Explaining Agility with a Process Theory of Change

Michael Wufka
Department of Computing Science and Information Systems
Douglas College
New Westminster, Canada
wufkam@douglascollege.ca

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

***Abstract*—While agile approaches have been widely adopted, our theoretical understanding of their foundations and impacts remains limited. This is due to conflating two entirely different meanings of "agile." We therefore unpack these two meanings and present our tentative understanding as a process theory. The theory posits that agility emerges from a dialectic interplay between recognizing and responding to needs for changes. Meanwhile, rather than directly affecting success, agility moderates the negative effects of need for change on success. Viewing agility this way helps address the research-practice gap by highlighting the need for skepticism of methods and practices, and by suggesting practically relevant research questions.**

***Keywords*-process theory, agile development, agility.**

I. Introduction

Agility refers to two fundamentally different things. In everyday English, *agile* means "able to move … quickly and easily" (nimble) and "able to change or be changed rapidly" (responsive) [1]. In Software Engineering (SE), however, *agile* refers to an anti-bureaucracy sociotechnical movement and its associated development methods (e.g. [2]-[4]), practices (cf. [5]) and values (cf. [6]). Overloading of the term *agile* is problematic in at least three ways:

1. Labeling as "agile" every method, practice and value that is *intended* to increase agility undermines the responsibility to empirically evaluate whether these things actually *cause* nimbleness and responsiveness to increase. While it is plausible that daily stand-up meetings, for example, increase agility, little empirical data supports these causal relationships and many plausible hypotheses turn out to be false.
2. It obscures differences and contradictions between the individual methods, practices and values, as well as variance in their levels of empirical support.
3. It exacerbates the gap between software engineers and scientists who understand agility differently.

Clearly distinguishing between agile the anti-bureaucracy movement and agile the nimbleness and responsiveness of a team raises many questions. How does agility (nimbleness and responsiveness) emerge in SE projects? How is it related to specific practices, tools and technologies? How does it affect developers, processes and project outcomes? What impacts do anti-bureaucratic values and principles have on software projects? How do they affect developers, processes and outcomes? And how do we measure any of this?

Finding little theory about agile software development, Dingsøyr et al. [7] urged "agile researchers to embrace a theory-based approach in their scholarship." As agility fundamentally concerns responding to change, and the above questions concern *how* people, technologies and practices interact, a process theory approach is particularly appropriate [8], [9]. Consequently, the purpose of this paper is as follows:

> ***Purpose**: To formulate an initial process theory that explains the role of agility in software projects.*

We proceed by reviewing existing research on process theories in SE and related disciplines (Section II). We then develop a series of insights concerning how a process theory could explain agility (Section III). Section IV presents our preliminary theory and Section V discusses its implications.

II. Process Theory in Software Engineering

Some researchers distinguish between *variance theories* and *process theories*. Variance theories attempt to explain and predict changes in dependent variables based on changes in independent variables. For example, Senapathi and Srinivasan [10] use variance in innovation as well as sociological, technological and organizational factors to predict variance in agile usage and consequently effectiveness. Similarly, Maruping et al. [11] relate variance in agile methodology use to variance in software project quality, with moderating variables requirements change, outcome control and self control. Variance theories are important for differentiating important factors from coincidental correlates, predicting important phenomena and determining what empirical studies should measure.

Process theories, contrastingly, attempt to explain and describe how entities change over time. For example, the Problem-Design Exploration Model [12] explains how genetic algorithms can design systems by evolving a population of solution candidates. Allison and Merali [13] developed a theory that explains how software processes are improved through a dialectic interplay with software development. Sensemaking-Coevolution-Implementation Theory explains how developers create software artifacts by alternating between making sense of an ambiguous context, oscillating between a context schema and a design space schema, and translating the latter into artifacts [14], [15]. Process theories allow for generalizations and predictions in situations with very complex causal relationships. They are important for understanding and communicating processes, organizing empirical observations of complex phenomena and providing dispassionate knowledge to counterbalance the prescriptive knowledge conveyed by methods [9], [14], [16], [17].

978-1-4673-7153-7/15 $31.00 © 2015 IEEE
DOI 10.1109/Agile.2015.10

Surprisingly, our literature review did not uncover any process theories of agility. Since process theories are concerned with how entities change, and agile software development is concerned with how teams, artifacts, process and project environments change and how those changes affect each other, process theories seem particularly appropriate for understanding agility.

For example, suppose we want to study the relationship between risk and agility. Adopting a variance theory approach, we can hypothesize that increases in risk lead to increases in bureaucracy, which decrease agility. In contrast, adopting a process theory approach, we can hypothesize that development processes are governed by a dialectic interplay between development and management, where development uses its political capital to decrease bureaucracy while management uses its political capital to increase bureaucracy. Identifying new risks gives management more political capital and fuels its pro-bureaucracy rhetoric. This change in the balance of power leads to imposition of new rules, which in turn prevent developers from effectively responding to environmental changes. The variance theory predicts changes in numerical variables (i.e. overall risk, bureaucracy, agility), while the process theory explains the sociotechnical process underlying these changes.

III. Insights Toward a Process Theory of Agility

This section describes three core insights underlying our view of agility.

A. The Core Phenomenon of Interest is Change

While our purpose here is to develop a process theory that explains agility, agility is not a process. Agility (i.e. nimbleness and responsiveness) is a property of a developer, team or organization. Therefore, we pursue a theory of some closely related process, wherein we can explore how agility emerges and affects other phenomena. Similarly, "fitness for survival" is an attribute of a species rather than a process, so we have a theory of evolution rather than a theory of fitness. However, the theory of evolution helps us understand fitness and how it relates to other phenomena including mutation and adaptation.

Agility, meanwhile, is principally concerned with responding to changing circumstances in software projects. Consequently, a theory of how software teams respond to change may help us understand agility and how it relates to developers, change, artifacts and stakeholders. Moreover, it seems reasonable to begin with teams rather than individuals or organisations as software is predominately developed by teams. A theory of team agility could later be adapted to individuals or multiple teams.

B. A Dialectical Approach

Process theories come in four ideal types: lifecycle, evolutionary, dialectic and teleological [8]. Some phenomena are particularly compatible with a particular type while other phenomena are conducive to multiple types. For instance, teleological theories particularly apply where a goal-oriented agent can choose its own actions (e.g. ship navigation). Lifecycle theories, in contrast, apply when an entity proceeds through a pre-figured, deterministic sequence of phases that it cannot control (e.g. a caterpillar's transformation into a butterfly). If a large population of entities changes over time, evolutionary theories may be useful. Dialectic theories apply where several entities with differing degrees of power conflict.

To explain how software teams respond to change, a dialectical approach appears promising. Numerous stakeholders including management, users, clients and developers likely affect how a software team responds to change. These stakeholders may vary in power and have conflicting needs, goals and values (cf. [18]). For example, introducing new information systems causes some stakeholders to gain power, while others lose power [19]. Some agile proponents similarly posit a fundamental conflict between *Business* and *Development* (e.g. [4]).

Sabherwal and Newman [20] argue for a dialectical view of system development projects. In this view, organizational actions are based on specific theses; for instance, the use of relational databases may be based on the thesis that relational databases are the best approach for most day-to-day applications. Over time, an opposing view (antithesis) may emerge; for instance, the belief that NoSQL databases consistently outperform relational databases. The resulting tension between the thesis and antithesis may be resolved by *persistence* of the status quo, *change* toward a new approach, or *a combination of persistence and change* (a synthesis). For example, the organization might continue with relational databases, completely switch to NoSQL databases or begin experimenting with NoSQL databases in some areas.

A teleological approach could also be useful. Software developers choose actions to achieve goals, as in teleological theories. Design is intrinsically teleological [21], [22] and software teams have previously been theorized as teleological agents [14], [15]. However, dialectal theories better capture conflict between different agents.

In contrast, evolutionary and lifecycle theories seem less applicable. Evolutionary theories are more appropriate for understanding changes in populations (e.g. open source projects, unit tests, software companies). Lifecycle theories assume that activity sequences are immutable; however, sociotechnical processes are rarely if ever immutable because human actors can and do wilfully deviate from prescribed activities [23]. Lifecycle theories, therefore, simply do not apply to human activities.

It is also possible to combine two or more ideal types (e.g. dialectical and teleological) to create a multiple-motor process [24] and to combine a process theory with a variance theory to create a hybrid variance/process theory (cf. [24], [25]).

C. Software Project Dialectics

Software development may involve many different dialectical interactions. For example, conflicts may arise between different types of stakeholders such as developers, users and management. Users may want more features to increase usefulness while customers may want fewer features to minimize costs. Similarly, developers may want a later release to reduce stress and overtime while project managers may want an earlier release to accelerate return on investment.

Another dialectic conflict concerns the processes and products of development. There is a constant oscillation between the development process and the software product

under development [13]. Meanwhile, the structure of the development team may mirror the software architecture; for instance, if a system has four independent subsystems than four teams can develop it simultaneously. If, however, two subsystems are merged, two of the teams may also have to merge, or at least coordinate, to avoid interference.

Other dialectics include schema coevolution [14], [26] (where developers' mental representations of the project context and design space inform each other), code-test coevolution (simultaneously improving the code base and test suite) and prototyping (where changes to a software artefact trigger changes in its environment and vice versa). These dialectics all involve multiple, conflicting entities.

IV. Theorizing Software Project Change

The insights above suggest examining agility through a dialectical process theory of how software teams cope with change (Figure 1). Specifically, we posit that: 1) need for change emerges from interacting stakeholders; 2) software teams iteratively recognize and respond to these needs. Agility (nimbleness and responsiveness) emerges from these recognition-response cycles. The team, software artifacts and development process all continually evolve throughout.

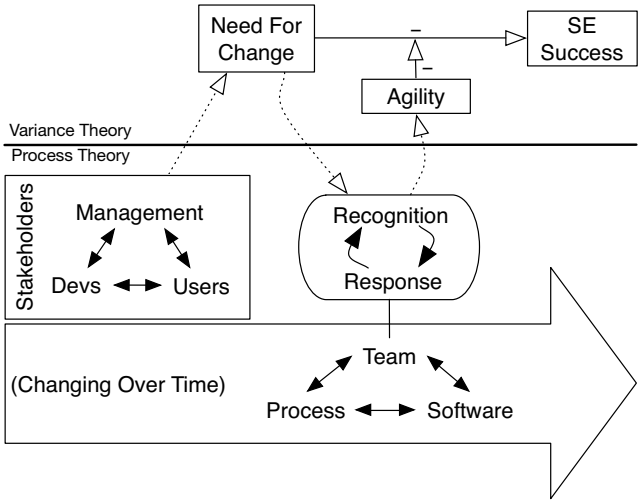

FIGURE 1. A Preliminary Process Theory of Software Project Change

While this paper focuses on process theory, Figure 1 also shows a simplified variance theory to further elucidate the key propositions. Specifically, we posit that need for change is negatively related to SE success. This proposition is based on the common belief that more unstable, unpredictable environments increase project difficulty. Rather than directly affecting success, agility moderates the negative relationship between need for change and success. That is, the benefits of responsiveness are only realized in the presence of changes that need responding to. While, including causal relationships complicates the emerging theory, the process would not make sense without the *Need for Change* and *Agility* constructs, and linking these to success clarifies the theory's context.

The proposed theory includes three dialectics – 1) the interplay between stakeholders, which produces the need for change; 2) the iteration between recognition and response, which produces change; 3) the interaction between the team, the develpment process and the software artifacts wherein changes take place. The core propositions of the proposed theory are as follows.

1. Software projects have stakeholders (e.g. developers, management, users), which have varying degrees of influence and experience conflict.
2. Stakeholders and their conflicts drive the need for change. While some changes are related to factors outside of a project, the *proximate* cause of a need for change is stakeholder reaction.
3. The need for a specific change and the need for change in general are quantifiable constructs in principle.
4. Prolonged, high need for change is negatively related to software project success.
5. Agility is a quantifiable construct in principle.
6. Agility moderates the relationship between the need for change and software project success. The higher the agility of a development team, the weaker the negative impact of need for change on project success, because higher agility entails more effective and efficient reaction to change.
7. High need for change triggers recognition and response cycles in the development team.
8. When a team recognizes the need for change, it may respond by changing itself, its development process or its software artifacts. It may take many recognition-response cycles to reduce the need for change. Recognizing the need for change creates pressure for change but does not guarantee that human actors will respond effectively or at all to this pressure.
9. Recognition of and response to needed changes is an ongoing interaction rather than a linear process. A specific response may not fully address the needed change or may demand further changes. Multiple responses on the team, process or software levels may be necessary and multiple (potentially conflicting) needs for change may emerge simultaneously.
10. The development team, the development process and the software under development coevolve, i.e., interdependently change over time. For example, dividing a system into three subsystems may lead to dividing the team into three subteams. This coevolution is affected by recognition-response cycles.

V. Potential Uses of the Proposed Theory

The research-practice gap around agility simultaneously stems from scientists' unwillingness to study practically relevant problems, practitioners' unwillingness to apply research results and a failure on both sides to demand evidence for the effectiveness of methods and techniques. The proposed theory addresses this in at least three ways:

1. Unpacking two different meanings of agile highlights the need to demand evidence that specific methods and practices increase the nimbleness and responsiveness of teams. Ostensibly agile practices may be empirically evaluated by analyzing their effect on recognition-response cycles.

2. Reframing research on agility as a process theory of change in software projects suggests research questions that are both practically relevant and scientifically tractable; for example, how do daily stand-up meetings affect recognition-response cycles? This is crucial because method evaluation is not scientifically tractable [27].
3. Presenting agility in terms of three relatable dialectics may help managers and developers better understand how to apply research results. For example, inconsistent findings around the effects of pair programming (cf. [28]) may be explained by differing levels of need for change between studies. The benefits of pair programming may only manifest in high need-for-change environments; consequently, experimental studies with different needs for change may produce conflicting results.

In addition, the proposed theory highlights the need for practices and technologies for managing stakeholder conflicts. It furthermore reveals how existing methods and theories oversimplify and over-rationalize change. For example, Zand and Sorensen [29] describe a simple process theory of organizational change where change consists of "unfreezing", "moving" and "refreezing". This over-rationalizes change, i.e., presents a chaotic reality in a lawful, organized manner [30]. Both software teams and organizations are in constant flux rather than frozen in static equilibria. By representing dialectical actors in a state of constant change, the proposed theory avoids and highlights such over-rationalizations.

VI. Conclusion

This paper makes two core contributions. It first clarified how the denotational overloading of *agility* obscures the responsibility to demonstrate empirically that ostensibly *agile* practices actually increase teams' nimbleness and responsiveness. It then explored a preliminary process theory of change in software development to explain how team agility emerges. The proposed theory specifically posits that 1) stakeholders and their conflicts produce needs for changes; 2) teams recognize and respond to needs for changes by altering themselves and their processes and artifacts; 3) agility emerges from the recognition-response dialectic; 4) agility moderates the negative impact of high need for change on success.

The proposed theory is limited in several ways. It does not attempt to explain agility at the organizational level or across projects. It includes only a small selection of elements and does not attempt to model how specific practices (e.g. continuous integration) affect agility. While agility entails both nimbleness and responsiveness, the proposed theory focuses on responsiveness. Moreover, the proposed theory is a work-in-progress and has not been empirically evaluated. Indeed, the purpose of this paper is to solicit feedback on the theory's face validity prior to testing.

These limitations suggest several implications and avenues for future research. First, we intend to test the proposed theory empirically. Existing guidance on process theory evaluation suggests that the process half of the theory may be evaluated using longitude field studies and questionnaire studies [31]. Meanwhile, the variance half may be evaluated using randomized control trials. It further suggests that to reveal the effects of agile methods or practices, studies may require high need-for-change environments or (better) directly manipulate need for change.

We hope that this paper and the proposed theory spur meaningful conversations about theorizing agility and rigorously scrutinizing ostensibly agile practices and technologies. We believe that good process theories are essential in addressing the research-practice gap around agility and development.

References

[1] Oxford English Dictionary, Oxford University Press, 2013.
[2] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2001.
[3] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 2003.
[4] K. Beck, *Extreme programming eXplained: Embrace change*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2005.
[5] P. Kruchten, "Contextualizing agile software development," *Journal of Software: Evolution and Process*, vol. 25, no. 4, pp. 351–361, Apr. 2013.
[6] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler et al., "Manifesto for agile software development," 2001. Available: http://www.agilemanifesto.org/.
[7] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, Jun. 2012.
[8] A. H. Van de Ven and M. S. Poole, "Explaining development and change in organizations," *The Academy of Management Review*, vol. 20, no. 3, pp. 510–540, 1995.
[9] A. H. Van de Ven, *Engaged scholarship: a guide for organizational and social research*. Oxford University Press, 2007.
[10] M. Senapathi and A. Srinivasan, "Understanding Post-Adoptive Agile Usage -- An Exploratory Cross-Case Analysis," in *Proceedings of AGILE*, IEEE, 2011, pp. 117–126.
[11] L. M. Maruping, V. Venkatesh, and R. Agarwal, "A Control Theory Perspective on Agile Methodology Use and Changing User Requirements," *Information Systems Research*, vol. 20, no. 3, pp. 377–399, 2009.
[12] M. Maher, J. Poon, and S. Boulanger, "Formalising design exploration as co-evolution: A combined gene approach," *Preprints of the Second IFIP WG5.2 Workshop on Advances in Formal Design Methods for CAD*, pp. 1–28, 1995.
[13] I. Allison and Y. Merali, "Software process improvement as emergent change: A structurational analysis," *Information and Software Technology*, vol. 49, no. 6, pp. 668–681, Jun. 2007.
[14] P. Ralph, "The Sensemaking-Coevolution-Implementation Theory of Software Design," *Science of Computer Programming*, vol. 101, pp. 21–41, 2015.
[15] P. Ralph, "Software Engineering Process Theory: A Multi-Method Comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory," *Information and Software Technology*, in press.
[16] M. L. Markus and D. Robey, "Information technology and organizational change: causal structure in theory and research," *Management Science*, vol. 34, no. 5, pp. 583–599, 1988.
[17] M. Poole, A. H. Van de Ven, K. Dooley, and M. E. Holmes, *Organizational change and innovation processes theory and methods for research*. Oxford University Press, 2000.
[18] P. Checkland, *Systems Thinking, Systems Practice*. Wiley, 1999.
[19] T. DeMarco, *Why Does Software Cost So Much?* Dorset House, 1995.
[20] R. Sabherwal and M. Newman, "Persistence and change in system development: a dialectical view," *Journal of Information Technology*, vol. 18, no. 2, pp. 69–92, 2003.
[21] C. W. Churchman, *The design of inquiring systems: Basic concepts of systems and organization*. Basic Books, 1971.
[22] E. A. Singer, *Experience and Reflection*. University of Pennsylvania Press, 1959.
[23] Y. Zheng, W. Venters, and T. Cornford, "Collective agility, paradox and organizational improvisation: the development of a particle physics grid," *Information Systems Journal*, vol. 21, no. 4, pp. 303–333, 2011.

[24] M. S. Poole and A. H. Van de Ven, “Empirical Methods for Research on Organizational Decision-Making Processes,” in *The Blackwell Handbook of Decision Making*, Nutt, P. C. and D. Wilson, Eds. Oxford: Blackwell, 2010, pp. 543–580.
[25] M. S. Poole, “On the Study of Process in Communication Research,” in *Communication Yearbook 36*, C. T. Salmon, Ed. New York, USA: Routledge, 2012, pp. 371–409.
[26] K. Dorst and N. Cross, “Creativity in the design process: co-evolution of problem–solution,” *Design Studies*, vol. 22, no. 5, pp. 425–437, Sep. 2001.
[27] P. Ralph, *Fundamentals of Software Design Science*. PhD Dissertation. University of British Columbia, 2010.
[28] J. Hannay, T. Dyba, E. Arisholm and D. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*. 51, 7, 1110–1122, 2009.
[29] D. E. Zand and R. E. Sorensen, “Theory of change and the effective use of management science,” *Administrative Science Quarterly*, 1975.
[30] M. W. Lewis, “Exploring Paradox: Toward a More Comprehensive Guide,” *The Academy of Management Review*, vol. 25, no. 4, pp. 760–776, Jan. 2000.
[31] P. Ralph, “Developing and evaluating software engineering process theories,” in *Proceedings of ICSE*, Florence, Italy, 2015.