Comparing Two Software Design Process Theories

Paul Ralph[1]

[1]Lancaster University, United Kingdom
paul@paulralph.name
http://paulralph.name

Abstract. This paper explores an ongoing conflict concerning the nature of software design. This conflict manifests itself as antagonism between managers and developers, debates about agile vs. plan-driven methodologies and aspiring developers' dissatisfaction with their courses. One side views design as a plan-driven information processing task involving rational decision-making (the Reason-Centric Perspective), while the other views design as an improvised, creative task involving naturalized decision-making (Action-Centric Perspective). Each perspective includes an epistemology, theory of human action and a software design process theory (an explanation of how software is created in practice). This paper reports the results of an exploratory questionnaire study that comparatively and empirically evaluated the two process theories. Results clearly favor the Action-Centric process theory: the Sensemaking-Coevolution-Implementation Framework.

Keywords: Design Science, Process Theory, Software Design, Questionnaire

1 Introduction

Software design science is the philosophical, theoretical and empirical study of software creation and modification including its phenomenology, methodology and causality. It is distinct from the "design-science research paradigm" [1, 2], where design is a research method. A key element of design science involves theories of the shape and organization of the design process [3]. Yet, the shape and organization of the design process of software, in particular, is not well understood [3-6], as most academic work on software design is prescriptive rather than explanatory or descriptive [see 7, 8] – hence, my primary research question (as follows).

> **Research Question:** *What is the process by which development teams create software in practice?*

To address this question, I evaluate competing process theories of software design, in terms of their descriptive and explanatory validity. A process theory is "an explanation of how and why an organizational entity changes and develops" [9, p. 512]. Process theories are distinct from process models – "A process model is an abstract description of an actual or proposed process" [10, p. 76]. A process theory seeks to explain how outcomes materialize *in general*, not simply one or several historical or possible activity sequences. Following [11], software design (verb) is the act of creat-

ing a specification of a software object, by an agent, intended to accomplish goals in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to set of constraints.

This research contributes to the field of *software design science,* the philosophical, theoretical and empirical study of the creation of virtual artifacts, including performance (phenomenology), methods, tools and practices (methodology), and antecedents and outcomes (causality). This should not be confused with the *design science research paradigm* [1, 2, 12], where knowledge is created by building and evaluating technological artifacts.

This paper is organized as follows. Section two categorizes existing research on software design into two mutually-exclusive clusters of interrelated theoretical and philosophical concepts (Reason-Centric and Action-Centric Perspectives). Section three describes the design and results of a survey study that comparatively tested process theories from both perspectives. Section four summarizes the contributions of this phase of the study and outlines the next.

2 Two Perspectives on Software Design

This section summarizes the Reason- and Action-Centric Perspectives [8], and defines the process theories used to operationalize each perspective.

2.1 The Reason-Centric Perspective

"According to the model of Technical Rationality – the view of professional knowledge which has most powerfully shaped both our thinking about the professions and the institutional relations of research, education, and practice – professional activity consists in instrumental problem-solving made rigorous by the application of scientific theory and technique," [13, p. 21]. Technical Rationality requires given problems – goals are agreed in advance and constraints are knowable. Schön argues that Technical Rationality is foundational to both positivism [14] and the Technical Problem-Solving design paradigm [3]. The latter posits that professionals design by optimizing or "satisficing" a design candidate vis-à-vis known constraints and objectives. They engage in rational decision making – choosing the best option from a known set [15].

Technical Rationality and the Technical Problem-Solving paradigm are consistent with the *cognitivist view* of human action, wherein actions are executed and understood through a plan and defined as "a sequence of actions designed to accomplish some preconceived end" [16]. Plans are prerequisites to action. Unanticipated conditions trigger replanning; evaluation is performed by comparing resulting and planned actions and outcomes. In this view, design is a form of plan-driven problem-solving, where an agent seeks a goal state by executing a plan within a field of constraints [17]. Moreover, this view is guided by an Information Processing metaphor – "The designer is seen as a machine capable of rationally selecting and connecting together elemental information to satisfy a set of constraints" [18, p. 309].

The Function-Behavior-Structure (FBS) Framework [19, 20] is an engineering design process theory, broadly consistent with the above. Figure 1 shows one representation of the FBS Framework; Tables 1 and 2 define its artifacts and processes.

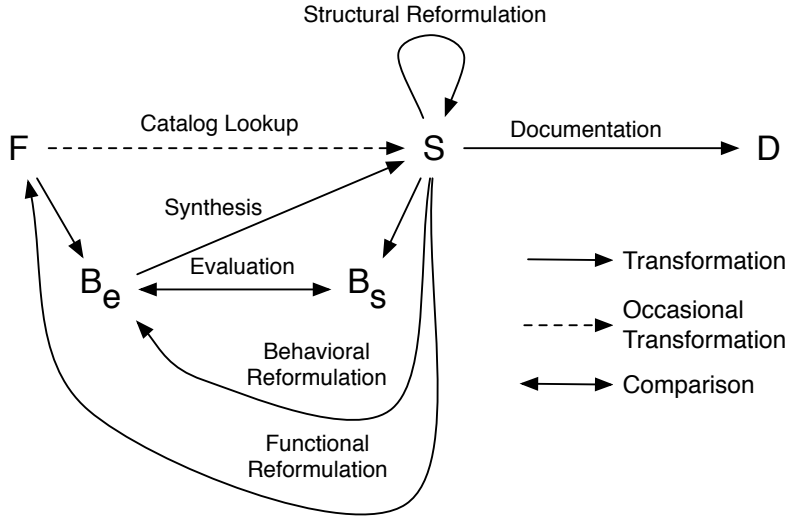

Fig 1. The Function-Behavior-Structure-Framework

Table 1. Artifacts of the FBS Framework (adapted from [19])

Symbol	Meaning
Be	expected (desired) behavior of the structure
Bs	"the predicted behavior of the structure" (p. 3)
D	a graphically, numerically and/or textually represented model that transfers "sufficient information about the designed artefact so that it can be manufactured, fabricated or constructed" (p. 2)
F	"the expectations of the purposes of the resulting artefact" (p. 2)
S	"the artefact's elements and their relationships" (p. 2)

The core claim of the FBS Framework is that "the purpose of designing is to transform *function*, F (where F is a set), into a *design description*, D, in such a way that the artefact being described is capable of producing those functions," [19, p. 2, original italics]. Gero posited three "intermediate artifacts" – structure (S), structure's behavior, (B_s) and expected behaviors (B_e). Gero and Kannengiesser (2002) situated function, behavior and structure in three different "worlds" – desired, internal and external – where each concept exists in each world (e.g., desired functions, the designer's interpretation of the functions of the current design candidate, and external representations of said interpretations). Kruchten [21] claimed that software design may be "cast" in the FBS Framework and consequently mapped the Rational Unified Process [22] and Waterfall Model [23] onto it.

Table 2. Operations of the FBS Framework (adapted from [19])

Operation	Inputs	Outputs	Meaning
Analysis	S	B_s	the process of deriving the behavior of a structure
Catalog Lookup	F	S	selecting a known structure that performs the required function
Evaluation	B_s & B_e	Differences Between B_s and B_e	comparing predicted behavior to expected behavior and determining whether the structure is capable of producing the functions
Formulation	F	B_e	deriving expected (desired) behaviors from the set of functions
Production of Design Documentation	S	D	transforming structure into design description suitable for manufacturing
Synthesis	B_e	S & B_s	"expected behavior is used in the selection and combination of structure based on a knowledge of the behaviors produced by that structure" (p. 3)

2.2 The Action-Centric Perspective

Social constructivism posits that knowledge is derived from social interactions [24]. Building from social constructivism and empirical studies of professional practice, Schön [13] devised the Reflection-in-Action design paradigm, where design is a reflective conversation between the designer and the situation. The designer alternates between framing (conceptualizing the problem), making moves (where a move is a real or simulated action intended to improve the situation) and evaluating moves. Multiple agents may collectively reflect in action using boundary objects [25].

Schön [13] argued that "when someone reflects in action, … he does not keep means and ends separate … he does not separate thinking from doing" (p. 69). This idea is elemental to the *ethnomethodological view* of human action (ethno-view), in which "the organization of situated action is an emergent property of moment-by-moment interactions between actors, and between actors and the environments of their action" [16, p. 179], while "plans are representations, or abstractions over action" (p. 186). Both Reflection-in-Action and the ethno-view imply that innovation is based on the creativity and experience of the designer; consequently, the guiding metaphor is creativity [18] and decision making is naturalistic [26].

The Sensemaking-Coevolution-Implementation (SCI) Framework [27, 28] (Figure 2, Table 3) is a design process theory that is broadly consistent with the above concepts. Unlike the FBS Framework, the SCI Framework is specific to software design. Its core claim is that software design includes three primary activities (in no set order) – making sense of context, iteratively evolving mental pictures of context and software artifact, and writing code based on the mental picture of the software.

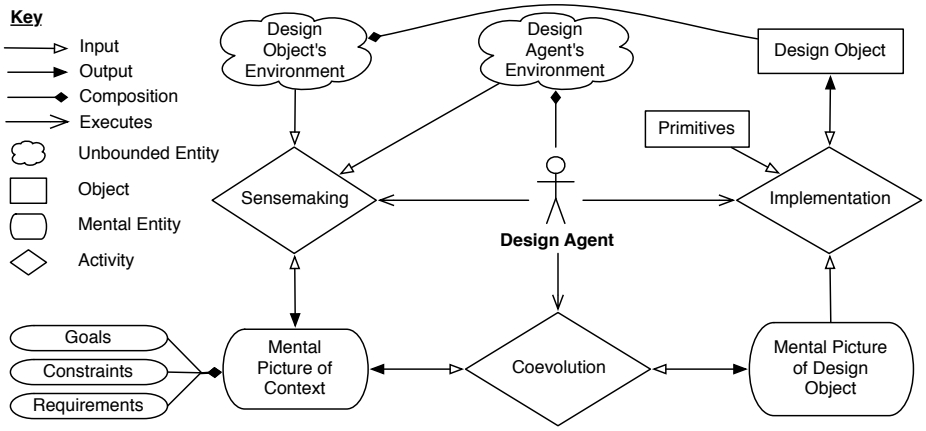

Fig 2. The Sensemaking-Coevolution-Implementation Framework

2.3 Comparative Analysis of Perspectives and Process Theories

Perspectives in Conflict. Table 4 contrasts the Reason- and Action-Centric Perspectives, which largely constitute the assumptions of the FBS and SCI Frameworks. The conflict between these perspectives is evident in Beck's discussion of common tensions between managers who try to drive projects through cost estimates and developers who cannot reliably estimate complex projects [29]. Similarly, Graham explains misalignment between programming education and practice – "I was taught in college that one ought to figure out a program completely on paper before even going near a computer. I found that I did not program this way.... I tended to just spew out code that was hopelessly broken, and gradually beat it into shape" [30]. Moreover, "the concept of method ... occupies an extremely privileged status in formal information systems development thought" while "the possibility that amethodical development might be the normal way" of building systems has "almost entirely elud[ed] the systems development literature" [31, p.54, 58]. The Reason-Centric perspective has occupied an analogously privileged status in design research despite little empirical evidence concerning its assumptions and ramifications.

Process Theory Similarities. Despite differing assumptions, the two process theories are similar in several ways:

1. They are both teleological process theories; i.e., 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, and monitors progress* [9, 32, 33]. Therefore, they share fundamental aspects of teleological process theories, including goals and an agent.
2. They share fundamental design concepts; e.g., the FBS Framework's *expected behavior* and *structure* concepts are similar to the SCI Framework's *requirements* and *mental picture of the design object* concepts (see Tables 1 and 3).

Table 3. Concepts of the SCI Framework (adapted from [28])

Concept	Meaning
Constraints	a restriction on a structural or behavioral property of the *design object*
Design Agent	an entity or group of entities that is 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's Environment	the totality of the surroundings in which the *design object* exists or is intended to exist
Design Agent's Environment	the totality of the surroundings of the *design agent*
Design Object	a (possibly incomplete) manifestation of the *mental picture of design object*, composed of *primitives*, in the *design object's environment*
Goals	optative statements (which may exist at varying levels of abstraction) about the effects the *design object* should have on the *design object's environment*
Mental Picture of Context	the collection of all beliefs, held by the *design agent*, regarding the *design agent's environment* and the *design object's environments*
Mental Picture of Design Object	the collection of all beliefs held and decisions made by the *design agent* concerning the *design object*
Primitives	the set of entities from which the *design object* may be composed
Requirements	a structural or behavioral property that a design object must possess
Sensemaking	the process where the *design agent* perceives its *environment* and the *design object's environment* and organizes these perceptions to create or refine the *mental picture of context*
Coevolution	the process where the *design agent* simultaneously refines its *mental picture of design object* based on its *mental picture of context*, and vice versa
Implementation	the process where the *design agent* generates or updates a *design object* using its *mental picture of design object*

Table 4. Comparison of Reason- and Action-Centric Perspectives

Dimension	Reason-Centric Perspective	Action-Centric Perspective
Epistemology	Positivist	Constructivist
Theory of Action	Cognitivist	Ethnomethodological
Design Paradigm	Technical Problem-Solving	Reflection-in-Action
Decision Making	Rational	Naturalistic
Guiding Metaphor	Information Processing	Creativity
Process Theory	FBS Framework	SCI Framework

1. *Both frameworks are consistent with models*. In the FBS Framework, the designer necessarily creates an external representation of the design artifact's structure and may also model functions and behaviors. In the SCI Framework, the design agent may model both the mental pictures of the context (conceptual models) or the design object (design models).

Process Theory Differences. Notwithstanding these similarities, the two theories differ in at least three ways.

1. Whether problem setting and problem solving are separate (FBS Framework) or cotemporal and inextricably linked (SCI Framework)
2. Whether the coding process is driven by prefigured decisions (FBS Framework) or evolves iteratively with the design process (SCI Framework)
3. Whether designers focus on models (FBS Framework) or code (SCI Framework)

The first difference results from the conflicting design paradigms underlying the two theories, the second from their dissimilar theories of action, the third from the differing guiding metaphor. Therefore, comparatively testing the two process theories on these dimensions may give insight into the descriptive validity of the underlying assumptions encompassed by the Action and Reason-Centric Perspectives.

3 Research Design and Results

Taking a comparative approach to testability [34, 35], my original research question may now be operationalized as *Which of the FBS and SCI Frameworks more accurately describes how software is created in practice?*

My literature review did not uncover any previous empirical evaluations of either theory in the software domain. Moreover, I uncovered little methodological advice on evaluating process theories. However, Wolfe [36] identified two common approaches to studying innovation processes – cross-sectional surveys and in-depth field studies. Considering the similarity between design and innovation, it would seem reasonable to adopt these methods here. Furthermore, combining the two approaches enables multi-method triangulation – the survey (phase 1) allows for larger sample size and reliability while the field study (phase 2) facilitates gathering deep insights into developer behaviors and cognitive processes. This paper focuses on the survey, which I designed based on well-known guidelines [37-39].

3.1 Hypothesis

I hypothesize that the SCI Framework is more accurate, as its underlying design paradigm (Reflection in Action) and theory of human action (Ethno-View) are better supported by empirical studies than their Reason-Centric alternatives [13, 16].

Hypothesis H1: *The SCI Framework more accurately reflects how software is created in practice than the FBS Framework.*

3.2 Instrument Development and Validation

The steps in the instrument development and validation were as follows.

1. The author identified differences between the two theories.
2. A colleague with expert knowledge of software design reviewed these differences, finding no bias in the interpretation of either theory.
3. The author generated approximately 80 items concerning these differences.
4. Items were reviewed by two MIS faculty, one with extensive experience in questionnaire-based research, the other with extensive knowledge of design.
5. A pilot was conducted with three professional developers and seven MIS PhD students to get research-oriented feedback. Items were revised.
6. A second pilot with 12 professional developers was conducted. Results indicated that the questionnaire was too long and difficult to understand. Most items were dropped; remaining items were simplified.
7. A third pilot with 10 professional developers was conducted. Minor revisions were made, resulting in the final version of the instrument.

Following this process, the questionnaire comprised 13 items (listed in the Appendix). Each item was constructed with six responses: one strongly supporting each framework; one supporting each framework; one neutral; one "Not Applicable / Don't know." The question order was randomized; the answer order varied by question. Please note, these items are **not** reflective indicators of latent constructs. Differences between process theories are not latent constructs and items do not reflect these differences as much as describe certain behaviors and attitudes related to the differences. For example, from Difference 1 (whether problem setting and solving are separate), the survey included the item "The process of designing the software has NOT helped my team better understand the context in which the software is intended to be used."

3.3 Sampling and Administration

The population of interest includes all members of all software development teams, worldwide. However, for practical reasons, I limit the sample to English speakers. Moreover, having no comprehensive population list, random sampling was impractical. Instead, participants were recruited through posts on popular software development blogs and through Twitter. The questionnaire was administered online.

Between December 2, 2009 and January 11, 2010, 1384 participants responded to the survey. The response rate cannot be calculated since, as in snowball sampling, the sample size is undefined. However, of the 4410 individual visitors to the survey page, 1384 completed it (31%), 1118 partially completed it and 1908 bounced (looked at the survey's front page and then left). Table 5 summarizes their key demographics.

Responses were received from 65 countries across six continents with concentrations in the United States (549), Canada (176), United Kingdom (118) and Australia (73). Participants indicated fulfilling varied roles (they could choose several), including developer (1325), analyst (569), quality assurance specialist (533), manager (266) and graphics designers (195). Respondents reported using a wide variety of agile (e.g., Scrum), plan-driven (e.g., the Rational Unified Process) and homegrown methodologies. When asked "Is your project more 'social' (like a website) or 'technical'

(like a device driver)", participants answered: more social - 34%; more technical - 29%; in between - 36%.

Table 5. Summary of Sample Demographics

Dimension	Mode	Minimum	Maximum
Years of Experience	1 to 5 years* (31.5%)	< 1 year (2.9%)	> 25 years (3.6%)
Education	Bachelor's Degree* (48%)	Some School (1.7%)	PhD (4.1%)
Company Size	1 to 10 (29%)	1 to 10 (29%)	>10 000 (10.5%)
Dimension	Mean	Standard Deviation	Range
Team Size	11 members	83 members	3000 members
Project Length	1.9 years	2.4 years	20 years

3.4 Results

Before presenting the results, I enumerate the possible patterns and their interpretations, assuming responses are coded from 1 (strong support for the FBS Framework) to 5 (strong support for the SCI Framework).

1. A symmetric distribution (median of 3) would indicate that neither framework is substantially more accurate than the other.
2. A positively-skewed distribution (median of 1 or 2) favors the FBS Framework.
3. A negatively-skewed distribution (median of 4 or 5) favors the SCI Framework.
4. A bimodal distribution (e.g., modes of 2 and 4) would indicate that developers can be categorized into two groups, one supporting each framework.
5. A combination of symmetric, positively and negatively skewed items would suggest a problem with the survey instrument.

The results are given in Table 6 (please note: columns do not total 1384 as each question had a "N/A" option). It is clear from inspecting Figure 3 that the overall distribution favors the SCI Framework. The same pattern is observed at both the item level (each item had a median response of 4 or 5) and the individual level (96.6% of respondents had a median response of 4 or 5). In summary, the response distribution is negatively skewed, supporting the SCI Framework.

Table 6. Questionnaire Results

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

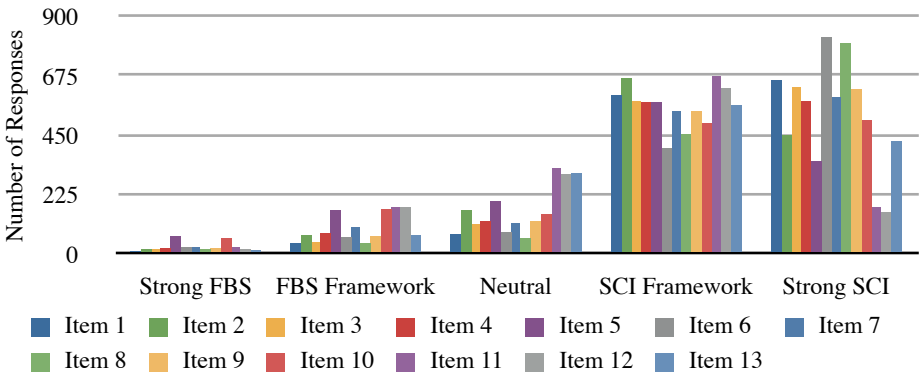

Fig 3. FBS/SCI Agreement Across 13 Items

Many methodologists and statisticians disagree as to whether Likert scales of the kind used in this research produce interval or ordinal data, and consequently as to whether to apply parametric or nonparametric tests [40]. Here, I take the more cautious route, treating the data as ordinal.

Nonparametric tests (such as chi-square) require an expected distribution to compare with the observed distribution. Since there is no *a priori* "FBS-supporting distribution", I generated one (for each item) by reflecting the observed distribution (subtracting each response from 6). The resulting chi-square statistics (with significance via the sign test) indicate that the observed distribution of each item is significant at $p < 0.001$ (Table 7). This answers the question, 'is the observed distribution significantly different from an equally compelling distribution supporting the alternative hypothesis?' Substituting normal and uniform distributions produced similar results.

Table 7. Chi-square Test Results - Observed vs. Reflected Distribution

Item	Z	Asymp. Sig. (2-tailed)	Item	Z	Asymp. Sig. (2-tailed)
1	-33.49	$p < 0.001$	8	-33.18	$p < 0.001$
2	-29.90	$p < 0.001$	9	-30.48	$p < 0.001$
3	-32.21	$p < 0.001$	10	-22.13	$p < 0.001$
4	-29.92	$p < 0.001$	11	-20.10	$p < 0.001$
5	-20.47	$p < 0.001$	12	-18.84	$p < 0.001$
6	-31.45	$p < 0.001$	13	-27.87	$p < 0.001$
7	-28.53	$p < 0.001$			

In addition to the thirteen items, several demographic and project variables were included in the questionnaire, including gender, education, experience, nationality, occupation, team size, project duration, firm size, methodologies in use, and the nature of the software. Although space does not permit a thorough presentation of the analysis, *none of these variables had a measurable effect on individuals' overall agreement with the FBS or SCI Framework*.

4 Implications for Research and Practice

4.1 Contributions

To the best of my knowledge, this study is not just the first empirical evaluation of the FBS and SCI Frameworks but of any software design process theory. The evidence supporting the SCI Framework (and vicariously the Action-Centric Perspective) lends further support to a growing body of evidence questioning the centrality of rational thought in design and other professional activity [e.g., 13, 18, 31, 41-43]. Moreover, the SCI Framework is immediately useful for both research, practice and teaching.

1. For researchers, it may facilitate evaluating and improving design methods, tools and practices. For example, in evaluating a design methodology (e.g., Extreme Programming), we may ask, "does this methodology provide guidance concerning all three fundamental design activities – sensemaking, coevolution and implementation?" If not, can the methodology be improved by considering those omitted?" Moreover, it may inform development of an antecedent theory of design project success. In a strict interpretation of causality, causal theories imply precedence relationships. The SCI-Framework dispenses with Waterfall-like, artificial activity sequences. Therefore, it may help eliminate extraneous causal relationships during theory building (e.g., the hypothesis that analysis quality causes design quality is incorrect *a priori* since analysis and design are cotemporal in practice).
2. For educators, it may inform evaluation and improvement of software design curricula. For example, the presented evidence implies that the SCI Framework is a better description of software design than the Waterfall Model [23] (which is a subset of the FBS Framework [21]); therefore, it may be more useful to teach the concepts of the SCI Framework in design-oriented courses.
3. For managers, it suggests that developers may resist attempts to pressure them to separate analysis from design, write code linearly or iterate on models; that implementing a tool, practice or method that is incompatible with iterative coding and simultaneous analysis and design will likely be ineffective without corresponding changes in development practices; and that managers who believe that their employees build software according to the Reason-Centric Perspective (that is, rationally) or using a Reason-Centric method (e.g., Waterfall) are likely mistaken or possibly actively being deceived. Furthermore, if developers do not understand the problems that they are solving until the solution is well into development, any upfront budget and schedule estimates lack substantive understanding of the problem. It seems incredulous that anyone could accurately estimate the cost of solving a problem without knowing what identifying the problem.

4.2 Limitations

The results of this study should be considered in light of four limitations:

1. The sample is not random and may include some bias. However, given the variety in the reported demographics, suggesting that the sample comprises only one or several fringe developer communities seems incredulous.

1. The limitations inherent to survey research, including lack of depth and responder bias, obviously apply here. Phase two of the study (described below) is designed to mitigate these shortcomings.
2. As the test was comparative, it does not indicate that the SCI Framework is unequivocally "right" or "true". It is simply more accurate than the alternative.

4.3 Future Work (Phase 2)

As mentioned above, a multimethodological research design combining a survey with one or more in-depth fields studies would provide more convincing evidence than either approach alone. Following this, the next phase of the current study involves comparatively evaluating the FBS and SCI Frameworks using a field study to corroborate (or contradict) and add nuance to the current evidence.

One form of field study with a rich methodological foundation in organizational research is the case study [c.f. 35, 44-47]. A case study is a "comprehensive research strategy" that "investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident … [and] relies on multiple sources of evidence, with data needing to converge in a triangulating fashion" [35, p. 13-14]. A case study approach is preferable when 1) the research focuses on how things are done in practice, 2) the research focuses on contemporary events, and 3) the research does not necessitate behavioral manipulations [35]. The present situation clearly meets these criteria.

I propose a three-case design comprising two literal replications and one theoretical replication (two studies of software development teams where the same result (SCI Framework superior) is predicted and one study of an engineering design team, where a different result (FBS Framework superior) is predicted). The proposed design is informed by the incisive summary of recommendations in [44]. Data collection may include interviews, recording meetings, direct observation and copying relevant artifacts (e.g., design diagrams). The resulting collection of statements, observations and artifacts can then be coded according to a closed coding scheme based on the two theories. Specifically, for each concept and relationship of each theory, related items of evidence would be classified as ether *supporting* or *opposing*. The extent of support for each theory would reflect the cumulative support for each concept and relationship. At least two coders will be used to facilitating measurement of reliability via intercoder agreement [35, 44, 46, 47].

4.4 Concluding Remarks

"The shape and organization of the design process is an essential component of a theory of design" [3, p. 130-1]. Since "the shape and organization of the design process" in the software domain is poorly understood [3-6], this study began with the question, *What is the process by which development teams create software in practice?* This question was operationalized as an empirical, survey study comparing two incompatible software design process theories; i.e., explanations of the shape and organization of the design process. The SCI Framework – in which design is modeled as an improvised, emergent activity wherein a self-directing agent alternates between three primary activities: 1) making sense of context; 2) iteratively evolving mental pictures of

context and software artifact; 3) writing code based on the mental picture of the software – was supported. Since the differences between the FBS and SCI Frameworks tested reflect differences in the assumptions comprising the Reason and Action-centric perspectives, this evidence also suggests that the Action-Centric Perspective is more consistent with the pragmatic reality of software design than the Reason-Centric Perspective. Since the Reason-Centric Perspective has held a privileged position in design literature for many years [31], this evidence calls into question much of the field's conceptual research, the potential usefulness of popular design methodologies, and the conventional wisdom surrounding how software designers are educated and how software projects are managed.

References

1. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision Support Systems 15, 251--266 (1995)
2. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28, 75-105 (2004)
3. Simon, H.A.: The Sciences of the Artificial. MIT Press, Cambridge, MA, USA (1996)
4. Freeman, P., Hart, D.: A Science of design for software-intensive systems. Communications of the ACM 47, 19-21 (2004)
5. Sullivan, K.: Preliminary Report: NSF Workshop on the Science of Design: Software and Software-Intensive Systems. University of Virginia Department of Computer Science, Airlie Center (2003)
6. Wynekoop, J., Russo, N.: Systems development methodologies: unanswered questions. Journal of Information Technology 10, (1995)
7. Wynekoop, J., Russo, N.: Studying system development methodologies: an examination of research methods. Information Systems Journal 7, 47-65 (1997)
8. Ralph, P.: The Sensemaking-Coevolution-Implementation Framework of Software Design. MIS Quarterly (under review), 76 pages (2010)
9. Van de Ven, A.H., Poole, M.S.: Explaining development and change in organizations. The Academy of Management Review 20, 510--540 (1995)
10. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Communications of the ACM 35, 75-90 (1992)
11. Ralph, P., Wand, Y.: A Proposal for a Formal Definition of the Design Concept. In: Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, W. (eds.) Design Requirements Engineering: A Ten-Year Perspective. Lecture Notes on Business Information Processing, pp. 103-136. Springer-Verlag (2009)
12. Walls, J.G., Widmeyer, G.R., El Sawy, O.A.: Building an information system design theory for vigilant EIS. Information Systems Research 3, p36 - 59 (1992)
13. Schön, D.A.: The reflective practitioner: how professionals think in action. Basic Books, USA (1983)
14. Hacking, I.: Scientific Revolutions. Oxford University Press, New York, USA (1982)
15. Luce, D., Raiffa, H.: Games and Decisions: Introduction and Critical Survey Wiley, New York, NY, USA (1957)
16. Suchman, L.: Plans and Situated Actions: The problem of human-machine communication. Cambridge University Press (1987)
17. Newell, A., Simon, H.: Human Problem Solving. Prentice-Hall, Inc. (1972)

18. Love, T.: Philosophy of Design: A Meta-theoretical Structure for Design Theory. Design Studies 21, 293-313 (2000)
19. Gero, J.S.: Design prototypes: A Knowledge Representation Schema for Design. AI Magazine 11, 26-36 (1990)
20. Gero, J.S., Kannengiesser, U.: The Situated Function-Behaviour-Structure Framework. Design Studies 25, 373-391 (2004)
21. Kruchten, P.: Casting Software Design in the Function-Behavior-Structure Framework. IEEE Software 22, 52-58 (2005)
22. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional (2003)
23. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of Wescon, (1970)
24. Berger, P., Luckmann, T.: The social construction of reality : a treatise in the sociology of knowledge. Penguin, London (1966)
25. Levina, N.: Collaborating on Multiparty Information Systems Development Projects: A Collective Reflection-in-Action View. Information Systems Research 16, 109-130 (2005)
26. Klein, G.: Sources of Power: How People Make Decisions. The MIT Press, Cambridge, MA, USA (1999)
27. Ralph, P., Wand, Y.: A Teleological Process Theory of Software Development. JAIS Theory Development Workshop, vol. 8(23), Sprouts: Working Papers on Information Systems, Paris, France (2008)
28. Ralph, P.: Theories of Software Design. MIS Quarterly (under review), 76 pages (2010)
29. Beck, K.: Extreme Programming eXplained: Embrace Change. Addison Wesley, Boston, MA, USA (2005)
30. Graham, P.: Hackers and Painters. (2003)
31. Truex, D., Baskerville, R., Travis, J.: Amethodical systems development: the deferred meaning of systems development methods. Accounting, Management and Information Technologies 10, 53-79 (2000)
32. Singer, E.A.: Experience and Reflection. University of Pennsylvania Press (1959)
33. Churchman, C.W.: The design of inquiring systems: Basic concepts of systems and organization. Basic Books, New York (1971)
34. Sober, E.: Testability. Proceedings and Addresses of the American Philosophical Association 73, 47--76 (1999)
35. Yin, R.: Case study research: Design and methods. Sage Publications, California, USA (2003)
36. Wolfe, R.A.: Organizational innovation: review, critique and suggested research directions. Journal of Management Studies 31, 405-431 (1994)
37. Straub, D.W.: Validating Instruments in MIS Research. MIS Quarterly 13, 147-169 (1989)
38. DeVellis, R.: Scale development: Theory and applications. Sage, Thousand Oaks, CA, USA (2003)
39. Fowler, F.J.: Improving survey questions: Design and evaluation. Sage, Thousand Oaks, CA, USA (1995)
40. Harwell, M.R., Gatti, G.G.: Rescaling Ordinal Data to Interval Data in Educational Research. Review of Educational Research 71, 105-131 (2001)
41. Parnas, D.L., Clements, P.C.: A rational design process: How and why to fake it. IEEE Transactions on Software Engineering 12, 251-257 (1986)
42. Nandhakumar, J., Avison, D.: The Fiction of Methodological Development: A Field Study of Information Systems Development. Information Technology & People 12, 176-191 (1999)

43. Zheng, Y., Venters, W., Cornford, T.: Agility, Improvisation and Enacted Emergence. In: International Conference on Information Systems, Montreal, Canada (2007)
44. Dube, L., Pare, G.: Rigor in Information Systems Positivist Case Research: Currect Practices, Trends and Recommendations. MIS Quarterly 27, 597-635 (2003)
45. Eisenhardt, K.M.: Building Theories from Case Study Research. The Academy of Management Review 14, 532-550 (1989)
46. Lee, A.S.: A scientific methodology for MIS case studies. MIS Quarterly 13, 33-50 (1989)
47. Benbasat, I., Goldstein, D., Mead, M.: The case research strategy in studies of information systems. MIS Quarterly 11, 369-386 (1987)

Appendix: Questionnaire Items

Participants were asked to respond to the following items on a 5 point agreement scale.
- 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)

Participants were asked to respond to the following items on a 5 point ranging from "Exclusively with models," to "Exclusively with code."
- I do detailed design...
- My team does detailed design...

Participants were asked to respond to the following item on a 5 point ranging from "Exclusively prediction," to "Exclusively observation."

Which of these is more consistent with how your team does testing? (Required)
1) Prediction: testers inspect models of the software and predict how code based on those models will behave (e.g., predict from a UML class diagram how the code will handle an error).
2) Observation: testers run the code and see what it does (e.g., unit testing, manually test the interface).