The Software Development Landscape

A Rationalization of Agile Software Development as a Strategy in the face of Organizational Complexity

by

Stephan Marnix Braams
(s1359185)

A thesis presented for the degree of:
Master of Science

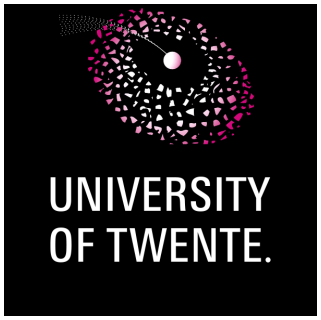
MSc Philosophy of Science, Technology and Society - PSTS

10 March 2020

Supervisor:
Dr. Miles A. J. MacLeod

Second Reader:
Dr. ir. Klaasjan Visscher

Faculty of Behavioural, Management and Social Sciences
University of Twente
Enschede, The Netherlands



Summary

‘Agile’ is a collective term for a group of development methods originating from the Software Development (SWD) domain. The most used of these methods is called ‘Scrum’. Agile is used in the vast majority of software development companies and has largely replaced the previously dominant Waterfall-style of software development. While Waterfall was a plan-driven, sequential, linear methodology, the Agile approach is based on an iterative and incremental process. Agile was originally developed for structuring IT projects, organizing roles and responsibilities in SWD teams as well as prioritizing and distributing work.

Strikingly, Agile is now increasingly spreading to other domains such as education, thesis supervision, controversy mapping and many more. Agile is thus widely used in response to complex problem-solving contexts, but currently, there is no well-developed and intuitive rationalization of Agile readily available. Synthesising concepts form Philosophy of Science, Complexity Science and the Agile Community, in this thesis, I show how the implicit rationale behind Agile can be articulated visually and intuitively .

The main research question that guides the research is: “What are some of the fundamental intuitions and rationalizations motivating the application of Agile in current core Agile documents, how effective are these rationalizations and how can they be improved by drawing on concepts from Philosophy of Science and Complexity Science?” The general research approach is based on critical reflection on personal experience in the SWD domain, akin to auto-ethnography. The epistemology of this thesis is therefore non-positivist and non-representationalist. First, the 2001 Agile Manifesto and the Scrum Guide are defined as the core documents of Agile. The justifications for the use and effects of Agile within these documents are analysed. Subsequently, a novel rationalization is proposed through a synthesis of several concepts, including David Snowden’s Cynefin Complexity framework and Weisberg and Muldoon’s epistemic landscape model. The landscape metaphor demonstrates that the appropriateness of approaches like Waterfall (plan-driven, linear) and Agile (iterative, incremental) depends on the epistemological conditions under which they are used.

Therefore, the main conclusion of the research is that it is useful to understand Agile as a reaction to, and strategy for dealing with, the inherent uncertainty present in organizational environments such as IT-projects. This uncertainty is comparable to the non-linear causality observed in complex adaptive systems. Since this type of complexity-related uncertainty can emerge in many contexts, this offers a plausible explanation for the observed spread of Agile outside the SWD domain.

The implications of the rise of Agile for the academic fields studying science and technology could be significant. If a new approach needed to be found to develop software, might there be a similar need for a new approach to understanding software? Can it be safely assumed that the knowledge,

theories and research methods developed for physical technologies directly apply to software or is there a need for adjustment or even replacement, comparable to what happened in the SWD domain? This thesis offers some of the groundwork needed to pull these kinds of questions into view and enabling them to be explored.

In a broader sense, this thesis also serves as an example of the potential of approaching the organization as a distinct object and level of analysis. The organization can be seen as a meso-level, nested between the micro/human level and the macro/society level, with its own particular distinguishing characteristics and influence. Since a lot of modern science takes place in organizations, and many technologies are developed in the context of organizations, the influence that this context has should be accounted for. One way that the organization has an influence, is through the structuring effect of organizational methodologies on prioritization, division of labour, collaboration and communication. Therefore, understanding the logic and rationale of methodologies such as Agile helps to explain certain decisions, dynamics and developments related to knowledge and technology production.

Acknowledgements

Firstly, I am immensely thankful to my supervisor, Miles MacLeod, for his patience, relaxed 'no-worries' attitude and incredibly valuable and insightful contributions to the project. Often, he could phrase what I wanted to be saying better than I could. I aspire to possess such clarity of thinking one day too.
Secondly, I want to thank the examiner, Klaasjan Visscher for elevating the project by challenging me to reflect on the process of reflection and articulate how the process of articulation took place, despite long periods of radio silence from my end.
Finally, this thesis would not exist without my invaluable support-system of friends and family. Your incredible generosity of time and patience will not be quickly forgotten.

I don't expect that the master thesis project is a purely academic task for anyone. For most people, it also signifies the end of one's time as a student. Being a student means still having something of a *tabula rasa* over you: somewhere you believe that everything is still possible. Leaving that time behind you means facing some questions on what is important and what you are willing to commit yourself to. It involves building good habits, taking responsibility, and putting in the time. With other words: Growing up. Therefore, I consider myself very lucky for having the time and space to go through this process.

Contents

Chapter 1

Introduction

1.1 Topic & Background

A reader of the Dutch news may have seen the terms ‘Agile’ or ‘Scrum’ popping up recently. Examples include the following headlines:

> Even Prime Minister Rutte is talking about Agile. Is everybody working in the scrum? (Janssen 2019)

> Rijkswaterstaat chooses external agencies for Agile Transformation (Consultancy.nl 2019)

> Bol.com: “Agile approach helped us with WMS implementation”(Dijkhuizen 2019)

The ‘Agile’ that the news articles are talking about is a collective term for several development methods that originated in the Software Development (SWD) domain (Dingsøyr et al. 2012; Dybå and Dingsøyr 2008). This collective consists of Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, Feature-Driven Development (FDD) and Pragmatic Programming (Beck et al. 2001). Out of these, Scrum is by far the most used (Lindsjørn et al. 2016). What unites this diverse collection of methods is the formation of the Agile Alliance and publication of the Agile Manifesto in 2001. Since then, Agile has had “[”p.1]a huge impact on how software is developed worldwide(Dybå and Dingsøyr 2009) and brought “unprecedented changes to the software engineering field”(Dingsøyr et al. 2012, p.1). According to the latest worldwide survey among software developers, 97% out of 1,319 respondents reported that their organization uses Agile techniques (VersionOne 2019, p.7).
While some see this rise of Agile as a Kuhnian paradigm shift within the SWD field (Beck and Gamma 2000, p.231) or even a revolution for how teams and organizations are structured in “virtually every industry”(Sutherland 2014, p.7), others dismiss it as a meaningless management trend (Cram and Newell 2016).
As attested by the news articles, Agile (and specifically Scrum) have also been growing in popularity outside of the SWD domain. Examples of this include the use of Agile in education (Sharp and Lang 2019; Parsons and MacCallum 2019; Bongaerts 2018), thesis supervision (Tengberg 2015; Mora et al. 2015; Magnuson et al. 2019), organization of universities (Twidale and Nichols 2013),

instructional design (Rawsthorne and Lloyd 2005), curriculum development (Mahnic 2011) construction management (Goodwin 2017; Owen and Koskela 2006; Adut 2016), crisis management (Zykov 2016), Sports Science (McCaffery 2018), defence acquisition (Messina, Modigliani, and Chang 2015), and Science and Technology Studies (specifically controversy mapping) (Vertesi and Ribes 2019, p472). Many organizations, both inside the IT domain and outside it are undergoing a so-called 'Agile Transformation'.

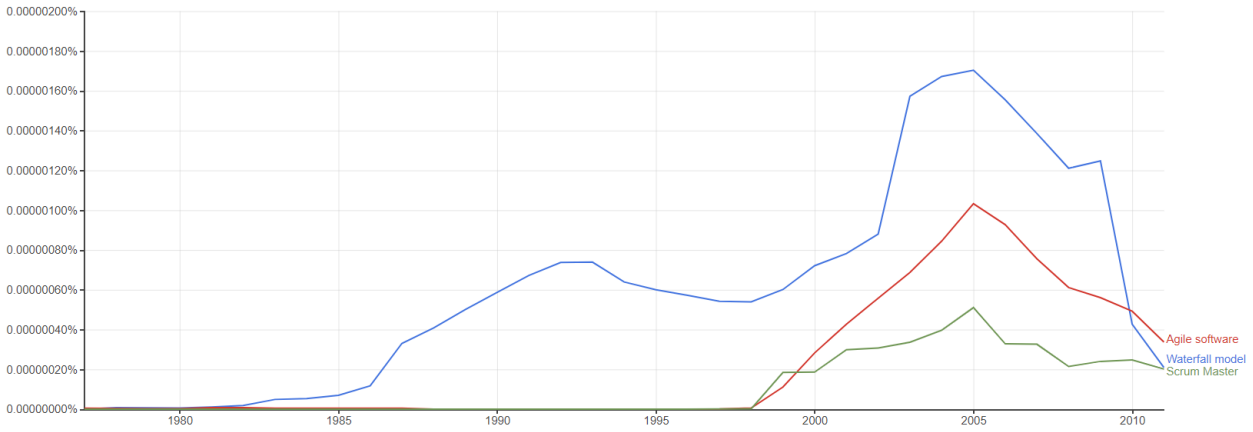


Figure 1.1: Indication of notability/popularity of concepts using Google Ngram. Blue is waterfall, Red is Agile, Green is Scrum

Despite its impact, there is no universally accepted definition of what it means to be Agile (Conboy and Fitzgerald 2004). An inclusive way to describe Agile methods would be a collection of practices, techniques, methods and ideas concerning how "software developers plan and coordinate their work, how they communicate with customers and external stakeholders, and how software development is organized in small, medium-sized and large companies"[1](Dingsøyr, Dybå, and Moe 2010, p.15)

Often, Agile is characterized in contrast to its predecessor, the previously dominant "Waterfall" (also called plan-driven) approach to software development (Ralph 2013). In the history section accompanying the Agile Manifesto Waterfall is characterized as a "document-driven, heavyweight software development process" (Beck et al. 2001). According to Conboy and Fitzgerald: "It is now widely accepted that these methodologies are unsuccessful and unpopular due to their increasingly bureaucratic nature. Many researchers and practitioners are calling for these heavyweight methodologies to be replaced by agile methods" (Conboy and Fitzgerald 2004, p.105). Since 1994 each year the Standish Group has published the 'Chaos Report', surveying IT-project success. In the 1994 Chaos Report, only 16% of IT-projects could be counted as successful under the definition of the Standish Group (Standish 2014). Agile proponents often attribute these figures to the failure of the Waterfall approach, but the validity of the numbers has come under question (Eveleens and Verhoef 2009). A causal network schema of the arguments I have personally observed pro-Agilist use is pictured in figure 1.9. In line with this approach of defining Agile through contrast with Waterfall, Nerur et al. offer the table pictured in figure 1.3.

1. Considering the use of Agile outside of software development, the word 'software developer' could be replaced here with 'team member' or something equivalent

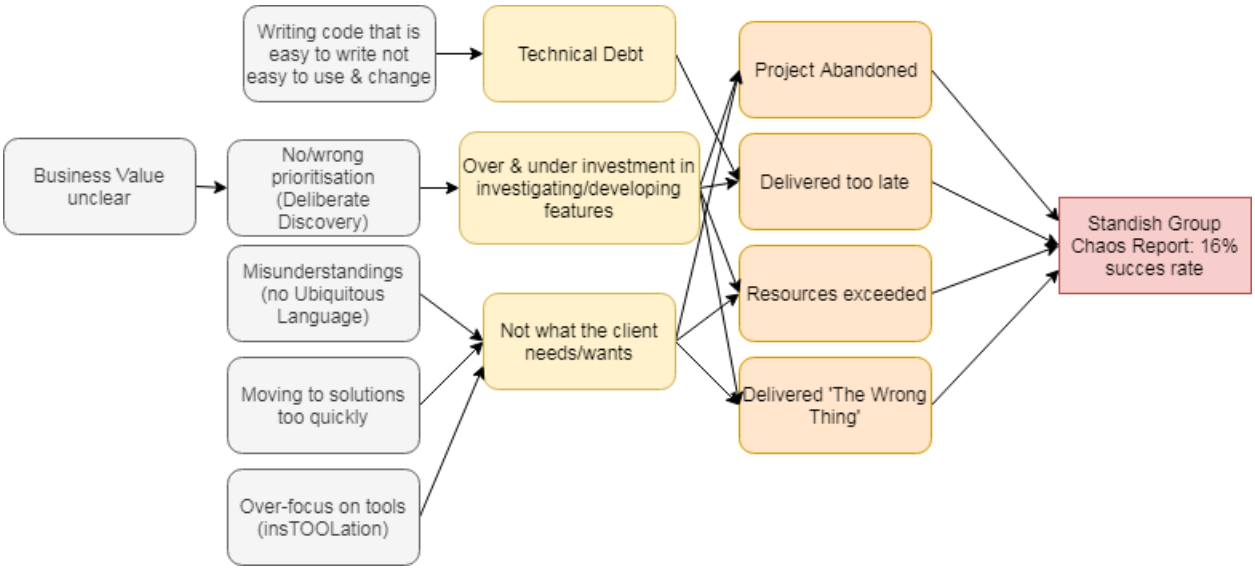


Figure 1.2: Anecdotal argument schema of supposed causes of low IT project success rate

	Traditional development	Agile development
Fundamental assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal
Development model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
Desired organizational form/structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality control	Heavy planning and strict control. Late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing

Figure 1.3: Main differences between traditional (Waterfall) development and Agile development according to (Nerur, Mahapatra, and Mangalaraj 2005, p.75)

Another common way that both the Waterfall and Agile approaches to software development are characterized is through schematic representations of the specified project structure. The sequential structure that gives Waterfall its moniker is depicted in figure 1.4[2]. Like a Waterfall, the project moves down through the phases in one direction.

In contrast Agile, and specifically Scrum (see figure 1.5[3]), is often represented with a circular diagram, showing the iterative nature.

For the remainder of this thesis, it is assumed that the reader has knows the basic elements of the Agile/Scrum process. For the reader where this is not the case, a summary of the Scrum Guide is included in appendix A.

One dimension along which IT projects have changed since the rise of Agile is the Lead Time: the time it takes a unit of functionality to go from being requested to being operational and actually

2. Image source: https://blog.standupti.me/post/127220134289/waterfall-software-development-isnt-dead
3. Image source: https://cdn.thinglink.me/api/image/693177623857070082/1240/10/scaletowidth

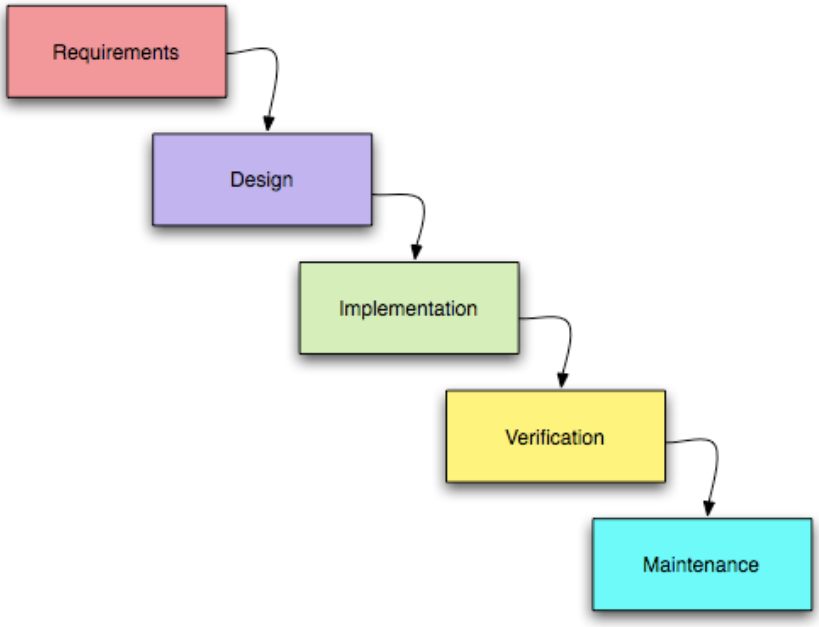


Figure 1.4: A Representation of the Waterfall process

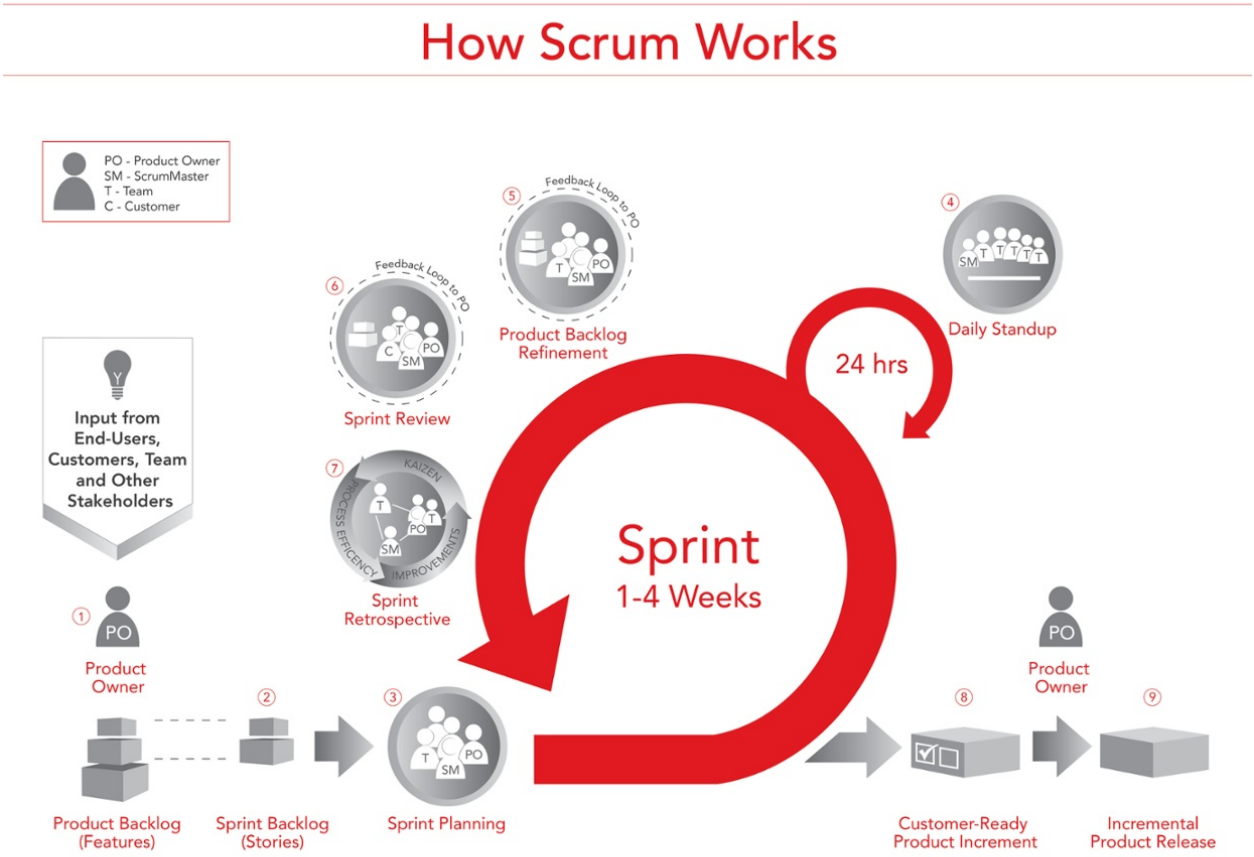


Figure 1.5: A Scrum Diagram

in-use by users. In the Waterfall era lead time would have to be measured in months or even years (Abbas 2009, p. 12). In contrast, today among the highest performers in the industry new software is released to users many times a day (Forsgren et al. 2019, p. 44). An example of this change in development practices is Spotify, which reportedly adjusts the functionality of its application hundreds of times per day, running different versions in particular time zones and countries to compare which implementation decision improves key metrics the most (Finnegan 2016).

Additionally, the scope of the development process has expanded to the point that it envelops virtually the whole lifespan of use. Any smartphone or computer user can attest to this due to the constant updates one receives. A recent example of the influence of the development process on software is Windows 7. The support from Microsoft for the Windows 7 Operating System ended January 14th (Windows 2020) which means that no new updates would be released for the software and all users were advised to switch to Windows 10. In other words, when the software was no longer in the development process it could no longer be used. Compare this to a piece of physical technology such as a chair. The influence of the development process of a chair is limited to the discrete phase of construction, or arguably even limited to the design phase. Understanding Technology has always included the need for understanding the process by which that technology came into being. If we want to understand why a chair is the way it is, understanding how it was designed is one important factor. For software technology understanding the development process is thus even more important, since to different degrees software is always under development. To understand the development process, the logic of the development methodology needs to be understood.

1.2 Problem Statement

Given the ubiquity and impact of IT on our western industrialized lives and society the need for understanding software, and therefore Agile methodology, is vital. However, critical attention for Agile within the academic fields that study Science and Technology (PSTS) such as Philosophy of Technology, Philosophy of Science and Science and Technology Studies is almost non-existent[4]. More generally, the organization is not often recognized as something to take into account in its own right in PSTS.

The ubiquity of Agile within the SWD domain suggests that there are certain ideas in Agile that are very useful. The spread of Agile to all of the other domains mentioned suggests that those ideas are also (becoming)[5] useful more generally.

However, a closer look at Agile quickly reveals a lot of uncertainty, ambiguity and confusion about what exactly these useful ideas are. Firstly, it is not immediately clear who or what gets to call themselves Agile and therefore can be counted as a member of the community, which inspires

4. See section 1.4
5. It could be that other domains have similar problems to the software domain and they are now discovering Agile as a useful solution, or it could be the case that other domains are evolving on certain aspects (perhaps information density, speed of change or degree of uncertainty) that introduces problems for which Agile proves to be a remedy

much debate (and confusion) about who and what can actually call itself Agile (Laanti, Similä, and Abrahamsson 2013). Examples of this are the controversy if scaling methods such as SAFe are Agile, and whether specific practices such as estimations are Agile. Secondly, that an organization calls itself Agile does not mean that it is 'actually' Agile. The Agile community has many names for these corrupted versions of their methodologies including Dark Agile, Fake Agile, Dark Scrum, Scrum In Name Only, and many more. An example of this is found in (Moreira 2013, p.15):"Although many companies say that they are doing Agile in some form, a large proportion of these are actually doing Fragile ("fake Agile"), ScrumBut ("I'm doing Scrum but not all of the practices"), ScrumFall ("I'm doing mini-waterfall in the sprints or phase-based Agile"), or some other hybrid variant that cannot deliver the business benefits of pure Agile". This constant accusation and redrawing of boundaries complicates the understanding of what Agile is further. On top of this, the confusion is not helped by all the different practices making a claim to the Agile name and fame. Agile is constantly expanding with new Next Big Things (e.g. DevSecOps, NoEstimates, AgileNext). Portman gives a "Bird's eye view of the Agile forest" in figure 1.6 (Portman 2019).

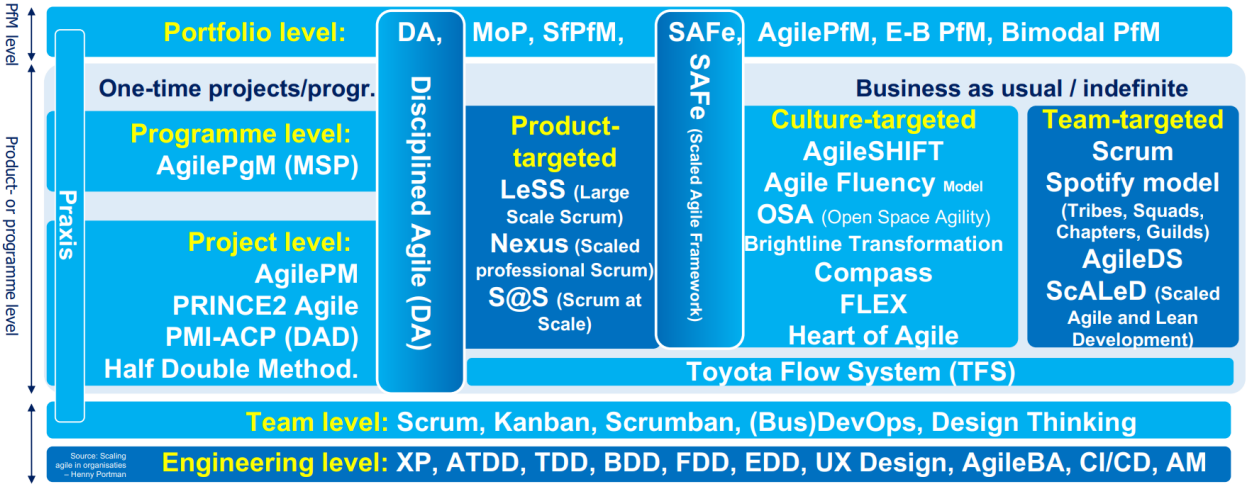


Figure 1.6: A bird's eye view of the Agile forest

The ambiguity about what Agile is and why it is used is also reflected in the news articles which were mentioned at the start of the chapter. The main reasons for adopting Agile mentioned there are that Agile helps an organization adapt better to changing circumstances. However, it remains implicit *how* Agile is supposed to achieve this effect and why this is suddenly desirable (did circumstance not need adjusting to in the past?). While the news articles note the ubiquity of Agile, they offer no underlying conceptual layer in the form of an explanation or justification for its success nor the claimed effects.

The articles also describe Agile and Scrum as 'management terms' which are "cringe inducing". The management field has a reputation for guru's, fashion trends and hypes (Cram and Newell 2016, p.156). The hype that is (sometimes deliberately) whipped up by consultants and certification bodies (ibid., p.156) around the Next Big Thing makes separating fact from fiction hard (Meyer 2014). In the face of this hype, septics believe that Agile is nothing new at all (Ågerfalk, Fitzgerald, and In 2006). However, if the principle of charity is followed, it can't be assumed that all of the success of Agile both within the software development domain and outside of it can be simply

written off as gullible people falling for a management scam. Agile must be addressing some need or problem that exists not only in the SWD domain but also widely outside of it.

What this need is does not become immediately clear from the usual descriptions and justification of Agile. For example, the term ‘flexibility’ is mentioned in the articles as if that is an explanation by itself:

> [The person using Agile] sees a few advantages: “Roles make it possible to handle changing situations more flexibly” Janssen 2019

It remains unclear what it is about the roles defined by Scrum the help flexibility, and from this quote it seems that flexibility is the motivating reason to adopt Scrum.

> It offers us the flexibility we need in our IT-landscape. (...) It has helped us greatly to be flexible. Dijkhuizen 2019

Looking at how Agile is discussed here, the only explanation on offer for why to adopt Agile is “flexiblity”. The same pattern can be observed with respect to the term ‘autonomous team’. From how the term is commonly used when discussing Agile it appears that autonomous teams are an inherent good that should be striven after for its own sake. In the public discussion of Agile, the justifications for the use of Agile care quite thin.

The journalistic sources analysed up to now may be excused for not delving deeper into the conceptual basis underlying Agile, but in the closest academic field, Software Engineering, similar issues pop up. While many specific practices and cases have been studied “not enough attention is being paid to establishing theoretical underpinnings” (Dingsøyr et al. 2014, p.1219). A more elaborate characterization of the Software Engineering literature concerning Agile can be found in section 1.4.1. In addition to an academic concern, this lack also has implications for practice: “without the clear understanding of underlying theoretical principles behind software development approaches, organizations are at the high risk of being non-adaptive”(Jain and Meso 2004, p.1667). In other words, if the core principles and ideas of Agile are not understood in an explicit manner, organizations cannot independently adapt and improve the methods to their own context and future novel situations.

I have also observed the absence of a coherent narrative explaining and motivating Agile in practice personally. In my experience, when pressed practitioners often refer to a ‘Agile Mindset’. It is then said that even if all of the surface-level practices of Agile are followed an organization is not ‘truly Agile’ until it adopts an Agile Mindset. However, what this mindset entails remains vague. Another way that this sentiment is stated is through referring to Agile as more of a ‘philosophy’ than a concrete method (Abbas, Gravell, and Wills 2008, p.95). Even Scrum, which specifies some very specific practices in its Scrum Guide states: “Scrum is not a process, technique, or definitive method. Rather, it is a framework within which you can employ various processes and techniques” (Sutherland and Schwaber 2017, p.3). What Agile actually has to offer, which problem it solves, remains hidden in this vague language. In practice, I have observed that this results in problems explaining Agile to newcomers, clients and other stakeholders. It usually take some experience in an Agile environment for someone to ‘get’ what its about. But, when asked, ‘getting’ Agile in that way does not mean that that person can then articulate what it is that they get. Marick therefore argues that adopting a new methodology requires a “gestalt switch” on the ontological level (Mar-

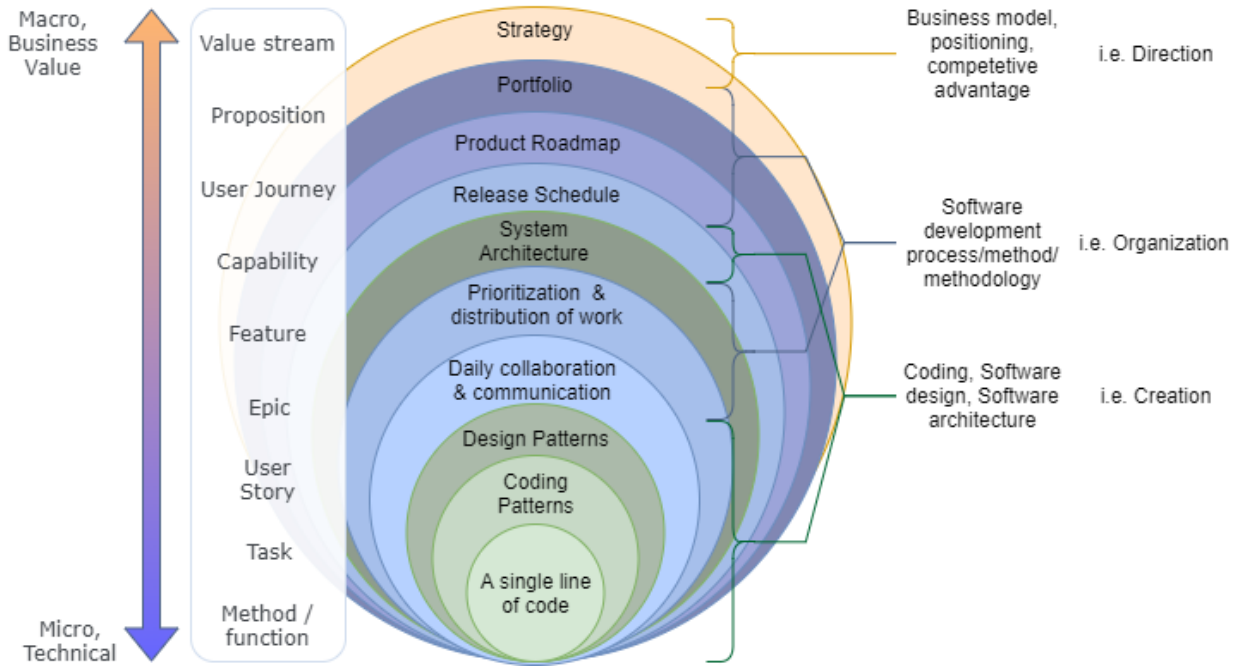


Figure 1.7: Levels of organization and analysis in software development organizations

ick 2004, p.64). The type of knowledge involved here appears to be of the type “I know it when I see it”, i.e. a type of tacit know-how. This “tacit knowledge problem” (Shull et al. 2002) impedes the transfer of knowledge, but also its development. If the core ideas of Agile are not explicitly understood, it is much harder to adapt them to a particular situation or organization. The improvement and innovation of the core ideas of Agile are therefore also stifled by remaining in a diffuse, implicit form.

Another way to phrase this observation is to compare it to cooking. You might be a quite decent cook, possessing the tact know-how of preparing several recipes. But this tacit knowledge is not easily transferred to others or use to develop new recipe’s. Only when you master the principles of cooking can you tech others and apply them in novel ways. The need then, observed both in the literature and in practice, is to isolate these core insights of Agile and put them in a form that can be understood without software development experience.

1.3 Research Approach and scope

Another way to state the identified problem is as there there is a need for a nuanced and intuitive rationalization of Agile methodology. The term ‘rationalizations’ is not mean in it’s psychological or sociological variations, but instead as identifying the underlying logic through (philosophical) analysis. This type of conceptual analysis can be compared with a philosophical project that took place in the philosophy of biology concerning the gene concept (Stotz and Griffiths 2004; Waters 2004). The use of the concept gene by biologists seemed inconsistent to philosophers. The goal of Stotz and Griffiths was to “re-interpret the knowledge of contemporary genetics by replacing sloppy thinking based on unclear concepts with more rigorous thinking in terms of precise concepts. Showing that scientists’ actual thinking does not align with the precise application of these

concepts would not refute the analysis supporting the classical gene or molecular gene concepts and it would not undermine the argument motivating the proposal for the new process molecular gene concept". While their goal was precision of the use of concepts, the goal of this thesis is articulating the tacit intuitions underlying Agile in an experience independent, intuitive way.

To avoid confusion, the level of analysis which this thesis concerns itself with should be specified. A number of different levels that could be discerned in figure 1.7. A lot has been written about the technical (green in the diagram, labelled 'creation') levels. The focus of this thesis lies on the organizational (blue) levels. The research methodology is elaborated on in chapter 2.

1.3.1 Research Questions

With the problem and research approach formulated, the research question can now be formulated. Therefore, the main research question of this thesis is:

> What are some of the fundamental intuitions and rationalizations motivating the application of Agile in current core Agile documents, how effective are these rationalizations and how can they be improved by drawing on concepts from Philosophy of Science and Complexity Science?

The main question is answered through the sub-questions:

1. What are the core documents of Agile?
2. What justification for the use and effects of Agile do the core documents offer?
3. What justification for the use and effect of Agile are used in the Agile community?
4. What are the limits of these current rationalizations?
5. How can the intuitions underlying Agile be better rationalized using notions from Philosophy of Science and Complexity Science?

With the 'better' in 'better rationalized' I mean more intuitive and independent of software development experience and knowledge of complexity science.
To situate the thesis, some related literature is now reviewed.

1.4 Related work

This is a multi-disciplinary thesis, and therefore multiple bodies of literature are relevant. These include the Software Engineering literature, the PSTS literature and the (non-academic) Agile Community. The search phrases, search engines and other sources consulted for the literature research can be found in Appendix B.
The goal of the literature research is not to arrive at a generalizable representation of opinion on Agile within the different bodies of literature. This is elaborated further on in chapter 2.

1.4.1 Software Engineering Literature

In a review of the available Software Engineering literature, Dingsøyr et al. state that while there are ample case studies and investigations into specific Agile practices such as peer programming,

the ‘core’ of Agile requires “a unified framework that brings coherence to the seemingly disparate streams of research being pursued” (Dingsøyr et al. 2012, p.1219). They continue:

> Our limited analysis of the theoretical perspectives used in prior agile development research suggests that not enough attention is being paid to establishing theoretical underpinnings, when investigating agile development and its various practices (...) sound theoretical roots help us glean the essential concepts, or the “truths” of software development that are methodology-independent (...) Therefore, we urge agile researchers to embrace a more theory-based approach in the future when inquiring into these promising research areas of agile development. ibid., p.1219

Jain and Meso support this characterization of the software engineering literature: “Much of the prior literature does not provide any theoretical basis for these [Agile] methodologies” (Jain and Meso 2004, p.1661). This sentiment is also present in Nerur et al.:

> While the growing popularity of agile development methodologies is undeniable, there has been little systematic exploration of its intellectual foundation. Such an effort would be an important first step in understanding this paradigm’s underlying premises. This understanding, in turn, would be invaluable in our assessment of current practices as well as in our efforts to advance the field of software engineering. Nerur et al. 2010, p.15

Conboy et al. give the following list of problems surrounding the understanding of Agile (Conboy, Fitzgerald, and Golden 2005, p.36):

- “Many different definitions of an agile method exist. However, this is not surprising given that IS researchers cannot even reach consensus on the definitions of the most basic terms such as information system, method, and technique.”
- “Many different agile methods exist. Each of these methods focuses heavily on some of the principles of the agile manifesto and ignore others completely, but yet are portrayed by some not only as an agile method, but as the best agile method.”
- Often not all elements of a method such as eXtreme Programming is used. It is unclear if this use counts as agile or not.
- “There are some, especially those using more traditional ISD methods, who disregard agile methods, as unstructured, ad hoc, glorified hacking”
- “Cockburn (2002a) even dismisses the existence of an agile method altogether, claiming that it is something to which developers can only aspire, and that only hindsight can determine whether an agile method was actually adhered to.”
- “Finally, there is a perception among the purveyors ofthe agile method that all prlor methods were non-agile.”

Northover et al. observe a broader need for foundational work in the software engineering literature: “Its [Software Engineering] philosophical foundations and premises are not yet well understood. In recent times, members of the software engineering community have started to search for such foundations. In particular, the philosophies of Kuhn and Popper have been used by philosophically-minded software engineers in search of a deeper understanding of their disci-

pline. It seems, however, that professional philosophers of science are not yet aware of this new discourse within the field of software engineering."(Northover et al. 2008, p.85)

1.4.2 PSTS Literatures

Searching the database for philosophy papers philpapers.org resulted in the collection of 17 relevant philosophical sources. The work of Lucas Introna specifically philosophically addresses software development ((Introna 1996; Introna and Whitley 1997; Coyne 1995)) but does not address Agile as such. The closest match to this thesis is Northover et al.'s "Agile Software Development: A Contemporary Philosophical Perspective" (Northover et al. 2007a), where a framework based on Kuhn is used to analyse eXtreme programming, and (Northover, Boake, and Kourie 2006) a Popperian perspective is taken in order to demonstrate the similarity between Agile methodologies and Popper's philosophy.
The lack of work on software development methodologies available in the PSTS fields is best demonstrated by reviewing the software-related issues that *are* focussed on. The significance of software as something to pay attention to is most obvious in the field of computer ethics. Here, the main focus has been on discussions about intellectual property (De Laat 2005), privacy (Johnson 2004) and recently algorithmic bias (Friedman and Nissenbaum 1996). Outside computer ethics, philosophical discussions related to software have mainly centred around issues about the nature of computation and information (Floridi 2008; Brey and Søraker 2009), open-source (Laat 2001) and the ontological status of digital objects (Irmak 2012; Hui 2016). Software-related issues also show up in adjacent fields such as the philosophy of cognitive science, e.g. the relation between computation and cognition (Vallverdú 2010; Wang 2003). Indirectly, issues in the philosophy of mind related to machine consciousness and Artificial Intelligence also involve software. Within Science and Technology Studies (STS) a lot of discussion takes place under the term 'digital', e.g. (Vertesi and Ribes 2019). Issues ranging from software-hardware boundary work (Jesiek 2006) to software and sovereignty (Bratton 2016) are discussed, but software development methodology is not often specifically addressed Perhaps this is because the notion 'design' has more of a history within STS, and the design of software is indeed discussed (e.g. (Johnson et al. 2014)) however, there is a crucial difference between design and development methodology. The notion of design sits closer to the practice of coding itself, while development methodology sits one step higher where the overall process is structured.

One way that discussions about software related issues take place without it standing out is through the use of the term. 'Algorithm' (e.g. Danaher 2016). Discussing software through the term algorithm can obscure that an algorithm is always (part of) a software programme. Furthermore, these software programmes are always embedded in a software development process. The same is true for terms such as Artificial Intelligence, Machine Learning and Smart City, or when (the effects of) a specific application is discussed. In this way the simple fact that all of the software that underlies the plethora of issues mentioned above is always rooted in a software development process remains obscured and evades critical attention.

1.4.3 Agile Community

The non-academic content from the Agile community indirectly informs the characterizations of Agile in this thesis. This influence is through the auto-ethnographic process described in section 2.1.2.

One source which drives constant change in the Agile community is the continuous and rapid development of the software domain itself. A second source is the ever-increasing adoption of Agile both within and outside the SW domain. These change drivers spur a rapid and constantly developing debate that mainly takes place on those across an intricate constellation of mediums that are most geared and suited to quick and succinct communication. The most used and popular sources to share knowledge and experience about the rapidly changing trends and developments in software in general and Agile specifically include personal blogs (e.g. `martinfowler.com`, `ronjeffries.com`, `lizkeogh.com`) and blogging platforms (e.g. `infoq.com`, `dzone.com` and `hackernoon.com`), 'microblogging' site Twitter, popular books ("Twice the work in half the time" (Sutherland 2014), "Accelerate"(Forsgren et al. 2019), "The Unicorn Project" (Kim, Behr, and Spafford 2019)) and practice-orientated conferences aimed at developers (the most visited include QCOn, OSCON, GOTO and Agile 20XX). Recordings of these talks and seminars are hosted on sites like YouTube and Vimeo and subsequently disseminated on Linked-in, Twitter and in 'the blogosphere'. An overview of these elements can be found in appendix C. A schematic representation of the components and relationships between the formal (academic) and informal (community) bodies of work can be found in figure 1.8.

The ideas making up Agile are constantly being formed and debated by a vast network of tweets, blogs, books, conference talks and in-person conversations. At this point, many thousands of people, teams and organisations use techniques, practices and methods that they would call Agile (VersionOne 2019). Every new person hearing about and starting to work with Agile ideas, every new paper or book published, every new blog, tweet and conference talk, every discussion and new idea represents a shift in what Agile is and what it means to any particular person. How can all conceptions and experiences of interaction with Agile ideas be done justice? Moreover, which perspectives are worth taking into account? For example, how much heavier should the opinion of a Manifesto signatory or a 'thought leader' count compare to the view of an unknown practitioner with decades of experience? There is no 'objective' way of deciding what the scope should be and therefore what the cut-off point should be regarding which voice to count in and which to count out. Borrowing two terms from structuralist linguistics, next to this synchronic problem (i.e. what to include in a map of a network at any *fixed* point in time), there is also the diasynchronic problem: i.e. how to account for the changes in the relationships within a network *over time*. How far in the past would views be included I, and since the debate is always ongoing how would recent developments account be accounted for? These problems are not solved by this thesis but are worth taking into account.

1.5 Aims and Contributions

There are several audiences that this thesis has been written for. In the first place, the goal is to propose a way of making sense of Agile for anyone confronted with it, both within and outside the

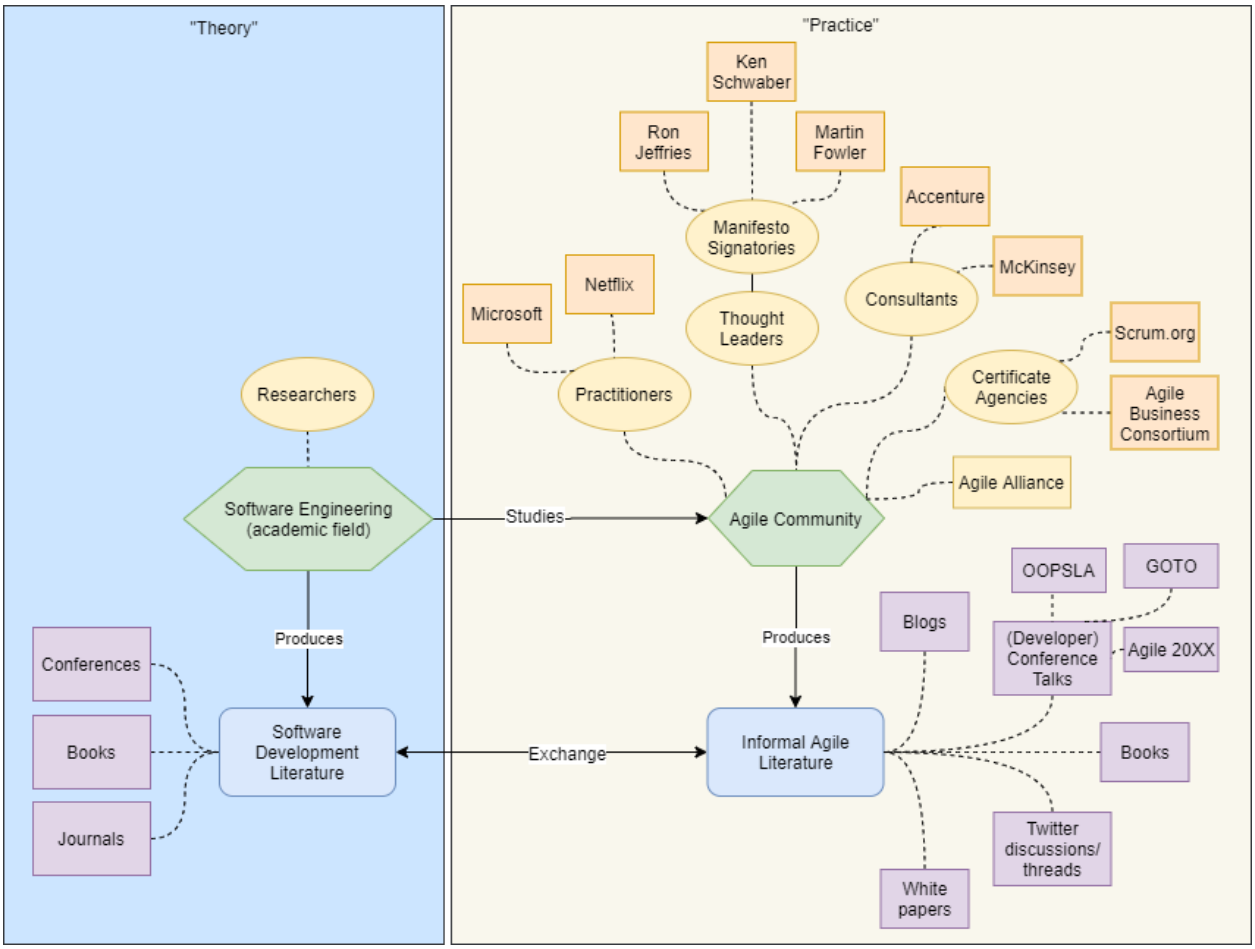


Figure 1.8: Formal and informal software engineering literatures

SWD domain. By no means is the proposed rationalization meant as the ‘best’ or ‘true’ account of Agile. Instead, the goal is a useful way to think that helps connect the dots of why Agile is the way it is.

Explicitly articulating the logic that informs the decisions and development of Agile in has a great potential for value in practice. This potential is captured in the well-known proverb:

> ”Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime”

Specifying a specific solution can help solve a specific problem. But if the underlying principles are uncovered, this has the potential of offering much greater value. Having the core principles in hand does not mean that they are in a form that is easily transferable to any reader. Often, a lot of jargon and domain experience is needed to understand what is being said. Otherwise, the statements seem like obvious no-brainers or quite meaningless. An immediate example is the Agile Manifesto itself: without a lot of prior knowledge, it isn’t clear what is so special about it and how it could have kick-started the current movement. The challenge is thus to translate it into a form that draws on a more common set of knowledge. In addition to facilitating understanding, communication, use and development of Agile in practice, an underlying goal of the thesis is to demonstrate the value

of paying attention to organizations in general and methodologies in particular for the academic fields interested in understanding science and technology (PSTS). There are several ways that this thesis is valuable for PSTS. First, the goal of understanding software technology follows from the general aim of PSTS to understand technology. This relation is represented in figure 1.3 (the orange levels represent the scope of this thesis).

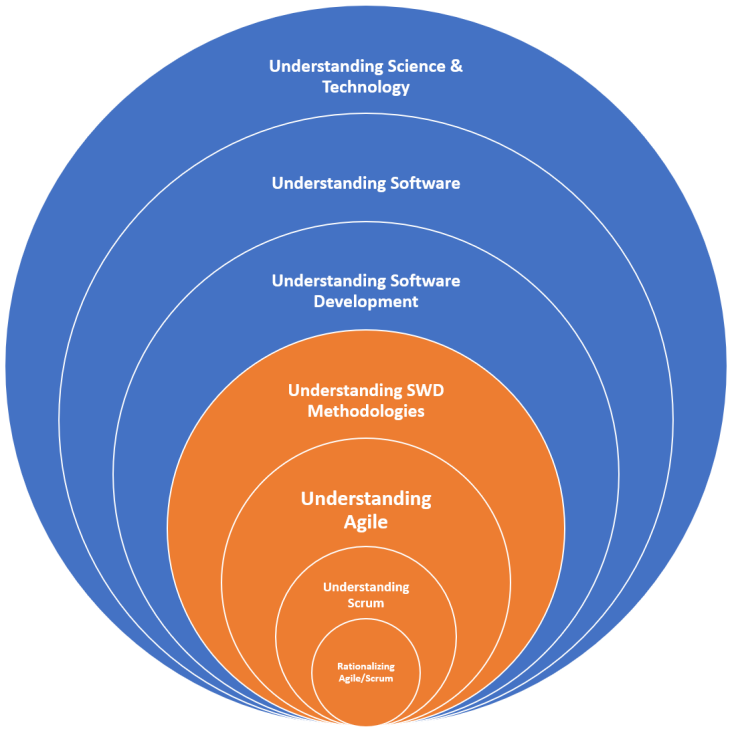


Figure 1.9: How the goals of this thesis follow from general PSTS aims

Understanding the dominant rationale in the software domain can aid understanding but also the communication from PSTS with the SWD domain. For example, sometime ethicists want to make normative recommendations about software. By having a specific understanding of Agile, both the current decisions and developments in the SWD can be understood better, and the recommendations can be put in a form that is tailored to make sense with respect to the Agile way of thinking. This potentially increases both the credibility and the effectiveness of the recommendations. On top of this, understanding Agile could be reflexively useful for the organizations and methods of the PSTS fields themselves. That is if some of the ideas that motivate Agile also make sense for the academic organizations that make up PSTS, some of the ideas may be adopted. The 'Data Sprints' in controversy mapping (Vertesi and Ribes 2019, p472) are an example of this. The academic relevance of this thesis to PSTS is discussed further in section 5.2.1.

1.6 Defining the terms

As noted above by Conboy, some of the very basic concepts in software engineering are surrounded by ambiguity. Gruner adds (Gruner 2011, p.295):

> Typically there will be a software development ‘process’ in a software engineering project, during which a software ‘system’ is being produced in accordance with a corresponding ‘model’. However, all those three concepts already have a long semantic history in the terminology of various sciences, such that software engineering has simply ‘inherited’ these terms and continued their usage without much language-analytic reflection about their historical semantics. Here I can also see aopportunity for interesting philosophical work in the future, such as to find out which aspects of the historical semantics of ‘system’, ‘process’ and ‘model’ have been preserved in the terminology of software engineering, and which aspects of their semantics have been modified or even lost.

For the purposes of this thesis, no particular distinction is made between IT, ICT or Software. Similarly, the thing that an IT project is producing is referred to interchangeably as a software system, application or software product.
Particularly tricky terms in the context of Agile are method and methodology. In general usage, these terms imply a specific procedure with defined steps. However, almost none of the methods that fall under the Agile umbrella specify specific steps about how code should be written. Probably the one that comes closest is eXtreme Programming with the Test-Driven Development Practice, which specified that a test should be written before the code that makes the test pass. Coding best practices (called patterns), bad practices (called smells or anti-patterns), system architecture and similar technical issues are not directly addressed by this thesis.

1.7 Roadmap to the thesis

The remainder of this thesis is structured as follows: Chapter 2 addresses the approach to answering the research questions and answers the first through definition, Chapter 3 presents the results to the second two sub research questions and, Chapter 4 answers the last sub question en the main research question by constructing the rationalization based adapting on the landscape metaphor. Finally Chapter 5 summarizes the results, explores some generalizations and discussion points and concludes with possible future research avenues.

Chapter 2

Research Methodology

In the previous chapter, the ‘Why’ and ‘What’ of this thesis have been addressed. This chapter is concerned with the ‘How’.

First, the research process and methods are explained. Next, for each sub research question the research approach is specified. In the section 2.2.1 the first research question is answered, which is:

1. What are the core documents of Agile?

Finally, the validity of the research methodology is discussed.

2.1 Research Methods and Process

The main research methods of this thesis are reflection and analysis and synthesis. The primary source of input for this thesis is my own experience in the software development domain. In this thesis, I am not a neutral, objective observer. Instead, the experience with Agile and software development generally that I already held at the start of the project forms the basis on which decisions are made and as the main input of ‘data’. This aspect of the research can be characterized as applied philosophy, namely: critical philosophical thinking applied to past experience. This has some similarities with auto-ethnography. While some argue for the merits of auto-ethnography as a research method within organizational research (Boyle and Parry 2007) its validity is controversial (Méndez 2013). The issue of validity is addressed further in section 2.3.

2.1.1 Research Process

A post-hoc characterization of the first phase of the research process is schematized in figure 2.1.

The second phase of the research process consists of the development and theoretic amelioration of the perception of software development methodology arrived at in the first phase. This second phase involves articulation, reflection and synthesis. This phase is schematized in figure 2.2.

Each of the research methods is now succinctly addressed.

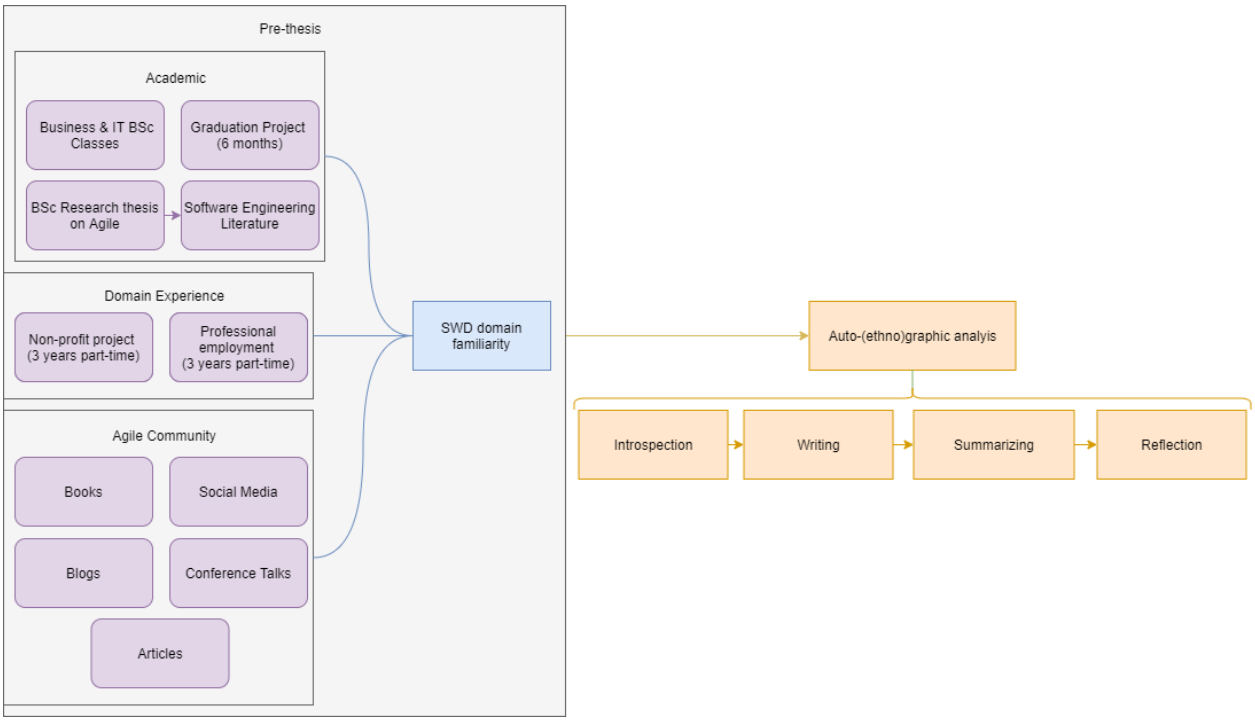


Figure 2.1: A schematic representation of the first phase of the research process

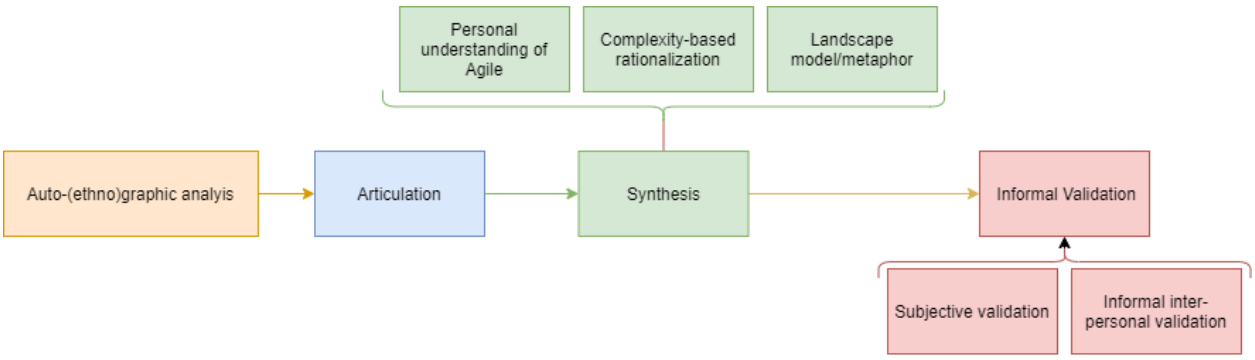


Figure 2.2: A schematic representation of the second phase of the research process

2.1.2 Auto-ethnography

(Hamdan 2012) mentions four main steps to auto-ethnography: the regressive, the progressive, the analytical, and the synthetical. These steps can be recognized in the first phase of the overall research process. In figure 2.1 the experiential basis is also represented. I was practically born into the software domain: my mother was the first female Computer Science graduate from the University of Twente, my father worked on JPEG and LaTeXcomponent Babel, my brother studies Computer Science and I have a Bachelors in Business Information Technology. My Bachelor's thesis was also concerned with Agile (specifically Behavior Driven Design and Model-Driven Development), and I have worked professionally and voluntarily in software teams and organizations since 2016. I have been keeping track of the Agile Community through various media since my bachelor's research. The accumulated familiarity with the ideas and concepts used there were extracted through writing, discussion, reflection and analysis. Of course, such a process results in a very particular perception of both the Software Engineering literature and the Agile Community

and gives no guarantee of general representativeness. But, as will be explained further in section 2.3, for the purposes of this thesis no *guarantee* of general representativeness is needed.

2.1.3 Articulation

'Articulation' here means taking something that was vague or diffuse and finding a way to state it in a sharper, clearer and more explicit way. The goal is to draw out what Agile is actually is claiming about the nature of Software Development. In this sense, I see the core of this thesis as restating the insights of Agile into a form which can be more readily understood.

2.1.4 Reflection

Through reflection, distance is taken from a subject which opens up a new perspective. Stepping to a meta-level allows seeing new connections and can lead to insights. The distance helps to distinguish the particulars from the general patterns. One way to think about this, taken from Deborah Johnson (Johnson 2007), is as the 'big idea' and the 'small idea'. In the original paper, Johnson uses these notions to differentiate the small idea of democracy, namely that citizens should have control over the factors that influence their lives, from the big idea of democracy, being the particular instantiation of democracy in a country, such as a first-past-the-post voting system in a parliamentary monarchy. By separating the two, the big idea can be criticized while preserving the small idea. That is, particular policies and decisions can be rejected without abandoning democracy as a whole. However, the small ideas are usually implicit: we encounter most ideas in their 'big' form and it takes critical analysis to tease out the small idea. In these terms, Agile is usually encountered in its 'big' form. In the particular instantiations of practice and in the descriptive accounts of the SE and informal literature the small idea of Agile is not immediately obvious.

When the foundations under Agile have been revealed through articulation, reflection can help isolate the core principles and expose its underlying rationale. The rationalization of Agile should thus lead to the 'small ideas' of Agile.

2.1.5 Synthesis

The different elements produced in the previous steps are combined, arriving at a new narrative as a result. This narrative serves the purpose of an analytical tool and a lens through which to view Agile. The way that this is achieved is by putting the reader, who is potentially unfamiliar with the dynamics of software development, in a situation that allows them to tap into knowledge and intuitions they already possess. A way to do this is through a metaphor. A metaphor makes a connection between an unknown situation and maps it onto a familiar one, which allows the transfer of existing knowledge and intuitions into the new subject (Pribram 1990). The metaphor used is one that taps into a fundamental familiarity that all embodied beings are expected to have: spatially navigating a landscape. This model of an agent on a landscape who tries to reach the highest peak is taken from Michael Weisberg and Ryan Muldoon (Weisberg and Muldoon 2009). The original model is adapted towards the SWD context by adjusting the way that the landscape behaves to reflect the dynamics of software development as discussed in Chapter 3. The insights

about software development generally and Agile specifically are integrated into the behaviour of the landscape, after which software development methodologies can be reinterpreted as strategies for dealing with that landscape. In light of this image of the landscape, it becomes clear how different approaches to software development make sense in particular circumstances. The rationalization of Agile presented in this thesis is thus in terms of a rational navigation approach to the SWD landscape.

2.2 Research Approach per Question

Table 2.2 provides an overview of how the methods and research questions match up.

RQ		Approach	Source Selection
1	What are the core documents of Agile?	Agile survey, Auto-Ethnography	Estimated Influence
2	What justification for the use and effects of Agile do the core documents offer?	Text-analysis	
3	What justification for the use and effect of Agile are used in the Agile community?	Auto-ethnography	Subjective
4	What are the limits of these current rationalizations?	Analysis	
5	How can the intuitions underlying Agile be better rationalized using notions from Philosophy of Science and Complexity Science?	Auto-ethnography, Reflection, Analysis, Synthesis	

2.2.1 Selection of Sources

With respect to the first research question, the core documents of Agile are defined here based on the mentioned auto-ethnographical basis. First, the Agile Manifesto stands as an eye in the storm of discussion of what is Agile. If the Manifesto is not Agile than what else is? Therefore the Manifesto is defined by this thesis as one of the core documents of Agile. In my personal experience, more people recognize the term Scrum than the term Agile. Looking at the market share of Scrum within the Agile domain (VersionOne 2019), in practice Agile often equals Scrum. Therefore, the Scrum Guide is the second document which is defined as a core Agile document for the purposes of this thesis.

With respect to the third research question, 'What justification for the use and effect of Agile are used in the Agile community?', again personal experience is the deciding factor. Within their review, Dingsøyr et al. offer an overview of theoretical perspectives (in order of frequency) for when Agile *is* rationalized in the software engineering literature (Dingsøyr et al. 2012, p.1217), pictured in figure 3.1

All of these perspectives can be investigated for their explanatory merits. This thesis limits itself to the complexity-based rationalization observed to be used in the Agile community. The choice for

Theoretical perspectives used in agile research.

Theoretical Perspective	Number of Articles	Article(s)
Knowledge management	9	Dingsøyr and Hanssen (2002) Holz and Maurer (2002) Sena and Shan (2002) Doran (2004) Fang et al. (2004) Bellini et al. (2005) Crawford et al. (2006) Salazar-Torres et al. (2008) Chan and Thong (2009)
Personality	6	Sfetsos et al. (2006) Choi et al. (2008) Layman et al. (2008) Sfetsos et al. (2009) Acuna et al. (2009) Hannay et al. (2010)
Organizational learning	1	Holz and Maurer (2002)
Double loop learning	1	McAvoy and Butler (2007)
Triple-loop learning	1	McAvoy and Butler (2007)
Complex adaptive systems	2	Meso and Jain (2006) Socha and Walter (2006)
Social facilitation	2	Arisholm et al. (2007) Balijepally et al. (2009)
Adaptive Structuration theory	1	Cao et al. (2009)
Chaos theory	1	Levardy and Browning (2009)
Complexity theory	1	Falessi et al. (2010)
Coordination theory	1	Pikkarainen et al. (2008)
Distributed cognition	1	Sharp and Robinson (2008)
Evolutionary theory of knowledge	1	Northover et al. (2006)
Fuzzy set theory	1	Mafakheri et al. (2008)
Game theory	1	Hazzan and Dubinsky (2005)
Graph theory	1	Zimmer (2003)
Socio technical	1	Johannessen and Ellingsen (2009)
Teamwork model	1	Moe et al. (2010)
Theory of diagnosis	1	Trinidad et al. (2008)

Figure 2.3: Theoretical perspectives used in agile research according to (Dingsøyr et al. 2012, p.1217)

this focus is based on previous familiarity, personal interest and expected philosophical depth of the complexity based rationalization.

Regarding the final research question, the form chosen for the rationalization is an adaptation of the Epistemic Landscape model by (Weisberg and Muldoon 2009). This choice was similarly not a selection among multiple choices based on objective criteria. By happenstance, my supervisor was already familiar with the paper and as soon as we started discussing it the explanatory potential became evident to us.

2.3 Validity

From a traditional scientific perspective on validity of research methods should be objective, repeatable and reliable (Cronin, Ryan, and Coughlan 2008). However, the aims and methods of this thesis do not align with this scientific perspective. This thesis does not aim to be exhaustive,

representative or ‘right’. No illusions are made of presenting the be-all-end-all answer to the question concerning Agile. rather, the attitude is ”what is a *useful way to think* about Agile?” However, ’useful’ is not a universal property: it is related to a particular actor pursuing a particular goal. In the case of this thesis, one of those actors are the collective PSTS fields. One presupposition of this thesis is that within those PSTS fields there is insufficient attention and knowledge about Software Development Methodologies and, moreover, that an increased appreciation for this topic holds potential benefit. ’What is a useful way to think?’ thus also means ’what do PSTS scholars need to know about the SWD domain generally and Agile specifically to:

- Grasp the underlying principles and assumption behind Agile
- Understand the rationale behind developments and decisions in the SWD domain
- Tailor their (normative) messages toward the SWD domain to connect with and fit in with the mentality of that domain

Another actor is a person confronted with (and perhaps confused by) Agile. For this actor, a useful way to think is a way that helps make sense of their situation. With other words what is the underlying logic of Agile? However, I hold no illusion that there is one ‘true’ logic to be found in or behind Agile. A notion that helps out here is empirical adequacy. With the notion of empirical adequacy, this thesis takes inspiration from Constructive Empiricism. Within Philosophy of Science, Empirical Adequacy has a very precise meaning about the nature of the relationship of theories and models to reality. As Van Fraassen defines it: “A theory is empirically adequate exactly if what it says about the observable things and events in the world is true” (Van Fraassen 1980, p.12) and thus for unobservable phenomena, the theory or model merely gives a good and coherent description of the structure of reality. This notion is one way of dealing with the problems that arise when it is assumed that the value of models and theories lie in the direct representational link with the structure of reality. Any dissimilarity between the theory and reality would then mean the theory has no value. In practice, it can be observed that models help to solve problems, even if they are not completely accurate descriptions of reality. In this same way, the description and rationalization of Agile presented in this thesis are not expected to be categorically accurate to the point that no one could dispute them. Instead, they are expected to be a “a good and coherent description” that helps to solve problems and bestows the reader with a sense of clarity. This attribute of empirical adequacy can be tested by testing the account constructed in this thesis in practice and seeing if it makes sense to practitioners and helps to solve problems.

The first test of this kind is subjective: i.e. does it make sense to me? This is represented in the research process figure 2.1 as subjective validation. The second test that has been performed is through conversations about the proposed rationalization with industry professionals. Additionally, in the context of this research, I visited a conference organized by the Dutch society for project managers ’BPUG’ as well as a workshop about Agile where I could observe rhetoric and justifications in practice and informally test out some of the ideas of this thesis. This is represented as ‘informal inter-personal validation’. If the account ’makes sense’ to the reader, if it evokes a feeling of clarity, then this is a good indicator (but no guarantee) that it is indeed a useful account. This is why moving away from direct representationalism does not immediately imply that ’anything goes’: only a specific subset of possible accounts help to ’make sense’ of the ideas surrounding software development methodology. The validation of the empirical adequacy of the proposed

rationalization can be taken up in a more systematic way in future research.

2.4 A guide, a translator, an archaeologist, a midwife and a gadfly

To summarize, this thesis takes the roles of a guide, a translator, an archaeologist, a midwife and a gadfly. It guides those with an interest in science or technology who are confronted with the software development domain to what I deem relevant area's of interest, offering a particular route, although others are possible. It translates the necessary terms into more accessible language and points out features to watch out for and remember. It acts as an archaeologist in taking a very close and careful look at the ideas and notions that make up the foundations of Agile. And, taking up the Socratic torch, while not being responsible for their origin, like the midwife it helps bring forth helpful ideas and like the gadfly it aims to sting enough to provoke a reaction.

Chapter 3

Current Rationalizations

In this chapter, the results of the second three research questions are presented. These questions are:

3. What justification for the use and effects of Agile do the core documents offer?
4. What justification for the use and effect of Agile are used in the Agile community?
5. What are the limits of the current rationalizations?

The structure of the chapter mirrors the order of the questions.

3.1 Justifications in the core documents

The reasoning for the selection of the Agile Manifesto and the Scrum Guide as the core documents to be analysed can be found in section 2.2.1

3.1.1 The Agile Manifesto

The Agile Manifesto consists of only 7 sentences containing 68 words. In full, it states (Beck et al. 2001):

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

Accompanying the manifesto are 12 principles, which read:

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

The manifesto thus puts forth a number of values but does not define a discrete or definitive methodology. This leaves a lot of room for interpretation if any particular practice, team or company 'is Agile' or not. Rather than going into specific scenarios or experiences, the manifesto points in the direction of a particular approach or perspective on developing software. In the colloquial sense of the word philosophy, Agile might thus (better) be called a software development 'philosophy' than a methodology. In one sense, the question 'What is Agile?' can thus be answered as a set of values concerning how software should be developed. A methodology or a specific method may be based on these values or conform to these values, but the manifesto itself is not a methodology.

Notably, the manifesto offers no outright justification or support for why these values and principles are important and should be followed. Of course, it is not unusual for a manifesto to be polemical and make bold claims and declarations. Nevertheless, a particular empirical position is implied in the wording of the manifesto. This (implied) justification is found in the first sentences: "We are

uncovering better ways of developing software *by doing it and helping others do it.*" The 'better ways of developing software' are thus not uncovered by thinking and reflecting about developing software, or by building a model or by applying theories but by "doing it and helping others do it". this grounding in experience is re-emphasized at the beginning of the next line of the manifesto: "*through this work* we have come to value." Rather than deducing these better ways of developing software top-down from abstract theory, the claims put forth have been formulated bottom-up from the trial-and-error know-how of seasoned professionals. The implication is that if the reader also develops software and/or helps others develop software they would (necessarily) agree with the authors since they would experience too that these values and principles make sense. The twenty thousand signatures seem to support this assertion.

3.1.2 The Scrum Guide

A summary of the guide can be found in appendix A. In a section called 'Scrum Theory' the early (2010) version of the Scrum Guide states (Guide, Schwaber, and Sutherland 2010, p.3): "Scrum is a framework for developing complex products and systems that is grounded in empirical process control theory". The specific reference given with respect to empirical process control theory is (Schuler 1996). This thesis focusses on the 'complex' part of this justification. In the later (2017) version (Sutherland and Schwaber 2017) this changes to "Complex Adaptive Systems"
Complexity is further mentioned in the following instances in the 2017 version:

> "As technology, market, and environmental complexities and their interactions have rapidly increased, Scrum's utility in dealing with complexity is proven daily." (p.4)
> "When the words "develop" and "development" are used in the Scrum Guide, they refer to complex work, such as those types identified above." (p.4)
> "The Scrum Team has proven itself to be increasingly effective for all the earlier stated uses, and any complex work."(p.6)
> "Large Development Teams generate too much complexity for an empirical process to be useful."(p.7)
> "When a Sprint's horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase."(p.9)
> "The Daily Scrum is held at the same time and place each day to reduce complexity."(p.12)
> "In complex environments, what will happen is unknown. Only what has already happened may be used for forward-looking decision-making."(p.16)

So the guide repeatedly states *that* it is an effective approach for dealing with complexity but offers no explanation of *how* and *why* this is the case. Neither is their conception of complexity defined, which implies that the Guide assumes familiarity with complexity science concepts.

3.2 Justifications in the Agile Community

Personal experience[1] shows that in practice the justification of Agile mirrors that of the manifesto: either none is given or the use of Agile is justified on its assumed empirical merits ('It just works, don't question it'). The most common justification for the use of Agile is no justification. This comes in the forms "everybody's doing it" and "Management decided we are doing Scrum now".
Arguably, the (perceived) failure of Waterfall feeds the need and readiness to try anything else. In the history section accompanying the Agile Manifesto Waterfall is characterized as a "document driven, heavyweight software development process" (Beck et al. 2001). According to Conboy and Fitzgerald: "It is now widely accepted that these methodologies are unsuccessful and unpopular due to their increasingly bureaucratic nature. Many researchers and practitioners are calling for these heavyweight methodologies to be replaced by agile methods" (Conboy and Fitzgerald 2004, p.105). Since 1994 each year the Standish Group has published the 'Chaos Report', surveying IT-project success. In the 1994 Chaos report, only 16% of IT-projects could be counted as successful under the definition of the Standish Group. Agile proponents often use these figures to argue for the failure of Waterfall and the need for adopting Agile. When the manifesto was first published online, it was possible to add one's name to the list of signatories, as a sign of agreeing with the content of the manifesto. Up until the possibility to publicly underwrite the manifesto was removed in 2016, the manifesto received more than 20.000 signatures (Nyce 2017). The manifesto had thus not only managed to find the common ground between the dozens of alternative software development methodologies that had emerged in the '90s but also managed to capture a sentiment that struck a chord among a significant amount of the software development community at the time.

A smaller contingent justifies the use of Agile through vague notions which are treated as ends in themselves, such as 'flexibility' or 'automous teams'. Within the Agile Community, a host of different theories make the rounds. Within their review, Dingsøyr et al. offer an overview of theoretical perspectives (in order of frequency) for when Agile *is* rationalized in the software engineering literature. (Dingsøyr et al. 2012, p.1217), pictured in figure 3.1.

Arguably, all of these perspectives can be investigated for their explanatory merits. However this thesis limits itself to the complexity-based rationalization observed to be used in the Agile community. One example of this is the simple complexity estimation method given by Liz Keogh (Keogh 2013).

1. Just about everyone in the world has done this.
2. Lots of people have done this, including someone on our team.
3. Someone in our company has done this, or we have access to expertise.
4. Someone in the world did this, but not in our organization (and probably at a competitor).
5. Nobody in the world has ever done this before.

Additionally, two complexity-based rationalizations have come to lead a life of their own within

1. No exhaustive systematic overview of all existing justifications in the community is given here. The method and source selection have been specified in chapter 2.

Theoretical perspectives used in agile research.

Theoretical Perspective	Number of Articles	Article(s)
Knowledge management	9	Dingsøyr and Hanssen (2002) Holz and Maurer (2002) Sena and Shan (2002) Doran (2004) Fang et al. (2004) Bellini et al. (2005) Crawford et al. (2006) Salazar-Torres et al. (2008) Chan and Thong (2009)
Personality	6	Sfetsos et al. (2006) Choi et al. (2008) Layman et al. (2008) Sfetsos et al. (2009) Acuna et al. (2009) Hannay et al. (2010)
Organizational learning	1	Holz and Maurer (2002)
Double loop learning	1	McAvoy and Butler (2007)
Triple-loop learning	1	McAvoy and Butler (2007)
Complex adaptive systems	2	Meso and Jain (2006) Socha and Walter (2006)
Social facilitation	2	Arisholm et al. (2007) Balijepally et al. (2009)
Adaptive Structuration theory	1	Cao et al. (2009)
Chaos theory	1	Levardy and Browning (2009)
Complexity theory	1	Falessi et al. (2010)
Coordination theory	1	Pikkarainen et al. (2008)
Distributed cognition	1	Sharp and Robinson (2008)
Evolutionary theory of knowledge	1	Northover et al. (2006)
Fuzzy set theory	1	Mafakheri et al. (2008)
Game theory	1	Hazzan and Dubinsky (2005)
Graph theory	1	Zimmer (2003)
Socio technical	1	Johannessen and Ellingsen (2009)
Teamwork model	1	Moe et al. (2010)
Theory of diagnosis	1	Trinidad et al. (2008)

Figure 3.1: Theoretical perspectives used in agile research according to (Dingsøyr et al. 2012, p.1217)

the community: the Stacey Matrix and the Cynefin Framework. Therefore these two are now discussed.

3.2.1 Stacey Matrix

The diagram pictured in figure 3.2, known as the Stacey Diagram or the Stacy (Complexity) Matrix, shows up innumerable times in different articles, blogs and presentations about (project) management generally and as a justification for Agile specifically:

Despite its apparent ubiquity, finding the original definition of the matrix is quite tricky. Sometimes a specific citation is accompanying the diagram, but these do not match. Furthermore, in the current editions of Stacey's books the diagrams cannot be found. Despite Stacey's best efforts, it seems the diagram has taken on a life of its own. In the 2012 edition of "Tools and Techniques of Leadership and Management", Stacey explains that he originally introduced the diagram in the 1996 second edition of the "Complexity and Creativity in Organizations" textbook: "The book introduced

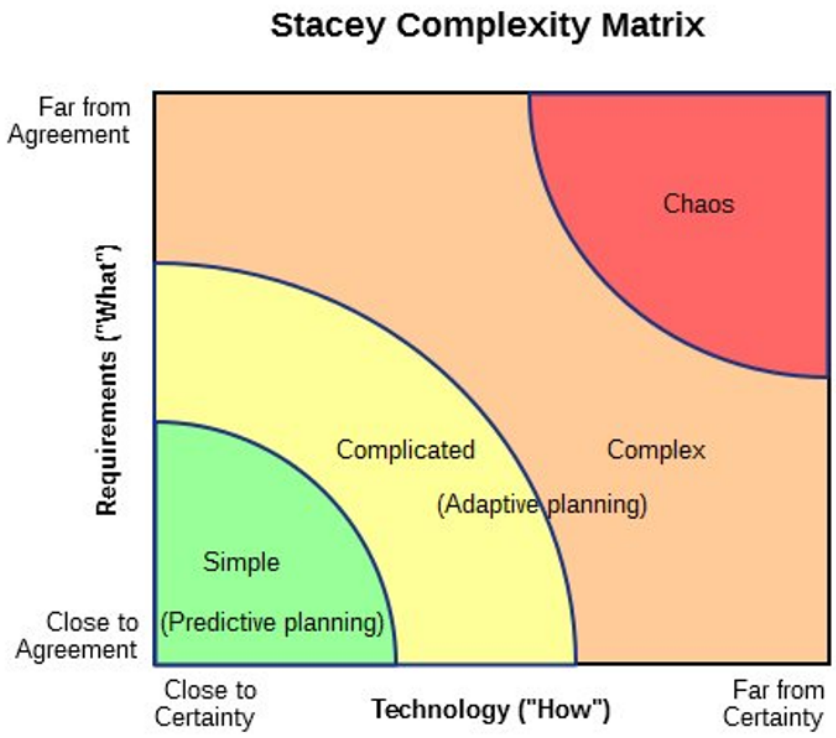


Figure 3.2: The Stacey Framework

a diagram which I have since come to regret. The y-axis of this diagram indicated movement from situations close to certainty to situations far from certainty, while the x-axis indicated movement from situations in which people were close to agreement with each other to situations in which they were far from agreement with each other" (Stacey 2012, p.151). Between these axes, Stacey defined 3 main zones: the ordered zone, the disordered zone and the zone of complexity.

The ordered zone is divided further depending on a high degree of certainty or a high degree of agreement. If both are high traditional management tools and techniques apply, such as planning specific paths of action to achieve outcomes and using monitoring as a form of control. According to Stacey, the majority of management theory and literature applies to this kind of situation. If certainty is high but agreement is low, 'political' approaches are called for, such as consensus-building. If on the other hand agreement is high but certainty is low, executing a predetermined plan will not work. The goal state can be defined but the path to the goal cannot be preset. In the disordered (or chaos) zone, only avoidance is offered as a strategy. The key difference cited by Stacey is that in the ordered zone past experience/behaviour of the system is a useful resource for forecasting the future state/behaviour while outside of it this is not the case. Confusingly, in Zimmerman's summary, 'certainty' is defined as "unique or at least new to the decision-makers" in addition to "extrapolating from past experience is not a good method to predict outcomes", mixing a subjective factor (has the decision-maker encountered the situation before?) with an objective attribute of the system/situation (are past experiences a good method for predicting outcomes?)The zone of complexity (or Edge of Chaos) is seen as a situation where traditional management approaches are ineffective or even counter-productive, but at the same time it is "the only dynamics in which the new and the creative can emerge" (ibid., p.151). Stacey describes

this zone as having the paradoxical quality of mixing predictability and unpredictability, where regularity and irregularity exist at the same time.

something of a shift can be discerned in Stacey's work around the mid-'90s, following the collaboration with his PhD students Patricia Shaw and Doug Griffin. While in the first phase Stacey's approach was to directly apply chaos theory and complexity science concepts to organizations and management, later he shifted to treating the complexity sciences more as a source domain for analogies, which need a significant deal of interpretation to be applicable to attributes of (self-)conscious, emotional, spontaneous, reflective human beings that have a degree of control over their actions. While in the first phase Stacey conceptualized organizations as complex adaptive systems, in the later phase he characterized them as 'complex responsive processes of relating', drawing on George Mead, Norbert Elias, Hans Joas, William James, John Dewey. The concept of emergence stemming for self-organizing processes of interactions becomes key: "It is in these local responsive processes that there emerge population-wide patterns of activity, culture and habitus. Organisational life is thought of as the game people are invested in and organising processes are understood to be the ordinary politics of everyday life."

So why the regret on Stacey's part for introducing this diagram? Well: "presenting things in this way suggests that managers can decide which kind of situation they are in and then choose the appropriate tools" and "I no longer use the diagram because it is simply interpreted in a way that sustains the dominant discourse while using the alternative jargon of complexity" (Ibid).

3.2.2 Cynefin Framework

The Cynefin sense-making framework was developed between 1999 and 2003 by David Snowden and Cynthia Kurtz (Fierro, Putino, and Tirone 2018, p.533). Cynefin is a welsh word meaning habitat or environment. The framework is made up of five domains: obvious (called simple in early versions), complicated, complex, chaos and disorder and can be seen in figure 3.3.

The domains represent different decision-making contexts. Is not always clear if these contexts are determined by attributes of a system (the domain is a property of the world), by the ability of the decision-maker (the domain is a state of the agent) or a mix (the domain is determined by the interaction between system attributes and agent capability). The point of the domains is that they describe what strategies are appropriate for operating in such a situation. In the figure, under the name of the domain the different strategies are specified (sense, categorize, respond, analyse, probe, and act). Additionally, for each domain, it is specified what kind of knowledge is possible (Best, good practice, emergent, and novel).
For example, an often-used example of a situation that would be categorized as obvious in the Cynefin framework is tying your shoes. You *sense* what is going on and can immediacy tell what is going on. You *categorise* which of the available steps should execute and you *respond* by executing said step. Cause and effect are directly related and immediately perceivable in such a situation and there is a *best practice* that reliably results in the desired outcomes.
An example of a complicated system would be a mechanical car engine (i.e. not one of the modern ones which are digitally managed and locked off for the owner of the car). A complicated system is made up of several (static) sub-components (e.g. the electrical, hydraulic, gas, oil, air systems

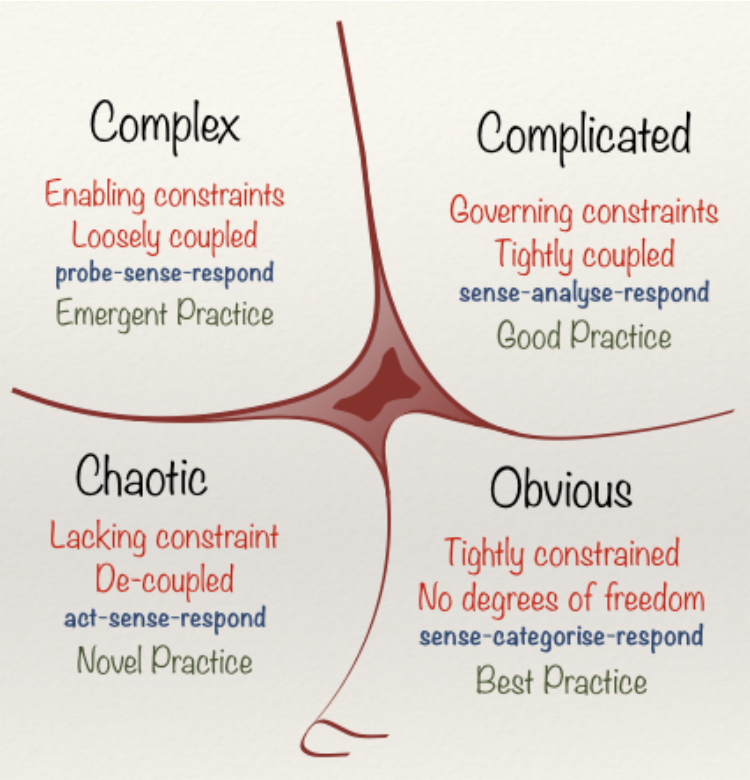


Figure 3.3: The June 1 2014 version of the Cynefin Framework

in a car). If the motor stops working it is not immediately obvious what the cause is, but through *analysis* the cause can be determined. If you do not possess the necessary knowledge for the analysis you turn to an expert, in this case a mechanic. Often there are multiple ways to solve the problem, i.e. *good practices*, depending on the circumstances or personal preference, but not one 'silver bullit' solution which always works reliably.

The domains that follow have a key difference. Both Obvious and Complicated are called Ordered Domains in Cynefin. In contrast, the Complex and Chaos domains are Unordered. This means that cause and effect are not linearly related and can therefore not directly be determined through analysis. The climate is an example of a complex, interconnected system. The climate is made up of many subsystems: Geosphere (hydrosphere, atmosphere, cryosphere, lithosphere) & Biosphere. Here, all subsystems are subject to constant change and interaction. If you influence one system it affects the other systems in unpredictable ways. Therefore what worked in one situation is not guaranteed to work in another similar situation. There is no way to measure a complex without affecting it in some way. An example is an election where conducting polls, meant to uncover cause (voter preferences) and effect (who is elected) influences how people vote and only when the election has taken place (and thus the result is fixed and can no longer be influenced by efforts of analysis) can explanations be arrived at (although full understanding of what caused someone to be elected might be unreachable due to practical constraints). Therefore the strategy specified by Cynefin is to *probe* the system through safe-to-fail experiments and *sense* the unpredictable effects. What qualifies an experiment as safe-to-fail is that you can *respond* by amplifying or dampening its effects as needed. The experiments can result in *Emergent Practices* such as the discovery of constraints in the system.

With this framework, some of the described phenomena in the software development domain can be explained. In such an analysis it is assumed that most of the activities in the manufacturing industry take place in the Complicated domain. The resulting practices, knowledge and approaches (such as Waterfall) are then mostly applicable to complicated problems. Secondly, it is claimed that developing software is often situated in the complex domain. Through this contrast of domains, the problems with applying manufacturing approaches to software start to become clear.

When an decision making actor assumes to be in the complicated domain they can adopt the corresponding epistemological strategy (Sense-Analyze-Respond). They might then assume that project failure is due to not analysing the situation enough instead of recognizing the mistake in recognizing the epistemic conditions. They might then spend more time in the design phase, thinking that if enough resources are invested here a satisfactory product must be the result. This cycle can repeat a few times until the (percieved) situation in ’90s with large IT project failure is the result. Thus, within the Cynefin framework the lack of success of the plan-driven approaches can be interpreted as the application of a proven strategy in the wrong domain, namely of Complicated strategy into the Complex domain.

Likewise, the success of Agile can be connected to the priority it lays on building prototypes of the software as soon as possible and seeing how stakeholders respond, i.e. experimenting/probing – sensing and responding. That is, the success of Agile development methodologies can be related back to the bringing into accordance with the domain in which software development is located and the strategy that works in that domain. Additionally, the resistance to Agile can be related to a mixing of strategies the other way around, namely of Complex strategies into Complicated or even Obvious situations. As Agile gained steam and popularity, its success in Complex software development projects led it to be seen as a Best Practice (which aren’t possible in the Complex domain) and as a panacea for all software projects, while it is not guaranteed at all that all software projects are complex. Smaller projects or projects where the functionality is very straightforward may be Complicated or even Obvious, and in those cases the Complex strategy doesn’t make sense and feels like a detour or unnecessarily indirect way of organizing the work and processes

3.3 Limits of the Current Rationalizations

Different ‘tools’ are useful for different jobs. The simple complexity score from 1-5 by Liz Keogh mentioned in the previous chapter can be understood and applied instantly. However, this ease of use comes at a cost. This 1-5 scale helps to compare phenomena to each other linearly, but otherwise gives limited information about the dynamics of complexity. With other words, it has limited explanatory power and a limited scope of the number of situations it can help make sense of.

Regarding the Stacey model, as discussed Ralph Stacey did all in his power to distance himself from the unfortunate way that the model that bears his name is being used. It should therefore not

be seen as an accurate representation of his thinking. Now, crucially, in the case of the Cynefin framework, the relative simplicity of the framework is the point.
Dave Snowden explains that he actively worked to reduce the intricacy of the model to the level that it can be sketched on a napkin. The idea behind this 'napkin test' is to prevent dependency on "esoteric knowledge" and to keep it imaginable (since people supposedly are only able to hold five concepts in their mind at the same time) (Snowden 2015). Keeping the framework easily drawable (and thus 2D) both promotes the easy adoption and use of the model and discourages the misappropriation by those looking for "accreditation revenue and consultancy dependency" (ibid.). Additionally, both models come from distinct projects with their own particular aims.
For this thesis, different considerations and priorities are at play and therefore different choices are made.

In addition to the limited nuance, the Cynefin framework by itself does not integrate an intuitive understanding of complexity. In the full body of work of Snowden, this is, of course, no problem, but in how the image is used in the community, knowledge of complexity science concepts and software development experience are needed to apply it to Agile. By finding a different form, this context and knowledge dependency can be eliminated

3.4 Chapter Summary

The second sub research question of this thesis is "What justification for the use and effects of Agile do the core documents offer?". For the Agile Manifesto this question has been answered with: an implied empirical justification but no theoretical account. Regarding the Scrum Guide, the early (2010) version mentions empirical process control theory (specifically (Schuler 1996)) and the later (2017) version mentions Complex Adaptive Systems.
Following up with the third research question, this complexity-based rationalization has also been observed in the Agile community, and two particular forms, the Stacey Matrix and the Cynefin Framework have been evaluated. The fourth sub research question, "What are the limits of the current rationalizations?" has been answered with: the current rationalizations are dependant on knowledge of complexity science and software development experience.
In the following chapter, the complexity-based rationalization is transformed in such a way that it makes use of basic intuitions and becomes independent of knowledge of complexity science and software development experience

Chapter 4

Proposed Rationalization

In this chapter, the results of the final sub research question are presented. This question is:

5. How can the intuitions underlying Agile be better rationalized using notions from Philosophy of Science and Complexity Science?

The structure of the chapter again mirrors the order of the questions. First, the basic elements are introduced through the example of gradient descent in neural networks.

4.1 Basic elements of the landscape schema

If the performance of a system can be calculated, this score can be plotted against the initial conditions of that system. An example of this is how gradient descent in neural networks can be visually explained (Kathuria 2018)[1]. Imagine an impossibly simple neural network with two nodes, whose task it is to classify if an image is an apple or not. The weight of a node determines if it activates or not. With the right weights, i.e. all nodes have a correct number assigned to them, for a particular input (an image) the network will give the right answer (it is an apple or not). For any particular combinations of weights, a loss value can be calculated. Together, the 2 input parameters (the weight of the two nodes) and the output (the loss value) make up a loss function. The best performance of the system happens at the lowest loss value. That is, the system is the best at recognizing apples if the weights result in the lowest value of the loss function. This function can be plotted as a surface, see image 4.1.

In reality, neural networks would involve millions or a billion parameters (ibid.), but we can't visualize a billion dimensions. At the start, it isn't known which weights have the lowest loss value so random weights are assigned. This is point A in figure 4.1. The global minimum of the function is represented by point B. The task is to adjust the weights in such a way that point B is reached. However, it is not possible to go directly to point B. It is only possible to calculate the gradient around point A and move downward. If very small steps are taken, it takes a very long time to reach B. If too large steps are taken, the goal can be overshot, see image 4.2.

A more complicated surface has local minima. A search strategy that simply moves down can get trapped in such a local minimum, see image 4.3. In a landscape like the one pictured in image 4.4, a very sophisticated search algorithm is needed to find the global minimum. And remember

1. All images in this subsection are taken from this page

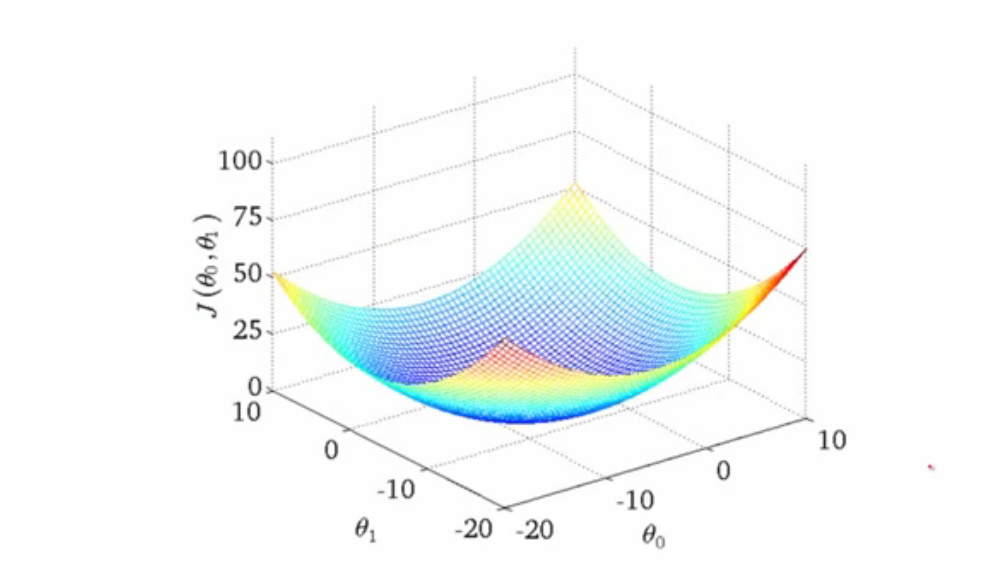


Figure 4.1: A loss landscape

that the ‘true’ landscape for a neural network is not 3 dimensional but has millions or billions of dimensions.

The basic schema of (1) input parameters, (2) an associated ‘succes’ score which together plot a surface, (3)starting at a random point on that surface and (4) finding an optimum through a search process can be used to represent other systems than neural networks too.

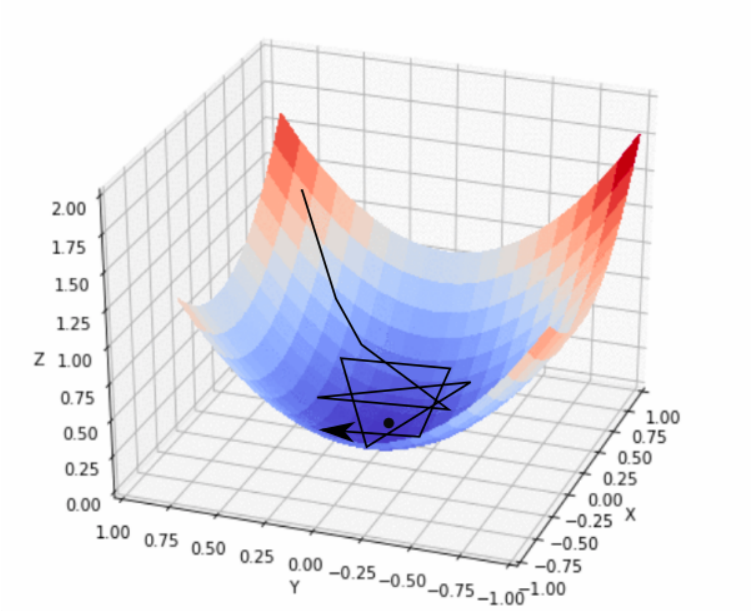


Figure 4.2: A search approach that overshoots the minimum

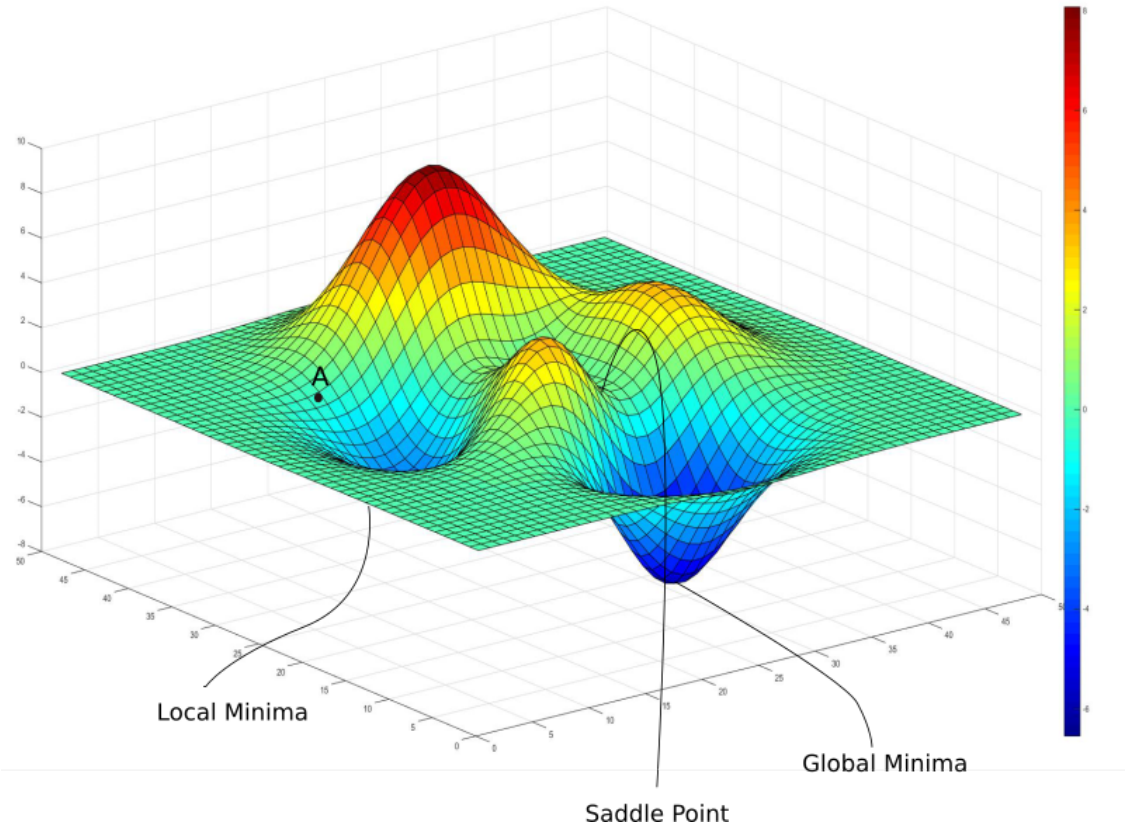


Figure 4.3: Local Minima in a landscape

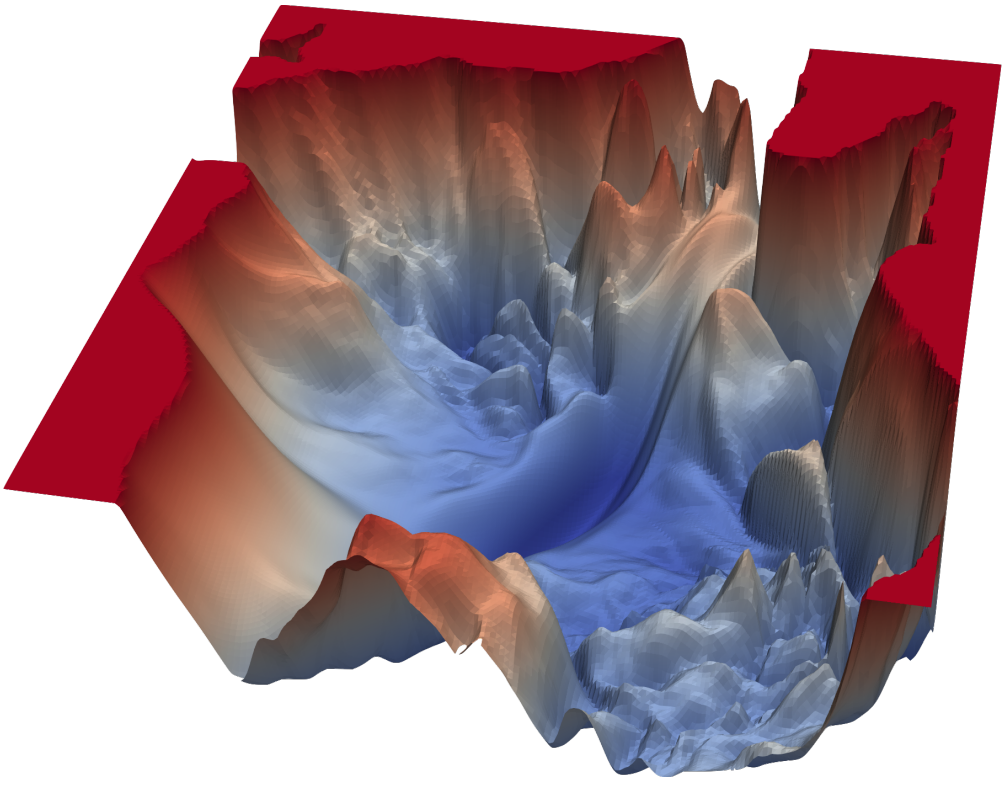


Figure 4.4: A hard to find global minimum

4.2 The Epistemic Landscape

An example of the use of this schema can be found in the 2009 Philosophy of Science paper “Epistemic Landscapes and the Division of Cognitive Labour” by Michael Weisberg and Ryan Muldoon (Weisberg and Muldoon 2009). With this paper, the authors offer a new way of understanding scientific progress through an agent-based model of scientific research. In their model, research is represented as a group-based activity of agents who use different strategies to navigate an unknown landscape. Specifically, they use a computer simulation to determine the influence (and therefore the desirability) of radically innovative groups (’Mavericks’) versus groups who adhere to the established status quo within the field (’Followers’). Through several idealizations and generalizations, they construct the landscape to be able to simulate the success and interaction of these different research groups. the epistemic landscape is supposed to represent a particular research topic within a scientific field. Therefore, the boundaries of the landscape represent the border of the topic. The X and Y-axis symbolise different choices that research groups make with regards to researching this topic. Figure 4.5 shows a top-down view of the epistemic landscape model and figure 4.6 shows a 3D plot.

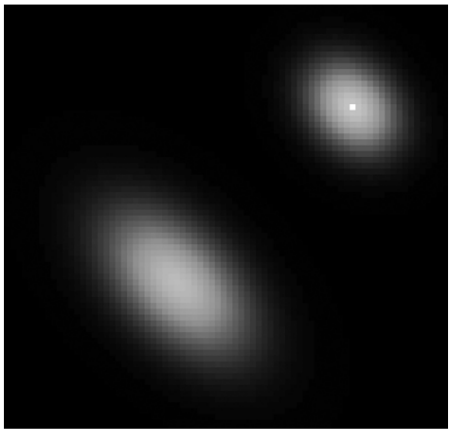


Figure 2. Two-dimensional representation of epistemic landscape. In the simulation, this grid is wrapped around a torus so that there are no edges. Lighter shades of gray correspond to greater significance.

Figure 4.5: Top-down view of the epistemic landscape (from Weisberg and Muldoon 2009, p. 235)

In the neural network example, a coordinate on the landscape straightforwardly represented two numbers: the weights of the two nodes. In Weisberg and Muldoon’s landscape, all of the choices involved in a research methodology are represented by a single coordinate. Accurately depicting the countless choices involved would require a model with countless dimensions; something that can’t be visualized. Therefore, we have to imagine ourselves that all those degrees of freedom are encoded into the X and Y axis.

For the neural network example, the Z-axis represented the outcome of the loss function. In the case of the epistemic landscape, the height represents the significance of the truth discovered by the strategy at that coordinate. A higher elevation on the landscape represents a more significant truth. The goal of the agent on the epistemic landscape is thus to find the global optimum. Here too the imagination is stretched. While the value of the loss function was just a straightforward mathematical result, the epistemic significance score is not something that can be calculated. We have to act as if epistemic significance can be represented by a simple number. So, with each

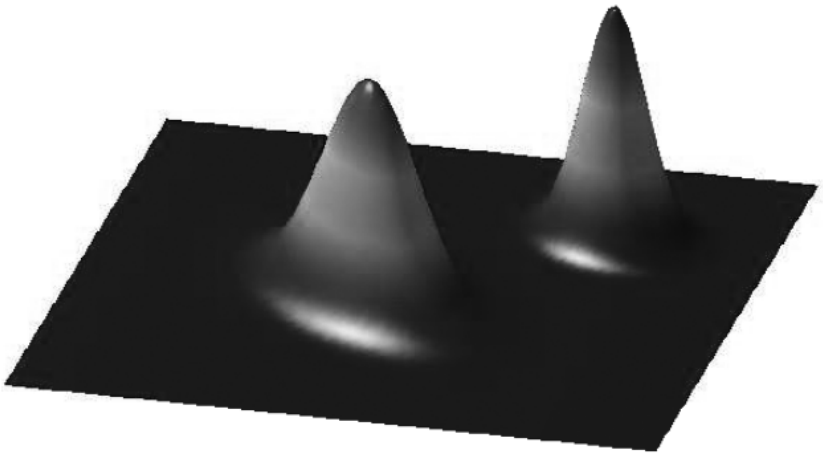


Figure 1. An example epistemic landscape of the form used for the models in this article.

Figure 4.6: 3D plot of the epistemic landscape (from Weisberg and Muldoon 2009, p. 235)

step, more information is encoded into this landscape model.
Finally, there is another complication. Where in the neural network example there was one single point that represented the current state of the system, to represent different research groups the epistemic landscape has a multitude of agents simultaneously navigating the landscape. All of these agents have the same goal (finding the global optimum) but they employ different search strategies which have different success rates. Importantly, an individual agent has limited knowledge about the landscape: they learn about it by exploring or observing others.

The simplest navigation strategy employed by the agents is called hill climbing with experimentation. Using this strategy, the agent only keeps track of its current and previous position on the landscape and takes no note of what the other agents on the landscape are doing. When the agent notices that the direction they are heading in leads to lower significance discoveries (i.e. does not slope up) they 'experiment', i.e. chose a new random heading hoping that this direction will turn out more fruitful. If we could see these agents traversing the landscape they would start on a certain patch, wander around until they find an upward slope and gradually move up the hill closest to them. This means that agents who are very far removed from the peaks will take a very long time to discover the truths that are represented by those peaks.

A more sophisticated search pattern is employed by the so-called 'Mavericks'. These agents take into account which approaches have successfully been explored previously and avoid them. If we could see these agents, instead of gradually ascending the nearest hill we would see them methodically jumping around the unvisited patches in their neighbourhood. In the simulations of Weisberg and Muldoon, this meant that the mavericks found the peaks much quicker than the 'Followers' who used the hill-climbing approach

To summarize, instead of mathematically representing a completely knowable system (like was the case in the neural network example) here the landscape schema is used to visualize an otherwise

invisible process. An added benefit is that the reader does not need to have personal experience with the particular dynamics of the interactions of research groups and the epistemic significance of scientific discoveries. All of this context information is encoded into the model of a landscape with a global optimum to be reached.

4.3 Constructing the Software Development Landscape

Each component of the landscape schema will now be adapted for the software development (SWD) context. The goal here is to encode the SWD context information into the properties of the landscape.

4.3.1 What determines the elevation of the SWD landscape

In the original model, the landscape was bounded by the borders of the scientific topic that was represented. In the SWD version, the landscape instead represents all possible versions of a software programme produced by a particular software project. Instead of epistemic significance, in the SWD version the Z-axis expresses the degree of success of the version of the software solution represented by that coordinate. It is assumed that a landscape has a global optimum (the highest peak), which represents a successful outcome of the project. There could be several peaks of different heights on the landscape, i.e. multiple satisfactory outcomes with different degrees of perceived success The job of the agent on the landscape is always to find the highest peak (i.e. implement that set of functionality and behaviour into the software system that satisfies the client/user).

4.3.2 Shifting, expanding Sands

The constant technical change in the SWD domain translates into the SWD landscape continuously expanding at the borders. The axes represent possible choices and since new technical possibilities mean new possible configurations for the IT project this means more X and Y-coordinates. If a project takes many years to complete this can mean that the technological choices settled on at the start of the project are obsolete by the end. This can mean that the software system is not maintainable because nobody can be found who still uses the particular technique of that system. Additionally, the communication standards have can have changed or, in the context of the market, the software system is no longer relevant because competitors use superior technology. In other words, the SWD landscape changes out from under the agent while it is standing still.

This is because the ’why’ and ’what’ questions are dependent on and are rooted in the ’user (or client) need’. And what that user needs is not a stable, set-in-stone, given fact. First of all, there is what the user think they need. And often what the user say they want does not coincide with what the user actually turns out to need. While the rate of technical change in the software domain is high, the rate of change for user need throughout a project is possibly much higher.

Now, crucially, the inclusion of these two dynamic components (technical change and client value change) that make up the Z-axis of the landscape means that the SWD landscape is not static

like the epistemic landscape. That is to say since both technical capability and the usefulness of a software programme evolves during the runtime of the project the very same software programme (i.e. a particular combination of X and Y-coordinates) has a different Z-value at any one time[2]

To recap, because of changing customer need, changing technological capabilities and the evolving understanding of all parties involved in IT projects The SWD landscape is very dynamic and incorporates with many degrees of freedom.

While it is impossible to show a moving landscape on paper, the visualization in figure 4.7[3] perhaps can help imagine the plethora of changing a moving optimum in the SWD landscape.

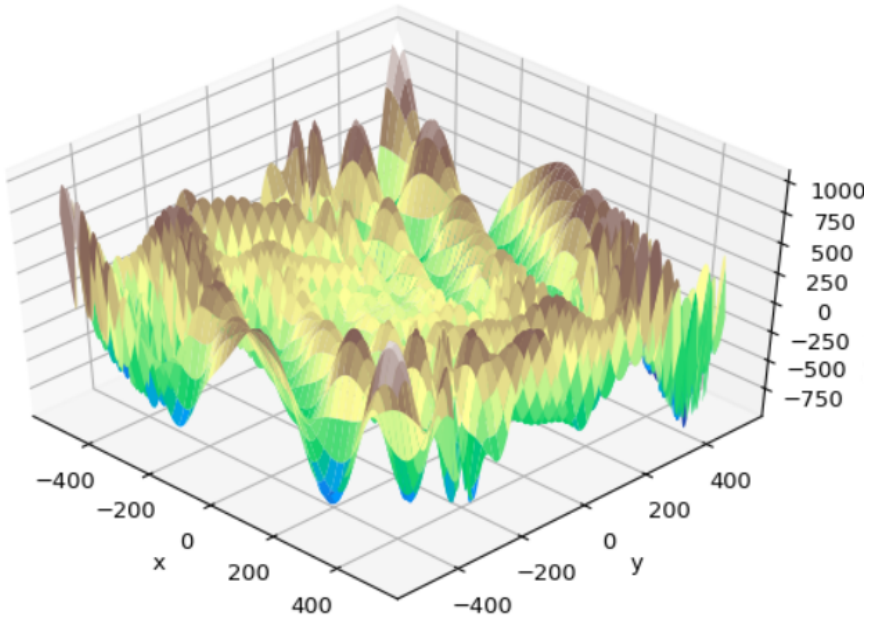


Figure 4.7: The dynamic SWD landscape

4.3.3 Unlucky Travellers

Now on top of this shifting, expanding landscape are the agents. While in the original model an agent represented a (group of) scientist, in the SWD version an agent represents a particular IT project. The Chaos report figures (Standish 2014) would suggest that only 16% of the agents on the landscape succeed in finding an optimum (i.e. can be counted as project success).

Following the complex system dynamics mentioned in section 3.2 examining a complex adaptive system affects that system. In the SWD model, this means that the movement of the agents on the landscape alters that landscape. So, in addition to technological change and evolving use-value, the delivery of working software to users reshapes the landscape.

2. Of course, different kinds of projects have different degrees of these dynamics. This would be represented by different speeds at which the landscape shifts: projects with low uncertainty and high agreement would have a relatively stable landscape while projects with high uncertainty have very dynamic landscapes

3. Image source: https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14

Now, this would already be a treacherous and confusing landscape to traverse when an agent is actually able to see it. But the agents are not directly aware of the elevation of the landscape. The only way to reveal the elevation (i.e. find out if the software is successful) is to have real users actually use the software. The longer between moments of users using the software means the longer the agent on the SWD has no idea of they have reached a peak or fallen into a deep crevasse. The solution for this is to make an approximation of the landscape that can be used in the meantime. Such an approximation is like an idealized map of the landscape. But how well of a map can be made of this landscape that we have described? In other words, how knowable is the landscape ahead of time?

4.3.4 Mapping the landscape

In the neural network example used at the start of the chapter, the landscape was the direct result of a mathematical formula. In the model of Weisberg and Muldoon, this became more abstract because the elevation is revealed when the truth found out by a particular set of methodological choices is published. As mentioned, for the SWD landscape model this Z-value is determined by project success, a composite of many factors, among which technical quality and use-value. The agents on the landscape (the IT projects) have to act, even though they do not know where the optimum on the landscape is. Therefore, the agents have developed techniques by which they try to map out the landscape beforehand. In terms of the landscape, this is like looking for the formula that defines the landscape. The (impossible) formula that produces the 'actual' landscape will be called F_{real}. While for the neural network such a formula was possible and available (the loss function), in the context of an IT project this is very doubtful If IT projects indeed qualify as complex adaptive systems it "cannot be adequately described by means of a simple theory" (Cilliers 2002, p.ix) and then a mapping function would be impossible.

Otherwise, it is the question of how well the agent can produce a function which approaches F_{real}. That is, can it be predicted what success will look like at the start of the project? The most important factor for the function that the agent produces is that it should match the peaks. The maxima of F_{agent} (what the map says are successful configurations of software) should match the highest values of F_{real} (software that turns out successful when deployed and used).

Different strategies can be employed to try to construct a mapping function that approaches F_{real}. However, because the elevation of the landscape is invisible until software is deployed in production and in-use by users, between deployments the agent has to blindly trust their mapping function. In these terms, Waterfall can be interpreted as trying to improve the map, while Agile is trying to reduce the need for a map by revealing the elevation directly.

4.4 SWD methodologies as search strategies

Encoding the SWD context information into the properties of the landscape has resulted in a fluctuating, expanding landscape being altered by the agents travelling around it who cannot perceive it and have to use approximations to navigate. With this image in mind, it will start to make sense

why different approaches to navigating the landscape are rational in such a landscape and others are not. Within this model, SWD methodologies can now be reinterpreted as search strategies for dealing with the dynamic nature of this landscape.

4.4.1 The search strategy of the Agile Agent

Instead of mapping the landscape and plotting a route, the Agile Agent (AA) first takes an inventory of goal height (success definition of the client) without committing to a specific peak of the landscape. The next action of the AA is to gauge the current elevation by taking the smallest possible step and 'collapsing' all the mapping functions onto F_{real}. This means that the smallest 'slice' of the desired functionality is taken and all of the phases of Waterfall are conducted for that single slice. Therefore working, tested, scalable code has to run in a production environment, being used by real users and presented and available to all relevant stakeholders (i.e. all potential sources of change for the elevation of the landscape). There is a deceiving simplicity to this approach:

- Take a step
- Evaluate the height of the landscape
- Identify an upward slope
- Take the next step on the slope

Where Waterfall deploys software with long intervals, which means that it is unsure if the software that is produced actually represents 'customer value', and thereby 'success' of the software application, Agile continuously and regularly delivers customer value, i.e. is constantly increasing in altitude on the landscape. Therefore, an Agile project should be able to stop at the end of any iteration and have working, valuable software in the hands of users, while if a Waterfall project is stopped prematurely only intermediate products are available and nothing may be salvaged.

With this image in mind, the importance of team autonomy mentioned should make more sense. Team autonomy means that the agent is free to incorporate the knowledge it gains about the landscape and which isn't clear to management and other stakeholders. In this context, team autonomy is thus not an emancipatory or democratic ideal that is pursued for its own sake, but due to this knowledge about the landscape the team is in the best, perhaps the only, position to make navigation decisions. This is why command-and-control hierarchies are much less effective in complex emergent environments such as the SWD landscape. And, importantly this requires the trust form those in power to transfer decision making capability as close to the surface of the landscape as possible.

It should now also be clear why in this kind of situation committing to a specific coordinate on the landscape is bound to end in failure or gross loss of resources. It is not just more efficient or flexible to continually readjust the heading of the agent, it is the only reasonable response to the behaviour of the landscape. In these terms, adopting an Agile approach is thus not simply one option among many, but the only sensible strategy. However, it should also be clear that if the dynamics of the landscape are understood that there is nothing special about Agile method specifically that makes them the only option. In the terms introduced in 2.1.4, Agile is the 'Big Idea' version of a response to a complex adaptive landscape, and through grasping the underlying principles better and more specific solutions can be derived that fit the exact landscape that you happen to find yourself in.

When we look back to the original Weisberg & Muldoon paper we can now also raise some critical points about Agile. Isn't the described approach setup to only ever find local optima, i.e. the peak closest to the starting point? Of course, finding *any* peak is an improvement compared to the many failures that large software development projects are known for. It appears that with the iterative search pattern all the while during the progress of a project you are more and more being entrenched since every choice you make entrenches you more and more in the given path. A contrast to this is the notion of 'pivoting' in start-ups: if you have developed a certain product but are not committed to a client, you can try out different spaces until you find the equivalent of a landscape that has a peak right at where the capabilities of your product lie. However, an intriguing idea is translating the search pattern of the Mavericks from the Weisberg & Muldoon paper into the SWD landscape. If Agile has been as successful as it has with a simple hill-climbing approach, the Maverick strategy promises even higher success rates With the core of Agile uncovered and rationalised, all kinds of questions like these can be explored.

4.4.2 The blind, shackled Waterfall agent with a faulty, outdated map

The (perceived) failure of Waterfall-style software development can now also be reinterpreted as a mistaken belief in the trustworthiness (or even the possibility) of mapping functions. The Waterfall approach to software development starts with a discrete phase of requirements gathering and analysis. In this phase, it is the task of the Business Analyst to produce a set of requirements which defines the software to be built. In other words, their task it to produce the mapping function, $F_{requirements}$ which should match (the main features of) F_{real}. However, there are several problems for requirement analysis which translate to sources of 'noise' in the construction of $F_{requirements}$. Due to these sources of noise, during the construction of the function $F_{requirements}$ which produces the landscape that the developers have to navigate, features are introduced into the landscape that do not exist in F_{real}. These sources include difficulty in tacit knowledge extraction, insufficient familiarity with technical feasibility and the introduction of unnecessary functionality. These sources are now each elaborated on.

The first hurdle for the mapping process is the source of information about the landscape. Since use-value is one-half of success as it is defined in the SWD model, the software user has a significant influence on the Z-score. Therefore, it would make sense to go to the user and base the $F_{requirements}$ map on what they say. However, there are several trapdoors here. First of all, it is not easy to map F_{user} because of the introduction of assumptions (e.g. not asking what is meant by a certain concept while the client may think of it differently than the analyst), the fact that humans are not able to fully articulate tacit knowledge introduces, and all the trappings and ambiguity of language in general. Second, the software solution that the user *say* they would value is often not what they will actually value once it is implemented. In other words, F_{user} does not match F_{real} and is therefore not a reliable source to base the map on since the global optimum (the highest peak of the landscape) of F_{user} does not necessarily match F_{real}. On top of this, direct access to F_{user} is not always used in the requirements process. If the Business Analyst is talking to a manager or other functionary who is not going to use the software themselves the Business Analyst is two degrees removed from actually mapping the landscape: $F_{manager} \sim F_{user} \sim F_{real}$.

Another difficulty for the mapping process is that unreachable peaks can be introduced if neither the client nor the Business analyst has sufficient knowledge about technical capabilities. This dynamic is captured well in the XKCD comic[4] pictured in figure 4.8.

Figure 4.8: XKCD comic number 1425

Additionally, false peaks can be introduced into $F_{requirements}$ by tacking on unnecessary functionality to the software programme. An example could be that an executive has heard the word Big Data and now wants their flagship project to use machine learning while it does not contribute to client value and therefore not to project success This introduces a peak in the mapping function that steers the agents off course.

The combination of these difficulties for the mapping process means that everything is set up for the Waterfall requirements analysis process to designate peaks in the landscape that are in fact not globally optimal for anyone. And even in the hypothetical situation where F_{real} could be mapped, the landscape is not static but dynamic. Therefore, $F_{requirements}$ is just a snapshot of the output ofF_{real} at the start of the project when the least is known about the SWD landscape.

After requirement analysis,$F_{requirements}$ is finalized (akin to signing off on a blueprint) and handed over to the IT/Delivery department to be implemented. At this point, the features of the software programme to be produced have all been meticulously documented in the requirements documentation. On the landscape, this means that the target coordinate has been selected and a route on the landscape has been plotted. The IT team has been given the set of directions and is asked to follow them to the letter. In certain contexts, the requirements are put into a contract and any changes require an official 'Request For Change' (RFC) process and associated forms and procedures. In such a procedure it isn't always clear who has the authority to make a decision or there are multiple decision-makers with conflicting interests. Due to these and other factors, a RFC can take months to get approved.

4. https://xkcd.com/1425/

In the language of the landscape, the navigation decisions have thus all been made and locked into place at the start of the journey before any working software has been produced. At the start of the project, when the least is known about the landscape all of the major decisions are locked in. In the Agile community, this is called ”locking-in ignorance”. According to Martin Fowler, chief scientist at ThoughtWorks and Snowbird attendee: “People would come up with detailed lists of what tasks should be done, in what order, who should do them, [and] what the deliverables should be . . . The variation between one software and another project is so large that you can’t really plot things out in advance like that” (Nyce 2017). Horror stories recounted by the authors of the manifesto about what the excesses of the requirements phase looked like include an entire bookshelf’s worth of requirements in binders and an 800-page document that had been translated across three different languages. Ken Schwaber, the cofounder of Scrum and founder of Scrum.org and manifesto signatory says Waterfall “literally ruined our profession. . . It made it so people were viewed as resources rather than valuable participants. . . With so much planning done upfront, employees became a mere cog in the wheel” (ibid.).

Now let’s follow the agent (the project) in our mind’s eye as it starts its path on the landscape under the Waterfall approach. All of the pitfalls of software engineering in general start to come into play:

- Mismatching Assumptions
- The trappings of language such as ambiguity and miscommunications
- stakeholders (client, sales, management or others) try to add functionality midway through the process (i.e. try to change the target coordinate halfway throughout the journey)
- Developers focus on making cool software instead of software that the user wants
- Technical Debt
- Programming errors (off-by-one errors, typo’s, deprecated functions)
- People start leaving the project out of frustration and thereby knowledge of how the landscape behaves leaves. the newcomers have to try to reinvent the wheel.

We can imagine these as the agent encountering trapdoors and bears on the road, skeletons in the closet, or getting stuck in quicksand. Meanwhile, it can take years before working software is delivered. This means that all this time the agent only has the ’map’ produced at the start of the project to go off of while all the while the landscape has been evolving. Due to the locked-in requirements, the Waterfall agent has a very limited capacity to make use of what it learns about F_real and there is a significant delay in any course corrections it wants to make. I.e. developers are not allowed to just change course when they spot the difference between $F_{requirements}$ and F_{real}. On top of this, these limiting decisions have been made at the point of the project when the least is actually know about F_{real}, since no working software has been produced at that point.

One of the reasons that the total project takes so long is that all of the requirements can be implemented but now the software has been tested. Since testing is a separate department, they are often not immediately available. While waiting for a test slot to open up, the agent stands still and the environment keeps changing. Furthermore, since the testers are a new group of people, they have to acquire all of the context knowledge about the project sometimes only through the documentation of the code. This documentation is like if the agent had been keeping notes about

its travels, all the while still being blind to the elevation of the landscape because no working software has been presented to the client and therefore the 'successfulness' remains hidden. In practice, these notes are often sporadic or confusingly written, and even the best travel notes do not transfer all of the tacit knowledge gained during the journey. In testing, several discoveries can be made about the software that has been produced. For example, in comparing the code to the requirements the testers can discover that a wrong turn was taken on the landscape by the development team. Adjusting course again has to go through the RFC process all the while the landscape is shifting.

Even when the agent is not completely blind to the elevation of the landscape (i.e. the customer and other stakeholders are involved periodically and working software is presented to them), any discoveries made require the arduous RFC process. On the landscape, this means that if the agent notices along the way that the chosen peak is not as high as though and tries to adjust course, the decisions themselves take a while to get approved and in the meantime, the landscape has changed again. In terms of the functions that map the landscape, the development team has only the landscape produced by $F_{requirements}$ to go off of. If they encounter one of the unreachable peaks of the landscapes or in some other sense find out that the map produced by $F_{requirements}$ is leading them the wrong way, in the Waterfall process there is very little room for the agent to adjust course. Because of the sequential nature of Waterfall, the later in the project a change is made, the more expensive it becomes.

The reasons for Waterfall-style software project failure as outlined in the Standish Chaos reports (Standish 2014) can now be reinterpreted within the context of the SWD landscape model. In all the trials and tribulations that the blind plan-driven agent goes through on their journey of the landscape the travel time can significantly increase. Meanwhile, there are limited resources available and this can result in the situation where the trip of the agent is cut short before ever reaching its destination. In the case where no working tested software is delivered at all throughout the implementation phase of the project, this means that plotting such a project on the landscape would look like an agent standing still for a very long time, all the while burning time and money until the project is stopped. Of course, a lot of code has been produced by such a project, but this does not translate to any movement in the SWD landscape model.

Another scenario is that software is produced, i.e. the agent moves over the landscape, but as mentioned the elevation remains unknown. When the final product is finally released, the elevation becomes clear (the value of F_{real} is revealed). At the start of the project, a coordinate was picked which at the time looked like a peak on the landscape. Due to the nature of the landscape, the only way to know what a peak is is to visit it. Such an estimation at the start of the project is therefore almost a total guess. In practice, it can suddenly turn out that either the coordinate never represented an optimum at all or in the time that it took to travel to the coordinate the landscape has evolved and it is no longer a peak.

All in all, it can be said that the Waterfall-style strategy for navigating the dynamic SWD landscape is both inefficient and ineffective. Making these navigation decisions upfront does not make use of the learning processes and discoveries regarding the environment. It should now be clear why Waterfall is an irrational strategy for the type of landscape modelled in the SWD landscape.

4.5 The right ‘tool’ for the right job

While the strategy of the Waterfall Agent appears to mismatch with the SWD landscape as it has been described in this chapter, this changes if the nature of the environment changes. In a static, knowable landscape the plan-driven approach is much more efficient and due to Agile’s possible tendency to get stuck in local optima, perhaps more effective.

In a historical context, the plan-driven approaches were developed in situations where they made sense, and therefore they got the reputation of best practices. When the activity of writing code grew in size and complexity in the 1950s, resulting in the software crisis in the 1960’s (Shapiro 1997, p. 20) and the call for professionalization in the form of ‘Software Engineering’ and ‘Software Factories’ it makes sense that methods and approaches were called on that were tried-and-true in the engineering and factory domains. If, however, these methods failed because the development of software systems of a certain size and intricacy is complex in the Cynefin sense than it also makes sense that they would fail. These failures were interpreted as due to human error or a lack of analysis which set the direction for a self-reinforcing pattern of investing more in upfront analysis after each failure until the resulting pressure on the organization of the over-rigid and bureaucratic methods that this pattern resulted in became too much to bear and a ‘rebellion’ broke out in the form of Agile, which relatively quickly replaced the plan-driven approach due to its effectiveness. Now, it is completely plausible that the way for dealing with complexity was found due to trial-and-error instead of a high-level understanding of what was going on. And since software developers were more interested in coding that in understanding why Agile worked, it quickly spread without a good theoretical account. Embedded in a higher-level perspective on what is going on, both strategies can be part of the ‘tool belt’ of organizational and methodological approaches and applied when appropriate.

4.6 Chapter Summary

The core goal of the proposed rationalization of Agile is to make it more nuanced and more intuitive. The rationalizations discussed in the previous chapter are both existing frameworks applied to Agile. The proposed rationalization is more nuanced because it is tailor-made for the software development domain. The existing rationalizations require either familiarity with complexity science or practical experience with software development. The proposed rationalization is independent of both this theoretical knowledge and practical experience because it integrates the dynamics directly.

By constructing a metaphor that functions as a cognitive bridge, the complexity-based rationalization of software development approaches has been de-contextualized. Therefore, the answer to the fifth sub research question is that Agile can be rationalized as a search strategy on the software development landscape model. In the following and final chapter, the results are discussed further and future research directions are indicated.

Chapter 5

Conclusion

The results of this thesis are now succinctly summarized, some ensuing implications are mentioned and future research opportunities are pointed at.

5.1 Summary

The first sub research question, “What are the core documents of Agile?” has been answered with: the Agile Manifesto and the Scrum Guide. The second sub research question of this thesis is “What justification for the use and effects of Agile do the core documents offer?”. For the Agile Manifesto this question has been answered with: an implied empirical justification but no theoretical account. Regarding the Scrum Guide, the early (2010) version mentions empirical process control theory (specifically (Schuler 1996)) and the later (2017) version mentions Complex Adaptive Systems.
Following up with the third research question, this complexity-based rationalization has also been observed in the Agile community, and two particular forms, the Stacey Matrix and the Cynefin Framework have been evaluated. The fourth sub research question, “What are the limits of the current rationalizations?” has been answered with: the current rationalizations are dependant on knowledge of complexity science and software development experience.
The fourth sub research question, “What are the limits of the current rationalizations?” has been answered with: the current rationalizations are dependant on knowledge of complexity science and software development experience. By constructing a metaphor that functions as a cognitive bridge, the complexity-based rationalization of software development approaches has been decontextualized.

The core goal of the proposed rationalization of Agile is to make it more nuanced and more intuitive. The existing rationalizations are both existing frameworks applied to Agile in the community. The proposed rationalization is more nuanced because it is tailor-made for the software development domain. The existing rationalizations require either familiarity with complexity science or practical experience with software development. The proposed rationalization is independent of both this theoretical knowledge and practical experience because it integrates the dynamics directly. The complexity-based rationalization has been transformed in such a way that it makes use of basic intuitions and becomes independent of knowledge of complexity science and software development experience. By constructing a metaphor that functions as a cognitive bridge, the complexity-based rationalization of software development approaches has been

de-contextualized. Therefore, the answer to the fifth sub research question is that Agile can be rationalized as a search strategy on the software development landscape model. In the following and final chapter, the results are discussed further and future research directions are indicated. Therefore, the answer to the fifth sub research question is that Agile can be rationalized as a search strategy on the software development landscape model.

5.2 Discussion & Implications

First, the broader relevance of the thesis for the PSTS fields is discussed. Next, the case is made for more attention to the organizational level of analysis. Finally, a point is made about epistemic responsibility.

5.2.1 PSTS relevance

There are a number of academic fields which have as their aim understanding (some aspect of) science and/or technology. These fields include the Philosophy of Technology, Philosophy of Science, Science and Technology Studies, the History of Science and Technology and Ethics of Technology. In this thesis, these fields will collectively be referred to as 'PSTS'. In these fields, Agile specifically, and software development methodology in general, rarely comes up (see section 1.4). However, there are a number of reasons that software development, and Agile in particular, warrant attention and critical discussion in the PSTS fields. Not only does paying attention to the software development methodology field hold the promise of enriching the current understanding of technology and improving the effectiveness of communication from PSTS toward the software development industry, due to the ubiquity, impact and particular nature of software letting these subjects stay unattended runs the risk of leaving PSTS unequipped for a future where it is increasingly confronted by software.

Any reader who is in the position to read this thesis probably needs little convincing (perhaps only a reminder) of the size of the role that software plays in (western, industrialized) daily life and society. It would hardly be an exaggeration to say that we live in a 'digital age'. The ubiquity of software is reflected in the fact that some of the largest companies are software companies, some of the largest public debates in recent years (such as about the influence of social media on the American presidential elections, Artificial Intelligence and Privacy, data use & surveillance) are about software. The rapid digitisation of organizational processes dubbed the 'Digital Transformation' (Stone and Levine 2019) is affecting virtually all organizations, including the sciences (Symons and Horner 2014). In the humanities, this influence of software is most visible in the emergence of the discussion about the 'Digital Humanities' (Berry 2011; Berry 2012).

Likewise, software has a steadily growing presence in PSTS. For the technology-oriented aspects of the PSTS fields, software has shown up with increasing frequency as both the subject and the context of inquiry. For example, discussions about Robots, Automation, Artificial Intelligence, Transhumanism and the Internet all involve software in direct and indirect ways. For the science-focused side of PSTS, this growing role of software has resulted most prominently in attention

for the impact of Big Data, e.g. (Fuller 2015) and Machine Learning, e.g. (Ratti 2019). All in all, the current ubiquity of software, which on top of this is only expected to keep increasing, by itself warrants a significant amount of dedicated investigation from the PSTS into software-related subjects. And, as discussed in section 1.4, there is indeed quite some attention within PSTS for Software in general. But the organizational and methodological aspects remain underdeveloped. In addition to being crucial for understanding software in general, there is another reason why an investigation into Agile is relevant to PSTS. The advent of Agile itself may challenge the viability of PSTS' approach to technology if software will continue its ubiquity and dominance.

One line of reasoning goes that the emergence of Agile reflects the particular nature of the software development process. For example, it is claimed that once the SWD process is understood as a knowledge production process, a learning process or as a design process instead of a production process that it becomes clear why Agile should be used and not other tried-and-true traditional approaches. This argument can be extended: the fact that a new way of developing and organizing needed to be found in order to deal with making software might mean that software is different than other technologies for which no other development methods needed to be found. This fact may translate to the methods and theories that the PSTS fields apply to technology: if software is indeed different in some capacity, can it safely be assumed that the traditional ways of approaching technology apply to software and don't need to be likewise adapted as the development method was? Possible specific examples of PSTS methods that may need to be adapted for software development include Technology Assessment or Value-Sensitive Design. In order to explore this question about the applicability and viability of PSTS methods and theories about technology, first, a clearer understanding of Agile is needed since this is what raised the questions in the first place.

There is thus an opportunity for a productive cross-pollination between PSTS and the SWD domain. While PSTS offers the analytical resources to punch through the haze of vagueness surrounding Agile and extract and articulate the underlying rationale, both PSTS (because of the increasing relevance of software) and the SWD domain stand to benefit from the availability of a clear understanding of Agile.

5.2.2 The organization as a distinct unit & level of analysis

One question that arises in the face of this thesis is: Why has agile remained unnoticed? While software in general has garnered a lot of critical attention, it might be that the right *analytical level* has been missing. The level on which analysis takes place is crucial since it reveals certain aspects and conceals others. Crucially, modern science always takes place in an organization, technology is usually developed in the context of an organization and also has influences on an organizational level, but the organization as a unit of analysis is rarely addressed in PSTS. I think it is not only important to critically investigate Software and Software development methodology because of the ubiquity of software and the resulting relevance of understanding software, or because of helping practitioners clear up their conceptions and debates but also because it pulls the organization and its dynamics as such into focus. In the reflective fields such as Philosophy of Technology, Philosophy of Science and Science and Technology Studies, different levels of analysis are used,

but the organizational level seems to be underdeveloped. I would argue the organization, i.e. groups of collaborating humans, or systems that consist of humans, is a relevant unit of analysis and context for analysis. It appears to me that much reflective analysis is conducted at the macro level of societies, or 'humanity' and the micro-level of the individual human, but the meso level of organizations is less common. However, it might be so that these different levels each know their own dynamics which makes it so that knowledge or approaches from one level cannot be extended directly to the other.

In this sense, investigating Agile also results in uncovering certain attributes or dynamics of the level of analysis or domain of organizations *as such*. Agile methodologies usually don't actually say anything about how to write software, but focus on how the work should be organized, what roles and responsibilities should be assigned, i.e. how to structure the organization in order to deal with certain dynamics that arise when that organization is performing complex work. Why I think the organizational level deserves more specific attention is because organisations make up a sizeable part of the world. For example, in Philosophy of Technology, micro-level questions are asked about how a technology impacts me as a human (micro) or how technological change impacts societal change (macro). But very often, and especially in the case of software technologies, these tools are used in the context of organizations and we can/should investigate the impact or meaning of a technology on an organizational level. If it is true that the dynamics of the meso level are not fully covered by the findings of the micro and macro level and thus that the findings on those levels do not (fully) apply to the meso level, the organization level could represent a blind spot in current analysis which is deserving of its own specific study. Likewise in Philosophy of Science questions are asked about conducting science on an individual or abstract level (how is knowledge constructed in scientific processes, how do explanations work), but these scientific processes almost always take place in the context of organizations, and if it is true that organizations know their own dynamics, again this is something worth investigating. Certain consequences or dimensions of the problems or processes that are studied may remain hidden when the meso-level of analysis is overlooked. Even if it this meso level dynamic does not exist and it is fully covered by findings or approaches from the higher and lower levels, this is a fact that it seems to me is worth uncovering, where the first and last finding of the organizational field science and technology studies would be that it doesn't need to exist.

5.2.3 Epistemic Responsibility

The SWD domain is infamous for its many, and continually expanding cast of guru's, methods and frameworks. One part of what keeps the guru cycle going is a particular attitude on the side of their public. Particularly management just wants to install the 'best practice' into their organization and be done with it. This attitude makes you susceptible to believe someone who comes along claiming they have found the magic silver bullet that will solve all your problems and lets you go back to worrying about whatever it is you rather concern yourself with. There is an asymmetry in the guru-follower relationship. The guru (or methodology, or other equivalents) prescribes what should be done and the follower has to execute these commands. This leaves no room for adjusting the command to the particular situation of the follower. The relation is therefore also rigid.

The catch is that the principles are abstract and need to be re-interpreted for a specific situation. Since it is impossible to anticipate all possible future situations, the 'user' of the principles needs to do this work of applying the principles to their particular situation themselves. The rub is that it often seems that people don't want to do this work and rather pass on this burden to an external party. The problem is that this external party cannot know the specifics of the situation of the original user and therefore the interpretation of the principles will always be somewhat lacking.

From this description, it can also easily be seen that this is not an attitude that is specifically causing problems in the SWD domain. In many different settings, the pattern can be observed of someone being confronted with a difficult problem (e.g., organizational complexity but in the classic guru setting also: identity, belonging, purpose etc.) and instead of taking ownership and responsibility of their situation and figuring out through trial and error what helps they turn to an external source which promises to have the answers. For simple situations, this is indeed the most direct route for dealing with a situation. Furthermore, this externalization of problem-solving mirrors the parent-child relationship and therefore it makes sense that it is a deeply rooted habit in how we approach the world. But however comfortable it may be to offload the cognitive burden of figuring out a complex situation, it is exactly in those complex situations that 'best practices' don't hold up.

Therefore, it appears to me that there is a small similarity to Existential philosophy here: people need to take responsibility and ownership for their situation and accept the burden that comes with that instead of looking for external sources of 'best practices' so that they don't have to think for themselves. In the context of the type of complex emergent landscape described in Chapter 4 there truly is no one else who can do the problem-solving work of dealing with the specific dynamics of the particular landscape: you need to be on the landscape to learn about it. Acknowledging this fact and taking up this task may thus be called accepting one's epistemic responsibility.

5.3 Future research

To conclude, some future avenues of research are pointed at. There are two main categories. First, there are possibilities for expanding, elaborating and improving the current research direction Secondly there are parallel research approaches that become clear because of the project.

5.3.1 Development of current project

The current thesis is based on personal experience. The results could be validated through a full agile community literature review, a full systematic software engineering literature review and/or a broad empirical study of rationalizations is practice (an approach critiqued by Waters in the Gene context (Waters 2004)). The empirical adequacy of the model can be more systematically tested by seeing if it makes sense to large numbers of practitioners and helps to solve problems.

Some possible future research questions could be:

- Is complexity a property of a system, a property of the cognitive capacity of a decision aiming agent, or a co-construction of both factors?
- What strategies for dealing have been observed in complexity science and how do they compare with Agile?
- How does the uncertainty in software development compare with physical product development?
- Based on this difference, should software be understood as a different kind of technology?
- If so how should theories and research methods be adapted to reflect this difference?
- Is it valid to treat IT projects/organizations as complex adaptive systems? Why?
- How can the notion of emergence and the body of work on process ontology develop the understanding of organizational processes?
- Can this approach help evaluate Brian Marick's claim that switching methodologies require a gestalt switch on the ontological level?
- How does the informal use of notions from complexity science in the agile community stack up with their formal definitions?
- How does the rhetorical use of the notion of Waterfall as doomed to fail and incompetent stack up to reality? I.e. does Waterfall actually exist or is it a strawman?
- How does the conceptual analysis based on the two core documents compare with the views on Agile and its rationalization held in the Agile community?

A possible qualitative research approach to the final question is pictured in figure 5.1.

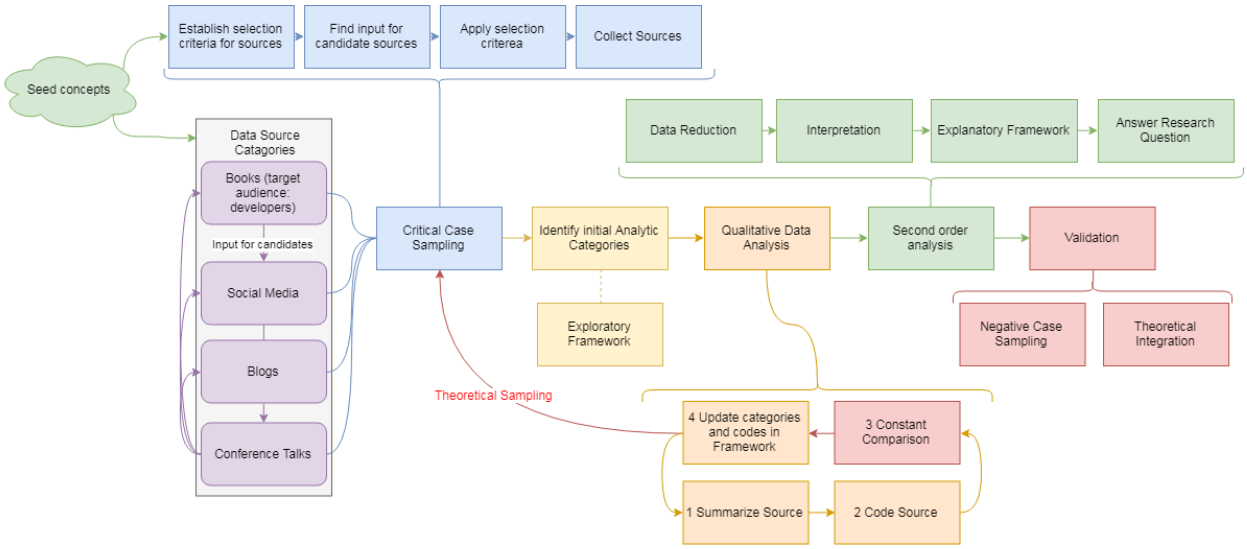


Figure 5.1: A qualitative research approach for the informal Agile Community literature

5.3.2 Roads not taken

Some of the other ways that the topic could be approached but which were not part of this thesis. Another way to arrive at a description and characterization of Agile is through a strongly empirical approach. In such an investigation the focus is not what people *say* why and what they do but to uncover and perhaps compare that with actual practices. An RQ would be: What is the relation between Agile theory/concepts and software development practices?

Similarly, another way to investigate the popularity of Agile is through a mainly sociological lense. The 1995 PSTS thesis about the spread of Object-Oriented Design “The Object is the Subject” by Bob Kennedy is this sort of investigation. Here one would use a conceptual framework consisting of the notions ‘design regime’ and ‘promotor’. With these notions in hand, the spread and adoption of the ideas can be traced. However, with such an approach the ideas themselves are left somewhat unexamined. Instead, this thesis is mostly focussed on these ideas themselves and aims at fully unpacking them.

The introduction of the early (2010) version of the Scrum Guide states (Guide, Schwaber, and Sutherland 2010, p.3): “Scrum is a framework for developing complex products and systems that is grounded in empirical process control theory”. The specific reference given with respect to the empirical process control theory is (Schuler 1996). This empirical process control theory could be investigated and compared to the rationalization proposed in this thesis.

All of the theoretical perspectives pictured in the overview by (Dingsøyr et al. 2012, p.1217) in figure 3.1 could be investigated. Of particular interest are the use of evolutionary concepts in the context of software development (Calcott 2014).

An approach more along Philosophy of Science lines: are Waterfall and Agile paradigms? Northover has done a lot of groundwork for this question (Northover, Boake, and Kourie 2006; Northover et al. 2007a; Northover et al. 2007b; Northover et al. 2008)

Finally, Gruner points to an interesting problem for a Philosophy of Software Engineering:

> Many software products seem to be inappropriate or do not fulfil their intended purpose simply because in many cases we just do not know ‘how to do things’; we lack the procedural knowledge in many domains and circumstances. Vice versa one could even assume a very radical epistemological position and declare: We do not have any knowledge about something unless it is procedural (algorithmic) knowledge about how to create it. This is related to the problem of what is ‘creativity’. For example, it is fair to say that we have very good procedural knowledge about how to create a compiler—in short: we ‘know’ compilers very well. On the contrary, we do not have procedural knowledge about how to create stunning original pieces of art—therefore, in radical terms, we do not know art, not in this strong sense of ‘knowing’ with which we know compilers (because we can produce them easily following standardized handbook procedures). Also in software engineering in general we still have very little (procedural) knowledge about how to create a software system which adequately fulfills some arbitrarily given purpose P. Gruner 2011, p.297

Bibliography

Abbas, Noura (2009). “Software quality and governance in agile software development”. PhD thesis. University of Southampton.

Abbas, Noura, Andrew M Gravell, and Gary B Wills (2008). “Historical roots of agile methods: Where did “Agile thinking” come from?” In: *International conference on agile processes and extreme programming in software engineering*. Springer, pp. 94–103.

Adut, Jonathan (2016). “Applying agile approaches in public construction and civil engineering projects: A study to identify opportunities for a more flexible projectmanagement process”. MA thesis. KTH Industrial Engineering and Management Industrial Management.

Ågerfalk, J, Brian Fitzgerald, and Old Petunias In (2006). “Flexible and distributed software processes: old petunias in new bowls”. In: *Communications of the ACM*. Citeseer.

Beck, Kent and Erich Gamma (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Beck, Kent et al. (2001). *Manifesto for agile software development*. URL: https://agilemanifesto.org/.

Berry, David (2011). “The computational turn: Thinking about the digital humanities”. In: *Culture machine* 12.

Berry, David M (2012). “Introduction: Understanding the digital humanities”. In: *Understanding digital humanities*. Springer, pp. 1–20.

Bongaerts, Fon (2018). “De invloed van Scrum@ School op de zelfregulatie van technasiumleerlingen in de onderbouw.” MA thesis. University of Twente.

Boyle, Maree and Ken Parry (2007). “Telling the whole story: The case for organizational autoethnography”. In: *Culture and Organization* 13.3, pp. 185–190.

Bratton, Benjamin H (2016). *The stack: on software and sovereignty.(Software studies)*.

Brey, Philip and Johnny Hartz Søraker (2009). “Philosophy of computing and information technology”. In: *Philosophy of technology and engineering sciences*. Elsevier, pp. 1341–1407.

Calcott, Brett (2014). “Engineering and evolvability”. In: *Biology & Philosophy* 29.3, pp. 293–313.

Cilliers, Paul (2002). *Complexity and postmodernism: Understanding complex systems*. routledge.

Conboy, Kieran and Brian Fitzgerald (2004). “Toward a conceptual framework of agile methods”. In: *Conference on Extreme Programming and Agile Methods*. Springer, pp. 105–116.

Conboy, Kieran, Brian Fitzgerald, and William Golden (2005). “Agility in information systems development: a three-tiered framework”. In: *IFIP International Working Conference on Business Agility and Information Technology Diffusion*. Springer, pp. 35–49.

Consultancy.nl (Nov. 4, 2019). “Rijkswaterstaat werkt samen met externe bureaus aan verbetercultuur”. In: *Consultancy.nl*. URL: https://www.consultancy.nl/nieuws/25674/rijkswaterstaat-kiest-externe-bureaus-voor-agile-transformatie.

Coyne, Richard (1995). *Designing information technology in the postmodern age: From method to metaphor*. Mit Press.

Cram, W Alec and Sue Newell (2016). "Mindful revolution or mindless trend? Examining agile development as a management fashion". In: *European Journal of Information Systems* 25.2, pp. 154–169.

Cronin, Patricia, Frances Ryan, and Michael Coughlan (2008). "Undertaking a literature review: a step-by-step approach". In: *British journal of nursing* 17.1, pp. 38–43.

Danaher, John (2016). "The Threat of Algocracy: Reality, Resistance and Accommodation". In: *Philosophy and Technology* 29.3, pp. 245–268. DOI: 10.1007/s13347-015-0211-1.

De Laat, Paul B (2005). "Copyright or copyleft?: An analysis of property regimes for software development". In: *Research Policy* 34.10, pp. 1511–1532.

Dijkhuizen, Bas (Oct. 23, 2019). "Bol.com: 'Agile-aanpak heeft ons geholpen bij WMS-implementatie'". In: *logistiek.nl*. URL: https://www.logistiek.nl/warehousing/nieuws/2019/10/bol-com-agile-aanpak-heeft-ons-geholpen-bij-wms-implementatie-video-101170036.

Dingsøyr, Torgeir, Tore Dybå, and Nils Brede Moe (2010). *Agile Software Development: Current Research and Future Directions*. 1st. Springer Publishing Company, Incorporated. ISBN: 3642125743, 9783642125744.

Dingsøyr, Torgeir et al. (2012). *A decade of agile methodologies: Towards explaining agile software development*.

Dingsøyr, Torgeir et al. (2014). *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation: XP 2014 International Workshops, Rome, Italy, May 26-30, 2014, Revised Selected Papers*. Vol. 199. Springer.

Dybå, Tore and Torgeir Dingsøyr (2008). "Empirical studies of agile software development: A systematic review". In: *Information and software technology* 50.9-10, pp. 833–859.

– (2009). "What do we know about agile software development?" In: *IEEE software* 26.5, pp. 6–9.

Eveleens, J Laurenz and Chris Verhoef (2009). "The rise and fall of the chaos report figures". In: *IEEE software* 1, pp. 30–36.

Fierro, Davide, Stefano Putino, and Lucio Tirone (2018). "The Cynefin Framework and Technical Competencies: a New Guideline to Act in the Complexity". In: *INCOSE International Symposium*. Vol. 28. 1. Wiley Online Library, pp. 532–552.

Finnegan, Matthew (Sept. 5, 2016). "How Spotify uses automation and microservices to gain speed advantage on larger rivals". In: *Computerworld*. URL: https://www.computerworlduk.com/it-management/spotify-uses-automation-microservices-gain-speed-advantage-3645974.

Floridi, Luciano (2008). *The Blackwell guide to the philosophy of computing and information*. John Wiley & Sons.

Forsgren, Nicole et al. (2019). "2019 Accelerate State of DevOps Report". In:

Friedman, Batya and Helen Nissenbaum (1996). "Bias in computer systems". In: *ACM Transactions on Information Systems (TOIS)* 14.3, pp. 330–347.

Fuller, Michael (2015). "Big Data: New Science, New Challenges, New Dialogical Opportunities". In: *Zygon* 50.3, pp. 569–582. DOI: 10.1111/zygo.12187.

Goodwin, Remington J (2017). "Construction Agility-An Integrated Management System". In:

Gruner, Stefan (2011). “Problems for a philosophy of software engineering”. In: *Minds and Machines* 21.2, pp. 275–299.

Guide, Scrum, Ken Schwaber, and Jeff Sutherland (2010). *Scrum guide*.

Hamdan, Amani (2012). “Autoethnography as a genre of qualitative research: A journey inside out”. In: *International Journal of Qualitative Methods* 11.5, pp. 585–606.

Hui, Yuk (2016). *On the existence of digital objects*. Vol. 48. U of Minnesota Press.

Introna, Lucas D (1996). “Notes on ateleological information systems development”. In: *Information Technology & People* 9.4, pp. 20–39.

Introna, Lucas D and Edgar A Whitley (1997). “Against method-ism: exploring the limits of method”. In: *Information Technology & People* 10.1, pp. 31–45.

Irmak, Nurbay (2012). “Software is an Abstract Artifact”. In: *Grazer Philosophische Studien* 86.1, pp. 55–72. DOI: 10.1163/9789401209182_005.

Jain, Radhika and Peter Meso (2004). “Theory of complex adaptive systems and agile software development”. In: *AMCIS 2004 Proceedings*, p. 197.

Janssen, Jonathan (Oct. 5, 2019). “Zelfs premier Rutte heeft het over ‘agile’. Werkt iedereen al in de scrum?” In: *Trouw*. URL: https://www.trouw.nl/economie/zelfs-premier-rutte-heeft-het-over-agile-werkt-iedereen-al-in-de-scrum~bb31463a/.

Jesiek, Brent K (2006). “The Sociotechnical Boundaries of Hardware and Software: A Humpty Dumpty History”. In: *Bulletin of Science, Technology & Society* 26.6, pp. 497–509.

Johnson, Deborah G (2004). “Computer ethics”. In: *The Blackwell guide to the philosophy of computing and information*, pp. 65–75.

– (2007). “Democracy, Technology, and Information Societies”. In: *The Information Society: Innovation, Legitimacy, Ethics and Democracy In honor of Professor Jacques Berleur sj*. Springer, pp. 5–16.

Johnson, Mikael et al. (2014). “The managed prosumer: evolving knowledge strategies in the design of information infrastructures”. In: *Information, Communication & Society* 17.7, pp. 795–813. DOI: 10.1080/1369118X.2013.830635. eprint: https://doi.org/10.1080/1369118X.2013.830635. URL: https://doi.org/10.1080/1369118X.2013.830635.

Kathuria, Ayoosh (2018). “Intro to optimization in deep learning: Gradient Descent”. In: *paperspace.com*. URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/.

Keogh, Liz (2013). “Estimating Complexity”. In: *lizkeogh.com*. URL: https://lizkeogh.com/2013/07/21/estimating-complexity/.

Kim, Gene, Kevin Behr, and Kim Spafford (2019). *The Unicorn project: A novel about Developers, Digital Disruption, and Thriving in the Age of Data*. IT Revolution.

Laanti, Maarit, Jouni Similä, and Pekka Abrahamsson (2013). “Definitions of agile software development and agility”. In: *European Conference on Software Process Improvement*. Springer, pp. 247–258.

Laat, Paul B. de (2001). “Open Source Software: A New Mertonian Ethos?” In: *Ethics and the Internet*. Ed. by Anton Vedder. Intersentia.

Lindsjørn, Yngve et al. (2016). “Teamwork quality and project success in software development: A survey of agile development teams”. In: *Journal of Systems and Software* 122, pp. 274–286.

Magnuson, Paul et al. (2019). “Getting Agile at School”. In: *Agile and Lean Concepts for Teaching and Learning*. Springer, pp. 115–132.

Mahnic, Viljan (2011). “A case study on agile estimating and planning using scrum”. In: *Elektronika ir Elektrotechnika* 111.5, pp. 123–128.

Marick, Brian (2004). “Methodology work is ontology work”. In: *ACM Sigplan Notices* 39.12, pp. 64–72.

McCaffery, Fergal (2018). “Adopting Agile in the Sports Domain: A Phased Approach”. In: *Software Process Improvement and Capability Determination: 18th International Conference, SPICE 2018, Thessaloniki, Greece, October 9–10, 2018, Proceedings*. Vol. 918. Springer, p. 275.

Méndez, Mariza (2013). “Autoethnography as a research method: Advantages, limitations and criticisms”. In: *Colombian Applied Linguistics Journal* 15.2, pp. 279–287.

Messina, Angelo, Peter Modigliani, and Su Chang (2015). “How agile development can transform defense IT acquisition”. In: *Proceedings of the 4th International Conference on Software Engineering in Defence Application (SEDA 2015)*.

Meyer, Bertrand (2014). *Agile!: The Good, the Hype and the Ugly*. Springer Science & Business Media.

Mora, Alberto et al. (2015). “FRAGGLE: A FRamework for AGile gamification of learning experiences”. In: *International Conference on Games and Learning Alliance*. Springer, pp. 530–539.

Moreira, Mario E (2013). *Being agile: your roadmap to successful adoption of agile*. Apress.

Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj (2005). “Challenges of migrating to agile methodologies”. In: *Communications of the ACM* 48.5, pp. 72–78.

Nerur, Sridhar et al. (2010). “Towards an understanding of the conceptual underpinnings of agile development methodologies”. In: *Agile software development*. Springer, pp. 15–29.

Northover, Mandy, Andrew Boake, and Derrick G Kourie (2006). “Karl Popper’s critical rationalism in agile software development”. In: *International Conference on Conceptual Structures*. Springer, pp. 360–373.

Northover, Mandy et al. (2007a). “Agile software development: A contemporary philosophical perspective”. In: *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. ACM, pp. 106–115.

Northover, Mandy et al. (2007b). “Extreme Programming: A Kuhnian Revolution?” In: *ICCS 2007*. Springer, pp. 199–204.

Northover, Mandy et al. (2008). “Towards a philosophy of software development: 40 years after the birth of software engineering”. In: *Journal for General Philosophy of Science* 39.1, pp. 85–113.

Nyce, Caroline Mimbs (2017). “The Winter Getaway That Turned the Software World Upside Down”. In: *The Atlantic*. URL: https://www.theatlantic.com/technology/archive/2017/12/agile-manifesto-a-history/547715/.

Owen, R and L Koskela (2006). “An agile step forward in project management”. In: *2nd Specialty Conference on Leadership and Management in Construction and Engineering*, pp. 216–224.

Parsons, David and Kathryn MacCallum (2019). “Agile Education, Lean Learning”. In: *Agile and Lean Concepts for Teaching and Learning*. Springer, pp. 3–23.

Portman, Henny (2019). “A bird’s eye view on the agile forest”. In: *PM World Journal* VIII.X. URL: https://pmworldjournal.com/article/a-birds-eye-view-on-the-agile-forest.

Pribram, Karl H (1990). “From metaphors to models: The use of analogy in neuropsychology”. In: *Metaphors in the history of psychology*, pp. 79–103.

Ralph, Paul (2013). “Software Engineering Process Theory: A Multi-Method Comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory”. In: *arXiv preprint arXiv:1307.1019*.

Ratti, Emanuele (2019). “Phronesis and Automated Science: The Case of Machine Learning and Biology”. In: *Will Science Remain Human?* Ed. by Fabio Sterpetti and M. Bertolaso. Springer.

Rawsthorne, Peter and David Lloyd (2005). “Agile methods of software engineering should continue to have an influence over instructional design methodologies”. In: *Unpublished Cape Breton University & Memorial University of Newfoundland, Retrieved from http://www. rawsthorne. org/bit/docs/RawsthorneAIDFinal. pdf*.

Schuler, H (1996). “Process Dynamics, Modelling, and Control. Babatunde A. Ogunnaike, W. HArmon Ray. Oxford University Press, Oxford 1995, 1260 Seotem. Zahlr. Abb. und Tab.,£ 65.-, ISBN 0-19-509119-1.” In: *Chemie Ingenieur Technik* 68.6, pp. 730–730.

Shapiro, Stuart (1997). “Splitting the difference: the historical necessity of synthesis in software engineering”. In: *IEEE Annals of the History of Computing* 19.1, pp. 20–54.

Sharp, Jason H and Guido Lang (2019). “Agile in teaching and learning: Conceptual framework and research agenda”. In: *Journal of Information Systems Education* 29.2, p. 1.

Shull, Forrest et al. (2002). “Replicating software engineering experiments: addressing the tacit knowledge problem”. In: *Proceedings International Symposium on Empirical Software Engineering*. IEEE, pp. 7–16.

Snowden, Dave (2015). “The table napkin test”. In: *cognitive-edge.com*. URL: https://cognitive-edge.com/blog/the-table-napkin-test/.

Stacey, Ralph (2012). *Tools and techniques of leadership and management: Meeting the challenge of complexity*. Routledge.

Standish, G (2014). “Chaos Report on Software Projects”. In: *Project Smart, The Standish Group, USA*.

Stone, Steven M and Levine (2019). *Digitally Deaf*. Springer.

Stotz, Karola and Paul Griffiths (2004). “Genes: philosophical analyses put to the test.” In: *History and philosophy of the life sciences* 26.1, pp. 5–28.

Sutherland, Jeff (2014). *Scrum: the art of doing twice the work in half the time*. Currency.

Sutherland, Jeff and Ken Schwaber (2017). “The scrum guide”. In: *crum. org* 268.

Symons, John and Jack Horner (2014). “Software Intensive Science”. In: *Philosophy and Technology* 27.3, pp. 461–477. DOI: 10.1007/s13347-014-0163-x.

Tengberg, Lars Göran Wallgren (2015). “The Agile Approach with Doctoral Dissertation Supervision.” In: *International Education Studies* 8.11, pp. 139–147.

Twidale, Michael B and David M Nichols (2013). “Agile methods for agile universities”. In: *Reimagining the Creative University for the 21st Century*. Brill Sense, pp. 27–48.

Vallverdú, Jordi (2010). *Thinking machines and the philosophy of computer science: concepts and principles*. IGI Global.

Van Fraassen, B. C. (1980). *The scientific image*. Oxford University Press.

VersionOne, CollabNet (2019). *State of Agile Survey 2019*. Tech. rep. URL: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508.

Vertesi, Janet and David Ribes (2019). *DigitalSTS: A Field Guide for Science & Technology Studies*. Princeton University Press.

Wang, Yingxu (2003). "On cognitive informatics". In: *Brain and Mind* 4.2, pp. 151–167.

Waters, C. (Feb. 2004). "What concept analysis in philosophy of science should be (and why competing philosophical analyses of gene concepts cannot be tested by polling scientists)". In: *History and philosophy of the life sciences* 26, pp. 29–58. DOI: 10.1080/03919710412331341631.

Weisberg, Michael and Ryan Muldoon (2009). "Epistemic landscapes and the division of cognitive labor". In: *Philosophy of science* 76.2, pp. 225–252.

Windows (2020). "Support for Windows 7 has ended". In: URL: https://www.microsoft.com/en-us/microsoft-365/windows/end-of-windows-7-support.

Zykov, Sergey V (2016). "Software Methodologies: Are Our Processes Crisis-Agile?" In: *Crisis Management for Software Development and Knowledge Transfer*. Springer, pp. 51–68.

Appendices

Appendix A

Scrum description

Scrum was one of the methodologies represented at the Agile origin meeting and is the most used methodology under the Agile banner. (VersionOne 2019). Due to this popularity, in everyday use Scrum has become equated with Agile to the point where the term Agile/Scrum is used as if there is no distinction between the two. To a lot of people developing software today, and to most of the people who first encounter Agile, the answer to the question 'What is Agile?' ends up being Scrum. Therefore, a closer look at Scrum is warranted.
Crucially, the guide states "Scrum is not a process, technique, or definitive method. Rather, it is a framework within which you can employ various processes and techniques." (p.3). Instead of specifying a discrete number of actions or steps, Scrum can thus best be compared to setting up the rules of a game. The framework doesn't actually define a specific game (i.e. the steps of a particular project), only which moves you can make within the game. What follows is a summary of the 2017 Scrum Guide (Sutherland and Schwaber 2017).

A.1 Actors in Scrum

The organizational unit that the Scrum Guide primarily focuses on is the 'Development Team'. This is supposed to be a multi-disciplinary group of persons who together have all the skills needed to create a version of the product. Scrum furthermore defines 2 team roles that serve different functions for the team. The first is the Scrum Master. The main responsibility of this role is to guard the Scrum process. This consists of reminding and educating people inside and outside of the team about the rules and agreements made about how the development process should take place. The second chief responsibility of the Scrum Master is to be a buffer between the team and the wider organization. The Scrum Master is supposed to prevent the development team from interruptions and being burdened with extra work or not directly relevant questions and issues. This role can be fulfilled by a dedicated, full-time team member but it can also be passed around periodically. The second role is the Product Owner. The Product Owner represents the wishes of the client and therefore they have the last word about the priority of the work (in Scrum terms: the order of the Project Backlog). The Product Owner thus also has a buffering function: all questions from both within and without the team are channeled through a single person.

A.2 Concepts and Artefacts in Scrum

A key concept in Scrum is the ‘Sprint’ which is a predefined length of time, for example 2 or 4 weeks. The goal of the Sprint is to produce a ‘Minimal Viable Product’—or MVP—which can then be shared with the client and other relevant stakeholders to gather feedback. A Sprint is essentially one iteration over all the phases (Planning, Design, Implementation, Testing, Deployment) of the software development process. The project is thus supposed to consist of sprints from beginning to end: there shouldn’t be things like Planning Sprint and Testing Sprints, since this would reintroduce the sequential waterfall structure into the process.
The individual requirements are written down in the User Story format: As a [stakeholder] I want [feature/functionality] in order to [achieve some business value]. Each story gets assigned an amount of ‘Story Points’ which are supposed to indicate the size of the task. The Story Point system is a relative measurement scale, meaning that a larger Story is defined as X times the smallest imaginable Story.
The User Stories are written on (physical or digital) cards or post-its. The overall ‘stack of cards’ of the project is called the Project Backlog or Product Backlog. The list of work that the team has committed to doing in the Sprint is called the Sprint Backlog. To keep track of and visualize the progress and status of the work during the sprint these cards are moved on a ‘Scrum Board’. This is a board with multiple columns, usually including ‘to-do’, ‘doing’, ‘testing’, and ‘done’. The team agrees on a Definition of Done, which provides a checklist of what should happen before a Story can be mode to the Done column.
Using the points assigned to the User Stories a graph can be made showing how much work was done when. At the start of the Sprint, the graph shows the amount of points that the team has committed to. As cards get moved to the ‘done’ column, points are subtracted from that total. Over time this forms a downward sloping line, which shows if the team is on track to reach the Sprint goal. This graph is called the Breakdown Chart and can also be made on the project level.

A.3 Recurring Meetings

Built into the Sprint are several recurring meetings, sometimes also called rituals or ceremonies. These are the Sprint Planning, Sprint Review, Sprint Retrospective, Backlog Refinement Meeting, and the Daily Stand-Up. During the Planning meeting, the Team reviews the Project Backlog and selects the User Stories that will make up the Sprint Goal for the coming Sprint. While the Sprint is taking place, the team (and interested stakeholders) have a 15-minute daily meeting in which the team updates the Scrum Board and the team members inform each other about what User Stories were finished yesterday, which stories are expected to be finished today, and what roadblocks they have encountered that other team members may be able to help them with.
At the end of the Sprint the results are presented to the client and other stakeholders and feedback is gathered. This meeting plays an important role in surfacing and aligning assumptions and possible misunderstandings of all parties. The Sprint Retrospective is an ‘internal’ team meeting, where the focus is not the work itself but the development process and way of working. This meeting provides a dedicated moment for reflecting on what practices are working and which can be adjusted.

The Backlog Refinement meeting prepares the backlog for the Sprint Planning meeting. Scrum advocates that the requirements in the Project Backlog should be loosely defined. Before the team can commit to a Sprint goal the User Stories therefore need to be broken down and specified further.

A.4 Team Autonomy

Scrum places a lot of emphasis on autonomy for the development team. Autonomy here is conceptualized as having the power to make decisions about:

- What to work on
- How much work to do
- What the development process should be like

Instead of being instructed what and how to work, Scrum aims to create a team that functions as a contained unit. Team autonomy is threatened by external influences and interests, both from inside the organization and from outside, such as the client. The team roles (Product Owner and Scrum Master), can thus be seen as a way to inoculate this threat by channeling those 'outside forces' in predefined ways, and importantly, away from direct unmediated contact with the development team itself.

One rationale that is offered by Scrum is that the team 'pulling' work itself from the backlog means that the team is structured as a Pull System, as opposed to a Push System. With these terms an appeal is made to Lean manufacturing theory, and the claim is that a Pull System can work faster and more efficient than a Push System.

Appendix B

Literature Research details

The following search phrases were employed:
("Extreme programming" OR scrum OR agile OR waterfall OR DSDM OR "rational unified process")
(Extreme & programming) | scrum | agile | waterfall | DSDM | (rational & unified & process)
(software OR ICT OR IT OR information technology OR information technologies OR digital) AND (organization OR management OR framework OR manifesto)
“software development” OR “software design” OR “software development methodology” OR “software development method” OR “software engineering” OR “software production” OR “software manufacturing” OR “IT development” OR “IT design” OR “IT development methodology” OR “IT development method” OR “IT engineering” OR “IT production” OR “IT manufacturing”

“ICT development” OR “ICT design” OR “ICT development methodology” OR “ICT development method” OR “ICT engineering” OR “ICT production” OR “ICT manufacturing” OR “information technology development” OR “information technology design” OR “information technology development methodology” OR “information technology development method” OR “information technology engineering” OR “information technology production” OR “information technology manufacturing”
“digital development” OR “digital design” OR “digital development methodology” OR “digital development method” OR “digital engineering” OR “digital production”

In total 966 documents were collected. Out of these, 398 were published in software engineering journals or conferences. 89 were published in philosophical journals. 46 were published in STS journals. These were entered into the research support tool Qiqqa and 614 were manually tagged. Additionally, Qiqqa provided automatic tags and sorted the sources into research themes. These themes are:

1. Teams
2. Conference; Case Study; case study; international conference; Analysis
3. Software development; agile software development; software development process; software engineering; Review
4. Agile
5. Agile; Agile Methods; agile methods
6. Systems; Information; information systems; systems development; systems engineering

7. Organization; Change; Culture; Information; Systems
8. Philosophy; Paradigm; computer science; Computer Science; History
9. Analysis; Survey; Introduction; Review; Systems
10. Social; STS; Policy; Analysis; Systems
11. Project management; Agile; SME; Framework; agile project management
12. Software engineering; Computer Science; computer science; Conference; Systems
13. Social; Culture; Information; History; Systems
14. Open source; Open Source; June; software development; Analysis
15. Information; Systems; Ontology; Analysis; Conference
16. Risk; Uncertainty; Analysis; Information; Change
17. Planning; Waterfall; Review; project management; Information
18. Change; Systems; Information; Learning; Framework
19. Agile; agile practices; Agile Practices; software development; agile adoption
20. Methodology
21. Framework; Review; requirements engineering; Analysis; Information
22. Complexity; Systems; Emergence; Creativity; Social
23. Software process; Software Process; software development; Process Model; process improvement
24. Information
25. Scrum; Teams; Planning; Review; software development
26. Lean; lean software; product development; lean software development; Systems
27. xp; XP; extreme programming; Planning; Agile
28. Collaboration; devops; Culture; product development; Analysis
29. Communication; Information; Collaboration; Social; Systems
30. Learning; knowledge management; Social; Planning; Change
31. Grounded Theory; grounded theory; Analysis; Social

The tags, themes and filtering capabilities facilitated locating relevant papers within the database.

While compiling a database of Software Engineering Literature 201 full conference proceedings or individual sources were collected (see table B).

Conference	Abbr.	# collected
International Congress (IC) on Software Process Improvement	CIMPS	8
IC Global Software Engineering	GSW	3
IC Computer Science and its Applications	CSA	16
Software and Data Technologies	SOFT	21
Lean Enterprise Software and Systems	LESS	2
Product-Focused Software Process Improvement	PROFES	72
IC Software Engineering for Defence Applications	SEDA	16
Software Process Improvement and Capability Determination	SPICE	37
IC Transfer and Diffusion of IT	TDIT	28
IC on Agile Software Development	XP	20

Appendix C

Agile Sources Overview

C.1 Books

(First) Author	(Distinct) Name	Goodreads Rating	GR # reviews	Amazon Rating	Amazon # Reviews	Amazon Best Seller	Book Authority	STC	Noop	Total
Jeff Sutherland	The Scrum Guide									
Robert C. Martin	Clean Code	1	1	0	1	1	1	0	0	5
Ken Schwaber	Software Develop-ment	0	0	0	0.5	1	1	1	1	4.5
Gene Kim	The Phoenix Project	1	1	1	1	0	0	0	0	4
Kenneth Rubin	Essential Scrum	0.5	0.5	1	1	1	0	0	0	4
Mike Cohn	User Stories Applied	0	0.5	0	0.5	1	1	0	1	4
Kent Beck	Test Driven Develop-ment	0	1	0	0	1	0	1	1	4
Robert C. Martin	Principles, Patterns, and Practices	1	0	0	0	0	1	1	1	4
Jim Highsmith	Creating Innovative Products	0	0	0.5	0	0	1	1	1	3.5
Jeff Sutherland	Scrum: The Art	0.5	1	0.5	1	0	0	0	0	3
Eric Evans	Domain-Driven Design	0.5	1	0	0.5	1	0	0	0	3

(First) Author	(Distinct) Name	Goodreads Rating	GR # reviews	Amazon Rating	Amazon # Reviews	Amazon Best Seller	Book Authority	STC	Noop	Total
Kent Beck	Extreme Programming	0	0.5	0.5	0	0	0	1	1	3
Alistair Cockburn	Crystal Clear	0	0	1	0	0	0	1	1	3
Gary McLean Hall	Adaptive Code	1	0	1	0	1	0	0	0	3
Chris Sims	Scrum: a Brief Introduction	0	0.5	0	1	0	1	0	0	2.5
Ken Schwaber	Project Management	0	0	0	0.5	0	0	1	1	2.5
Mike Cohn	Succeeding with Agile	0	0.5	0	0	1	1	0	0	2.5
Andrew Stellman	Learning Agile	0.5	0	0	0	1	1	0	0	2.5
Mitch Lacey	Scrum Field Guide	0.5	0	1	0	0	1	0	0	2.5
Gene Kim	The DevOps Handbook	1	0	0.5	0.5	0	0	0	0	2
Ron Jeffries	Extreme Programming Installed	0	0	0	0	0	0	1	1	2
James O. Coplien	Organizational Patterns	0	0	0	0	0	1	1	0	2
Eric Brechner	Project Management with Kanban	0	0	0.5	0	1	0	0	0	1.5
Ron Jeffries	Nature of Software Development	0	0	0	0	0	1	0	0	1
Jeff Langr	Agile in a Flash	0	0	0	0	0	1	0	0	1
Alistair Cockburn	Agile Software Development	0	0	0	0	0	0	0	0	0

(First) Author	(Distinct) Name	*Goodreads Rating*	*GR # reviews*	*Amazon Rating*	*Amazon # Reviews*	*Amazon Best Seller*	*Book Authority*	*STC*	*Noop*	*Total*
Alan Shalloway	Lean-Agile	0	0	0	0	0	0	0	0	0
Jim Highsmith	Ecosystems	0	0	0	0	0	0	0	0	0
Alexis Leon	Evaluating the Methods	0	0	0	0	0	0	0		

C.2 Conferences

GOTO
Agile Singapore
Agile Australia
OSCON
Agile Africa
YOW!
Qcon
Norwegian Developers Conference
Agile Prague
Lean, Agile & Scrum
Agile Tour
Agile Serbia
Agile by Example
Better Software
Agile Open
Agile on the Beach
Agile NZ
Agile Turkey
Agile India
Agilia Conference

Atlassian Summit
Deliver: Agile
Global Scrum Gathering
Mile High Agile
Agile and Beyond
Agile + Devops
Agile Alliance
XP
ACE!
Agile NL
Agile in the City
Agile Trends
Lean Agile
Agile London

C.3 Blogs

Blogs	Individual		Scrum-master-toolbox.org	Ignite	2017 Alexa Ranking	Feedspot	Traffic (5 month average)	Sites linking-in
Martin Fowler	https://martinfowler.com	5	1210000	4716	697	56	56	19
Mike Cohn	https://www.mountaingoatsoftware.com/blog	8	3	491000	715	684	53	5
Alistair Cockburn	http://alistair.cockburn.us/	30	70000	687	218	41	41	17
Dan North	dannorth.net/category/agile/	50000	341	456	28	28	16	
Roman Pichler	http://www.romanpichler.com/blog/	16	120000	305	249	44	44	15
Gojko Adzic	https://gojko.net/posts.html	x	60000	282	176	30	30	14
Bob Martin	https://blog.cleancoder.com/	150000	248	152	30	30	13	
Johanna Rothman	https://www.jrothman.com/articles/	44	40000	238	300	28	28	12
Mark Needham	http://www.markhneedham.com/blog/category/agile/	18	148130	229	50	41	41	11
Ron Jeffries	https://ronjeffries.com	46	51830	162	757	28	28	10
Jeff Sutherland	https://www.scruminc.com/scrum-blog/page/36/	x	74930	136	396	29	29	9
Jonathan Rasmusson	https://agilewarrior.wordpress.com/articles/	x	19	60000	127	13	32	32
Jeff Patton	https://www.jpattonassociates.com/blog2	40000	108	78	17	17	7	
Liz Keogh	lizkeogh.com	30000	83	466	13	13	6	
Ken Schwaber	https://kenschwaber.wordpress.com/	61	300000	73	238	31	31	5
Luis Concalves	luis-goncalves.com	12	125330	51	148	20	3	23
Chris Matts	https://theitriskmanager.com/	50000	2	245	12	12	3	
J.D. Meier	https://blogs.msdn.microsoft.com/jmeier	1	1214	23	23	2		
Dave Snowden	https://cognitive-edge.com/blog/author/dave-snowden/	35	2840	14	14	1		

Blogs	Individual		*Scrum-master-toolbox.org*	*Ignite*	*2017 Alexa Ranking*	*Feedspot*	*Traffic (5 month average)*	*Sites linking-in*
Organizations								
Scrum.org	https://www.scrum.org/resources/blog	x	7	6	1040000	557	1080	
ScrumStudy	https://www.scrumstudy.com/blog	17	132000	1392	453			
Target Process	https://www.targetprocess.com/blog/category/agile/	13	118070	290	190			
Xebia	http://blog.xebia.com/	15	144770	352	357			
Version One	https://blog.versionone.com/	11	70000	412	1720			
Scaled Agile	http://www.scaledagileframework.com/blog/	10	450000	492	340			
CA Technologies	https://www.ca.com/en/blog-highlight/tag/agile-management	9	235180	1140	131			
Atlassian	https://www.atlassian.com/blog/agile	4	1510000	3116	156			
Hackernoon	https://hackernoon.com/tagged/agile	2	21240000	6795	990			
Medium	https://medium.com/tag/agile	13830000	154	581				
ThoughtWorks	https://www.thoughtworks.com/blogs	225320000	9211	48				
InfoQ	https://www.infoq.com/agile/articles/	600000	1256	1154				
Agile Zone	https://dzone.com/agile-methodology-training-tools-news	2080000	5377	8650				
Agile Alliance	https://www.agilealliance.org/community/blog	x	3	12510000	5569	231		
Scrum Alliance	https://www.scrumalliance.org/education-blog	x	12	1	304720	763	43	

C.4 Social Media

Twitter	Followers	Tweets
Martin Fowler	266711	7857
Kent Beck	140521	12911
Robert C. Martin	125448	23919
Dan Tousignant	58295	64460
Jeff Sutherland	45517	2438
Mike Cohn	44121	4887
Jez Humble	41029	11674
Gene Kim	39903	30806
Ron Jeffries	35293	72488
Dan North	33339	32743
Ken Schwaber	30743	345
Dave Snowden	25391	32451
Alistair Cockburn	25030	25926
Eric Evans	23812	2151
Kane Mar	18599	3155
Nicole Forsgen	17043	14271
Liz Keogh	13522	18316
Roman Pichler	11794	1682
Jim Highsmith	11159	861
Geoff Watts	7290	15921
Alan Shalloway	6686	57646
Chris Matts	5166	9913
Ken Rubin	4411	3501
Michele Sliger	4141	17752
Mitch Lacey	4077	4313
Jeff Langr	1934	9754
Chris Sims	1878	5188
Ilan Goldstein	1538	3033
Andrew Stellman	729	3542
Gary McLean Hall	480	2186