Fit for Continuous Integration: How Organizations Assimilate an Agile Practice

Completed Research Paper

Alexander Eck
University of St.Gallen
alexander.eck@unisg.ch

Falk Uebernickel
University of St.Gallen
falk.uebernickel@unisg.ch

Walter Brenner
University of St.Gallen
walter.brenner@unisg.ch

Abstract

Despite being a cornerstone of agile development, surprisingly little is known about how organizations assimilate continuous integration (CI) and what organizational changes the practice implies. Through a systematic literature review complemented by case study research we address this gap and develop a conceptual framework describing organizational implications of CI assimilation. We employ adoption theory of technology-induced innovation as theoretical lens and argue that as organizations transform through the assimilation stages, ambiguity increases, forcing them to think in alternatives and take trade-offs. To the existing body of knowledge on agile development assimilation we add an in-depth study of a distinct agile practice.

Keywords

Continuous integration, agile, systems development, assimilation stages, innovation adoption.

Introduction

Despite being a cornerstone of agile development, there is surprisingly little knowledge about how organizations assimilate continuous integration (CI) and which implications CI introduction entails. CI is a practice where developers integrate and test frequently (Fowler 2006). CI was introduced to agile software development with Extreme Programming (XP) (Beck and Andres 2004) and the Agile Principles (Fowler and Highsmith 2001). CI relies on an underlying IT infrastructure (Humble and Farley 2010) that enables a process which can be summarized as depicted in Figure 1: based on a common codebase to which all developers contribute, the source code is compiled and integrated, deployed on a test environment and tested automatically, with the possibility to deploy working software into production (Humble and Molesky 2011). Although seemingly straightforward, CI – and as extension any agile practice – is an agent of change for any organization that implements it. As Toleman et al. (2004) observed, there is no binary choice to be made – assimilation of CI rather is a process that transforms the (IT) organization.

With this study we aim to identify which organizational implications CI exerts and therefore ask:

How do organizations assimilate CI and what organizational changes does the practice imply?

Ståhl and Bosch (2014) produced a systematic overview of CI practices and their differences. In contrast to their work, we do not focus on the technical aspects of CI (e.g. compilation frequency) but rather on the assimilation process. Still we regard their work as a valuable foundation, as the authors discuss some organizational implications, e.g. division of labor. In pursuing our research, we follow the lead of two prior studies (Russo et al. 2013; Conboy et al. 2007) and choose adoption theory of technology-induced innovation (Cooper and Zmud 1990) as theoretical lens for investigation. According to this model, organizations undergo three *pre-implementation* and three *post-implementation* assimilation stages, respectively. We limit our investigation to the latter, namely *acceptance*, *routinization*, and *infusion*: When in *acceptance* stage, organizational members start to accept the innovation and use it in their daily

work. An innovation enters the *routinization* stage when usage is perceived as normal activity and organizational processes are adjusted accordingly. The *infusion* stage is characterized as sophisticated and extensive usage.

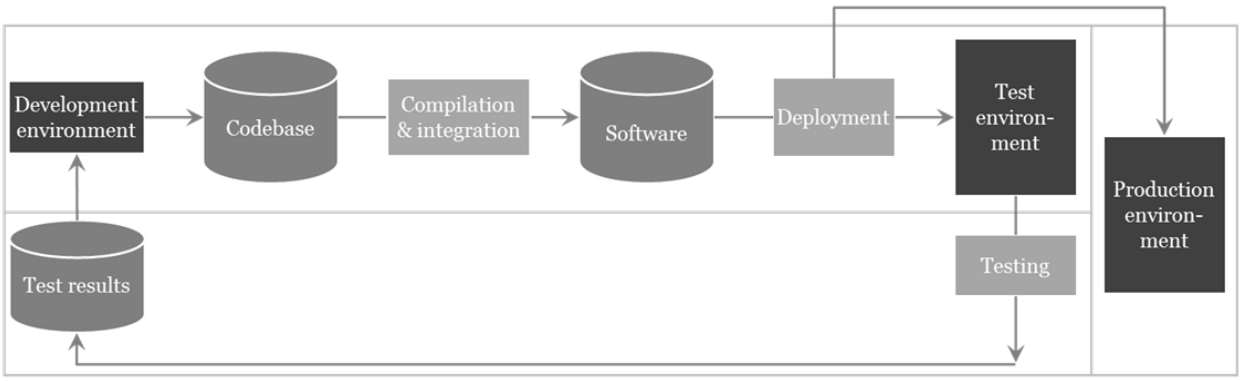

Figure 1. Illustrative and simplified architecture of a CI infrastructure

The remainder of this paper is structured as follows. In the next section the research approach is presented, which aimed at deriving a conceptual framework for CI assimilation. In the subsequent section we discuss its individual elements, i.e. the identified organizational implications of CI. We conclude with a summary of our main contribution, a reflection of the limitations, and suggestions for future research.

Research Approach and Derivation of Conceptual Framework

The objective of this study is to increase our understanding of the organizational implications that CI assimilation entails. As a large stream of literature already exists in the area of agile practices (Dingsøyr et al. 2012), we selected a concept-centric literature review (Webster and Watson 2002) as our primary method to carry out research.

Because we expected that few sources would specifically address organizational implications, we performed full-text keyword search on several databases to identify the body of literature on CI. The search unearthed *649 hits*, which we subsequently reduced to *34 relevant studies* after full content analysis against the qualitative exclusion criteria (Table 1 and Table 2). We anticipated such a large reduction, because our keyword search was broad while our qualitative criteria were narrow. We then performed a backward/forward search and located *9 additional studies*. From this list of *43 studies*, we extracted statements relevant to organizational implications of CI and grouped them by thematic similarity, in a procedure informed by the *open coding* technique developed by Corbin and Strauss (2008).

Inclusion criteria (keyword search string)	
	"continuous integration" OR "continuous build" OR "continuous deployment" OR "continuous delivery"
AND	agile or agility
AND	"information system*" OR "information technology"
AND	"system* development" or "software development"
Exclusion criteria (qualitative criteria)	
1	Does not investigate CI, just mentions the term
2	Does not investigate CI in an industry setting – educational or academic setting instead
3	Does not discuss any organizational implication of CI assimilation – technical discussion instead
4	Does not present any empirical data – conceptual work instead
5	Does not present qualitative data obtained from case study or action research

Table 1. Literature review inclusion and exclusion criteria

Source	Hits	Relevant
Keyword search		
AISeL	43	2
EBSCOhost	0	0
Emerald	6	0
IEEE Xplore	135	6
ScienceDirect	70	4
SpringerLink	358	20
Wiley Online Library	37	2
Forward/backward search		9
Total	*649*	*43*

Table 2. Literature review results

In total we identified *14 distinct organizational implications* of CI (see Table 4). For being able to locate each of these implications on the three assimilation stages (*acceptance*, *routinization*, *infusion*) we conducted primary research, as complement to the results obtained via literature review. We selected a multiple case study design, because CI assimilation and its organizational implications cannot be studied outside its context (Yin 2009). We opted for theoretical sampling with replication logic (Dubé and Paré 2003) and selected IT departments of five large financial institutions active in Continental Europe that practiced agile development. We chose the financial industry because it is highly dependent on IT-driven innovations (Broadbent and Weill 1993). The dominant form of data collection was interviews, partially complemented by company-provided material. In order to minimize interviewer bias (Patton 2002), we chose to follow a semi-structured interview guide that mentioned CI, but not the individual terms identified through literature review. Instead, matching of the organizational implications encountered at the companies was performed retroactively with help of the interview transcripts – exemplary statements and associated codes are given in Table 3. We coded the interviews exclusively for the 14 organizational implications distilled from literature review, and did not intend to uncover additional items.

Following this data analysis we estimated the CI assimilation stage of each company. We applied the Rasch heuristic (Lahrmann et al. 2011) to the case study data, with the goal to elicit which organizational implication is associated with which assimilation stage. The Rasch heuristic estimates the relative sophistication of an item. In line with similar research (Conboy et al. 2007; Russo et al. 2013) we assume that assimilation progress is reflected by a higher degree of sophistication. Following this approach the individual organizational implications are associated with a particular assimilation stage. This procedure enabled us to correlate the likely association of identified organizational implications with assimilation stages; that is, we were able to place the individual items on the acceptance, routinization, or infusion stage, respectively.

Company	Org. implication	Exemplary statement
A	Providing CI at project start	*We have a CI infrastructure, which has the goal to make the status transparent from start of the project.*
B	Institutionalizing CI	*The development environment with Eclipse, SVN and Jenkins is given. If a developer wants a change, he needs to formulate his request.*
C	Dealing with test failures right away	*Really with every check-in [...] a unit test runs. Only when these were successful we run load and performance tests automatically. And just then [...] the package is created and published [...] on a system test environment for acceptance testing.*
D	Devising an assimilation path	*All the bank moves from SVN to Git [...]. Today there are a lot of manual steps involved, and part of this project is to [...] improve production integration flow.*
E	Introducing CI for complex systems	*It was a challenge to have automated deployments on the mobile platform as well. [...] But now every morning we build the e-banking and deploy on both platforms for test.*

Table 3. Exemplary quotes from company interviews

The result – a conceptual framework describing organizational implications of CI assimilation – is given with Table 4. Additionally to the three assimilation stages, we clustered the findings into four thematic groups to increase clarity.

	Acceptance	S	C	Routinization	S	C	Infusion	S	C
Purposeful assimilation	Devising an assimilation path	4	A,D				CI assimilation metrics	3	
Processes & organization	Overcoming initial learning phase	3	C,D	Institutionalizing CI	3	A,B,C,D	Devising a branching strategy	4	
				Clarifying division of labor	5	A,B,D			
				CI and distributed development	5	A			
Testing	Dealing with test failures right away	4	A,B,C,E	Mastering test-driven development	10	A,C	Decreasing test result latency	15	
							Fostering customer involvement in testing	2	
IT infra-structure	Introducing CI for complex systems	7	A,C,E	Providing CI with project start	6	A,B,C,D	Extending CI beyond source code	4	
Count	4			5			5		

S: number of sources in which organizational implication is described

C: case study companies which have implemented the organizational implication

Table 4. Post-introductory organizational implications of CI

Organizational Implications of CI Assimilation

In the following, we discuss each conceptual framework element, organized along the four groups introduced above.

Purposeful Assimilation

Purposeful assimilation relates to devising a CI assimilation plan and monitoring its progress.

Devising an Assimilation Path

Sources suggest various approaches to drive CI assimilation, which we characterize as *low-hanging fruits*, *extend nucleus*, and *challenge-oriented*. All approaches emphasize putting CI in a larger context of overall agile development assimilation. Stober and Hansmann (2010) propose a *low-hanging fruits* approach: those aspects should be introduced first that are easily implemented and require few behavioral changes. As the organizational members get accustomed to the change it can then explore more sophisticated practices, one of which is CI. Olsson et al. (2012) suggest an *extend nucleus* approach: starting with a core CI system that automates integration and testing, the ambition and reach of CI is extended continuously, until CI is used to leverage operational software for experimentation. Finally Conboy et al. (2007) suggest a *challenge-oriented* approach: after prioritization of current challenges, the organization should implement those agile practices first that are likely to have the highest return. In conclusion, there is consensus that CI is not a means in itself and should be seen as component of agile development. Differences arise on who should have the initiative – ranging from a bottom-up attitude like *low-hanging fruits* up to a top-down stance as *challenge-oriented*.

CI Assimilation Metrics

Several authors discuss how to measure assimilation of an agile development practice such as CI. Stober and Hansmann (2010) propose an *outcome* metric, namely the number of integration errors towards the end of a release cycle. A high number indicates that CI has not been adopted well. As CI helps development teams to start integration activities early while continuing development until late in a project, there should be no observable spike in integration effort. An *input* metric is proposed by two sources (Conboy et al. 2007; Wang et al. 2012): CI assimilation – and to this end any agile development practice – can be expected to have progressed into a later stage if reflective activities such as continuous improvement processes have been institutionalized. It is worth noticing that authors dismiss *time since*

introduction as being suitable, because this metric has no qualitative component to it (Conboy et al. 2007; Wang et al. 2012). To summarize, we were able to identify one *outcome* metric and one *input* metric that complement each other.

Processes & Organization

Under *processes & organization* we cluster all activities that aim at successfully assimilating CI, for example changes in governance or newly established organizational bodies.

Overcoming Initial Learning Phase

Learning a complex practice such as CI takes time – it is very likely that productivity decreases before any positive effects materialize. Authors identified several sources of learning areas, both on *individual level* and on *group level*. *Individual* learning relates to handling the CI tools and understanding the processes which they facilitate (Silva et al. 2005). *Group* learning refers to accepting necessary changes in collaboration, e.g. a change in how testing is carried out (Williams et al. 2011). A proven way to accelerate learning on both levels is providing training (Nyman et al. 2010). In conclusion, it is critical that organizations prepare for a dip in productivity, accept this initial negative impact as investment, and minimize the learning phase through targeted training.

Institutionalizing CI

Three approaches have been discussed for how to institutionalize CI: *shared service*, *committee*, and *communities of practice*. Offering CI as *shared service* means establishing a dedicated team that maintains and improves CI, and which centrally provides training and other support. On the downside, all teams involved with CI depend on the central roadmap and cannot deviate easily (Heidrich et al. 2013). Alternatively or as complement, a *committee* might be established. It consists of subject matter experts across the organization, who discuss the CI roadmap and issue authoritative guidelines for CI use (Williams et al. 2011). A further alternative are *communities of practice*: usually self-emergent, they are sustained by company-internal thought leaders who drive CI improvements forward and provide guidance on request. Through granting these thought leaders time for engaging in such activities, organizations choose to promote this model (Woodward et al. 2010). To summarize, different models exist to foster CI use and sophistication, ranging from centralized to decentralized.

Clarifying Division of Labor

CI connects previously separated organizational functions through process automation, e.g. development and release management. A number of authors suggest that it is vital to clarify division of labor, such as establishing hand-over procedures along the CI chain (Maruping 2010; Thiyagarajan and Verma 2009) or introducing suitable policies, e.g. who is allowed to perform branch merging (Williams et al. 2011). However, organizations should resist re-establishing exactly those silos that CI is supposed to help overcome. A possible approach is to additionally assign source code ownership: across the entire CI chain, owners are made responsible to resolve integration errors affecting their code (Moore and Spens 2008; Stober and Hansmann 2010).

CI and Distributed Development

CI has the potential to increase collaboration among spatially distributed development through *replication, synchronization,* and *transparency*. Modern CI systems support local *replication* of the codebase, which makes the CI chain resilient against connectivity disruptions (Sauer 2010; Poole 2004). The different development teams *synchronize* their work with every integration (Hillegersberg et al. 2011; Avritzer et al. 2010). There is a trade-off between synchronization frequency and communication needs, however. If changes are instantly synchronized against the full codebase, then communication between locations is likely to increase in case of encountered integration errors (Avritzer et al. 2010). Lastly, CI creates *transparency* about the project status, because every team member has access to the codebase (Bose 2008; Avritzer et al. 2010). In summary, the sources argue that the various elements of CI collectively foster collaboration in distributed development settings.

Devising a Branching Strategy

A major feature of any CI system is *branching*, i.e. instantiating a copy of the common codebase, locally working with this copy, and feeding the changes back into the codebase after local testing. In its most basic use branching enables concurrent development. Practitioners leverage this feature to implement sophisticated strategies, which we characterize as either *feature-motivated* or *test-motivated*. A commonly observed *feature-motivated* strategy is to branch out every new feature, in order to not jeopardize integrity of the current codebase (Wilcox et al. 2007). Building on this idea, Stober and Hansmann (2010) propose cross-functional development teams when exploring risky technology changes not yet on the roadmap. These teams exist as long as the branch does, i.e. they are being dismantled once their goal is accomplished. To maintain consistency of the overall codebase, long-lived source code ownerships are assigned in parallel. Of the various *test-motivated* approaches, Dowling (2013) presents a variant in which a testing branch is created every two weeks. A dedicated team performs tests on this branch, while development continues uninterrupted. All encountered errors that could be fixed within this period are fed back into the codebase; a new test branch is then created and the cycle begins anew. Similarly, Ali et al. (2012) propose performing complex and time consuming tests on a dedicated branch. Interestingly, none of these authors mention a hybrid strategy, combing *feature-motivated* and *testing-motivated* approaches.

Testing

Although conceptually identical to the *processes & organization* group above, we created a separate *testing* cluster, because testing is a major consideration in CI.

Dealing with Test Failures Right Away

Several authors argue that CI should be leveraged to perform automated testing as frequently as possible. Developers should be forced to deal with encountered failures right away and aim for high-quality code from project start, instead of letting errors accumulate until late in the project (Talby et al. 2005; Avritzer et al. 2010; Bos and Vriens 2004). If test granularity is high, the root cause of an error can be located fairly easy (Grenning 2001).

Mastering Test-driven Development

CI enables *test-driven* development, a practice in which a test is written prior to functional implementation. Developers alternate between writing tests and implementing functionality on a minute-by-minute basis (Beck 2003), which is a significant change of habit from the traditionally sequential process of implementing and testing (Toleman et al. 2004; Greene 2004; Stober and Hansmann 2010; Nyman et al. 2010). To achieve large code coverage, i.e. tests exist for a large part of the source code, a high granularity of testing is aimed at. Coupled with the dynamic of constant alternation, it becomes infeasible to elaborate exhaustive test plans (Stolberg 2009). Developers hence need to learn effective testing skills that they can apply with dexterity (Pikkarainen and Wang 2011; Heidrich et al. 2013; Ambu and Gianneschi 2003; Stolberg 2009). Although data shows that writing tests is not time-consuming (Abrahamsson and Koskela 2004), developers need to grow confident that this extra effort results in increased quality (Ambu and Gianneschi 2003).

Decreasing Test Result Latency

Ideally there is no discernible time lag between checking in code and receiving test results, in order not to disturb the flow of development work (Rasmusson 2004). As a project progresses, however, the latency between code check-in and test results is likely to increase (Rogers 2004). We summarize the numerous strategies tackling this challenge as *infrastructure upgrade*, *automation*, *test optimization*, *testing hierarchy*, and *decomposition*. Two straightforward approaches are seeking an *infrastructure upgrade*, either in form of higher-performance hardware or improved software (Andersen and Amdor 2009; Rasmusson 2004), as well as increasing *automation* of the CI chain (Lippert et al. 2001; Nyman et al. 2010). To this end, several authors explore the possibilities of cloud computing (Riungu-Kalliosaari et al. 2012; Oza et al. 2013; Karunakaran 2013). *Test optimization* refers to having developers optimize those tests that need longest to complete, e.g. through simplifying its logic (Rogers 2004; Rasmusson 2004). A

more sophisticated strategy is establishing a *testing hierarchy*, with tests of single code fragments on the lowest level and usage behavior tests on the highest level, respectively. Depending on the level, tests are carried out at different frequencies (Greene 2004; Jääskeläinen et al. 2008; Ali et al. 2012; Rogers 2004). For example, Jääskeläinen et al. (2008) propose to perform *unit tests* with every code check-in, while executing *use case tests* less frequently. These tests might be executed by specialized teams, in order to lower the workload on developers (Stober and Hansmann 2010; Wildt and Prikladnicki 2010; Moore and Spens 2008). The most common form of d*ecomposition* is creating software modules which are tested independently, with full integration and testing carried out less frequently (Rasmusson 2004; Nyman et al. 2010). Another approach is splitting the CI chain in different segments and providing the developer with feedback after each step (Talby et al. 2005). In conclusion, a number of authors stress the importance of achieving a low latency between check-in and test results with proposals ranging from obvious (*infrastructure upgrade*) to intricate (*decomposition*).

Fostering Customer Involvement in Testing

CI is commonly seen as practice aimed to support the development team. But it can also be utilized to foster customer involvement, for example through integrating customers in testing activities. Sources suggest that customers should formulate *acceptance tests* for later automated evaluation on their own. To accept this demand, customers need to be convinced that it is an efficient way to formulate requirements and ultimately leads to better software (Kane et al. 2006; Toleman et al. 2004).

IT Infrastructure

Under *IT infrastructure* we subsume all organizational implications emanating from the technology part of CI.

Introducing CI for Complex Systems

From an IT infrastructure perspective CI faces complexity stemming from *target environment*, *testing suite*, and *technology*. When software is intended to be deployed on heterogeneous target environments – e.g. mobile devices – the CI system should provide the necessary facilities to integrate and test on these environments simultaneously (Wolff-Marting et al. 2013). The *testing suite* also adds complexity: it is likely that unit tests and use case tests mandate different tools, for example (Mangalaraj et al. 2009). Finally, it is common for software to be developed with different *technologies* simultaneously, e.g. through combing programming languages or developing embedded systems that integrate hardware and software components (Kasoju et al. 2013; Waterman et al. 2013). In conclusion, it is very likely that CI infrastructure itself becomes heterogeneous and complex. Mordinyi et al. (2011) propose a common service bus architecture around which the various elements are brought together. There might be cases when appropriate tools do not exist, e.g. when developing with specialized, legacy, or emergent technology. Organizations can either accept the fact and hope that appropriate tools become available in future (Kane 2003), or they develop and maintain necessary IT infrastructure components themselves (Kasoju et al. 2013; Waterman et al. 2013; Heidrich et al. 2013).

Providing CI with Project Start

It might be that CI is not available directly with project start: extensive manual preparation might be needed to set up the tool chain, or it needs to be amended with new elements that are required for the current project (Conboy et al. 2007; Poole 2004; Verweij, P. J. F. M. et al. 2010). Several authors argue that such delays may jeopardize CI altogether. If CI becomes available in the course of a project, developers are likely to abandon it (Kasoju et al. 2013; Avritzer et al. 2010). In addition, major architectural changes might become necessary to prepare the codebase for effective CI (Kane 2003).

Extending CI Beyond Source Code

A number of sources propose capturing all artifacts of software development with CI, not just the source code (Bos and Vriens 2004; Kane 2003). Otherwise, changes in uncontrolled artifacts, e.g. configuration files, firmware, etc. might cause integration and testing errors that are hard to reproduce (Larsson et al.

2009). However, organizations should carefully evaluate the benefits against its costs; it might make sense to purposefully leave gaps in the CI system (Talby et al. 2005).

Conclusion, Research Limitations, and Future Work

Analysis of the extant literature shows that CI is an agile development practice inducing numerous changes as it is being assimilated by the organization. Through the theoretical lens of innovation adoption (Cooper and Zmud 1990) we suggested a conceptual framework consisting of 14 organizational implications of CI. Scholars who intend to quantitatively examine the extent of CI assimilation in organizations might find this framework to be a solid foundation for creating a set of operationalized constructs.

From a practitioner's point of view, the comparison of each stage suggests that as organizations progress with CI assimilation, ambiguity increases: While at *acceptance* stage a number of *good practices* exist, e.g. how to handle test failures, organizations on later stages have to weigh trade-offs, such as deciding how exactly to pursue branching. But there are exceptions: Our case study data shows that organizations implement CI for complex systems rather early in their assimilation process, despite its ambiguity. A case in point is Company E, which introduced CI for two target environments, despite not being proficient in other areas of CI. This observation might be skewed by our case study design, as we regarded only one industry setting. The developed conceptual framework and in particular the framework elements might gain validity through investigation in other industries.

This study shows that CI and testing are intricately linked. Future research might capitalize on this insight and examine its innovative potential. For example the branching mechanism of CI allows creating variants of the same software cost-efficiently. This could be leveraged to derive working prototype software that is presented to the end-user, which would intensify customer interaction during software development. Seeds of this idea have already been proposed (Olsson et al. 2012).

According to our data, none of the organizations in our case study reached the *infusion* stage. While this might be simply attributed to missing information, it would be worthwhile to ask why this is the case. Conboy et al. (2007) offer a plausible explanation: organizations progress from one stage to another only in the presence of imminent challenges. A corollary should then also hold true: a strategic advantage might result when transitioning towards a later assimilation stage faster than the competition. Future work might explore possible correlations between CI assimilation rate and competitive edge, e.g. in the form of higher increased innovation output. To this end, it might be worthwhile to investigate assimilation of other agile practices in a similar manner as was done in this study with CI.

We note that our research has limitations that are attributed to the chosen research method. We mainly systematized existing knowledge from literature. It is possible that we did not capture all existing sources with our search. Also, we assume it likely that additional organizational implications of CI exist. As first mitigating action we performed comple–mentary data collection through case study research, and those results were consistent with the literature review findings. Future research might return to the case study method and feature more in-depth longitudinal data collection, e.g. through means of shadowing (Patton 2002).

References

Abrahamsson, P., and Koskela, J. 2004. "Extreme programming: a survey of empirical data from a controlled case study," ISESE 2004 Proceedings , pp. 73–82.

Ali, N. B., Petersen, K., and Mäntylä, M. V. 2012. "Testing highly complex system of systems: An industrial case study," ESEM 2012 Proceedings , pp. 211–220.

Ambu, W., and Gianneschi, F. 2003. "Extreme Programming at Work," in Extreme Programming and Agile Processes in Software Engineering, M. Marchesi, and G. Succi (eds.), Springer, pp. 347-350.

Andersen, T. J., and Amdor, L. E. 2009. "Leveraging Maven 2 for Agility," AGILE 2009 Proceedings , pp. 383–386.

Avritzer, A., Bronsard, F., and Matos, G. 2010. "Improving Global Development Using Agile: How Agile Processes Can Improve Productivity in Large Distributed Projects," in Agility Across Time and Space, D. Šmite, N. B. Moe, and P. J. Ågerfalk (eds.), Springer, pp. 133-148.

Beck, K. 2003. Test-driven development, Addison-Wesley.
Beck, K., and Andres, C. 2004. Extreme programming explained, Addison-Wesley.
Bos, E., and Vriens, C. 2004. "An Agile CMM," in Extreme Programming and Agile Methods - XP/Agile Universe 2004, C. Zannier, H. Erdogmus, and L. Lindstrom (eds.), Springer, pp. 129-138.
Bose, I. 2008. "Lessons Learned from Distributed Agile Software Projects: A Case-Based Analysis," Communications of the AIS (23), pp. 620–632.
Broadbent, M., and Weill, P. 1993. "Improving business and information strategy alignment: Learning from the banking industry," IBM Systems Journal (32:1), pp. 162–179.
Conboy, K., Pikkarainen, M., and and Wang, X. 2007. "Agile practices in use from an innovation assimilation perspective: a multiple case study," ICIS 2007 Proceedings.
Cooper, R. B., and Zmud, R. W. 1990. "Information Technology Implementation Research: A Technological Diffusion Approach," Management Science (36:2), pp. 123–139.
Corbin, J. M., and Strauss, A. L. 2008. Basics of qualitative research: Techniques and procedures for developing grounded theory, Sage Publications.
Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. 2012. "A decade of agile methodologies: towards explaining agile software development," Journal of Systems and Software (85:6), pp. 1213–1221.
Dowling, P. 2013. "Successfully transitioning a research project to a commercial spin-out using an agile software process," Journal of Software (in Press).
Dubé, L., and Paré, G. 2003. "Rigor in information systems positivist case research: current practices, trends, and recommendations," MIS Quarterly (27:4), pp. 597–635.
Fowler, M. 2006. Continuous integration. http://martinfowler.com/articles/continuousIntegration.html. Accessed 30 January 2014.
Fowler, M., and Highsmith, J. 2001. "The agile manifesto," Software Development (9:8), pp. 28–35.
Greene, B. 2004. "Agile Methods Applied to Embedded Firmware Development," Agile Development 2004 Proceedings , pp. 71–77.
Grenning, J. 2001. "Launching extreme programming at a process-intensive company," IEEE Software (18:6), pp. 27–33.
Heidrich, J., Oivo, M., Jedlitschka, A., and Baldassarre, M. (eds.) 2013. Product-Focused Software Process Improvement, Springer.
Hillegersberg, J., Ligtenberg, G., and Aydin, M. 2011. "Getting Agile Methods to Work for Cordys Global Software Product Development," in New Studies in Global IT and Business Service Outsourcing, J. Kotlarsky, L. Willcocks, and I. Oshri (eds.), Springer, pp. 133-152.
Humble, J., and Farley, D. 2010. Continuous Delivery, Addison-Wesley.
Humble, J., and Molesky, J. 2011. "Why Enterprises Must Adopt Devops to Enable Continuous Delivery," in Devops: A Software Revolution in the Making? P. Debois (ed.), Arlington, MA, USA, pp. 6–12.
Jääskeläinen, A., Katara, M., Kervinen, A., Heiskanen, H., Maunumaa, M., and Pääkkönen, T. 2008. "Model-Based Testing Service on the Web," in Testing of Software and Communicating Systems, K. Suzuki, T. Higashino, A. Ulrich, and T. Hasegawa (eds.), Springer, pp. 38-53.
Kane, D. 2003. "Introducing agile development into bioinformatics: an experience report," Agile Development 2003 Proceedings , pp. 132–139.
Kane, D. W., Hohman, M. M., Cerami, E. G., McCormick, M. W., Kuhlmman, K. F., and Byrd, J. A. 2006. "Agile methods in biomedical software development: a multi-site experience report," BMC Bioinformatics (7:1), pp. 1-12.
Karunakaran, S. 2013. "Impact of Cloud Adoption on Agile Software Development," in Software Engineering Frameworks for the Cloud Computing Paradigm, Z. Mahmood, and S. Saeed (eds.), Springer, pp. 213-234.
Kasoju, A., Petersen, K., and Mäntylä, M. V. 2013. "Analyzing an automotive testing process with evidence-based software engineering," Information and Software Technology (55:7), pp. 1237–1259.
Lahrmann, G., Marx, F., Mettler, T., Winter, R., and Wortmann, F. 2011. "Inductive Design of Maturity Models: Applying the Rasch Algorithm for Design Science Research," in Service-Oriented Perspectives in Design Science Research, H. Jain, A. Sinha, and P. Vitharana (eds.): Springer, pp. 176-191.
Larsson, S., Myllyperkiö, P., Ekdahl, F., and Crnkovic, I. 2009. "Software product integration: A case study-based synthesis of reference models," Information and Software Technology (51:6), pp. 1066–1080.
Lippert, M., Roock, S., Tunkel, R., and Wolf, H. 2001. "Stabilizing the XP process using specialized tools," XP 2001 Proceedings , pp. 38–41.

Mangalaraj, G., Mahapatra, R., and Nerur, S. 2009. "Acceptance of software process innovations – the case of extreme programming," EJIS (18:4), pp. 344–354.
Maruping, L. M. 2010. "Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges," in Agility Across Time and Space, D. Šmite, N. B. Moe, and P. J. Ågerfalk (eds.), Springer, pp. 11-30.
Moore, E., and Spens, J. 2008. "Scaling Agile: Finding your Agile Tribe," AGILE 2008 Proceedings, pp. 121–124.
Mordinyi, R., Moser, T., Biffl, S., and Dhungana, D. 2011. "Flexible Support for Adaptable Software and Systems Engineering Processes," SEKE 2011 Proceedings , pp. 608–612.
Nyman, R., Aro, I., and Wagner, R. 2010. "Automated Acceptance Testing of High Capacity Network Gateway," in Agile Processes in Software Engineering and Extreme Programming, A. Sillitti, A. Martin, X. Wang, and E. Whitworth (eds.), Springer, pp. 307-314.
Olsson, H. H., Alahyari, H., and Bosch, J. 2012. "Climbing the "Stairway to Heaven" - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software," SEAA 2012 Proceedings, pp. 392–399.
Oza, N., Münch, J., Garbajosa, J., Yague, A., and Gonzalez Ortega, E. 2013. "Identifying Potential Risks and Benefits of Using Cloud in Distributed Software Development," in Product-Focused Software Process Improvement, J. Heidrich, M. Oivo, A. Jedlitschka, and M. Baldassarre (eds.), Springer, pp. 229-239.
Patton, M. Q. 2002. Qualitative research and evaluation methods, Thousand Oaks, Sage.
Pikkarainen, M., and Wang, X. 2011. "An Investigation of Agility Issues in Scrum Teams Using Agility Indicators," in Information Systems Development, W. W. Song, S. Xu, C. Wan, Y. Zhong, W. Wojtkowski, G. Wojtkowski, and H. Linger (eds.), Springer, pp. 449-459.
Poole, C. J. 2004. "Distributed Product Development Using Extreme Programming," in Extreme Programming and Agile Processes in Software Engineering, J. Eckstein, and H. Baumeister (eds.), Springer, pp. 60-67.
Rasmusson, J. 2004. "Long Build Trouble Shooting Guide," in Extreme Programming and Agile Methods - XP/Agile Universe 2004, C. Zannier, H. Erdogmus, and L. Lindstrom (eds.), Springer, pp. 13-21.
Riungu-Kalliosaari, L., Taipale, O., and Smolander, K. 2012. "Testing in the Cloud: Exploring the Practice," IEEE Software (29:2), pp. 46–51.
Rogers, R. O. 2004. "Scaling Continuous Integration," in Extreme Programming and Agile Processes in Software Engineering, J. Eckstein, and H. Baumeister (eds.), Springer, pp. 68-76.
Russo, N. L., Fitzgerald, G., and Shams, S. 2013. "Exploring adoption and use of agile methods: a comparative case study," AMCIS 2013 Proceedings.
Sauer, J. 2010. "Architecture-Centric Development in Globally Distributed Projects," in Agility Across Time and Space, D. Šmite, N. B. Moe, and P. J. Ågerfalk (eds.), Springer, pp. 321-329.
Silva, A., Kon, F., and Torteli, C. 2005. "XP South of the Equator: An eXPerience Implementing XP in Brazil," in Extreme Programming and Agile Processes in Software Engineering, H. Baumeister, M. Marchesi, and M. Holcombe (eds.), Springer, pp. 10-18.
Ståhl, D., and Bosch, J. 2014. "Modeling continuous integration practice differences in industry software development," Journal of Systems and Software (87:0), pp. 48–59.
Stober, T., and Hansmann, U. 2010. "Mix and Match," in Agile Software Development, Springer, pp. 145-170.
Stolberg, S. 2009. "Enabling Agile Testing through Continuous Integration," AGILE 2009 Proceedings , pp. 369–374.
Talby, D., Nakar, O., Shmueli, N., Margolin, E., and Keren, A. 2005. "A process-complete automatic acceptance testing framework," SwSTE 2005 Proceedings, pp. 129–138.
Thiyagarajan, P. S., and Verma, S. 2009. "A Closer Look at Extreme Programming (XP) with an Onsite-Offshore Model to Develop Software Projects Using XP Methodology," in Software Engineering Approaches for Offshore and Outsourced Development, K. Berkling, M. Joseph, B. Meyer, and M. Nordio (eds.), Springer, pp. 166-180.
Toleman, M., Almeida, A., and Darroch, F. 2004. "Aligning Adoption Theory with Agile System Development Methodologies," PACIS 2004 Proceedings, Paper 36.
Verweij, P. J. F. M., Knapen, M. J. R., de Winter, W. P., Wien, J. J. F., te Roller, J. A., Sieber, S., and Jansen, J. M. L. 2010. "An IT perspective on integrated environmental modelling: The SIAT case," Ecological Modelling (221:18), pp. 2167–2176.

Wang, X., Conboy, K., and Pikkarainen, M. 2012. "Assimilation of agile practices in use," Information Systems Journal (22:6), pp. 435–455.
Waterman, M., Noble, J., and Allan, G. 2013. "The Effect of Complexity and Value on Architecture Planning in Agile Software Development," in Agile Processes in Software Engineering and Extreme Programming, H. Baumeister, and B. Weber (eds.), Springer, pp. 238-252.
Webster, J., and Watson, R. T. 2002. "Analyzing the Past to Prepare for the Future: Writing a Literature Review," MIS Quarterly (26:2), pp. xiii-xxiii.
Wilcox, E., Nusser, S., Schoudt, J., Cerruti, J., and Badenes, H. 2007. "Agile Development Meets Strategic Design in the Enterprise," in Agile Processes in Software Engineering and Extreme Programming, G. Concas, E. Damiani, M. Scotto, and G. Succi (eds.), Springer, pp. 208-212.
Wildt, D., and Prikladnicki, R. 2010. "Transitioning from Distributed and Traditional to Distributed and Agile: An Experience Report," in Agility Across Time and Space, D. Šmite, N. B. Moe, and P. J. Ågerfalk (eds.), Springer, pp. 31-46.
Williams, L., Brown, G., Meltzer, A., and Nagappan, N. 2011. "Scrum + Engineering Practices: Experiences of Three Microsoft Teams," ESEM 2011 Proceedings, pp. 463–471.
Wolff-Marting, V., Hannebauer, C., and Gruhn, V. 2013. "Patterns for tearing down contribution barriers to FLOSS projects," SoMeT 2013 Proceedings, pp. 9–14.
Woodward, E. V., Bowers, R., Thio, V. S., Johnson, K., Srihari, M., and Bracht, C. J. 2010. "Agile methods for software practice transformation," IBM Journal of Research and Development (54:2), Paper 3.
Yin, R. K. 2009. Case study research: Design and methods, Sage.