Cite as: Haig-Smith, T., & Tanner, M. (2016). Cloud computing as an enabler of agile global software development. *Issues in Informing Science and Information Technology, 13,* 121-144. Retrieved from http://www.informingscience.org/Publications/3476

Cloud Computing as an Enabler of Agile Global Software Development

Timothy Haig-Smith and Maureen Tanner
University of Cape Town, Cape Town, South Africa

hgstim001@myuct.ac.za, mc.tanner@uct.ac.za

Abstract

Agile global software development (AGSD) is an increasingly prevalent software development strategy, as organizations hope to realize the benefits of accessing a larger resource pool of skilled labor, at a potentially reduced cost, while at the same time delivering value incrementally and iteratively. However, the distributed nature of AGSD creates geographic, temporal, socio-cultural distances that challenge collaboration between project stakeholders. The Cloud Computing (CC) service models of Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are similar to the aspirant qualities of AGSD as they provide services that are globally accessible, efficient, and stable, with lower predictable operating costs that scale to meet the computational demand. This study focused on the 12 agile principles upon which all agile methodologies are based, therein potentially increasing the potential for the findings to be generalized. Domestication Theory was used to assist in understanding how cloud technologies were appropriated in support of AGSD. The research strategy took the form of case study research. The findings suggest that some of the challenges in applying the agile principles in AGSD may be overcome by using CC.

Keywords: Cloud Computing, Agile Software Development, Agile Global Software Development, Agile Principles, Scrum.

Introduction

Globalization offers organizations the prospect of a larger customer base in addition to improvements in productivity and cost reduction. The geographically distributed nature of globalization and advancements in technology enable organizations to change their operating models, as well as how and where they source capital, customers, resources, and human capital (Esbensen, Jensen, & Matthiesen, 2014; Smirnova, 2013). These operational and sourcing changes also affected how organizations develop software, giving rise to the phenomenon of Global Software Development (GSD) (Richardson, Casey, Burton, & McCaffery, 2010). GSD refers to the practice of having software development teams (SDTs) distributed geographically with the intended purposes that include reducing labor costs, increasing software development capacity, and 24-hour productivity (Al-qadhi & Keung, 2014).

Another concept affecting software development has been the evolution and increased adoption of agile methodologies. The formulation of the Agile Mani-

Editor: Eli Cohen
Submitted: December 26, 2015; Revised: March 13, March 21, 2016; Accepted: March 28, 2016

festo (Agile Alliance, 2001) and related formulation of the Agile Principles, by software development thought leaders, was a seminal moment for what later became known as the "Agile Movement" (Linkevics, 2014). The period since has seen rapid acceptance of agile principles, with pervasive adoption by industry and extensive research by academia (Nilsson & Karlsson, 2014). The agile principles focus on delivering products that meet customer requirements, being responsive to change, while at the same time not compromising on quality (Highsmith & Cockburn, 2001). A core tenet in the agile manifesto stresses a greater importance of collaboration and interaction between individuals than that of tools and processes for project success (Cockburn, 2006).

Organizations, having appreciated the benefits of following agile principles, are increasingly integrating agile practices into their GSD; this concept is termed Agile Global Software Development (AGSD). The goal of AGSD is to capitalize on the benefits of globally distributed production while still being more responsive to change, maintaining software quality, and controlling costs (Kamaruddin, Arshad, & Mohamed, 2012). While having many potential benefits AGSD also presents with the same challenges of GSD over and above the social challenges of distributed agile teams (Nilsson & Karlsson, 2014).

Another development affecting the software development ecosystem, with increasing prominence, has been cloud computing (CC). The main characteristics of CC, which include reduced cost, scalability, performance, multi-tenancy support, and distributed availability, align with the needs of GSD (Cocco, Mannaro, & Concas, 2012). Academic research into the application of the benefits of the cloud to agile principles is low (Tuli, Hasteer, Sharma, & Bansal, 2014). However, research by Gill and Bunker (2013) found that CC could enable agile principles of communication through cloud-based social technologies. Examples of these social technologies include video conferencing, knowledge management, and web portals. Furthermore, given that CC significantly assists in the practice of GSD, it has the potential to aid in the following of agile principles, and potentially improve productivity by amalgamating CC with AGSD (Smirnova, 2013).

This research study investigated how CC has the potential to reduce the conflict between agile principles and GSD, to enable organizations to realize the benefits of AGSD. The 12 agile principles provide an overarching framework for this study as all agile methodologies align to those principles (D. Cohen, Lindvall, & Costa, 2003).

The primary research question guided this study:

- How may cloud computing be used as an enabler of the agile principles whilst performing AGSD?

This empirical research study adopted an interpretivist research stance, applying a qualitative research approach to collect and analyze information, with data sourced from two organizational case studies (Saunders, Lewis, & Thornhill, 2009). Two case studies were used in an attempt to understand the phenomena through the interpretation the participants had of their context (Runeson & Höst, 2009). The first case (C1) investigated a large multinational organization conducting AGSD from offices located in Australia (Melbourne), Brazil (São Paulo), Republic of Georgia (Tbilisi), Mexico (Mexico City), South Africa (Cape Town), and the United States of America (Charlotte). The second case (C2) investigated a small, distributed agile team developing while being located in various cities within South Africa (Cape Town, Durban, and Johannesburg). This aligned with the theory selection, as domestication theory-based research is commonly interpretivist in nature (Hynes & Richardson, 2009).

The remainder of this paper is organized as follows. A literature review is first presented discussing issues around agile software development, agile global software development, and cloud

computing. The methodology employed for the study is then described, followed by a detailed discussion of the findings. The paper is then concluded.

Literature Review

Agile Software Development

The following sub-sections describe the 12 agile principles and how Scrum, the most popular agile software development methodology, aligns with these principles. The core values of agile software development as listed in the Agile Manifesto relate to concepts that place an emphasis on people interacting and collaborating, and an acknowledgment that requirements will change, and that responding to such changes is important (Agile Alliance, 2001). Williams (2012) found that the agile principles remain as relevant to contemporary agile practitioners as at the signing of the agile manifesto. Table 1 lists the 12 principles that form the basis for agile software development. Each of the principles can be associated with a core value (Highsmith & Cockburn, 2001).

Table 1. The 12 Agile Principles (Fowler & Highsmith, 2001, p. 35)

1.	"Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software."
2.	"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
3.	"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
4.	"Business people and developers must work together daily throughout the project."
5.	"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
6.	"The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation."
7.	"Working software is the primary measure of progress."
8.	"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."
9.	"Continuous attention to technical excellence and good design enhances agility."
10.	"Simplicity — the art of maximizing the amount of work not done — is essential."
11.	"The best architectures, requirements, and designs emerge from self-organizing teams."
12.	"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

The Scrum methodology

Agile methodologies are approaches that provide techniques which prescribe processes to assist SDTs in adhering to agile principles (Cockburn, 2006). There are numerous agile software methodologies each underpinned by the agile principles (D. Cohen et al., 2003). However, Scrum is the most widely adopted agile methodology (Azizyan, Magarian, & Kajko-Mattson, 2011; Kim, 2013). The study will thus focus on Scrum as a form of agile software development methodology. Scrum is an agile process framework developed by Ken Schwaber and Jeff Sutherland. Schwaber and Sutherland (2013, p. 3) define Scrum as “A framework within which people can address

complex adaptive problems while productively and creatively delivering products of the highest possible value". Scrum aligns with the agile principles in the follows ways:

- **Principle #1: Deliver Value to the Customer**

Scrum aligns with Principle #1, as sprints (iterations) occur with regular frequency, are shorter than a month, and the intended goal of each sprint is a usable and potentially shippable product (Schwaber & Sutherland, 2013).

- **Principle #2: Welcome Changing Requirements**

Scrum supports Principle #2 as changes to requirements and priorities are encouraged at various points of the Scrum cycle. Scrum also includes specific events designed to develop and communicate changes. However, Scrum does not permit changes to the sprint goal once a sprint has commenced (Schwaber & Sutherland, 2013).

- **Principle #3: Deliver Frequently**

Scrum sprints are time-boxed for less than four weeks, closely aligning with Principle #3.

- **Principle #4: Business & Developer Collaboration**

The business and development collaboration of Principle #4 of Scrum occurs both informally during a Sprint and at specific meetings (Schwaber & Sutherland, 2013). The role of the product owner creates open communication on requirements as they interface directly with the customer (Kim, 2013; Takkunen, 2014).

- **Principle #5: Motivated and Supported Team Members**

Organizations that adopt Scrum also align with Principle #5, as the cross-functional nature of the team requires support from the business to equip the teams appropriately with individuals with the required competencies. An outcome of the function of the scrum master to shelter the team from external influences creates the right environment for developers to focus solely on the tasks of the Scrum team (Darwish, 2014).

- **Principle #6: Face-to-Face Communication**

Scrum facilitates the face-to-face communication described in principle #6, through the sprint planning, daily scrum, sprint review, and sprint retrospective meetings (Schwaber & Sutherland, 2013).

- **Principle #7: Working Software Indicates Progress**

Principle #7 aligns with Scrum practices through the sprint review event, where the team is able to demonstrate useable software completed during the sprint, to assess the state of the product backlog (Darwish, 2014).

- **Principle #8: Sustainable Software Development**

Principle #8 states that the rate of progress should be sustainable and consistent (Fowler & Highsmith, 2001). Significant variance between a sprint velocity and team velocity identifies changes in the rate of progress, prompts the team to investigate and address the cause. Thus, team velocity allows Scrum teams to monitor the consistency of the rate of their progress (Pomar, Calvo-Manzano, Caballero, & Arcilla-Cobián, 2014).

- **Principle #9: Technical Excellence**

Scrum does not prescribe how software is developed therefore does not mention technical or architectural aspects of Principle #9 (Cockburn, 2006).

- **Principle #10: Simplicity**

The design of the Scrum framework aligns with Principle #10. While Scrum is difficult to master it is simple to understand (Schwaber & Sutherland, 2013).

- **Principle #11: Self-organizing Teams**

Scrum teams mirror Principle #11, as they are self-organized and the team is autonomous but accountable for decisions made. Scrum does not prescribe how the team should develop the software but emphasizes team and individual accountability. This is reflected in the self-organizing nature of Scrum teams who collectively decide how best to complete project tasks (Cockburn, 2006).

- **Principle #12: Continuously Adapt and Improve**

The sprint review of Scrum fully aligns to Principle #12. Scrum also instills a culture of improvement, evidenced by every Scrum event being an opportunity to optimize and improve (Schwaber & Sutherland, 2013). The coaching function of the scrum master also aligns with the goal of individual and team effectiveness.

Agile Global Software Development

Organizations are increasingly coupling agile practices with GSD with the intended purpose of realizing the competitive advantage of developing software globally. However, AGSD poses both technical and social challenges for organizations due to the distributed nature of the software teams collaborating on projects (Nilsson & Karlsson, 2014) as well as to the application of the agile principles (Kamaruddin et al., 2012). Table 2 provides a mapping of the AGSD challenges affecting the application of the agile principles. As demonstrated in Table 2, three agile principles most affected in the AGSD context are Principle #6, Principle #5, and Principle #4. It is plausible that negatively affecting communication, collaboration, and SDT motivation could also negatively influence the value generated from developed software.

Table 2. AGSD Challenges Impact on Agile Principles (Beck et al., 2001; Kamaruddin et al., 2012)

AGSD Challenge	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
Lack of customer involvement	●			●	●	●						
Bandwidth limitations					●				●			
Cultural differences					●	●						
Different project background		●				●						●
Different working hours				●	●							
High communication costs					●				●			
Inadequate tool support				●					●			
Lack of commitment	●					●						
Lack of frequent face-to-face contact				●		●						
Lack of trust					●	●					●	
Language differences		●		●		●						
Miscommunication of requirements				●		●						
Poor Communication Infrastructure					●				●			

Cloud Computing

The National Institute of Standards and Technology (NIST) defines CC as "… a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell & Grance, 2011, p. 2). The model comprises Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (Mell & Grance, 2011). Despite some challenges, the use of cloud-integrated tools and platforms have been proven to increase cost effectiveness and aid enterprises throughout the Software Development Life Cycle (SDLC) by allowing developers to focus more on building software with minimal impedance (Al-qadhi & Keung, 2014). Cloud services specifically IaaS, PaaS, and SaaS support AGSD.

IaaS consumers have access to virtualized servers and network accessible infrastructure hosted by the Cloud Service Provider (CSP) (Liu et al., 2012). IaaS supports software development through virtualization, allowing the provisioning of servers based on the demand of the project. This enables project teams to work in parallel, minimizing parallel work streams conflicting for resources (Mwansa & Mnkandla, 2014).

PaaS offers cloud consumers the ability to use frameworks and other software on preconfigured instances, provisioned and hosted by CSP (Liu et al., 2012). Standardized instances of virtual services with preinstalled and configured software assist in the SDLC process of custom-built software, through the rapid availability of new clean instances (Schneider & Sunyaev, 2014). Examples of PaaS include Cloud Foundry, Google App Engine, Heroku, and Microsoft Windows Azure (B. Cohen, 2013).

SaaS users include system administrators and end users within organizations, as well as individual users who consume the software directly from a CSP (Kavis, 2014). Users access SaaS services via a thin client (Jula, Sundararajan, & Othman, 2014). SaaS examples include Google Apps, Microsoft 365, and Salesforce (Al-qadhi & Keung, 2014). SaaS supports development in several ways (Al-qadhi & Keung, 2014; Schneider & Sunyaev, 2014). Firstly, SaaS provides a delivery platform for software development with global access to users that have access to the Internet. Secondly, updating software happens at the server, limiting disruption to clients and can eliminate issues related to legacy versions of the software. Thirdly, software developers can use APIs delivered through SaaS, to integrate as a component of their software, promoting code reuse and standardization, in addition to being a potential source of revenue for the API provider. Fourthly, SaaS protects the income stream for software development producers by lowering the risk of software piracy, as SaaS supports the ability to enforce a pay per use fee model.

Cloud as an Enabler of AGSD

The IaaS and PaaS layers of CC assist agile development teams in delivering value two ways (Dumbre, Senthil, & Ghag, 2011). Firstly, the rapid provisioning of environments through virtualization saves the Scrum team both time and effort in setting up the development environment. Secondly, cloud interfaces facilitate the deployment of software and environments, directly with the developer's Integrated Development Environment (IDE), reducing the need for assistance from outside the Scrum team.

SaaS provides a delivery platform for cloud-based agile management software tools that provide similar functionality as physical Scrum boards; examples include JIRA, Mingle, Rally, ScrumWorks, Trac, VersionOne, and XPlanner (Azizyan et al., 2011). These tools provide benefits to AGSD teams in four ways (Tuli et al., 2014). Firstly, being cloud based, the tools are globally accessible to the whole AGSD team, providing a single source of information across the whole team. Secondly, the tools are readily available. Thirdly, they do not require internal development

and deployment. Fourthly, the tools have the ability to scale, as adding new users does not degrade performance.

Reduced feedback latency

Agile software development relies on frequent feedback between developers and business users (Wang, 2011). CC reduces the time and effort to test and deploy software, thereby reducing the latency between completing development and receiving feedback on software errors, improving productivity by product owners and users (Guha & Al-Dabass, 2010). The deployment mechanisms supported by CC enable continuous integration. Continuous and frequent code deployment within a sprint enables product owners and stakeholders to review work done during a sprint and take corrective action mid-sprint. This allows developers to confirm and realign their understanding of the requirements as early as possible, eliminating wasted effort (Wang, 2011).

The Sprint review is an important agile practice where software developers provide feedback to the product owners and other stakeholders, through demonstration of functionality developed during the sprint, in line with Principle #7 (Schwaber & Sutherland, 2013). Deploying software that is globally accessible allows stakeholders of the project to review the code, regardless of their location (Dumbre et al., 2011). In instances of large temporal distances and teams are unable to meet simultaneously, the environment used for the review and code can remain active until all relevant team members have reviewed and provided feedback. This is possible as cloud environments are not limited in the same ways as physical servers. Once testing has concluded the environment can be decommissioned (Hossain, Bannerman, & Jeffery, 2011). While the potential for software bugs remain, there is a reduced possibility of failure due to the software running on a CC platform (Wang, 2011). This cultivates trust between the business stakeholders and developers as environment instability has the potential to create incorrect perceptions of poor software development practices (Cockburn, 2006; Schwaber & Sutherland, 2013).

Collaboration

The agile principles stress the importance of frequent collaboration and communication (Schwaber & Sutherland, 2013). Esbensen et al. (2014) found that frequent informal communication between AGSD team members across distributed locations creates a sense of team unity and reduced the time to resolve issues. The most commonly used collaboration technologies were calendars, email, instant messaging, screen sharing, shared document spaces, social media, source code environments, telephones, and video conferencing. These collaboration technologies are supported by CC or have cloud-based equivalents; however, CC has the benefits of reduced cost and pervasive availability (Esbensen et al., 2014; Gill & Bunker, 2013).

CC nurtures accountability, transparency, simplicity and trust emphasized in Scrum (Jula et al., 2014; Schwaber & Sutherland, 2013). By capturing information electronically using cloud-based tools, the entire team has access to the same information, eliminating delays or stale information. CC supports high levels of automation, reducing the need to capture information manually as in the case of code commits to a source repository, automated builds, and the results of automated tests (Schwaber & Sutherland, 2013).

Source code management

Source code repository software facilitates source code management, with the primary functions being to store, version, and prevent loss of source code and configuration files; they are essential tools for developer collaboration (Amin, Hasab, & Faraahi, 2014). Cloud-based software code repositories enable AGSD teams to distribute and integrate project files globally and help identify code changes with each check-in submission. SaaS code management products include Code Spaces, GitHub, GoogleCode, SourceForge, and Unfuddle (Fuggetta, Nitto, & Milano, 2014;

Pulkkinen, 2013). Examples of PaaS products with integrated code management include Heroku, Engine Yard, Windows Azure, Google App Engine, and Force.com (B. Cohen, 2013). Cloud-based code management systems promote transparency as all team members can inspect the history of each file, identifying which sections have been changed, when, why, and by whom (Pulkkinen, 2013). This transparency allows organizations performing AGSD to enforce uniform coding standards across an organizational code base, and assist in problem-solving and knowledge transfer (Fuggetta et al., 2014; Pulkkinen, 2013; Wang, 2011).

Automation

Automation of continuous integration is one way that CC can reduce the feedback cycle. Continuous integration is a practice endorsed by agile software development (Cockburn, 2006). The process requires source code management where every team member commits their changes to a centralized repository every time a new change or a task is completed (Pulkkinen, 2013). Continuous integration is a complex, time-consuming, and resource-intensive process. The process includes the committing of source code to a central repository, building the software from the integrated source code, running automated tests on the software (including performance testing), and deploying the software. PaaS can provide this requisite scalable resourcing by its ability to provide computing resources as required (Pulkkinen, 2013)

Uniform development environments

AGSD projects that do not make the whole environment available to the entire SDT run the risk of significant integration problems once insourced or outsourced software is integrated or deployed with onsite systems. To minimize this risk, internal software teams that are outsourcing software development work must first assess their software environments for portability, identifying all the related dependencies and configuration. Portability issues may require tactical projects to change the code base prior to outsourcing the software development (Stankov & Datsenka, 2010). PaaS can support uniformity between teams in two ways. Firstly, PaaS environments are accessible through the Internet, resulting in all team members having access to the same or identical instances of the development environments (Stankov & Datsenka, 2010). Secondly, certain PaaS platforms are able to deliver development tools such as IDEs as a service, thereby eliminating the need to install and configure an IDE (Ghohandizi, 2014).

Theoretical Framework

Domestication theory is an approach concerned with the understanding of the adoption and use of technology within households and institutions. The theory is concerned with practical, temporal, and the socio-cultural aspects associated with a technology and covers the pre and post-adoption stages of a particular technology (Haddon, 2006). The theory was initially proposed for understanding the adoption and use of media technology within the context of a household. However, it has also been used to understand the appropriation of technology by other entities such as educational institutions, businesses, and other groups (Chigona, Chigona, Kayongo, & Kausa, 2010; Harwood, 2011; Hynes & Richardson, 2009; Sandtrø, 2012). Prior domestication studies have mostly been related to technology associated with a physical device such as motor vehicles, televisions, laptops, and mobile devices (Brussel, 2013; Haddon, 2006; Harwood, 2011; Hynes & Richardson, 2009). However, there are examples of the use of domestication theory for non-physical technology such as Virtual Learning Environments (VLE) (Sandtrø, 2012). Consequently, it is appropriate to assume that domestication theory would be relevant to the understanding of how CC is used in AGSD.

The domestication process covers four non-discrete phases or dimensions: commodification, objectification, incorporation, and conversion (Chigona et al., 2010; Haddon, 2007). Domestication

theory was used because the framework describes and analyzes the processes of user acceptance, rejection, use, and integration of technology into the routine activities of individuals or organizations (Lee, Smith-Jackson, & Kwon, 2009). The use of the four phases or dimensions of domestication theory as an explanatory framework promoted an understanding of the process of cloud technology adoption and use within an organization. This assisted in understanding how CC enables the application of the agile principles within an AGSD context. The domestication phases provided a structure upon which to base the interview questions, in addition, the thematic analysis of the data. The subsections that follow discuss the four phases of domestication theory.

Commodification

The commodification phase focuses on the process undertaken for the consumer to possess a particular technology and is also referred to as "appropriation" by some researchers (Bakardijeva et al., 2006). Both potential and actual consumers begin to develop mental images of the usability and functionality of the technology as they evaluate the technology based on their needs (Hynes & Richardson, 2009). This phase pertains to the route followed for a specific technology, from the point of marketing of a product to the user, as well as to the user's motives for approaching the technology (Lee et al., 2009).

Objectification

The objectification phase covers the point at which a particular technology has been acquired. In this phase the consumers begin to decide on the meaning the technology has and in what aspect of their lives it inhabits (Haddon, 2006). However, possession of the technology does not imply its acceptance by the consumer (Hynes & Richardson, 2009).

Incorporation

Incorporation is the phase focusing on the point where the use of the technology becomes routine and forms part of the consumer's regular activities, both informally - by way of routines - and formally as part of a defined procedure (Chigona et al., 2010). Consumers acquire technology with specific functionality and applications in mind. The incorporation phase also covers the usability aspects of the technology for the consumer, as some technologies may not align with required functionality or routines of particular consumers (Lee et al., 2009).

Conversion

In the conversion phase, consumers show the adoption of the technology by sharing their experience of the technology with others (Chigona et al., 2010). While still using and being reliant on the technology, the consumer no longer consciously thinks about the technology. Having mastered the technology, the consumer may also begin adapting the use of the technology in ways different to the original intentions of the designers and marketers (Haddon, 2006; Lee et al., 2009).

Methodology

The research design for this case study aligns with the steps described by Voss, Tsikriktsis, and Frohlich (2002), namely, (1) Development of the research framework, constructs, and questions, (2) Case selection, (3) Research instruments and protocols selection, (4) Ensuring Reliability and Validity and (5) Data Analysis. The research followed a deductive approach, as it used domestication theory as a theoretical lens. The research adopted an interpretivist research stance to the phenomena, aligning with the qualitative research method (Runeson & Höst, 2009). This aligns with the theory selection, as domestication theory-based research is commonly interpretivist in

nature (Hynes & Richardson, 2009). The research study was empirical, applying a qualitative research approach to collect and analyze information, which comprised two case studies (C1 and C2) (Saunders et al., 2009).

The first case study (C1) focused on a software project completed by a multi-national organization with 17 globally distributed offices. The organization provides global trade management solution to manage the export and import of goods. The project entailed the development and delivery of a bespoke trade integration system that complied with the Brazilian Customs authority. Table 3 lists the participants that were interviewed for C1. Software development to deliver these requirements occurred at four office locations: USA, Australia, Brazil, Mexico, and South Africa. Business analysts and client service managers are also globally distributed. The SDTs follow the Scrum development methodology.

Table 3: Case 1 Interview Participants

Respondent Pseudonym	Job Title
C1.C	Technical Director/Enterprise Architect
C1.M	Project Manager
C1.R	Product Owner/Business Analyst

The second case study (C2) was conducted in a small South African, software development company with developers operating from Cape Town, Durban, and Johannesburg. C2 is a Microsoft Solution Provider, with Gold Partner certifications in application development and application integration. C2 provides software development resourcing, consulting services, and bespoke software solutions on the web and mobile platforms. The software development methodology used is Kanban. The organization does not have formal offices, and developers work from home or onsite at client offices. The selection of C2 was because of how different it is when compared with C1. C2 did not have to contend with different time zones, language differences within the SDT, nor international operations. The small size relative to C1 allows the organization to be more responsive to change than C1. The embedded case takes the form of a single software project for a time tracking and resource management solution for internal consumption and as a SaaS product for customers. Table 4 lists the participants that were interviewed for C2 using respondent pseudonyms to protect their identities at the time of the interview.

Table 4: Case 2 Interview Participants

Respondent Pseudonym	Job Title
C2.W	Director/Technology Specialist
C2.P	Software Developer

Data collection

Case research typically employs multiple data collection methods, that include interviews, observations, archival data, documents, and physical artifacts (Benbasat, Goldstein, & Mead, 1987). The primary data sources were semi-interviews and direct observation. The interview questions comprised four sections, structured as described in Table 5.

Table 5: Interview Questions Sections

Section	Description
One	Global software development processes performed within the organization.
Two	Alignment of software development practices to the agile principles.
Three	The pre and post adoption of CC for use within the software development practices of the organization.
Four	Open-ended: where the respondent was asked to suggest other ways that CC can enable the agile principles.

Direct observation takes the form of documenting actions of participants in the context of the environment under investigation (Yin, 2009). For this research study, direct observation included the demonstration of the tools used in relation to the study. Secondary sources related to the case included documentation related to meeting minutes, emails, instant messaging (IM) conversations, project artifacts, context diagrams of solutions, and configuration files (Benbasat et al., 1987; Yin, 2009). The primary secondary sources were the websites of the case organizations, as they created background and context to case organizations.

Data analysis

NVivo is software designed to support qualitative and mixed methods research, designed to handle non-numeric data such as interviews, open-ended survey responses, literature reviews, and web content (Runeson & Höst, 2009; Yin, 2009). The researcher used NVivo software to aid in the thematic analysis of the data. The analysis followed similar steps to those described by Yin (2009). The process was highly iterative; the actual steps performed were as follows:

1. Each interview was transcribed using NVivo.
2. Additional sources such as marketing material and web pages relating to the cases were also ingested into NVivo.
3. Initial set themes corresponding to domestication theory, agile principles, and AGSD related concepts were created.
4. Each theme was created in NVivo as a node.
5. Sub-nodes were also created to refine each theme.
6. Interview responses were then mapped to the corresponding themes.
7. Word frequency and text queries were also used to identify new themes.
8. The initial themes were then grouped and refined through several iterations, using Microsoft Excel.
9. The themes were then mapped to research questions.
10. The relationships were also created between themes that allowed for the formulation of conclusions.

Findings

This section discusses the findings derived from the deductive thematic analysis process. To reduce duplication and structure the findings the 12 agile principles have been grouped into three categories of communication and collaboration, technical excellence, and frequent delivery.

Communication and Collaboration

This section provides an overview of findings of how CC enabled AGSD teams to adhere to the agile principles that relate to collaboration, communication, support of the team, and process improvement.

Business & SDT collaboration

Business users and customers are part of the project team and CC enables the collaboration of business and developers. For example, using agile management software exposed through SaaS eliminates the need for specialized tools, and access can be granted to project stakeholders, who would have been unable to access the information due to firewall restrictions. This potentially increases the level of transparency − and possibly trust − as business users can see the status and track the progress of activities through the project. Project administration is further reduced if other software used on the project, such as IDEs, is also integrated with the agile software. This is supported by C1, as covered in the following extract of a conversation between C1.C on the need for a single tool to manage an agile project:

> *When you want to work with your customers, you want them to use the same tool, they might not see the same content as you. Most of these tools allow you to link issues together. I can have certain product codes under which the customer can log issues. Then I can link those issues to internal trackable issues, which they can't see, but you want that traceability at the end of the day and you can't do that when you've got two tools because then a human being must do the traceability link. Issue 500 over here and the customer system is now issue 200 here and everything that happens here must be filtered back here again (C1.C).*

Face-to-face communication

Whilst principle six stresses that face-to-face, communication is the preferred way for the SDT to communicate, it is not always possible in an AGSD context. However, frequent communication between team members is essential and cloud-based software such as email, IM, screen sharing, shared document spaces, and video conferencing aids communication. Each of these tools has a proper use and teams may need to use more than one communication channel to be correctly understood. The frequent communication also allows individuals to feel they are part of a team, which in turn supports agile principle five. This is supported by the following extracts:

> *There was a Skype group that has been setup since the day I joined and it is still going strong.… Slack came around the thing that I liked about Slack was the fact that you could paste different type of content in it. It would actually render images and links and is searchable. Whereas Skype's chat history is not that usable and easily accessible (C2.P).*

> *Communication is on the phone, IM and email. That's pretty much it. Also often do a secure meeting with them, it is like WebX you basically setup a secure meeting, view their desktop and they will do a demonstration of what they are busy working on this is what they have a question on. And then obviously I can take control of the desktop and I just click around and show them that is how what I mean. So I think that is something, what is nice about talking to guys here is you can get up and draw on the whiteboard and say this is what I mean. Where is it a bit more difficult to convey that in a medium where you don't have complete artistic freedom and go I want this change moved over there and all that sort of stuff. You kind of got to highlight it and say "I want this changed to here and more of a, you have to really outline something quite well." (C1.R).*

Motivated team members using the right environment

The three service models of CC support principle five in an AGSD context as they provide the SDT with the right computing resources upon which to develop and reduce the latency between assigning an AGSD team to a project and the SDT performing tasks that are directly related to producing value from the first iteration. In C1, the private cloud provides uniform software environments required for development and testing. Creating virtual servers or a pristine virtual machine (VM) for a developer is a routine activity, as shown in the following extract:

> *We send an email out to IT I think and they have templates within the virtualization environment where they select a template they want and it spins up in about five minutes. They will assign us a Partner ID, which will be for development or like a new project. Like our one's 2042 and 2040 which is for the development team (C1.M).*

Security restrictions are in place to prevent developers from copying any text or file data off their development image. C1.R expressed a level of frustration with the security restrictions. C1.R also voiced a consensus view that the primary motivation for the way the PaaS environment was implemented was to enforce security controls and that the ease of maintenance and rapidly available clean development machine instances were unintended benefits.

> *The whole VM Ware thing is nice from … I think where they can apply patches and changes as needed to, but I don't think that is their intent. I think their intent was source code safety like C1.C said, even if you steal the whole source code base you're not going to get to roll it out … So what is the real point? The VMs are locked down so you can't copy anything off of the VM unless you do a request to copy a file and then the request goes via the development manager. So the Development Manager will say, "You are trying to copy source code off of your VM. You are not allowed to that, but you can copy a pre-built file." For instance he needs deploy or something, they are very careful (C1.R).*

Self-organizing teams

Agile principle eleven stresses team accountability for the outcome of a project and trust in the team to make the best decisions possible (Kim, 2013; Schwaber & Sutherland, 2013; Takkunen, 2014). CC can potentially promote this by allowing the SDT to collaborate and express themselves using various tools. Trust in an AGSD context can be built through the SDT delivering consistently. Cloud-based agile management software can be used to track the current progress of an SDT as well as their progress over time. Transparency is achieved through the tracking of each team member's contribution. In C1, extensive use was made of Team Foundation Server (TFS) for source code management, agile project management, and real-time tracking of progress made by individuals working on each requirement, as evidenced by the following extract:

> *I can also then see [what has been done] because I have added the requirement to TFS and then I can see when they are actually checking-in against that requirement. So, I can see if they are making progress or not (C1.R).*

To become truly efficient, AGSD teams must overcome cultural issues that lead to fear and build trust between all team members. Cloud communication tools could be used to build relationships between team members and make all team members feel safe within the team. By example in C1, the product owner described the fear-based reluctance of members of his team to seek further clarity on particular requirements they did not understand. C1.M added that they intended using video conferencing to improve relationships.

> *[W]e discussed just last week, Monday I can't remember, with the development manager in the States about buying the Georgian team a camera and a couple of screens to talk. If somebody new to the company is going to a meeting with more than two or three people,*

it is very difficult to map a name, a title to voice, it is really difficult. When you can see them I hope it will be a little easier (C1.M).

Technical Excellence

This section provides an overview of findings where CC enabled AGSD teams to adhere to the agile principles that relate to technical excellence.

Simplicity

CC supports the tenth agile principle of simplicity, by abstracting the complexity of maintenance and support of the server environments, releasing the AGSD team to focus on the activities that deliver value. In C2, PaaS reduced the complexity, effort, and lead-time to provision and configure used in the SDLC as described in the following extract:

You can pre-configure them. You can get a blank one and do whatever, but you can actually say for any of the servers, "I want SQL this version, SharePoint this version, whatever, whatever", and it will spawn up a server for you (C2.W).

Technical excellence and good design

The ninth agile principle states that a constant focus on technical excellence and good design enhances other agile practices that are being followed (Jasemian, Mortensen, & Boje-Nielsen, 2007). Using CC for hosting of source code and following continuous integration practices exposes the code base and design to the entire AGSD team. This allows the SDT to conduct code reviews and ensure correct practices are followed consistently. In C1, the organization was able to enforce consistency through the use of a single internal framework executed on a uniform.

We have a framework that has been written by the Georgian office and that being reused again and again. So, the coding that is done here [Cape Town], needs to align with the way his framework is set out. So if he needs to add action buttons or we to need to add a new report that needs to be generated or something like that. There's a section pretty much just needs to be copied and pasted, we've already done all the work in terms of what or how you generate the report, it is just put it in the code and then let it run (C1.R).

Welcome changing requirements

The agile principle two of changing requirements requires consistent adherence to the other eleven agile principles. However, once requirements have changed the entire AGSD team must be notified and have updated requirements documentation. CC can support this by providing a secure, globally accessible location to store documents, and the team and the means to notify the SDT using appropriate communication mediums. The following extract from C1 illustrates how an architect notified the AGSD team of changing requirements:

They [SDT] have access to the UML model there they can make their changes and what we do is generate a document out of the tool which is then a static document that we can give over to the developers. Then we also make use of DropBox to share the specification so the static specifications which customs [government agency] gives us. We hand that over. So we are all working with the latest version, so if there is an update I will warn them and say [using Skype], "Look there is an updated spec we no longer using version X, we're using version X+1" (C1.R).

Frequent Delivery

This section provides an overview of findings where CC enabled AGSD teams to adhere to the agile principles that speak to frequent delivery.

Deliver frequently

CC enables the third agile principle of frequent delivery in an AGSD context is through continuous integration. Continuous and frequent code deployment within an iteration enables the product owners and stakeholders to review work done during a sprint and take corrective action before each iteration is complete. The process of continuous integration can be automated using centralized source code management software and a build server. In the following extract, the C1.M describes a typical sequence of events followed to deliver software that is ready for testing:

> *So the developer will develop on their VM, which is..., they can test their stuff, well those screens on their Dev VM. When they are ready, they commit it to TFS. The builds will happen in TFS, there is a Continuous Integration build setup. They would then pick the components of the solution they changed and which they need as part of their project and push it or copy it to the integration server. Which the analysts and testers would then they say, "Well this is now our next version of [project name redacted] development." (C1.M).*

CC enables frequent delivery by allowing AGSD teams to create multiple parallel environments for demonstration and indefinite testing (Ghohandizi, 2014). In C1, as described in the following extract, business users have a dedicated environment for them to test:

> *Sometimes we do the demo on a product owner's server instance. We do the demo, show them what it can do, and they go play on their server on their own (C1.M).*

Sustainable development

As previously discussed, frequent delivery decreases the amount of long-term planning required by the AGSD team, by having smaller time intervals which enable the SDT to more accurately estimate and manage their time. In C1, the ability to perform multiple progress demonstrations encourages the AGSD team to work consistently.

> *What is nice about these demos is that it focuses the team on getting goals ticked off, because if you just leave it, if they can work for three months it will, they will take it easy for those two months and then start working overtime the last month. You need to constantly say "Ok, guys I want to see that we are on the right page" (C1.C).*

Deliver value to the customer

Product owners or customers determine the value and priority of features delivered by an AGSD project (Sverrisdottir, Ingason, & Jonasson, 2014). CC can support this principle using SaaS agile management software as it enables the "customer" to set the priority, giving the AGSD team a clear view of what must be performed within a sprint. In C2, the product owner sets the priority of issues performed within the sprint, using agile management software as described in the following extract of C2.P who discusses how they used TFS to prioritize sprint deliverables:

> *Each task is drag and drop. The product manager would want to see everything so he can prioritize what is put into a Sprint (C2.P).*

Cloud Support for Agile Requirements

Lightweight agile methodologies such as Scrum and Extreme Programing (XP) recommend an evolutionary approach to system design, as requirements and designs are refined and improved

through iterative cycles (Jasemian et al., 2007). CC can potentially support this requirement maturation process via different tools for the different phases.

> *...[D]epending on the life cycle of the document. We put it, we normally start with GoogleDocs, then DropBox, and then we probably go to TFS (C1.M).*

Waiting for a document to reach a fully matured state may delay opportunities for collaboration and use − well past the point where the document would begin adding value (Ambler, 2002). C1 used GoogleDocs for the initial phases of requirements documentation maturity cycle. Once requirements documents were created, they were also accessible to the entire team. GoogleDocs also supports concurrent editing of documents by multiple authors with real-time updates. The reason for this approach is to support team collaboration and to make sure the team works off the latest requirements as described by C1.C in the following extract:

> *Once this team is busy writing documents, I want to know what [person name] has been working on in the last 30 minutes and I can open the document and instantaneously I can see the contribution of each team member on the document. I don't want them to check it in first because people are hesitant to check things in, it is a mental thing. There is a hesitation to check-in or save your document or whatever it is. The team is updated every six hours or every five days of this person's work. Where you work on something like GoogleDocs it is immediately shared there is no save button. When I change a letter now everybody sees it and with documents I find it handier to use GoogleDocs for those initial phases. Write it, draft it and by all means, get to a point where the company now needs to know of it take it, "Save As" word document, upload it onto TFS (C1.C).*

Artefacts management

CC offers a variety of tools with which to store and develop artifacts such as requirements documents, diagrams, and other project related artifacts. However, using different tools and storage mechanisms means that relevant documentation can be in multiple locations. AGSD teams need to have clearly defined processes and mechanisms to support the storage and retrieval of documents. In C2, all project artifacts not stored in the source code repository are referenced by a single document maintained, by the SDT to track metadata such as the version and location of project artifacts, as described in the following extract:

> *... [T]his is an example of a GoogleDoc which is document map. Any document people need to share or reference. This is a landing page [Shows a spreadsheet of documents and metadata related to the documents]. So people don't have to look in their emails or whatever, they go in here look, do a search or whatever and there is a purpose description, who the owner is, whether it is released or not, and at the end here [points to the last column] where is this thing. Is it in DropBox, is in the GoogleDocs, and soon where will it be in TFS (C2.M).*

Table 6 is presented in the Appendix to summarize how CC was found to enable SDTs in adhering to the agile principles while performing AGSD comparing the findings with literature.

Discussion

Analysis of the research findings suggests that CC assists in reducing feedback latency between the stakeholders of a project, through the support of both a value based approach to project artifacts and continuous integration. The study found frequent delivery could be achieved by adopting continuous integration practices. CC provided a platform for centrally accessible source code management as well as build servers that generated builds once developers checked-in source code. Hosting the source code management software and build server using cloud resources

meant that SDT productivity was not affected by frequent integration builds. CC also enabled parallel server environments to be available for testing; this allowed development to continue by using other environments while business users were testing. These findings are in line with research conducted by Fuggetta et al. (2014) and Lin (2014).

The tools used by AGSD projects may vary based on the needs of a project as it was found that when standard tools inhibited a project the SDT began using other tools that provided the required functionality. While the software must adequately support each AGSD project (and may vary as a result), using multiple software tools for a similar purpose may also present a project with different challenges. In the case of this study, users were reluctant or unwilling to use new collaboration tools due to company policy or firewall restrictions, in addition to not wanting to update yet another tool.

The study found that through the use of a range of cloud-based communication tools such as video conferencing, IM, voice chat, screen sharing has the potential to reduce the impact of infrequent face-to-face communication within an AGSD context. IM was found to support AGSD teams in communicating synchronously when the temporal distance was small and asynchronously where office hours did not overlap. IM chat information persistence allows SDTs to use the IM content as a historical record of events and decisions made over an extended period. IM also creates an awareness of availability for communication and the status of team members. IM can be tactically within an iteration to allow distributed team members to collaboratively address technical issues. IM could be used to notify the SDT of new or changed requirements documentation being available. Synchronous communication was found to be the preferred means of communicating shown by teams shifting meetings − such as stand-ups − traditionally held at the start of the day to later in the day to include as many team members as possible. Scheduled meetings used a standard set of communication tools. However, informal and ad hoc with preferred communication method was negotiated between participants. The study also found instances where more than one tool such as email, IM, and screen sharing was required to make sure all parties understood each other. The use of multiple CC tools resulted in an additional administrative overhead of finding project artifacts. To mitigate this, a centrally accessible document was made available to the team that contained metadata such as the filename, purpose, version, and importantly the location of the document. However, this required diligent maintenance that could have been mitigated had fewer tools and storage locations been used.

The study found that CC was used to enforce company policy of code security and data loss prevention. The use of a PaaS environment was found to promote standardization of environments assisting in IT maintenance and rapid provisioning of software environments, as well as resilience to the loss or failure of hardware.

It was also found that migrating the server environment to a CC environment improved stability, created software development capacity allowing developers to focus on tasks directly related to delivery. This increased development capacity came as a result of a reduction in the time and effort required to setup and configure servers, improved stability, and availability of the server platform, thus reducing the resources allocated to fixing server infrastructure issues. Integrating CC tools with the IDEs used by the SDT reduced the administration overhead of updating the status of project tasks. This allowed teams to focus communication more on why tasks were at a current status and collaborating on how to resolve issues.

Research Limitations

As a multiple case study, the generalizability of the results is limited. Participants in the interviews represented most of the roles that form part of an AGSD team. However, the full cross-section was only achieved across cases and not within each case. While participant responses

from both cases were congruent, including more participants could potentially further strengthen the triangulation of the findings, as well as expose additional insights and themes related to the study. The primary limitation to low participant numbers was work commitments, which affected the availability of the interview participants. The hope is that future case research will be conducted with a larger number of participants that are globally distributed.

Conclusion

Communication challenges are a recurring theme throughout the GSD and AGSD literature. Agile software development requires coordination, feedback, and trust - all of which are challenges in a distributed context. Temporal issues may require modifying agile methodologies, for example, when an AGSD team is using Scrum and where offices hours between locations do not overlap, they may need to adapt a sprint review meeting to be an asynchronous event.

For this research, domestication theory was used as a theoretical lens. The four dimensions of domestication theory assisted the researcher in developing interview questions and developing themes to gain insight into appropriation of CC within each case.

Following from the empirical findings and an analysis process, five key findings have been identified, namely:

1. Using CC reduces feedback latency.
2. CC provides a range of communication tools designed for different communication requirements.
3. The use of CC in conjunction with AGSD requires more discipline.
4. The software tools used on an AGSD project should support the project context.
5. CC enables an SDT to focus more on project delivery than ancillary project activities.

A research contribution of this study is that it was formulated − in part − as a response to a call for further research by Amin et al. (2014) for empirical case studies of how CC could be used to address challenges of AGSD. The contribution towards practice includes practical examples of the use of CC in both an AGSD and collocated context.

In conclusion, the paper successfully provides an understanding of how, through the use of CC, some of the challenges in applying the agile principles in an AGSD context may be overcome.

References

Agile Alliance. (2001). *Manifesto for agile software development*. Retrieved April 5, 2015, from http://agilemanifesto.org/

Al-qadhi, M., & Keung, J. (2014). Cloud-based support for global software engineering: Potentials, risks, and gaps. In *Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices* (pp. 57–64). Hong Kong: ACM.

Ambler, S. W. (2002). In T. Hudson (Eds.), *Agile modeling* (1st ed.). New York, USA: John Wiley & Sons, Inc. doi:10.1017/CBO9780511817533.018

Amin, S., Hasab, M., & Faraahi, A. (2014). An overview of applying cloud computing in addressing agile global software development challenges. *Reef Resources Assessment and Management Technical Paper*, *40*(5), 184–191.

Azizyan, G., Magarian, M. K., & Kajko-Mattson, M. (2011). Survey of agile tool usage and needs. In *Proceedings of the 2011 Agile Conference* (pp. 29–38). Salt Lake City, USA: IEEE.

Bakardijeva, M., Thomas Berker, Haddon, L., Hartmann, M., Hynes, D., Rommes, E., … Ward, K. J. (2006). In T. Berker, M. Hartmann, Y. Punie, & K. J. Ward (Eds.), *The domestication of media and technology* (1st ed.). New York, USA: McGraw-Hill Education.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *Principles behind the agile manifesto*. Retrieved April 5, 2015, from http://www.agilemanifesto.org/principles.html

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems case research. *MIS Quarterly*, *3*(3), 369–386.

Brussel, V. U. (2013). *Smartphone domestication in the urban environment*. Vrije Universiteit Brussel.

Chappell, D. (2012). How SaaS changes an ISV's business. Retrieved from http://www.davidchappell.com/writing/white_papers/How_SaaS_Changes_an_ISVs_Business--Chappell_v1.0.pdf

Chigona, A., Chigona, W., Kayongo, P., & Kausa, M. (2010). An empirical survey on domestication of ICT in schools in disadvantaged communities in South Africa. *International Journal of Education and Development Using Information and Communication Technology*, *6*(2), 21–32.

Cocco, L., Mannaro, K., & Concas, G. (2012). A model for global software development with cloud platforms. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 446–452). Çeşme, Turkey: IEEE.

Cockburn, A. (2006). *Agile software development: The cooperative game* (2nd ed.). Boston, USA: Addison-Wesley Professional.

Cohen, B. (2013). PaaS: New opportunities for cloud application development. *IEEE Computer Society*, *46*(9), 97–100.

Cohen, D., Lindvall, M., & Costa, P. (2003). *Agile software development*. Data & Analysis Center for Software.

Darwish, N. R. (2014). Enhancements in Scrum framework using extreme practices. *International Journal of Intelligent Computing and Information Science*, *14*(2), 53–67.

Dumbre, A., Senthil, S. P., & Ghag, S. S. (2011). *Practicing agile software development on the Windows Azure platform*. Retrieved April 25, 2015, from http://www.infosys.com/cloud/resource-center/Documents/practicing-agile-software-development.pdf

Esbensen, M., Jensen, R. E., & Matthiesen, S. (2014). Does distance still matter? Revisiting the CSCW fundamentals. *ACM Transactions on Computer-Human Interaction*, *21*(5), 1–26.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28–35.

Fuggetta, A., Nitto, E. Di, & Milano, P. (2014). Software process. In *Proceedings of the on Future of Software Engineering* (pp. 1–12). Hyderabad, India: ACM.

Ghohandizi, F. A. (2014). *Cloud-based software development for a federated cloud* (Masters thesie). Tampere University of Technology.

Gill, A. Q., & Bunker, D. (2013). Towards the development of a cloud-based communication technologies assessment tool: An analysis of practitioners' perspectives. *Vine*, *43*(1), 57–77.

Guha, R., & Al-Dabass, D. (2010). Impact of Web 2.0 and cloud computing platform on software engineering. *Proceedings of the International Symposium on Electronic System Design*, 213–218.

Haddon, L. (2006). The contribution of domestication research to in-home computing and media consumption. *The Information Society*, *22*(4), 195–203.

Haddon, L. (2007). Roger Silverstone's legacies: Domestication. *New Media & Society*, *9*(1), 25–32.

Harwood, S. A. (2011). The domestication of online technologies by smaller businesses and the "busy day." *Information and Organization*, *21*(2), 84–106.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120–122.

Hossain, E., Bannerman, P. L., & Jeffery, R. (2011). Towards an understanding of tailoring Scrum in global software development. In *Proceedings of the International Conference on Software and Systems Process* (pp. 110–119). Honolulu, USA: ACM.

Hynes, D., & Richardson, H. (2009). What use is domestication theory to information systems research? In Yogesh Kumar Dwivedi, B. Lal, & M. D. Williams (Eds.), *Handbook of research on contemporary theoretical models in information systems* (1st ed., pp. 482–494). Hershey, Pennsylvania: Information Science Reference.

Jasemian, T., Mortensen, R. S. K., & Boje-Nielsen, T. (2007). *Measuring agility: A quantitative survey among IT professionals*. Aalborg University.

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, *41*(8), 3809–3824.

Kamaruddin, N. K., Arshad, N. H., & Mohamed, A. (2012). Chaos issues on communication in agile global software development. In *Proceedings of the IEEE Business, Engineering and Industrial Applications Colloquium* (pp. 394–398). Kuala Lumpur, Malaysia: IEEE.

Kavis, M. (Ed.),(2014). *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, and IaaS)* (1st. ed.). Hoboken, NJ, United States of America: John Wiley & Sons, Inc.

Kim, D. (2013). *The state of Scrum: Benchmarks and guidelines.* Retrieved April 20, 2015, from https://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/Files and PDFs/State of Scrum/2013-State-of-Scrum-Report_062713_final.pdf

Lee, Y. S., Smith-Jackson, T. L., & Kwon, G. H. (2009). Domestication of technology theory: Conceptual framework of user experience. In *Proceedings of the 27th CHI Conference* (pp. 1–5). Boston, USA: ACM.

Lin, C. C. (2014). Toward a cloud based framework for facilitating software development and testing tasks. In *Proceedings of 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing* (pp. 1–2). London, United Kingdom: ACM.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2012). *NIST cloud computing reference architecture: Recommendations of the national institute of standards and technology (Special Publication 500-292)*. Retrieved May 10, 2016, from http://www.isaca.org/Groups/Professional-English/cloud-computing/GroupDocuments/909505.pdf

Linkevics, G. (2014). Adopting to agile software development. *Applied Computer Systems*, *16*(1), 64–70.

Mani, P., Jayakumar, S., & Gopalakrishnan, R. (2014). Enhancing agile software development using cloud computing : A case study. *International Journal of Research in Management & Business Studies*, *1*(1), 127–129.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology*. Retrieved October 5, 2014, from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Moe, N. B., Cruzes, D. S., Dyba, T., & Engebretsen, E. (2015). Coaching a global agile virtual team. In IEEE (Ed.), *Proceedings of the 10th International Conference on Global Software Engineering* (pp. 33–37). Ciudad Real, Spain.

Münch, J. (2013). Impact of cloud computing on global software development challenges. In *Proceedings of Cloud-Based Software Engineering* (pp. 34–39). Helsinki, Finland: University of Helsinki.

Mwansa, G., & Mnkandla, E. (2014). Migrating agile development into the cloud computing environment. In *2014 IEEE International Conference on Cloud Computing* (pp. 818–825). Anchorage, USA: IEEE.

Nilsson, C. G., & Karlsson, D. (2014). *Implementing agile project methods in globally distributed teams*. Karlstad University.

Pomar, F. A., Calvo-Manzano, J. A., Caballero, E., & Arcilla-Cobián, M. (2014). Managing the software process with a software process improvement tool in a small enterprise. *Journal of Software-Evolution and Process*, *26*(9), 481–491.

Pulkkinen, V. (2013). Continuous deployment of software. In *Proceedings of Cloud-Based Software Engineering* (pp. 46–52). Helsinki, Finland: University of Helsinki.

Richardson, I., Casey, V., Burton, J., & McCaffery, F. (2010). Global software engineering: A software process approach. In I. Mistrík, J. Grundy, A. Hoek, & J. Whitehead (Eds.), *Collaborative software engineering* (Vol. 1, pp. 35-56). Berlin: Springer-Verlag.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164.

Sandtrø, T. (2012). How the domestication process of a VLE came to closure. *The Online Educational Research Journal*, *3*(7), 2–8.

Saunders, M., Lewis, P., & Thornhill, A. (2009). *Research methods for business students*. Harlow, Essex: Pearson Education Ltd.

Schneider, S., & Sunyaev, A. (2014). Determinant factors of cloud-sourcing decisions: Reflecting on the IT outsourcing literature in the era of cloud computing. *Journal of Information Technology*, 1–31.

Schwaber, K., & Sutherland, J. (2013). the scrum guide. Retrieved April 24, 2015, from http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

Smirnova, I. (2013). Impact of cloud computing on global software development challenges. In *Proceedings of Cloud-Based Software Engineering* (pp. 71–78). Helsinki, Finland: University of Helsinki.

Spinellis, D. (2014). Developing in the cloud. *IEEE Software*, *31*(2), 41–43.

Stankov, I., & Datsenka, R. (2010). Platform-as-a-service as an enabler for global software development and delivery. *Multikonferenz Wirtschaftsinformatik*, 555–566.

Sverrisdottir, H., Ingason, H., & Jonasson, H. (2014). The role of the product owner in Scrum-comparison between theory and practices. In *Proceedings of 27th IPMA World Congress* (Vol. 119, pp. 257–267). Dubrovnik, Croatia: Elsevier B.V.

Takkunen, M. (2014). *Scrum implementation in a virtual team environment* (Master's thesis). Helsinki Metropolia University of Applied Sciences.

Tuli, A., Hasteer, N., Sharma, M., & Bansal, A. (2014). Empirical investigation of agile software development: A cloud perspective. *ACM SIGSOFT Software Engineering Notes*, *39*(4), 1–6.

Voss, C., Tsikriktsis, N., & Frohlich, M. (2002). Case research in operations management. *International Journal of Operations & Production Management*, *22*(2), 195–219.

Wang, W. (2011). *Reinforcing agile software development in the cloud*. Retrieved May 10, 2016, from https://www.open.collab.net/media/pdfs/CollabNet%20Whitepaper_Reinforcing%20Agile%20Dev%20in%20the%20Cloud.pdf?_=d

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

Yin, R. K. (2009). *Case Study Research: Design and Methods*. (V. Knight, S. Connelly, L. Habib, C. M. Chilton, & G. Dickens, Eds.) (4th ed.). Thousand Oaks, USA: Sage Publications, Inc.

Appendix

Table 6: Summary of Cloud Enablers of the Agile Principles while performing AGSD

Category: Regular Delivery		C1	C2
Principle	Cloud Based Support		
P01 Deliver value to the customer	• Parallel development environments (Stankov & Datsenka, 2010).	√	√
	• SaaS: Platform for deploying software (Chappell, 2012).	√	√
	• Product owners determine, prioritize, and track value using agile management software (Azizyan et al., 2011).	√	-
P03 Deliver Frequently	• Continuous integration (Mani, Jayakumar, & Gopalakrishnan, 2014).	√	√
	• IaaS, PaaS: fewer delays in freeing developers to focus on development work (Smirnova, 2013).	√	√
	• SaaS: Platform for rapid software deployment (Chappell, 2012).	√	√
P07 Working Software Measure	• Continuous integration (Mani et al., 2014).	√	√
P08 Sustainable development	• Agile management software: automated tracking of delivery rate and activities (Hossain, Bannerman, & Jeffery, 2011).	√	-
	• Better time management due to the ability to deliver frequently (Smirnova, 2013).	√	√
Category: Communication and Collaboration		C1	C2
Principle	Cloud Based Support		
P04 Business & SD Collaboration	• Agile management software: allows product owner to prioritize value with the collaboration of whole team (Azizyan et al., 2011).	√	–
	• Continuous integration allows for regular feedback between business people and developers (Mani et al., 2014).	√	√
	• PaaS and SaaS are globally accessible (Chappell, 2012).	√	√
	• SaaS: No specialized tools for business users, transparency (Chappell, 2012).	√	√
P05 Motivated Team Members, Environment, Trust	• Agile management software highlights when tasks are taking too long allowing the whole team to address issues that may go unnoticed (Tuli et al., 2014).	√	–
	• Cloud-based code management provides developers with source code that is current (Pulkkinen, 2013)	√	√

	• PaaS: supports the team by providing a uniform development environment between locations, limits the required software required for setup and configuration (Münch, 2013).	√	√
P06 Face to Face Communication	• The impact of distance distributed teams can be minimized through the use of CC tools email, instant messaging, screen sharing, shared document spaces, and video conferencing (Gill & Bunker, 2013).	√	√
P11 Self-organizing teams	• Agile management software – improvements can be tracked (Azizyan et al., 2011).	√	–
	• CC allows for more options to overcome AGSD challenges	√	√
P12 Continuously Adapt and Improve	• Code Management tools allow for code reviews to highlight areas of improvement (Moe, Cruzes, Dyba, & Engebretsen, 2015).	√	–
Category: Technical Excellence		C1	C2
Principle	Cloud Based Support		
P10 Simplicity	• IaaS, PaaS: Lower IT skills required for setup of stable and scalable development and production environments (Spinellis, 2014).	√	√
P09 Technical Excellence	• Centralized Code Management supports code reviews, enforces standardization, and eliminates duplication (Stankov & Datsenka, 2010).	√	√
	• Cloud platforms are robust, scalable, and highly reliable (Chappell, 2012).	√	√
	• Automated testing begins at the point of committing code and at every integration point (Lin, 2014).	–	–
P02 Welcome Changing Requirements	• Resilience to the impact of change requires proper adherence to the other 11 principles.	√	√
Key: Present =√ Absent/Not explicitly covered = –			

Biographies

Timothy Haig-Smith obtained a BCOM (Hons) degree from the University of Cape Town specializing in Information Systems in 2015, after a long absence from academia. He has worked in a variety of industries as a software developer and software development manager for several years. He is presently pursuing a Masters degree in Information Systems. His research interests include software development team dynamics, software development practice, and technical debt.

Dr Maureen Tanner has been teaching systems analysis and design at the Department of Information Systems of the University of Cape Town since 2009. Her research interests lie in Agile software development related issues (for both collocated and distributed teams), UML, software engineering and social aspects of social engineering, global software development, virtual teams, and team collaboration.