani, Is Requirements Engineering Inherently Counterproductive? in: Proceedings of the 5th International Workshop on the Twin Peaks of Requirements and Architecture, Florence, Italy, 2015.
[80] P. Ralph, The Illusion of Requirements in Software Development, Requirements Eng. 18 (2013) 293–296. doi:10.1007/s00766-012-0161-4.
[81] P. Ralph, Y. Wand, A Proposal for a Formal Definition of the Design Concept, in: K. Lyytinen, P. Loucopoulos, J. Mylopoulos, W. Robinson (Eds.), Design Requirements Engineering: a Ten-Year Perspective, Springer-Verlag, Cleveland, OH, USA, 2009: pp. 103–136. doi:10.1007/978-3-540-92966-6_6.
[82] R. Mohanani, P. Ralph, B. Shreeve, Requirements Fixation, in: Proceedings of the International Conference on Software Engineering, ACM, Hyderabad, India, 2014: pp. 895–906.
[83] N. Cross, Design cognition: results from protocol and other empirical studies of design activity, in: C. Eastman, W.C. Newstetter, M. McCracken (Eds.), Design Knowing and Learning: Cognition in Design Education, Elsevier Science, Oxford, UK, 2001: pp. 79–103.
[84] E. Sober, Testability, Proceedings and Addresses of the American Philosophical Association. 73 (1999) 47–76.
[85] W. Van Orman Quine, Pursuit of Truth, Harvard University Press, 1990.
[86] I. Lakatos, The Methodology of Scientific Research Programmes: Volume 1, Cambridge University Press, 1978.
[87] K. Popper, The Logic of Scientific Discovery, Basic Books, New York, NY, USA, 1959.
[88] D. Avison, G. Fitzgerald, Information Systems Development: Methodologies, Techniques and Tools, Third edition, McGraw-Hill Education, New York, 2003.
[89] T. Hall, N. Baddoo, S. Beecham, H. Robinson, H. Sharp, A systematic review of theory use in studies investigating the motivations of software engineers, ACM Trans. on Soft. Eng. and Methodology. 18 (2009) 1–29. doi:10.1145/1525880.1525883.
[90] J. Parsons, C. Saunders, Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse, IEEE Transaction on Software Engineering. 30 (2004) 873–888.
[91] P. Ralph, Toward a Theory of Debiasing Software Development, in: S. Wrycza (Ed.), Research in Systems Analysis and Design: Models and Methods: 4th SIGSAND/PLAIS EuroSymposium 2011, Springer, Gdansk, Poland, 2011: pp. 92–105.
[92] P. Ralph, P. Kelly, The Dimensions of Software Engineering Success, in: Proceedings of the International Conference on Software Engineering, ACM, Hyderabad, India, 2014: pp. 24–35.
[93] K. Molokken, M. Jørgensen, A review of software surveys on software effort estimation, in: Proceedings of the International Symposium on Empirical Software Engineering, ACM-IEEE, 2003: pp. 223–230.
[94] P. Ralph, Improving coverage of design in information systems education, in: Proceedings of the 2012 International Conference on Information Systems, AIS, Orlando, FL, USA, 2012.
[95] J. Grudin, J. Pruitt, Personas, Participatory Design and Product Development: An Infrastructure for Engagement, in: Proceedings of the Participatory Design Conference, Participatory Design Conference 2002: pp. 141–161.
[96] T. Proctor, Creative problem solving for managers: developing skills for decision making and innovation, 3rd ed., Routledge, Oxon,

UK, 2010.
[97] M. Prats, S. Lim, I. Jowers, S.W. Garner, S. Chase, Transforming shape in design: observations from studies of sketching, Design Studies. 30 (2009) 503–520.
[98] 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, 2013: pp. 35–38. doi:10.1109/GTSE.2013.6613868.
[99] M. Bergman, K. Lyytinen, G. Mark, Boundary objects in design: An ecological view of design artifacts, Journal of the Association for Information Systems. 8 (2007) 546–568.
[100] D.M. Wegner, Transactive memory: A contemporary analysis of the group mind, in: B. Mullen, G.R. Goethals (Eds.), Theories of Group Behavior, Springer, New York, 1987: pp. 185–208.
[101] E. Bjarnason, K. Smolander, E. Engström, P. Runeson, Alignment practices affect distances in software development: a theory and a model, in: Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering, ACM, 2014: pp. 21–31.

Appendices

Appendix A: Example Interview Questions

The following is an example case study interview guide. The precise questions asked were tailored to each case and each participant; however, these questions provide a sense of the core topics. Follow-ups and probes are not included.

1) Positioning the individual
   a) What is your position at [company name]?
   b) What project(s) are you involved with?
   c) What are you building?
   d) What are your responsibilities?
   e) What are your roles in this project?
2) Organization of Work
   a) How is software development organized in your company?
   b) What are the important roles?
   c) Who is involved?
   d) What are their respective responsibilities?
   e) What tools do you use?
   f) What kind of documents do you use?
   g) What kind of models or diagrams do you use?
   h) Who are your clients, customers or users?
3) Individual Activities
   a) What do you spend most of your time doing?
   b) On the first day of the last [iteration/cycle/sprint/whatever is indicated in theme one], what is the first thing you did?
   c) The second?
   d) Then what?
   e) Etc.
4) Conclusion
   a) Is there anything else you want to tell me?
   b) Is there anything else you think I should know?

Appendix B: Coding Scheme

Table 15 illustrates how qualitative evidence was categorized and organized. The completed table is too large to show, for example, the cell for Case 1 / for / Coevolution includes 1120 words.

Table 15
Coding scheme.

	Function-Behavior-Structure Theory							
	Evidence For				Evidence Against			
Theory Component	Case 1	Case 2	Case 3	Case 4	Case 1	Case 2	Case 3	Case 4
Functions								
Expected Behaavior								
Predicted Behavior								
Structure								
Formulation								
Synthesis								
Analysis								
Evaluation								
Reformulation								
Design Description								
Tight Coupling								

	Sensemaking-Coevolution-Implementation Theory							
	Evidence For				Evidence Against			
Theory Component	Case 1	Case 2	Case 3	Case 4	Case 1	Case 2	Case 3	Case 4
Sensemaking								
Coevolution								
Implementation								
Context								
Context Schema								
Goals								
Design Space Schema								
Design Object								
Primitives								
Loose Coupling								

Appendix C: Questionnaire Items

Although the actual questionnaire was conducted online, its content is more clear in the following paper version than it would be from screenshots [continued on next page].

About You and Your Project

Please answer all questions to the best of your knowledge. If you don't know the answer to a question, skip it, but please do not skip questions unnecessarily. The survey should take 10 to 15 minutes.

1. What is your gender?
   - ◯ Male
   - ◯ Female
   - ◯ Other

2. What is the highest level of education you have attained?

Some school
High school Diploma
Some college, university, trade school, etc.
Diploma from technical college, trade school, etc.
Bachelor's degree
Master's degree
PhD

3. How long have you been involved in the software development industry? (In-house, off-the-shelf, for-client, etc.)

less than 1 year
1 to 5 years
6 to 10 years
11 to 15 years
16 to 20 years
21 to 25 years
more than 25 years

4. What is your current occupation (check all that apply):
   - ☐ business or requirements analyst
   - ☐ graphics designer / animator
   - ☐ team lead
   - ☐ project manager
   - ☐ programmer / developer
   - ☐ quality assurance / tester
   - ☐ other

5. How many employees does your company have? (Required)

1 - 10
11 - 100
101 - 1000
1001 - 10 000
More than 10 000
Not applicable / Don't Know

6. Definition: For the purposes of this survey, your project is the software development project on which you are currently working. If you are not currently working on a project, your project is the last one you worked on. If you are working on multiple projects, your project is the one on which you spend the most time. Please answer all questions based on the same project.

What roles have you held in your project? (check all that apply) (Required)

- [ ] business or requirements analyst
- [ ] graphics designer / animator
- [ ] project manager
- [ ] programmer / developer
- [ ] quality assurance / tester
- [ ] other - please list:

7. Definition: For the purposes of this survey, your team consists of everyone with whom you collaborate closely on your project, regardless of their geographic location or official title. Your team is all the people you feel like you work with.

Approximately how many people are currently on your development team? (If the project is complete, what was the largest number of people who were on the team at one time?)

8. Please list any development methods your team is explicitly using (e.g., RUP, Extreme Programming, SCRUM, Service Oriented Architecture).

9. Approximately how long has your project been in progress?

years:

months:

10. Is your project more "social" (like a website) or "technical" (like a device driver)?

- ( ) More Social
- ( ) Somewhere in between
- ( ) More Technical
- ( ) Don't know

Propositions 1 and 3

11. Please indicate the extent to which you agree with the following statements. (Required)

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	Not applicable / don't know
No one thing drives all design decisions – they are made based on a variety of information	○	○	○	○	○	○
Changes to my team's understanding of what the software is supposed to do were triggered by changes in our understanding of the problem/situation	○	○	○	○	○	○
My understanding of what the software is supposed to do has been influenced by several factors (e.g., management, marketing, clients, the dev team, standards, my own values, experience on previous products, etc.)	○	○	○	○	○	○
My understanding of the software's purpose has been influenced by several factors (e.g., management, marketing, clients, the dev team, standards, my own values, experience on previous products, etc.)	○	○	○	○	○	○
The process of designing the software has NOT helped my team better understand the context in which the software is intended to be used	○	○	○	○	○	○
A complete, correct specification of low-level design decisions was available before coding began (*e.g., whether to use a hashtable or array to store usernames)	○	○	○	○	○	○
The software was coded iteratively	○	○	○	○	○	○
My team has revised the software code based on new information (e.g., bug reports, failed unit tests, feedback from Quality Assurance, etc.)	○	○	○	○	○	○
My team now understands what the software is supposed to do better than we did when we started coding	○	○	○	○	○	○
Low-level design decisions* were primarily made before the first line of code was written (*e.g., whether to use a hashtable or array to store usernames)	○	○	○	○	○	○

Appendix D: Extended Methodology

Many methodologists and statisticians disagree on whether Likert and semantic differential scales produce interval or ordinal data and, consequently, on which statistical tests to apply. In the interest of caution, this paper uses nonparametric tests. Nonparametric tests (including the chi-square goodness of fit test) may be used to evaluate the statistical significance of an observed distribution compared to an expected distribution. As no *a priori*, theoretically-justified distribution is available, the "expected distribution" must be generated somehow. Three alternatives are apparent:

1. Uniform distribution - on any given item, responses split evenly between all categories.
2. Pseudo-normal distribution - an approximated normal distribution on a five point scale.
3. FBS-supporting distribution - a distribution that favors FBS to the same extent that the observed distribution favors SCI.

The uniform and normal distributions are automatically generated from the data by the SPSS Statistics software package when using the Kolmogorov-Smirnov (K-S) Test. The FBS-supporting distribution was generated by inverting the observed distribution (i.e. subtracting each response from six). Using the normal distribution addresses the question *is the extent of negative skew in the observed distribution significant?* Using the FBS-supporting distribution addresses the question *is the observed distribution significantly different from an equally compelling distribution supporting the alternative hypothesis?*

However, generating a pseudo-normal distribution using the K-S test involves calculations (e.g. mean) inconsistent with interpreting Likert scales as ordinal data; therefore, these statistics should be interpreted with caution. The most defensible test is the chi-square goodness of fit test (with significance via the sign test) using the reflected (FBS-supporting) distributions as this directly compares Hypotheses H_1 and H_2 with minimal assumptions. Given the sample size and magnitude of skewness of the observed distribution, these results are fairly insensitive to changes in the expected distribution. Table 16 shows the results of testing the observed distributions against the three alternative theoretical distributions.

Table 16. Chi-Square Test Results

	Uniform Distribution		Pseudo-Normal Distribution		FBS-Supporting Distribution	
Item	K-S Test Z	Significance	K-S Test Z	Significance	Sign Test Z	Significance
1	24.59	$p < 0.001$	10.45	$p < 0.001$	-33.49	$p < 0.001$
2	21	$p < 0.001$	10.35	$p < 0.001$	-29.9	$p < 0.001$
3	23.21	$p < 0.001$	9.851	$p < 0.001$	-32.21	$p < 0.001$
4	21.9	$p < 0.001$	9.727	$p < 0.001$	-29.92	$p < 0.001$
5	15.92	$p < 0.001$	10.25	$p < 0.001$	-20.47	$p < 0.001$
6	23.51	$p < 0.001$	12.64	$p < 0.001$	-31.45	$p < 0.001$
7	21.32	$p < 0.001$	9.648	$p < 0.001$	-28.53	$p < 0.001$
8	24.72	$p < 0.001$	12.57	$p < 0.001$	-33.18	$p < 0.001$
9	22.12	$p < 0.001$	9.677	$p < 0.001$	-30.48	$p < 0.001$
10	17.7	$p < 0.001$	9.837	$p < 0.001$	-22.13	$p < 0.001$
11	13.65	$p < 0.001$	10.82	$p < 0.001$	-20.1	$p < 0.001$
12	12.94	$p < 0.001$	10.35	$p < 0.001$	-18.84	$p < 0.001$
13	17.5	$p < 0.001$	8.782	$p < 0.001$	-27.87	$p < 0.001$