

Winter 12-31-2017

The Empire Strikes Back: The end of Agile as we know it?

Jeffry Babb

West Texas A & M University, jbabb@wtamu.edu

Jacob Nørbjerg

Copenhagen Business School, jno.digi@cbs.dk

David J. Yates

Bentley University, dyates@bentley.edu

Leslie J. Waguespack

Bentley University, lwaguespack@bentley.edu

Follow this and additional works at: <http://aisel.aisnet.org/iris2017>

Recommended Citation

Babb, Jeffry; Nørbjerg, Jacob; Yates, David J.; and Waguespack, Leslie J., "The Empire Strikes Back: The end of Agile as we know it?" (2017). *Selected Papers of the IRIS, Issue Nr 8 (2017)*. 8.
<http://aisel.aisnet.org/iris2017/8>

This material is brought to you by the Scandinavian (IRIS) at AIS Electronic Library (AISEL). It has been accepted for inclusion in Selected Papers of the IRIS, Issue Nr 8 (2017) by an authorized administrator of AIS Electronic Library (AISEL). For more information, please contact elibrary@aisnet.org.

The Empire Strikes Back: The end of Agile as we know it?

Jeffrey S. Babb¹, Jacob Nørbjerg², David J. Yates³ and Leslie J. Waguespack³

¹ West Texas A&M University, Canyon, Texas, USA

² Copenhagen Business School, Frederiksberg, Denmark

³ Bentley University, Waltham, Massachusetts, USA

jbabb@wtamu.edu, jno.digi@cbs.dk, {dyates,lwaguespack}@bentley.edu

Abstract. Agile methods have co-evolved with the onset of rapid change in software and systems development and the methodologies and process models designed to guide them. Conceived from the lessons of practice, Agile methods brought a balanced perspective between the intentions of the stakeholder, the management function, and developers. As an evolutionary progression, trends towards rapid continuous delivery have witnessed the advent of DevOps where advances in tooling, technologies, and the environment of both development and consumption exert a new dynamic into the Agile oeuvre. We investigate the progression from Agile to DevOps from a Critical Social Theoretic perspective to examine a paradox in agility – does an always-on conceptualization of production forestall and impinge upon the processes of reflection and renewal that are also endemic to Agile methods? This paper is offered as a catalyst for critical examination of and as a call to action to advocate for sustaining and nurturing reflective practice in Agile and post-Agile methods, such as DevOps. Under threat of disenfranchisement and relegation to automation, we question how evolution towards DevOps may alter key elements in the tenets and principles of the Agile methods phenomenon.

Keywords: Agile Methods, Continuous Delivery, Critical Social Theory, DevOps, Iteration Pressure, Learning, Reflective Practice.

1 Introduction

In 2001, a wonderfully disruptive phenomenon was formally proffered to the world of software and systems development in the form of the Agile Manifesto [13, 29] – an espousal of principles and values which advocated for a progressive view on the art and craft of software and systems artifact realization. Levied in the context of classic “waterfall” conceptions of systems development, the set of methodologies gathered under the “agile” umbrella was a response to changes in the context of software and systems development. Also, the proliferation of information and knowledge, wrought by a world rapidly inter-connecting via the Internet, likely played its own part. The Agile Manifesto may be rightly considered an utterance of emancipation from staid and ossified beliefs and norms regarding the practice of development, manifested in – at least in the eyes of some – the CMM-inspired software process improvement

efforts of the previous decade [18, 45, 46, 54]. Sixteen years later, the extent of disruption and transformation brought about by a world extensively interconnected by the Internet is staggering if not even incomprehensible.

Agile software and systems practices heralded (and mostly delivered on) an accentuation of the co-creative possibilities inherent in a stakeholder-developer partnership, a partnership that is mediated with rituals and habits centered on regular discursive emergence, learning, and reflection. Among the other compelling aspects of Agile is its focus on the rapid delivery of customer value. To wit, some of the reflective and learning-centered aspects of Agile methods were frequent casualties of the “first shots” of many Agile-driven software projects. Rapid delivery, and the network effects of prolific delivery, have somewhat saturated the development space (with tools, frameworks, automation, and knowledge) where expectations of pace may outstrip learning and reflection inherently [5, 6, 30]. Thus, we begin to see a paradox in agility take shape: it is nimble in the face of uncertainty and risk, and yet susceptible to iteration pressure.

In some circumstances and settings, Agile practices are evolving to newer iterative software development paradigms such as DevOps – integrating development, deployment and operations tasks and responsibilities [22, 36] – and its companions, continuous delivery [41] and deployment [34]. We will use the moniker DevOps in the remainder of the paper to represent this family of methods [25].

DevOps offers a new conceptualization of Agile development which is consistent with the logics of accumulation and automation. DevOps practices complement Agile practices by emphasizing operations input to software and systems development [11], hence the contraction of “Development” and “Operations” to DevOps. In contrast, Agile practices emphasize customer input to development [1, 59]. DevOps emerged from the tenets of Agile software development pertinent to short release cycles [11], high automation in tooling [38], thus extending the continuous mode of design, development, testing and integration to delivery, deployment and monitoring [25, 38]. It could be argued that while Agile development espouses an ethos of organizational change through collaboration and learning, DevOps places more emphasis on implementing organizational change to achieve business goals (e.g., shorter lead times, better security, etc.) and on standardizing processes, automation, and data-driven improvements to process and product [23, 25, 41, 51].

In this paper, we reflect on the shift from Agile methods to DevOps, discursively and expositoryly, and from a critical philosophy of science, to examine a paradox of agility: will the embrace of the customer and managerial benefits of Agile methods, evident in DevOps and continuous evolution, undermine the learning and renewal aspects of Agile methods? In cases where a shift to DevOps has occurred, we see potential for automation and speed of delivery to alter the focus on reflective practice.

The paper is organized and fashioned after the canonical “Star Wars” metaphor, which itself follows a familiar “hero’s journey” story arc. In the prequel to the Agile-to-DevOps story, there is the “Empire” – the steadfast adherence to a technically-rational epistemology for practice that ensures that software processes can be systematically perfected with measurement and optimization on measurement. The “Rebellion” to this are a set of methods that value reflective practice and fluidity in the face of changing requirements and environments. Thus, we summarize different stages of Agile theory and practice over the past 18-20 years with examples drawn

from our own field studies, the studies of others, and the original descriptions of Agile processes. We then posit that, with an emphasis on augmenting people with technology and process automation, DevOps presents the possibility that the “Empire” will regain dominance as DevOps discourse focuses on technology and process and less on people and culture. We do not posit that this has happened, or is fated to happen, we simply reflect on Agile practice and foresee that reflection and learning are often exchanged for meeting production deadlines and various attendant “crises:” shall we expect that increasing technical rationality in the equation may change the Agile balance?

Finally, in lightly appropriating the “Star Wars” story arc, we take a critical theoretic view to advocate for emancipation for the Agile/DevOps software practitioner: to caution against yielding ground to automation when automation becomes detrimental to the reflective practices necessary for learning and renewal. In this respect, we relate to the small, but important, body of critical research exploring the relationship between work and new technology, see; e.g.; [15, 32, 48]; and we call for a critical examination of how the combination of managerial and organizational practices and priorities with new development technologies – Agile and DevOps – shape the work of software professionals. Further to this, we appeal to the Scandinavian research traditions where critical theory is a natural component of inquiry in cases where understanding the social, historical and ideological context of changes in practice is normative. As such, we question Agile’s future by considering its past and how it has fared in practice. We begin with a prequel: The Capability Maturity Model and Software Process Improvement.

We structure this paper in the following manner. In section two, we introduce the matter of software processes and software process improvement as a byproduct of the maturing that is software and systems development. In sections three and four, we characterize the evolutionary point at which iterative and continuous software delivery has challenged initial conceptualizations of software processes. These sections track a progression from agile methods towards software processes that focus further on continuous delivery and DevOps. In section five, we progress to utilizing Critical Social Theory to reflect upon the nature of progression from Agile to DevOps with respect to the tradeoffs that may impact human reflection and learning. Of particular interest are the emancipatory aspects of Critical Social Theory. We then conclude with an eye towards further Critical Social Theoretic inquiry into the emerging phenomenon that is DevOps.

2 Prequel: The Capability Maturity Model and Software Process Improvement

In the early 1990’s a structured and disciplined approach to improve software development and management gained traction: Software Capability Maturity Models (CMM) [26, 54]. These maturity models contained a set of managerial, developmental and organizational processes for software development organizations to adopt and apply to reach higher levels of process maturity, and thereby increasing levels of predictability, control, and, ultimately, efficiency, of their software development.

With the help of these maturity models, an organization could assess their software processes and initiate improvements; i.e. changes to processes, practices, management and organizations, needed to climb to a higher level.

The models sparked an interest in Software Process Improvement among software development organizations and researchers [28, 39] as well as much criticism [18]. Software engineers criticized the models for aiming to convert software development into an industrial assembly line process, which would impinge on the creativity and flexibility required to control the uncertainties inherent in software development. Particularly the data-driven continuous improvements (Level 5) were believed to result in incremental improvements to a fundamentally flawed process in need of radical change [18]. Others have pointed to the strong authoritarian and bureaucratic perspective on organizations and management espoused by the models and to the lack of appreciation of the organizational dynamics and politics of software developing organizations [45, 46]. Critics have also claimed that the models are too cumbersome and costly for small to medium sized software companies to use [10, 55].

3 Espoused Rebellion: Agile Principles and Practices

The Agile Manifesto was remarkable in its attempt to balance historically competing forces in software and systems development – the demands of the customer, the concerns of management, and the efficacy of the development team. When considering the tenets proffered in the Agile Manifesto, traditional software process values are acknowledged for their utility, but they are augmented with principles that highlight balance, largely between the developers and customer. Processes, tools, documentation, contracts, and planning are all concepts central to the inherent desire for management to control risks, costs, and productivity. These are natural byproducts of creating systems where profitability is at stake. The language used in the Agile Manifesto’s twelve principles clearly describes a balance that is customer focused and outcomes oriented. Some themes are emergent in the principles where management are scarcely mentioned and, when mentioned, are then referred to as “business people.”

Agile methods arguably co-evolved with the proliferation of Internet use and ubiquitous access to the World Wide Web. What is certain is that many aspects of responding to “Internet speed” [9] are fatiguing to the human element of software development and technical operations, even with advent of more capable and more sophisticated tools. Some responses – frequent iterations culminating in continuous delivery and continuous deployment [34, 41]; gravitating to fixed architectural patterns; componentization and reuse, for example, as embodied in microservices [42]; performing quality assurance earlier and more frequently [56]; amplified feedback; and, method tailoring – all present challenges to the human element. The high velocity of the current environment, producing such a frenzy around emerging tooling and frameworks causes visible and apparent fatigue [19] and even burnout [22].

It is reasonably self-evident that Agile methods have had great impact on the software and systems development world [1, 21, 24]. Not stated in the Agile

Manifesto and principles are the means of routinizing and controlling these methods of practice [16]. Thus, there are epistemological concerns afoot in our consideration of Agile methods and whether they have lost their way. While the original principles behind the Agile Manifesto may seem simple guidelines, they espouse an ethos and epistemology of practice that remains important. They speak to a “whole package” that includes customer orientation, individual excellence, reflective practice, technical excellence, and responsiveness to change. A 2012 survey of practitioners bears this out [61]. When asked whether (and what) would be changed about the Agile Manifesto’s principles, most suggestions focused on communication, learning, and collaboration. At issue is whether these connect to viable and working product, which depends on the competency and disposition of the team [14]. To put Agile methods into practice can be difficult, and its focus on customer feedback, where quicker cycles offer early detection of problems, is just one ingredient. Another, that is perhaps losing ground, is renewal via learning and reflection.

Toward this end, we will briefly discuss the application of three principles associated with Agile software development: ongoing customer contact; learning teams; and empowered and self-organizing teams. Based on findings from studies of Agile practice we will discuss how these principles are implemented in practical Agile software development projects.

3.1 The Principle of Ongoing Customer Contact

The principle of daily interaction between the development team and the customer is often abandoned or modified in practical applications of Agile software development [7, 31]. Particularly, small software companies with few developers and many customers have found it impractical or impossible to have customers and teams communicate frequently. Among other reasons for infrequent or lacking customer contact are: customers’ lack of time commitment and understanding of Agile practices, distance (e.g., in off shore development), and the customer representative’s insufficient skills and experience [31].

Development teams and companies apply different tactics to overcome the lack of customer contact, particularly a customer proxy or product owner, who acts on behalf of the customer when defining and interpreting user stories, planning a development cycle, and assessing outcomes. The lack of direct contact between the developers and the customer will, however, cause information distortion and delays, leading to misunderstood requirements, rework, increasing costs, and potential loss of customers [31, 49].

3.2 The Principle of Learning Teams

The Agile Manifesto values learning and self-empowered teams that reflect upon and improve their skills and practices on an ongoing basis. Agile methods such as XP and Scrum embody this principle in practices such as pair programming, frequent customer contact, stand-up meetings, reviews and retrospectives. Our research show that these practices are often adapted or omitted in Agile software development

projects with dire implications for in-team learning and reflection [5, 6, 30]. Developers frequently refer to lack of time and an increasing focus on producing software, when they explain why learning and reflection practices are omitted or strongly adapted. Particularly very small companies with limited resources and a strong need to ship (and be paid for) software tend to omit practices such as pair programming and retrospectives. Elsewhere, we have warned that this *iteration pressure* and the increasing attention to productivity at the expense of team – and individual – development and improvement, may have long term negative effects on the team’s performance and agility [6, 30].

3.3 The Principle of Empowered Self-organizing Teams

Agile software development is supposed to be organized in teams who work on a single project for a customer. The team is empowered and self-organizing, meaning that it manages the backlog of tasks, prioritizes and selects tasks for a Sprint or timebox, and distributes work among the team members.

Our observations of Agile software practices indicate, however, that these practices are susceptible to erosion. In very small companies, where customers far outnumber the developers, each developer is effectively a team of one, which is allocated to several projects; i.e.; one for each customer. Customer contact, task management and prioritization are furthermore the responsibility of the manager/owner in such organizations [7]. With variations, we have observed similar patterns emerge in other organizations, as briefly illustrated in the following examples from a preliminary analysis of observations and interviews in software development organizations in Denmark and the US.

The startup. The startup develops an innovative software product, and employs about 10 developers, all working in the same room, but loosely organized into teams based on the product architecture. Stories are defined and managed by a management group and allocated to sprints and teams. The team breaks the stories down into tasks, which are allocated to individual developers. The team uses Kanban boards, burndown charts, and other information radiators to manage the Sprint.

The mature SME. The mature SME is a web-agency, which develops web-sites and portals for different customers. The customers range from small to very large private and public companies and organizations. The relationship can extend for several years beyond the initial development of a site. The developers are organized into teams, each working for several customers. The team structure is not fixed, however, with developers being moved between teams to close resource gaps. Each team has a project manager, who is the primary liaison between the company and the customer, although other team members can participate in meetings with customers.

Tasks are negotiated with the customer, and assigned to developers by the project manager in two-week sprints. A developer can, in other words, work on several different “projects” during a Sprint!

These are just a few examples, representative of what we have observed in companies in both Denmark and the United States. Although we are in the early stages of analysis, we believe the examples reveal a general trend in the application of Agile software development, at least in certain kinds of software development organizations.

3.4 Erosion of Agile Principles

We observe that modern software organizations embrace the Agile ideals of evolutionary development, short cycles, and adaptive planning, but that several of the principles – or ideals – associated with Agile development seem to have been abandoned or heavily modified: the customer proxy or product owner has replaced the “customer on site,” and team learning and reflection has given in to iteration pressure and frequent deliveries. Loosely coupled individuals, managed by a project or product manager, have replaced the self-organizing team, and a Sprint is a planning frame where each developer is assigned a selection of tasks to solve for several customers.

There are probably several causes behind these developments, but the quest for higher productivity and shorter turnaround times – note that the duration of a Sprint or timebox has been reduced to only two weeks (or less!) over the past decade – seem plausible candidates.

4 DevOps: The Lingering Empire?

The agile approach to software and systems development brought programmers, testers and quality assurance employees together to ensure closer collaboration as a team, and shorten the time between software releases from several months or years to weeks. The DevOps approach aims to further increase the IT organizations’ capabilities to react quickly and release new software versions frequently – possibly several times per day – by removing organizational and practical boundaries between development and operations [25]. In practice, this is accomplished by automating operations oriented tasks, such as hardware configuration and set-up, systems integration, deployment, and monitoring. It reflects an emergent environment of operations which has progressed beyond the scope of the operational context under which Agile methods were conceived.

DevOps and its companions, continuous integration and continuous deployment, aim for a continuous and highly automated flow from programming, to test, delivery, and deployment. It depends on standardized architectures and sophisticated tools automating build, test, deployment, and monitoring processes. Measurements of the software development process and the product in use are fed back to development to inform changes or additions to the product and improvements to the process [51].

While there is still not much research about DevOps practices and their implications, it appears that the move to DevOps risks a further reduction in team control and authority towards outside managers, supported by monitoring and metrics

that, when applied blindly to improve processes, serve the business but don't account for their impact on people and culture [36].

4.1 Emerging Contexts of Practice

As this paper critically examines the impact of DevOps directly on the developer as a reflective practitioner, we must take care to examine DevOps as it is. Software and systems development, at “Internet speed” creates its own unique context. With continuous delivery comes continuous integration and the need to interact with the operational realities of adjusting features and content in real time. In this context, the discourse surrounding agility changes as practical considerations about the cycle of design, build, implement, measure, react, learn and change involve the highly detailed (and technically rational) world of IT and systems operations. Thus, the DevOps movement is, in part, about addressing imbalances of power and/or understanding that exist between developers and operations people [12, 22, 34, 42, 59]. In addressing human problems, DevOps focuses on building culture, teamwork, and serving the mission of the teams on which one serves. DevOps espouses themes that include collective ownership, servant leadership, trust, empathy, etc. [36]. Principles and practices supporting these themes are challenged, however, if either “Dev” or “Ops” has all or most of the power in an organization.

4.2 Evolution is Resilience

We consider DevOps, in part, an evolution of Agile methods in a new context. DevOps is positioned as such where Agile methods are the accelerant and catalyst for the “Dev” component of the symbiosis [12, 59]. Thus, in the need for high availability in a continuous mode, it was necessary to look beyond “potentially shippable” products and extend Agile into the culture and conditions of operations. It could be argued that Agile, in celebration of continuous customer contact, relegated the developed system as something you “threw over the wall” to operations when the customer expressed satisfaction. Agile principles and practices, however, essentially establish the roots and referents necessary to critically examine the essence and accidents inherent in the use of both Agile and DevOps.

The DevOps reality is that the operating modes of Amazon, Facebook, Google, and Netflix, for example, have little precedent and have elicited DevOps as a resilient response where the traits of Agile that have proven to be of worth – a focus on small teams continually delivering high-quality code to customers – have been hybridized and retained [36]. However, our central question remains: How will Agile's focus on learning and reflection fare in a merger with operations, and the increased focus on rapid delivery and “automated” decision-making?

5 Return of the Jedi: Adopting a Critical Response

To examine the implications for reflection and learning in a merger of development and operations, we adopt a critical theoretic perspective. In doing so, we also appeal to a sociotechnical epistemology of theory and practice. We assert that the sociotechnical perspective is aligned with Agile and DevOps according to the democratizing aspects assumed on issues related to open design; early customer value; egalitarian views on power, authority, and information; and continuous improvement and continual learning [22, 50]. Agile and DevOps also favor system theoretic perspectives in that problem solving and setting require a balanced perspective on matters such as complexity, the proclivities of the participants and stakeholders of the system, and role of chaos and entropy in design that favors early and iterative development [24, 36]. In this regard, Checkland's Soft Systems Methodology [20] strongly considers a general systems theoretic component.

From a sociotechnical perspective, we consider the impacts of iteration pressure and how it has transformed Agile practice. Specifically, we appeal for a consideration of these issues via the lens of the critical/neohumanist paradigm [43]. Relegation of the developer to code-producer is a departure from the tenets of Agile methods and Critical Social Theory (CST) encourages researchers to assume a value-laden inquiry with aim to question the shift to "neo-Taylorism" afoot in the evolution of Agile methods [47]. It is useful, if not overly simplistic, to consider the phase shift that Agile may be experiencing as it has encountered and digested aspects of the "Lean" movement and similar influences from the Japanese automobile manufacturing from the late 20th century.

Whereas some Agile methods have taken their clues from the *Lean* phenomenon from the start, there is a distinct end of the spectrum of Agile methods that is arguably aligned with a "human-centrism." This is evident in aspects of Extreme Programming [12] and Scrum [59]. Whereas Agile methods such as Scrum took care in minimizing the "us-vs-them" dichotomy between management and workers, developer relegation under iteration pressure reintroduces these aspects [47]. To wit, it would seem that some "democratic Taylorism" is envisioned in Agile methods' evolution towards the *Lean* and *Continuous* paradigms [2] inherent in DevOps.

Without considerable time afforded for renewal and learning through reflection, the "trappings" of the Agile Manifesto and principles may be visible at the surface, but are they still implementable with 1-week sprints or daily deliverables? The necessity of standardization inherent in all process optimizations is understandable, but the epistemology of technical rationality [57] inherent in these optimizations brings into question how learning will occur. Despite the inherent wisdom in "refactoring mercilessly to patterns" – as an example – questions arise as to how this inherently technically rational view will allow for the adaptation and innovation also inherent in Agile methods.

To appropriate CST in this case, we uphold its assertions: researchers are capable of inquiry that is value laden and that seeks to expose injustice. The creativity and freedom assumed in the original characterizations of Agile methods are in danger of being subsumed into a knowledge interest that is purely technically rational and practical. The relevance of our inquiry, as engaged scholars desirous of direct action,

would naturally lead to a knowledge interest rooted in emancipation. Consistent with the underlying concerns outlined by Habermas [27], to take a critical theoretic perspective on the evolution of the Agile paradigm that charts a course away from a human-centricity, is to consider the mediating and moderating role of technology in the social relations upon which Agile methods are founded [44]. Further, we argue that an emancipation imperative exists in the call to action that is the Agile Manifesto and its principles. The Agile Manifesto and its principles introduce a dialectic that seeks to maximize benefits to developers, management, and customers with equanimity and equality.

5.1 The Emancipation Imperative

Central to a critical theoretic response to not just the lacuna we characterize in Agile’s epistemology, but to the continued mis-calibration between the necessity of technical rationality and the imperative to recognize that human potential is shaped by our own innovations [52]. We would be naïve to think that this reshaping is always for the better. As it has been suggested that critical theoretic treatments are tantamount to a “missing paradigm” in information systems (IS) research, its value may persist inherently given the perturbations our own systems cause to known order.

Howcroft and Trauth [33] outline key themes in critical research that are relevant to the “agile” paradox. First, the emancipatory component of critical theory has a focus on freeing individuals from adverse or detrimental power relations which lead to disenfranchisement, alienation, and domination. Further, a willingness to undertake critique of tradition – to disrupt the status quo by revealing and highlighting incongruences, anomalies, and inequities to foment positive change. A non-performance (conformance) theme highlights tools and mechanisms designed to bolster managerial efficiency over human considerations.

5.2 Strategies for Emancipation

Poignantly, CST calls for critique of the technological determinism also known as technical rationality. In Agile’s progression towards DevOps, the efficacy and efficiency of the artifact is often the sole determinant of quality. Further, CST values reflection in a manner where some advocacy and interest is inherent in the researcher, making the process value-laden: in this sense, it is to act in advocacy for justice. Myers and Klein [40] offer a set of principles for critical research from which some validation of the arguments made in this paper is possible. See Table 1 below.

Table 1 Applying the Principles of CST to the Agile Paradox. (Based on Myers and Klein [40])

Myers and Klein Critical Theory Elements	This Paper’s Position
<i>Insight</i>	The “agile” paradox: Agile’s progression towards continuous delivery and lean principles may unwittingly upset the balance between learning/renewal and production to

	disenfranchise developers.
<i>Critique: Core Concepts from Critical Social Theorists</i>	Habermas: Reason in practice requires reflective judgment and critique such that the renewal of practice is possible by “seeing” the totality of the problem.
<i>Critique: Taking a Value Position</i>	The lean-influenced continuous delivery evolution of Agile is relegating the reflective practice component of Agile development for the developers involved.
<i>Critique: Revealing and Challenging Norms</i>	In a neo-Taylorist manner, DevOps emphasizes the feedback from automation and artificial intelligence over the time for reflection.
<i>Transformation: Individual Emancipation</i>	Pauses for reflection required for the renewal of human insight and repertoire. This time and occasion is characterized as an afterthought in the nascent elaborations on DevOps.
<i>Transformation: Improvements to Society</i>	As a result of rapid feedback vis-à-vis automation, are we becoming smarter? Or, will innovation in software and systems dry up as quick cycles stifle the reflection, learning and renewal needed to innovate?
<i>Transformation: Improving Social Theories</i>	Argyris and Schön [4] theory of action and Schön’s [57, 58] epistemology of reflective practice remains relevant and a context from which critical investigations are possible.

We appropriate these principles here in a call to action to IS researchers. In a previous call to action (emerged about the same time that Agile methods had emerged), which largely resulted in the contemporary design science movement in IS research, various voices arose asking a simple question: Where is the IT artifact in our research [3, 53, 60]? We extend this call here by suggesting that Agile methods, and their evolutionary progression, so central in delivering many compelling IT artifacts, are worthy of our inquiry. The rush to the incorporation of “lean” and “smart” processes into our development cycles requires some pause, caution, and reflection in order to appreciate what is gained and lost.

5.3 Appealing to Scandinavian Traditions in IS Research

We consider that the critical research that we call for is firmly rooted in Scandinavian traditions of research and scientific inquiry. While the transition to DevOps in response to the challenges of continuous delivery and deployment does not present a uniquely Scandinavian problem, research traditions in Scandinavian are at least instructive. Bansler [8] characterizes three traditions in Scandinavian research in systems development that have some direct bearing on our characterization of the progression of Agile methods: systems-theoretical, socio-technical, and critical. This is so as Scandinavian research on systems development, and the antecedent and

guiding theories often referenced, consistently reflect Agile principles in their own espoused world-view. In Table 2 below, we utilize a rigorous scholarly digest and reflection on the impact of Agile and reference Scandinavian research traditions for analysis.

Table 2 Relating Traditions in Scandinavian Research to Agile Principles. (Adapted from Nerur et al. [50])

Agile Principle	Related Research Tradition
Emphasis on Individuals	Sociotechnical Systems
Emphasis on Balancing Quality with Human Concerns	Sociotechnical Systems Critical Theory
Learning and Adaptation	Systems Theory
Participative Development	Sociotechnical Systems
Accepting and Leveraging Change	Sociotechnical Systems Critical Theory
Self-organization	Systems Theory
Minimum Viable Product	Sociotechnical Systems
Reflective Practice	Critical Theory

Thus, as a final point, we appeal to continued application of the Scandinavian tradition in investigating the Agile-to-DevOps shift currently afoot. Systems thinking and the socio-technical perspective, considered from a critical theoretic footing, hold promise to develop insight for understanding. We did not seek to commit an “act” of technological determinism by giving the impression of vilifying DevOps. Rather, we hold the same concerns that others expressed on the advent of Agile methods [17], and appeal for a mindful, reasoned and considered appreciation of this arising.

6 Conclusion

This paper has presented a position which characterizes how an evolution towards continuous delivery and DevOps presents a number of paradoxical conundrums. In the face of these developments, we take the position that the implications for learning and reflection are understudied and present the principle call to action around which this paper is designed. Where speed is paramount, quality becomes harder to sustain and cost is difficult to manage. Fred Brooks talked about building “one to throw away” and it is likely that, given “Internet speed,” many projects are “throwaways” as the foundations upon which they are built are now irrelevant and perhaps unsupportable. Managers and developers face an “always on” mode where the boundaries between projects, be they parallel or linear, are grey and fuzzy. When there is no beginning and no end, what is the subject of a Sprint review or retrospective? What is the basis for learning? What is a Sprint when operation is continuous? Increasingly, learning is at least partially – if not fully – delegated to algorithmic machine learning based on data-driven tooling, which is far more capable of learning through aggregation without leveraging the very human use of metaphor. When quality is negotiable versus, for example, reliability and security, and the

creativity wrought by metaphor is subsumed, then high velocity development is at risk of yielding a “lowest common denominator” product where distinction based on quality is irrelevant.

Whereas Agile emanated from seasoned professionals who had the benefit of a pre- and proto-Internet era to cultivate their ideas, the cadence of high velocity is likely so fast that their innovation would have been missed in a contemporary environment. This may be why DevOps has a more hurried and urgent feel to it. As a cultural movement, it lacks a manifesto and lacks consistent prescriptive methodologies. These phenomena make the discourse around DevOps fluid at best and confusing or incomprehensible at worst. It is therefore not surprising that DevOps is still evolving and a successful implementation, even in the most capable organizations, requires a journey that takes years of effort and often remains challenging to scale.

Essentially, we return to our core line of inquiry: How has the characterization of the “agility” changed in the Agile to DevOps transition? Have things regressed from an ideal of the “agile” (nimble and reflective) practitioner as an artisanal master of craft with a keen eye to productivity, learning and renewal in a reflective practice, to a “mechanical Turk” able to produce software and systems “tidbits” akin to the way a short-order cook delivers fast food [35, 37]? And will data-driven improvement replace reflection, learning and creativity with incremental optimization of flawed products and processes, similar to the critique raised against the CMM? Or, is DevOps a response to a new context and environment where the canonical articulations of Agile methods, such as Scrum and XP, no longer fit operational reality? While the former characterization may appear too brash on first blush, it is worth consideration at this progressed juncture in the history of Agile methods, as they have intermingled with the iterative, lean, and continuous delivery aspects of the uptake of Agile methods that have ontological and epistemological considerations for IS researchers.

This paper is offered as both a metaphoric “discussant” and a call to action. The emphasis on rapid delivery and accrual of value, even when necessary in web and cloud environments, calls into question when and where renewal through reflective practice may occur. Ships can’t stay at sea indefinitely; they must rest and refit at regular intervals – just like learning from experience and re-calibrating repertoire is necessary for the reflective software developer. We have presented an argument that the learning paradox arising from agility may have deleterious effects not only on the quality of the product, but also on developer enfranchisement to the process. To adopt a critical social theoretic stance in this issue is to consider how to emancipate both the developer, and perhaps Agile methods, from this growing imbalance. Another approach would be to disavow Continuous and DevOps from its Agile past – which, although possible, is not practical. We call on more engaged scholarship, and in utilization of an action learning cycle, to better understand the “agile” learning paradox and its implications for future practice. This is so as the design, development, implementation, and maintenance of systems remains a core concern of the IS discipline.

References

1. Abrahamsson, P., Conboy, K., & Wang, X. (2009). 'Lots done, more to do': The current state of agile systems development research. *European Journal of Information Systems* 18, 281-284.
2. Adler, P. (1995). 'Democratic Taylorism': The Toyota Production System at NUMMI. In *Lean work: Empowerment and exploitation in the global auto industry*, 207-219.
3. Alter, S. (2003). Sidestepping the IT Artifact, Scrapping the IS Silo, and Laying Claim to 'Systems in Organizations'. *Communications of the AIS* 12(1), 30.
4. Argyris, C., & Schön, D. A. (1974). *Theory in Practice: Increasing Professional Effectiveness*. Jossey-Bass, San Francisco, CA.
5. Babb, J. S., Hoda, R., & Nørbjerg, J. (2013). Barriers to Learning in Agile Software Development Projects. *Agile Processes in Software Engineering and Extreme Programming, 14th International Conference*, H. Baumeister, & B. Weber. Springer Verlag, Vienna, Austria 149, 1-15.
6. Babb, J. S., Hoda, R., & Nørbjerg, J. (2014). Embedding Reflection and Learning into Agile Software Development. *IEEE Software* 31(4), 51-57.
7. Babb, J. S., Hoda, R., & Nørbjerg, J. (2014). XP in a Small Software Development Business: Adapting to Local Constraints. *5th Scandinavian Conference on Information Systems (SCIS)*, T. H. Commissio, J. Nørbjerg, & J. Pries-Heje, Ringsted, Denmark, Springer.
8. Bansler, J. (1989). Systems development research in Scandinavia: Three theoretical schools. *Scandinavian Journal of Information Systems* 1(1), 3-20.
9. Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., & Slaughter, S. (2003). Is 'Internet-speed' Software Development Different? *IEEE Software* 20(6), 70-77.
10. Basri, S., & O'Connor, R. V. (2010). Understanding the Perception of Very Small Software Companies towards the Adoption of Process Standards. *17th EuroSPI Conference*, Springer, Grenoble, France.
11. Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley, Boston, MA.
12. Beck, K. (2004). *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley, Reading, MA.
13. Beck, K. et al. (2001). *Manifesto for Agile Software Development*. Agile Alliance. At <http://agilemanifesto.org/>, Retrieved 10 December 2016.
14. Berczuk, S. (2007). Back to basics: The role of agile principles in success with a distributed scrum team. *Agile Conference (AGILE)*.
15. Bergvall-Kåreborn, B., & Howcroft, D. (2014). Persistent problems and practices in information systems development: A study of mobile applications development and distribution. *Information Systems Journal* 24(5), 425-444.
16. Binstock, A. (2012). Interview with Alan Kay, *Dr. Dobb's*, July 10.
17. Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer* 35(1), 64-69.
18. Bollinger, T. B., & McGowan, C. (1991). A critical look at software capability evaluations. *IEEE Software* 8(4), 25-41.
19. Cancialosi, C. (2016). DevOps, culture change and the brass ring of velocity. *Forbes*, March 28.
20. Checkland, P. B. (1981). *Systems Thinking, Systems Practice*. John Wiley & Sons, Chichester, UK.
21. Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research* 20(3), 329-354.
22. Davis, J., & Daniels, K. (2016). *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*. O'Reilly Media, Sebastopol, CA.

23. Dennehy, D., & Conboy, K. (2016). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software*.
24. Dybå, T., & Dingsøy, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology* 50(9-10), 833-859.
25. Fitzgerald, B., & Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*.
26. Haase, V., Messnarz, R., Koch, G., Kugler, H. J., & Decrinis, P. (1994). Bootstrap: Fine-Tuning Process Assessment. *IEEE Software* 11(4), 25-35.
27. Habermas, J. (1972). *Knowledge and Human Interests*. Heinemann Educational, London, UK.
28. Hansen, B., Rose, J., & Tjørnehøj, G. (2004). Prescription, Description, Reflection: The Shape of the Software Process Improvement Field. *International Journal of Information Management* (24), 457-472.
29. Highsmith, J., & Fowler, M. (2001). The agile manifesto. *Software Development Magazine* 9(8), 29-30.
30. Hoda, R., Babb, J. S., & Nørbjerg, J. (2013). Toward learning teams. *IEEE Software* 30(4), 95-98.
31. Hoda, R., Noble, J., & Marshall, J. (2011). The impact of inadequate customer collaboration on self-organizing Agile teams. *Information and Software Technology* (53), 521-534
32. Howcroft, D., & Taylor, P. (2014). 'Plus ca change, plus la meme chose?'—Researching and theorising the 'new' new technologies. *New Technology, Work and Employment* 29(1), 1-8.
33. Howcroft, D., & Trauth, E. M., Eds. (2005). *Handbook of Critical Information Systems Research: Theory and Application*. Edward Elgar Publishing, Cheltenham, UK.
34. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, Reading, MA.
35. Ipeirotis, P. G., Provost, F., & Wang, J. (2010). Quality management on Amazon Mechanical Turk. *ACM SIGKDD Workshop on Human Computation*, 64-67.
36. Kim, G., Debois, P., Willis, J., & Humble, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, Portland, OR.
37. Levitt, G. M. (2000). *The Turk, Chess Automation*. McFarland & Company, Incorporated Publishers.
38. Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. (2016). *The Practice of System and Network Administration: DevOps and Other Best Practices for Enterprise IT*. Addison-Wesley, Boston, MA.
39. Mathiassen, L., Pries-Heje, J., & Ngwenyama, O., Eds. (2002). *Improving Software Organizations. From Principles to Practice*. Addison-Wesley, Reading, MA.
40. Myers, M. D., & Klein, H. K. (2011). A Set of Principles for Conducting Critical Research in Information Systems. *MIS Quarterly* 35(1), 17-36.
41. Neely, S., & Stolt, S. (2013). Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). *Agile Conference (AGILE)*.
42. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Sebastopol, CA.
43. Ngwenyama, O. K. (1991). The critical social theory approach to information systems: Problems and challenges. *Information systems research: Contemporary approaches and emergent traditions*, 267-280.
44. Ngwenyama, O. K. (1993). Developing end-users' systems development competence. An exploratory study. *Information and Management* (25), 291-302.

45. Ngwenyama, O., & Nielsen, P. A. (2003). Competing Values in Software Process Improvement: An Assumption Analysis of CMM from an Organizational Culture Perspective. *IEEE Transactions on Engineering Management* 50(1), 100-112.
46. Nielsen, P. A., & Nørbjerg, J. (2001). Assessing Software Processes: Low Maturity or Sensible Practice. *Scandinavian Journal of Information Systems* 13(1-2), 23-36.
47. Niepcel, W., & Molleman, E. (1998). Work design issues in lean production from a sociotechnical systems perspective: Neo-Taylorism or the next step in sociotechnical design? *Human relations* 51(3), 259-287.
48. Nørbjerg, J., & Kraft, P. (2002). Software practice is social practice. *Social Thinking - Software Practice*, R. Klischewski, Floyd, C. & Dittrich, Y. Eds. MIT Press, Cambridge, MA.
49. Nørbjerg, J., & Shakir, S. N. (2015). The End of the Line: Project Management Challenges in Small Software Shops in Pakistan. *Strategic Project Management*, K.-M. Osei-Bryson & C. Barclay. CRC Press, Boca Raton, FL, 107-131.
50. Nerur, S., Cannon, A., Balijepally, V., & Bond, P. (2010). Toward an Understanding of the Conceptual Underpinnings of Agile Development Methodologies. *Agile Software Development: Current Research and Future Directions*, Springer, Germany, 15-29.
51. Olsson, H. H., & Bosch, J. (2014). Post-deployment Data Collection in Software-Intensive Embedded Products. *Continuous Software Engineering*, J. Bosch, Springer, 143-154.
52. Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research* (2), 1-28.
53. Orlikowski, W. J., & Iacono, C. S. (2001). Research commentary: Desperately seeking the 'IT' in IT research—A call to theorizing the IT artifact. *Information Systems Research* 12(2), 121-134.
54. Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability Maturity Model for Software, v. 1.1. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
55. Pino, F. J., Garcia, F., & Piattini, M. (2008). Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Control* 16(2), 237-261.
56. Roche, J. (2013). Adopting DevOps Practices in Quality Assurance. *Communications of the ACM* 56(11), 38-43.
57. Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, NY.
58. Schön, D. A. (1987). *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. Jossey-Bass, San Francisco, CA.
59. Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (vol. 1). Prentice Hall, Upper Saddle River, NJ.
60. Weber, R. (2003). Still desperately seeking the IT artifact. *MIS Quarterly* 27(2), iii-xi.
61. Williams, L. (2012). What agile teams think of agile principles. *Communications of the ACM* 55(4), 71-76.