CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

The Empire Strikes Back

The End of Agile as we Know it?

Babb, Jeffry S.; Nørbjerg, Jacob; Yates, David J. ; Waguespack, Leslie J.

Document Version
Final published version

Publication date:
2017

License
CC BY-NC-ND

Citation for published version (APA):
Babb, J. S., Nørbjerg, J., Yates, D. J., & Waguespack, L. J. (2017). *The Empire Strikes Back: The End of Agile as we Know it?*. Paper presented at The 40th Information Systems Research Seminar in Scandinavia, Halden, Norway.

Link to publication in CBS Research Portal

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact us (research.lib@cbs.dk) providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. okt.. 2026

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

Jeffry S. Babb[1], Jacob Nørbjerg[2], David J. Yates[3], Leslie J. Waguespack[3]

[1] West Texas A&M University, Canyon, Texas, USA
[2] Copenhagen Business School, Frederiksberg, Denmark
[3] Bentley University, Waltham, Massachusetts, USA

Abstract. Agile methods have co-evolved with the onset of rapid change and turbidity 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 intensions 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 – what 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 and as an overt call to action to engage in emancipatory scholarship in advocacy for the Agile development team. Under threat of disenfranchisement and relegation to automation, we question how a tilt towards DevOps will preserve 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 – 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 [14, 38, 39, 46]. Sixteen years later, the extent of disruption and transformation brought about by a world extensively inter-connected

by the Internet is staggering if not even fully comprehensible from the perspective of the halcyon days of the *fin de siècle*.

While Agile practice 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, among the 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, 25].

Agile practices themselves are giving way to the evolutionary phenomenon that is DevOps [18, 30] or even its companion, Continuous Delivery [34] and Deployment [28] (Continuous* – we will use the moniker DevOps in the remainder of the paper) [21]. DevOps offers a new conceptualization of Agile practice which is consistent with the logic of accumulation. DevOps emerged from the benefits of Agile software development pertinent to shortened release cycles. Thus, a central tenet of DevOps is to reach a state where applications are released faster and more frequently. In this context, a process management response is to introduce tools to increase automation and continuous delivery [28]. While Agile software development constituted an ethos of theory and practice focused on organizational change through its collaboration and learning focus, DevOps places more emphasis on implementing organizational change to achieve organizational goals and on traditional notions of standard processes, automation, and – not the least – data driven process and product improvements [19, 21, 34, 43].

In this paper, we present a position on Agile's evolution towards DevOPs as a means of developing discourse, from a critical philosophy of science, to better understand a paradox of agility: will the embrace of the customer and managerial benefits of Agile methods, evident in the DevOps and continuous evolution, undermine the learning and renewal aspects of Agile methods? With Agile methods, strains of legitimate democratization of the developer as reflective practitioner were resonantly and resolutely clear. With DevOps, where developer is primarily engaged in rapid delivery, we see the developer in danger of being relegated to an automaton and cog.

In this paper, offered to generate discussion and inquiry, we ponder a paradox evident when we consider Agile's high pace in contrast with its espoused values centered on learning and renewal. 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. Directly, we question Agile's future by consider its past and how it has fared in practice. We begin with a prequel: Software Process Improvement and the Capability Maturity Model.

2 Prequel: The Capability Maturity Model and Software Process Improvement

In the early 1990's a novel approach to improve software development and management gained traction: Software Capability Maturity Models. See for example [22, 46]. The 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 [24, 32] as well as much criticism. Software engineers criticized the models for aiming to convert software development into an industrial assembly line process, which would stifle 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 [14]. 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 [38, 39]. Critics have also claimed that the models are too cumbersome and costly for small to medium sized software companies to use [10, 47].

3 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 all 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." This may or may not be naiveté, but that is unlikely considering the gravitas of the signatories.

It is reasonably self-evident that Agile methods have had great impact on the software and systems development world [1, 17, 20]. Not stated in the Agile manifesto and principles are the means of routinizing and controlling these methods of practice [13]. Thus, there are epistemological concerns afoot in our consideration

of Agile methods and whether they have lost their way. While the original tenets and 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 [53]. 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 [12]. Agile is difficult, and its feedback focus, 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 three core values often associated with Agile software development, which can be extracted from the 12 principles: 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 (and their associated practices) have unfolded in practical Agile software development projects. Further in our exploration of these aspects of Agile methods, we seek to develop an important core line of inquiry. How has the characterization of the "agility" in these methods 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" available to produce software and systems "tidbits" akin to the way a short-order cook delivers fast food [29, 31]? While such a characterization may appear too brash on first blush, it is worth entertaining 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 Information Systems researchers.

3.1 Ongoing customer contact

The principle of a customer on site being in close contact with the development team was among the early casualties in the practical application of Agile software development [7, 26]. Particularly small software companies with few developers and many customers found it impractical or impossible to have customers and teams communicate frequently. Among other reasons for infrequent or lacking customer contact cited are: customers' lack of time commitment and understanding of Agile practices, distance (off shore development), and the customer representative's insufficient skills and experience [26].

The developers 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 [26, 41].

3.2 Learning teams

Agile software development espouses the idea of self-empowered teams that reflect upon and improve their skills and work 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 and retrospectives. These practices are, however, strongly adapted or omitted in Agile software development projects with dire implications for in-team learning and reflection [6, 25]. 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 bill) 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, 25].

3.3 Empowered self-organizing teams: Agile or short term resource management?

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.

Observations of Agile software practices reveal, however, an erosion of these practices and a general change in the meaning of 'team' and 'project'. In very small companies, where customers far outnumber the developers, each developer is effectively 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 seen similar patterns emerge in larger organizations in Denmark as well as in the United States, as briefly illustrated in the following examples.

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. In other words, can a developer 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. We believe they reveal a general trend in the application of Agile software development, at least in certain kinds of software development organizations.

3.4 Whatever happened to the 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 drivers behind these developments, but the quest for higher productivity and shorter turn-around 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 Continuous integration, continuous deployment and DevOps: Standardize, measure, improve!

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 organisations' capabilities to react fast and release new software versions frequently – possibly several times per day. [21].

Continuous integration, continuous deployment and DevOps aims towards a continuous flow from task definition over programming to test, delivery and deployment. It depends on standard architectures and automated build, test, and deployment processes. Measurements of the software development process and the product in use are fed back to development to define changes or additions to the product and improvements to the process.

While there is still not much research about DevOps practices and their implications, it appears that the move to DevOps, among others, is accompanied by a further reduction in team control and authority towards outside managers supported by extensive metrics [19].

5 Discussion: Adopting a Critical Response

Agile methods arguably co-evolved with the proliferation of Internet use and ubiquitous access to the World Wide Web [9]. What is certain is that many aspects of responding to "Internet speed" 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 [28, 34]; gravitating to fixed architectural patterns; componentization and reuse, for example, as embodied in microservices [35]; performing quality assurance earlier and more frequently [48]; 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 [15] and even burnout [18].

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/guiding theories often referenced, consistently reflect Agile principles in their own espoused world-view. See Table 1 below.

Table 1 Relating Traditions in Scandinavian Research to Agile Principles

Agile Principle	Related Research Tradition
Emphasis on Individuals	Sociotechnical Systems
Emphasis on balancing quality with human concerns	Sociotechnical Systems Critical Theory
Learning and adaptation	Appreciative Systems
Participative Development	Participatory Design
Accepting and Leveraging Change	Soft Systems Methodology
Self-organizations	Complex Adaptive Systems
Minimum Viable Product	Sociotechnical Systems
Reflective Practice	Reflective Practice

(Adapted from Nerur et al. [42])

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 learning [18, 42]. 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 [20, 30]. In this regard, Checkland's Soft Systems Methodology [16] strongly considers a general systems theoretic component.

From both 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 [36]. 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 [40]. 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 [11] and Scrum [51]. 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 [40]. 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 Agile's Manifesto and principles may be visible at the surface, but are they still implementable with 1-week or 2-week sprints? The necessity of standardization inherent in all process optimizations is understandable, but the epistemology of technical rationality [49] 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 [23], 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 [37]. 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 [44]. 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 research, its value may persist inherently given the perturbations our own systems cause to known order. Rhetorically, are we hoist by our own petard?

Howcroft and Trauth [27] 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 nonperformance (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 [33] offer a set of principles for critical research from which some validation of the arguments made in this paper is possible. See Table 2 below.

Table 2 Applying Principles the of CST to the Agile Paradox (Myers and Klein, 2011)

Myers and Klein Critical Theory Elements	**This Paper's Position**
Insight	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.
***Critique**: Core Concepts from Critical Social Theorists*	Habermas: reason in practice requires reflective judgment and critique such that the renewal of practice is possible by "seeing" the totality of the problem.
***Critique**: Taking a value position*	The lean-influenced continuous delivery evolution of Agile is relegating the reflective practice component of Agile development for the developers involved.
***Critique**: Revealing and Challenging Norms*	In a neo-Taylorist manner, DevOps emphasizes the feedback of automation and artificial intelligence over the time

	for reflection.
***Transformation**: Individual Emancipation*	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.
***Transformation**: Improvements to Society*	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 innovation?
***Transformation**: Improving Social Theories*	Argyris and Schön [4] theory of action and Schön's [49, 50] 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 Information Systems Researchers. In a previous call to action (emergent about the same time that Agile methods had emerged), which largely resulting 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, 45, 52]? 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 "smart" and "lean" processes into our development cycles requires some pause, caution, and reflection in order to appreciate what is gained and lost.

6 Conclusion

This paper has presented a position which characterizes how an evolution towards continuous delivery and DevOps that 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 retrospective or review? 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 and DevOps 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 also 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.

This paper is offered as both a metaphoric "discussant" and a call to action. The emphasis on rapid value and delivery, 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; some rest and refit – learning from experience and (re)calibrating repertoire – is necessary. We have presented an argument that the learning paradox (another form of unpaid debt) arising from agility may have deleterious effects not only on the quality of the product (which has proven to be negotiable in the "Internet speed" era), 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 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. Beck, K. (2004). *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley, Reading, MA.
12. Berczuk, S. (2007). Back to basics: The role of agile principles in success with an distributed scrum team. *Agile Conference (AGILE)*.
13. Binstock, A. (2012). Interview with Alan Kay, *Dr. Dobb's*, July 10.
14. Bollinger, T. B., & McGowan, C. (1991). A Critical Look at Software Capability Evaluations. *IEEE Software* 8(4), 25-41.
15. Cancialosi, C. (2016). DevOps, culture change and the brass ring of velocity. *Forbes*, March 28.
16. Checkland, P. B. (1981). *Systems Thinking, Systems Practice*. John Wiley & Sons, Chichester, UK.
17. Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research* 20(3), 329-354.
18. Davis, J., & Daniels, K. (2016). *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*. O'Reilly Media, Sebastopol, CA.
19. Dennehy, D., & Conboy, K. (2016). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software*.
20. Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology* 50(9-10), 833-859.
21. Fitzgerald, B., & Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*.
22. Haase, V., Messnarz, R., Koch, G., Kugler, H. J., & Decrinis, P. (1994). Bootstrap: Fine-Tuning Process Assessment. *IEEE Software* 11(4), 25-35.
23. Habermas, J. (1972). *Knowledge and Human Interests*. Heinemann Educational, London, UK.
24. 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.
25. Hoda, R., Babb, J. S., & Nørbjerg, J. (2013). Toward Learning Teams. *IEEE Software* 30(4), 95-98.
26. 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
27. Howcroft, D., & Trauth, E. M., Eds. (2005). *Handbook of Critical Information Systems Research: Theory and Application*. Edward Elgar Publishing, Cheltenham, UK.
28. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, Reading, MA.
29. Ipeirotis, P. G., Provost, F., & Wang, J. (2010). Quality management on Amazon Mechanical Turk. *ACM SIGKDD Workshop on Human Computation*, 64-67.
30. 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.

31. Levitt, G. M. (2000). *The Turk, Chess Automation*. McFarland & Company, Incorporated Publishers.
32. Mathiassen, L., Pries-Heje, J., & Ngwenyama, O., Eds. (2002). *Improving Software Organizations. From Principles to Practice*. Addison-Wesley, Reading, MA.
33. Myers, M. D., & Klein, H. K. (2011). A Set of Principles for Conducting Critical Research in Information Systems. *MIS Quarterly* 35(1), 17-36.
34. Neely, S., & Stolt, S. (2013). Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). *Agile Conference (AGILE)*.
35. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Sebastopol, CA.
36. 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.
37. Ngwenyama, O. K. (1993). Developing end-users' systems development competence. An exploratory study. *Information and Management* (25), 291-302.
38. Ngwenyama, O., & Nielsen, P. A. (2003). Competing Values in Software Proces Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. *IEEE Transactions on Engineering Management* 50(1), 100-112.
39. Nielsen, P. A., & Nørbjerg, J. (2001). Assessing Software Processes: Low Maturity or Sensible Practice. *Scandinavian Journal of Information Systems* 13(1-2), 23-36.
40. 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.
41. 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.
42. 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.
43. Olsson, H. H., & Bosch, J. (2014). Post-deployment Data Collection in Software-Intensive Embedded Products. *Continuous Software Engineering*, J. Bosch, Springer, 143-154.
44. Orlikowski, W. J., & Baroudi, J. J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research* (2), 1-28.
45. 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.
46. 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, Pitssburgh, PA.
47. 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.
48. Roche, J. (2013). Adopting DevOps Practices in Quality Assurance. *Communications of the ACM* 56(11), 38-43.
49. Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, NY.
50. Schön, D. A. (1987). *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. Jossey-Bass, San Francisco, CA.
51. Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (vol. 1). Prentice Hall, Upper Saddle River, NJ.
52. Weber, R. (2003). Still desperately seeking the IT artifact. *MIS Quarterly* 27(2), iii-xi.

53. Williams, L. (2012). What agile teams think of agile principles. *Communications of the ACM* 55(4), 71-76.