Trading Discipline for Agility? Questioning the Unfaithful Appropriation of Agile Software Development Practices

Akbar M. Saeed
Wilfrid Laurier University, Canada
asaeed@wlu.ca

Abstract

Agile software development practices are rapidly replacing traditional and apparently more disciplined methodologies. However, empirical evidence suggests that organizations experience varying levels of success as more structured processes are traded for more agile ones. Using an autoethnographic approach, we reflect on how the various practices of XP discipline time-space relations amongst developer, customer and code. In this new form of disciplining, we contend that each actor is located in time and space in disciplined or controlled ways. We conclude that the faithful appropriation of the entire complement of agile development practices seems to be critical to the novel disciplinary positioning that they together collectively promote.

1. Introduction

In today's uncertain business environments, organizations are increasingly relying on information technology to enhance their agility and allow them more flexibility in choosing strategic direction [1][2]. Consequently, information systems need to be regularly modified to support emergent business needs [3][4]. Traditional software development processes have been unable to respond effectively to the rapid pace of change in the business environment due to the overwhelming focus on documentation and process [5]. As a result, many software projects seem to become outdated even before they are finished [6]. In an attempt to cope with such volatility, many managers are looking to 'lighter' more agile development methodologies to promote manoeuvrability and speed of response [1][7][8][9].

A recent industry survey found that 88% of 3051 respondent organizations were practicing some form of agile management techniques to develop internal applications [10]. However, upon closer inspection, this appropriation seems to be occurring in an unfaithful manner as companies adapt only particular agile development principles to suit the needs of different contexts [11]. The appropriation perspective asks "[...] whether people use the technology as its designers […] intended" [12] p15. The user has the ability to deviate from what was originally intended by the designer and use the features in a different way. Desanctis and Poole originally referred to this as unfaithful appropriation [13]. For example, even though Daily Standup meetings were practiced by 85% of respondent organizations, only 55% were using Coding Standards, only 30% were using Pair Programming and only 25% were using Continuous Deployment. In other words, managers seemed to be adopting those agile methods that aligned well with pre-existing company processes. This is not surprising considering empirical evidence that suggests the leading cause of failed agile projects was 'company philosophy or culture at odds with core agile values' [10]. Similar findings occurred in a recent case study of a software startup organization that had faithfully adopted some XP techniques while blending others with traditional approaches (see Table 1) [14]. Overall, evidence from the field suggests that most organizations are using a more blended approach [15][16], as agile methods are often integrated with some upfront design and formal methods often involve some form of iteration [17]. Without doubt, the promises of faster time to market, better management of changing priorities and better alignment of IT/business objectives will ensure that agile approaches continue to grow in popularity amongst software development organizations [10]. However, academic research on agile methodologies is still lacking as a more theory-based approach to scholarship in this area is urgently needed [18].

TABLE 1.
ADOPTION FAITHFULNESS OF XP PRINCIPLES

XP Principle	Adoption Level	Summary
40-Hour Work Week	Full	Developers worked flexible but regular workdays.
Coding Standards	Low to Partial	Standards were initially avoided but later implemented.
Collective Code Ownership	Partial	Code was officially shared but developers exhibited possessiveness.
Continuous Integration	Full	Code was rarely broken and was continually linked and compiled.

URI: http://hdl.handle.net/10125/41873
ISBN: 978-0-9981331-0-2

Continuous Testing	Partial to Full	Testing was continuous but advance scripts were not created. Black box testing was phased.
On-Site Customer	Full	The CEO and Analytic Director acted as customers.
Pair Programming	Low	Programmers were independent except when difficulties or interdependencies existed.
Planning Game	Full	Value engineering balanced features against time and budget.
Refactoring	Full	Modules were constantly improved. Periodic bursts of and dramatic improvement occurred.
Simple Design	Full	Working software was favored.
Small Releases	Full	Frequent (weekly) build cycles
System Metaphor	Full	Communication was simple and informal but unambiguous.

Adapted from [14].

This paper unfolds as follows. We begin by locating the traditional approach to software development within the fabric of history that has colored its evolution. Next, we attempt to identify the traditional means by which discipline is understood. The Capability Maturity Model (CMM), a methodology designed to foster process discipline in the software development process, will aid with such an identification. We then use the declaration known as the Agile Manifesto (www.agilemanifesto.org) to illustrate how agile development methods have been defined in opposition to structured, apparently more disciplined methodology. Consequently, we will imply that an absence of that which is considered good discipline results in current popular characterizations of agile development methodology. We then proceed to disrupt this implied continuum by suggesting that agile methods do not exist in binary opposition to structured methods, but instead, they actualize a different form of discipline. By way of example, we use first-hand experiential observation data to model how the agile programming methodology Extreme Programming (XP) unobtrusively disciplines the time-space relations between its actor entities (the developer, the customer and the code). In this new form of disciplining, we contend that each actor is located in time and space in disciplined or controlled ways. We conclude that the faithful appropriation of the entire complement of agile development practices seems to be critical to the novel disciplinary positioning that they together collectively promote.

2. Software development approaches

2.1. The structured approach

In 1968, over fifty experts of the NATO Science Committee convened to plot a future course for what was then a fledgling software industry. They specifically discussed the looming 'software crisis' resulting from the production of software with low quality and a lack of reliability [18]. Unable to chart an adequate course, they nonetheless agreed to define the field of software engineering as 'the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software' [5] p87. Consequently, the software engineer was born into a somewhat nefarious world, thrust into an unfamiliar milieu where tried and trusted engineering methodology was heralded as the 'silver bullet' that could finally slay the legendary werewolf [19].

In reaction to the pending crisis and a need for greater legitimacy, the software industry collectively centred on defining a more disciplined software development process [20]. Consequently, four distinct areas were defined: requirements gathering, designing and developing of the software, testing the results, and overall project management [4]. Through the years, many rational methods have been forged along these trajectories e.g. waterfall, prototype, iterative, rapid application development and spiral-based methodologies [21]. Referred to as heavyweight methodologies, due to their heavy documentation and process orientation, they attained legitimacy by drawing on well established engineering principles. Predictably, software development came to be characterized as a prescriptive, deterministic and mathematical process [22]. Structured approaches involved rigid definitions of the roles to be played, activities to be performed and artifacts to be produced [23]. Business requirements were solicited, specified and extensively documented early in the cycle, and the 'freezing' of such requirements, early in the process, was thought to make the application less vulnerable to volatility [24]. Requirements were then coded in a manner that focused on efficiency and as a result process variation was minimized. By keeping the development processes structured, the software product that emerged was an authentic reflection of the original business requirements and 'a way to put some order into the chaotic jumble of thoughts' [25].

From this mode of thinking emerged the waterfall method, aptly described as an 'attempt to put discipline into the software development process by forcing understanding and documentation of the requirements before going on to design, by forcing understanding and documentation of design before going on to coding, by forcing testing of the code while coding each module'

[25] p57. However, such understanding is difficult and elusive, and in particular, documentation is a 'pain' in the highly chaotic software development environment [17][26][25].

The framework of the Capability Maturity Model (CMM) was created to deal with such chaos. Originally developed in 1993 as a tool to objectively assess government contractors' development processes in order to predict their ability to deliver a contracted project within specification and on time, CMM quickly became popular as a general process maturity framework. The Software Engineering Institute of Carnegie Mellon promoted the CMM as a way to define the key elements of what they considered to be an effective software development process, as they argued that the benefits of better methods and tools could not be realized in the 'maelstrom of an undisciplined, chaotic project' [27] p1. Table 2 outlines the maturity levels of CMM, with bolding (not in the original) on those phrases that seem to reflect its underlying disciplinary principles. Notably, the CMM is at once a reference model for appraising software process maturity and a normative model for helping software organizations progress along an evolutionary path from an ad hoc, chaotic process to a mature, disciplined software processes [28]. In other words, CMM provides software organizations with 'guidance on how to gain control of their processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence' [27] p4.

TABLE 2.
MAJOR CHARACTERISTICS OF THE LEVELS OF MATURITY IN CMM

CMM Level	Major Characteristics
1 Initial	The software process is characterized as **ad hoc, and occasionally even chaotic. Few processes are defined**, and success depends on individual effort and heroics.
2 Repeatable	Basic project management processes are established to **track** cost, schedule and functionality. The necessary **process discipline** is in place to **repeat** earlier successes on projects with similar applications.
3 Defined	The software process for both management and engineering activities is **documented, standardized, and integrated into a standard software process for the organization.** Projects use an **approved**, tailored version of the organization's standard software process(es) for developing and maintaining software.
4 Managed	**Detailed measures** of the software process and product quality are collected. Both the software process and product are **quantitatively understood and controlled.**
5 Optimizing	**Continuous process improvements** is facilitated by **quantitative feedback** from the process and from piloting innovative ideas and technologies

Adapted from [27] (bolding not in original)

A quick glance at the description of the different CMM levels reveals some of the key elements that are commonly understood to constitute process discipline. First, development processes must be defined and thereby rendered controllable by traditional project management techniques. For instance, software process maturity is defined as 'the extent to which a specific process is explicitly defined, managed, measured and controlled' [27] p4. These processes must be documented, standardized and integrated into a standard set of software processes. The development process must be used and modified only in controlled and approved ways. Furthermore, detailed metrics must be established and continuous process improvements must be institutionalized using a highly quantitative approach. Overall, the themes of definition, control, documentation, standardization and measurement stand out as being indicative of the process discipline inherent in the CMM reference model.

Despite repeated efforts to engineer improved methodologies, software development continues to be in somewhat of a crisis. Over 30 years ago, Fred Brooks observed the development of IBM's 360 operating system and made the less than intuitive conclusion that adding more people to a project team running late would only make it still run later [29]. These findings brought attention to the possibility that software engineering may be unlike other traditional forms of engineering. Indeed, software is not tangible in the sense that other engineered products are [30]. In a timeless article, Brooks argued that there is no 'silver bullet' approach to making team based software development easy [29]. He contended that knowing what to build was still the single, hardest part of building a software system. Many years later, we are still faced with the 'inevitable pain of software development', as we struggle to deal with increasingly volatile business requirements [25][31]. Even though there is unlikely to be a silver bullet approach, agile software development methodologies have proven to be better at dealing with such requirements volatility, more than many more structured methods [1][32][33][34].

2.2. The agile approach

In business terms, agility can be defined as the ability to 'detect opportunities for innovation and seize those competitive market opportunities by assembling requisite assets, knowledge, and relationships with speed and surprise' [3] p1. Agile methods are usually dynamic, context-specific, aggressively change-embracing and growth-oriented activities [35]. However, in an organizational context, they also need to be skillfully balanced against the need for institutional order. For instance, agile project management balances needs of the highly structured project management process against those of the creative technical team [37].

As suggested earlier, perhaps the major challenge to software engineering is to figure out how to deal with

ever-changing user requirements [25]. Agile methods offer an effective response to such a problem [23][37]. By allowing business requirements to change throughout the development process, rather than freezing them in time, the developed application has a greater chance of satisfying the evolving needs of the user population [38]. Also, the placement of the customer (or customer representative) in close proximity to the development team increases the chances of meeting evolving requirements. In this way, change is embraced, rather than controlled [39].

In 2001, leaders of the agile movement met to create a manifesto that would embody some of the core principles that they had already come to informally embrace (www.agilemanifesto.org). In that manifesto, certain types of activities were given a preference over other more traditional methods of building applications: individuals and interactions were valued over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan.

By valuing individuals and interactions over processes and tools, there is a definite recognition of the social aspects of software engineering [40]. This line of thinking is contra to the historically grounded rational approach described previously. However, it is very much in line with the thinking that has more recently emerged in the field of information systems [1][2][41]. By valuing working software over comprehensive documentation, the focus of development is on delivery of the end product and not as much on the method of getting there. Empirical evidence suggests that developers prefer to reference the code itself over the code's documentation [42]. Again, this is contra to traditional methods that focus on process discipline and heavy documentation in order to faithfully create the product as specified by the business requirements. By valuing customer collaboration over contract negotiation, the customer is placed in close proximity to the development team. In some instances, as in the case study experience described in this paper, a product manager may even substitute for the customer role [43]. However, the key factor is that the customer continually collaborates with the software development team as the product is being developed. In a structured world, contracts specify what needs to be delivered. In an agile world, the customer specifies the needed functionality as the project progresses, founded on an underlying trust between customer and developer [42]. Finally, by valuing response to change over following a plan, the ongoing agility of the development process is sustained. Taken together, those activities encouraged in the Agile Manifesto promote the underlying agile culture that is an integral part of any agile software development process.

3. Trading discipline for agility: an apparent continuum

From the preceding discussion, it appears that those activities more valued in agile development are seemingly in opposition to those valued in traditional development. Individuals, interactions, working software, collaboration and responding to change displaces processes, tools, documentation, contracts and plans. Those items in the latter group are considered to constitute discipline, as confirmed by our discussion on CMM, and those in the former group are considered to constitute agility, as confirmed by our discussion on the Agile Manifesto. Although, it is clear that there is value in the latter items, most agilists seem to value the former items more [17]. The underlying implication is that these items exist in binary opposition to each other along a continuum of sorts. Indeed, those companies that do not achieve a balance and fall on the extreme ends of the apparent continuum can suffer detrimental consequences [2]. For instance, a lack of project management and control was found to be one of the main reasons for many abandoned information systems development projects [45]. Conversely, Microsoft remains flexible by not adopting too many structured software-engineering processes, like CMM or ISO [24].

In the next section of the paper, we attempt to weaken the assumption of an implied continuum between discipline and agility. Our contention is that XP, an exemplary agile development methodology due to its overt specification of core principles, also exudes discipline, albeit in a different way. I will endeavor to show how XP disciplines time-space relations between its actor elements. Actor elements are those entities that are able to influence the development process. This includes human actors like the customer and the developer as well as a non-human element like the code itself. Each of these plays an important role in influencing the way that the XP development process unfolds [4]. My theoretical argument will be illustrated through a discussion of each of the practices of XP and the relationship that is disciplined by that practice. By demonstrating how this form of disciplining is quite unlike traditional forms, I question the apparent continuum that may be hindering the faithful appropriation of agile processes. However, in order to be able to proceed with this argument, I must first briefly discuss the concept of time-space relations.

4. Time-space relations

Historically, management theory has assumed a more scientific view of time, that is linear, chronological, objective, universal, independent, quantifiable and homogeneous [46]. Some researchers have suggested that effective management action has actually been impeded

by such simplistic understandings of time [47]. Others have called for a more pluralist conception that could lead to a fuller appreciation of the diversity in time-ordering systems that occur in organizational life [48]. Time is experienced through a process of temporal structuring that characterizes people's everyday engagement in the world. Furthermore, these temporal structures specify parameters of acceptable conduct and are also modified by the actions they inform, thereby establishing a temporal rhythm [49]. Notably, there may also be different modes of social times, which may exist side by side in an organization [50]. This however can prove to be problematic as temporal asymmetry can be a source of ongoing conflict [51].

As basic categories of human existence, both time and space can be considered fundamental to defining the context in which technology interacts within organizations [51]. A general kind of 'time-space ecology' can perhaps help us better understand the different kinds of interactions that occur between humans and their environments. We suggest that social processes are always situated in a particular time-space context that provides both enabling and constraining influences on these processes [51]. More simply put, all activity is situated as it occurs for a specific duration of time in a specific place [52]. We will now turn our attention towards the specific practices of XP and the time-space relations that exist between them.

5. Extreme programming

Even though XP is not the most widely used of the agile software development methodologies [10], it is perhaps the most disciplined as it specifies core values, core principles and core practices [4]. As our main interest centers on trying to understand how discipline emerges through the performance of agile practice, we therefore chose to focus on XP[1]. XP is highly agile and lightweight by nature, therefore rendering it quite reactive to both the internal and external elements in which it operates. It has 12 core practices that are by no means all-inclusive [6]. These practices are not necessarily new, but take existing principles and practices to extreme levels [39]. Agile methods offer generative rules, a set of principles from which a multitude of practices may be produced and understood [52][53]. They interact in concert with the team that implements them, by specifying a minimum set of practices that should be performed in all circumstances to generate an appropriate practice for a unique situation [6]. Even though it is recommended to start adopting all the practices 'by the book' before adapting any of them [54], the key according to many XP gurus, is to implement all the rules concurrently as they develop a certain synergy when they interact. Being interdependent, the weakness of one practice is made up by the strengths of others [39]. These rules allow a wide range of possibilities as opposed to demanding a priori specification. In fact, XP is self-adaptive in that the rules are meant to change through time and use [54]. Local adaptation of the rules is therefore highly encouraged [39].

In the field of software development, it is often difficult to differentiate a computer program's technical aspects from the influence exerted by the socio-cultural background of the software development team [24]. Likewise, XP can be envisioned as a sociotechnical ensemble that is created by the simultaneous influence of both people and technology [41]. In this view, neither people nor technology deserve a privileged position in shaping ongoing practice but it is the interplay that is important. For the purposes of our argument, we would suggest that there are human and non-human entities that are indispensable to the software production process in XP methodology. A similar view was promoted by Meso and Jain when they suggested that the process of agile development involved interactions amongst three dimensions: stakeholders, process-related guidelines and software artifacts [55].

In XP, there are two main human roles that are usually identified [39]. The first role is played by the developer who is the actual writer of the code. The second role is the customer who is the one that is responsible for deciding the features that are to be included in the product. Typically, the customer would represent the user community and be on the development staff full time [56]. The other entity that is important to consider, in terms of influence in the development process, is the code itself. The code, as a contributing element in the XP development network, enters into relations with both the developer and the customer. Consequently, we suggest that understanding how these relations are controlled in time and space will enable an understanding of the discipline that is inherrent in the XP development principles and practices.

Various actors in XP continually engage in time-space relations with other actors throughout the development process. This is what makes agile projects so challenging to manage, as actors are given the leeway to continously adapt to changing circumstances rather than having rigid controls imposed upon them. As a result, the project manager ends up 'steering from the edge' [57]. However, we argue that if these time-space relations can be disciplined then the activity of the actors themselves will be disciplined and accordingly the development process will end up becoming more disciplined overall. Put differently, agile methods exude a different type of discipline, one that depends more on locating actors in controllable ways rather than relying on what we would

[1] This was also somewhat opportunistic as the author was already employed within a software company that was intent on employing XP practices for the development of software

consider more traditional forms of control. In the next section, we will use the notion of time-space relations to discuss this novel mode of disciplining that we suggest is exerted throughout the XP development process.

6. Extreme disciplining of time and space

With the advent of information technologies in the workplace, traditional forms of managerial power have been displaced by more modern forms [58][59]. Some have argued that technologies structure social relationships within organizations and therefore complement, and occasionally compete with institutional modes of governance [60]. For instance, disciplinary power acts in a largely unobtrusive manner as it regulates movements and establishes calculated distributions [58]. By disciplining time and space, the actions of the body are enabled and constrained in predictable ways. More particularly, the disciplining of time involves the timetable as a central mechanism [58]. Its goal is to eliminate the danger of 'wasting' time and to establish regularity to activities. In this way, a collective and obligatory rhythm is imposed from the outside. On the other hand, the discipline of space involves the distribution and organization of individuals in an analytical area such that 'each individual has his own place; and each place its individual' [58]. Such a linear distribution makes people amenable to discipline and ultimately establishes their presences and absences i.e. where they can and can not be at particular times. Discipline individualizes bodies by a location that does not give them a fixed position, but more accurately distributes them and circulates them in a network of relations. Once individuals are enclosed in identifiable, ranked, serialized and functional spaces, their activity can be more efficiently controlled. Our contention is that through this mode of disciplining time-space relations emerges the so-called 'rhythm' of an XP project emerges [4]. In the next section, we describe our research approach and then go on to discussing our reflections on the way that each of the XP practices helps to discipline the time-space relations between the developer, the customer and the code. We then conclude the paper with a discussion of some implications.

7. Research approach and motivation

This research stems from multiple reflections that were acquired during an extended period of time during which the author played the key role of customer in an XP development process. A case oriented approach was employed which is useful in investigating a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident [61]. As mentioned before, XP relies heavily on contextual factors in order to remain agile and therefore isolation of the development process from its context is quite difficult. The case study occurred in a software development organization, with approximately 150 employees, that developed e-billing software. The paper's author was embedded in the organization for a period of approximately one year during which he was on the management team responsible for adapting XP practices into the software development process. During that time, the author also engaged directly in agile development processes by playing the customer role i.e. the one who represents the interests of the system user, as well as focusing on issues of integration. The motivation of the research was to understand how what seemed to be such a supposedly undisciplined and agile process could remain disciplined. To that end, the author engaged in a form of 'autoethnographic' research which constitutes an approach to research and writing that seeks to describe and systematically analyze personal experiences in order to understand cultural experience [62]. Our interest was to uncover the means by which the XP system disciplined its actor entities. Our contention was that this disciplining was an element of 'XP culture', exuding from norms, and as such, the author's personal experiences were importantly a part of the very same system that was being studied. In other words, reflections on being disciplined and observations of how other entities (human and non-human) were being disciplined were critical and formed the basis of the assertions presented in this paper. Overall, this study importantly contributes to an area of research where there seems to be a serious lack of attention to the theoretical underpinnings of agile development practice [34].

Data collection occurred over a period of approximately a year during which informal discussions about the development process were conducted with various company employees at various levels in various jobs. Also, many observations were made of how the XP process unfolded in relation to those who were enacting it. The specific focus of the research was to reflect on the means through which discipline was enacted. As a form of ethnography, the author was studying and reflecting upon agile development culture in order to uncover some of its relational practices, shared values and beliefs, and shared experiences [63] that specifically related to disciplining. Some of these cultural values have already been studied in some detail using an ethnographic approach [42] and others have surmised that XP culture may have five key values: communication, simplicity, feedback and courage, with respect underlying the previous four [64]. Through the observation of the unfolding of the development process, the author both experienced and observed his own and others' co-participation within the ethnographic encounter such that 'both the self and others are presented together within a single narrative ethnography, focused on the character and

process of the ethnographic dialogue' [65] p69. In the next section, we describe our findings about how time-space relations are disciplined in the XP development system.

8. The disciplining of time-space relations in the XP development system

As mentioned earlier, XP consists of three main roles and twelve key principles. In Fig. 1, these principles are mapped out along the particular time-space relation that we suggest they discipline. We will now proceed to describe how each time-space relation is disciplined by describing XP principles (italicized in the discussion) that pertain to that particular relation.

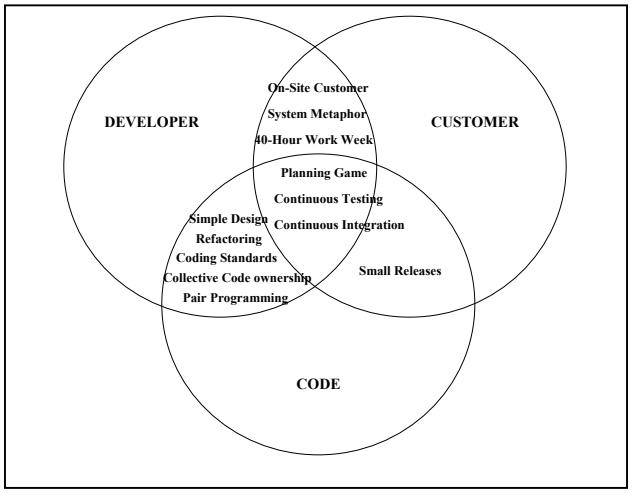

Figure 1 Mapping of XP practices

8.1. The developer and the code

Time-space relations between the developer and the code are disciplined through the enactment of several practices that define the way that the developer must approach the creation of the code. The principle of *Simple Design* encourages designs that marginally satisfy the specified functional requirements that appear in the form of user stories. This limits the tendency of developers to over design [66]. *Refactoring* is the activity of regularly reviewing the code to remove redundancy, eliminating unused functionality and rejuvenating obsolete designs thereby leading to a certain efficiency of expression. This activity occurs with confidence, as the continual running of the test code ensures that no pre-existing functionality is broken. Consequently, any changes to code that disrupt functionality will be surfaced immediately. *Coding Standards* ensure that all written code is formatted and written in the same way. XP uses a common system of names and a coding standard for all developers, thereby improving cross-communication. For a team to work effectively in pairs, and to share ownership of all code, programmers need to write code in the same way, with generative rules that make sure the code communicates clearly. Unlike traditional methods where developers 'own' particular code segments, in XP there is *Collective Code Ownership* where the whole team is responsible for all the code. This collective ownership is essential in order to support refactoring and scheduling activities. Finally, in *Pair Programming*, arguably one of the most researched principles of XP [67], two developers work together at one computer, continuously collaborating on the same design, algorithm, code or test [68]. This has been shown to improve productivity and quality, as together the developers are more than twice as fast at programming, think of more than twice the number of solutions to problems, have a higher defect prevention and defect removal rate and overall they learn more [68, 69]. This may not be intuitive to many project managers, just as Brooks observation that putting more people on a project team would only make it run later [29].

Taken together, these aforementioned practices discipline the time-space relations between the developer and the code that he/she produces. Looked at from another angle, they help establish the norms that govern the way that the developer should interact with the code, thereby specifying the correct and acceptable approach to development work. In this way, the relationship between the code and the developer is spatially and temporally disciplined within the work environment.

8.2. The developer and the customer

Having an *On-Site Customer*, or a customer representative on-site, changes the overall dynamic of software development. In structured approaches, the business requirements are specified upfront. As discussed previously, requirements are not allowed to change and the customer is often in an antagonistic role when functionality is not delivered that should have been there. Even the physical distance between the user and technical support group can influence how effectively problems are solved, or knowledge is transferred [70]. The customer, being in close proximity to the development team, is able to work as a part of the development team. Space and consequently time is disciplined in this way. The *System Metaphor* is a simple analogy for what the system should be like or do. It allows for a reduction in documentation but more importantly it allows for a more interpretive understanding of the product to be built. The metaphor enables at the same time as it constrains. The *40 hour workweek* is another example of disciplining time, as anything other than voluntary overtime has an immediate and dramatic negative effect on productivity [66]. Both developer and customer mutually understand that activities must be executed efficiently to adhere to a limited forty-hour workweek.

8.3. The customer and the code

Small Releases discipline the time-space relationship between the customer and the code. Traditionally, the customer would not have any relationship with the code until the final product is delivered. In XP, there are small releases of functional code as the system is driven into working form in every iteration [4]. In this way, the 'project proceeds in a steady rhythm of delivering more functionality' [4] p45, and also any problems are quickly made visible. Through this visibility, the customer is able to exert influence to steer the development of the product. Decisions can be made as to when the product has enough functionality to be released. This gives heightened control to the customer who previously was practically helpless in trying to get an unfinished product released. This is a notable advantage, especially in turbulent environments that require companies to be highly agile and reactive to competition [2][3].

8.4. The developer, the customer and the code

The *Planning Game* disciplines both release planning activities and iteration planning activities. The release plan is typically done before the project is started, whereas the iteration plan occurs at the beginning of each iteration. By having regular meetings at the beginning of each iteration, and daily stand-up morning meetings, a certain temporal rhythm is established [4]. This temporal rhythm makes project progress quite visible and therefore correctable if needed. Traditional approaches often result in problems not being identified until well into a project. In the *Planning Game*, the developer and customer together determine what is feasible and desirable to get built in each iteration and consequently in the release. Importantly, this is by no means a fixed proposition, like the 'frozen' requirements of structured methods. The customer always has the option of adding more functionality into the product, thereby requiring more time, or releasing a product as is [71]. Often times, during the product development process, the customer comes to realize what is possible which results in the need for new requirements. With regular regimented communication, both emerging requirements and problems are uncovered quickly and are dealt with collaboratively. More developers may in fact be assigned to a priority activity that is falling behind or an in-trouble developer may be assigned less work in a following iteration. Either way, there is built-in dynamism in the process as inefficiencies are uncovered in a timely fashion and dealt with swiftly. In the planning meetings, business requirements are typically specified in the form of user stories, which represent features that the customer desires in the final product. The user stories are ideally business-oriented, testable and estimable [39]. They initially appear on index cards (or an equivalent) with a description of the feature in the language and terms used by the customer. The user story has been eloquently described as 'a promise of a conversation' between developer and customer [39]. With the collaboration of the developer, who accepts and 'owns' the assignment, the amount of time required to complete the job is estimated. However, the exact method of execution is left to the creativity of the developer.

The principles of *Continuous Testing* and *Continuous Integration* are also very crucial. One of the less than intuitive principles of XP is that test code is written first. A developer chooses a particular user story to work on, deduces the technical tasks required to make it happen and then writes test code. This test code is based on functional criteria specified by the customer. Initially the test code fails until needed supporting code is implemented. Overall, the test code ensures that the particular user feature that it represents is still functioning even as code surrounding it is constantly modified. The functional test mitigates some of the inherent risks of the highly interdependent programming languages of today [72]. The developer can run builds on their own machines before integrating changes into the main body of code. The old approach to builds was a problematic process, as haphazard integration would break the build easily. As a result, developers would waste countless hours trying to understand which particular piece of code broke the build. In XP, the developer tests new code against his/her own local build on a local machine. The build test code is mainly made up of previously specified functional tests. Once this runs satisfactorily, the new code is integrated into the main build on the build machine.

Any broken tests are dealt with daily thereby further contributing to the temporal rhythm. Through an automated unit testing framework, unit tests and functional tests can be run on a continuous basis [73]. All efforts are focused on delivering working software in a timely fashion and keeping it working. The test scripts are crucial to such an initiative as they keep the code 'clean' as new code is added to existing code. This also allows for business requirements to change as needed, with minimal disruption, as the code itself serves as documentation. In traditional methods, documentation becomes quickly outdated as problems are encountered in development, workarounds are devised and often not documented. *Continuous Integration* also reduces the introduction of bugs by mandating that the developers merge their code, only after it has been tested on a local machine. Indeed, developing a disciplined and automated build process is essential to a controlled project [72].

The regular planning activities of the *Planning Game*, combined with *Continuous Testing* and *Continuous Integration* approaches, discipline the time-space relationships that exist between the developer, customer and code. In other words, these practices work together to

control the rhythm of the overall project. The *Planning Game* imposes a temporal rhythm on the work, as well as controlling the ongoing development of the code and the recurring activities of the developers. The user stories identify the functionality that is to be developed, in what time frame it is to be accomplished and also which developer is assigned to it. Developers become assigned to working in specific spaces thereby making them more amenable to disciplinary tactics. *Continuous Testing*, enacted in an automated and regular fashion by the test server, allows for a disciplining of the evolving code and the functionality that the code is consequently able to deliver.

9. Developing the future

This paper responds to a call for using more theory-based approaches in the study of agile methodologies [34]. We began by presenting evidence from extant research that organizations may not be faithfully appropriating agile practices in practice [16]. Then, structured methodology was compared to agile methodology in efforts to understand what kinds of practices constituted discipline and what kinds of practices consituted agility. The apparent continuum between the two was described and then we proceeded to dismantle it by suggesting that discipline is not traded but transformed. By way of example, we used the agile methodology of XP to show how its practices discipline the time-space relations between its actor elements, namely the developer, the customer and the code. We tried to grasp the essence of this different form of disciplinary power with the awareness that this type of discipline cannot be easily understood using traditional frames of reference. Our conclusion was that the developer, the customer and the code are located in time and space in very disciplined and controlled ways. Both are individualized by location that does not necessarily fix them, but distributes them and circulates them in a network of relations [58]. It is the apparatus of XP, as a whole with all its practices, that embodies power and distributes individuals in this permanent and continuous field. The end result is that a heterogeneous mass of actors are turned into a homogenous controllable social order, with disciplined time-space relations, through the combined influence of the various XP practices. Similarly, others have suggested that core XP practices may interact with each other, thereby confounding efforts to study them individually [67].

In today's era of increasingly rapid change, there is a distinct need to balance discipline and agility in software development initiatives [74]. As information technology plays an increasingly bigger part in the viability of companies, the software industry may arguably become one of the world's most important [75]. As we have endeavored to show, agile methodologies can also be quite disciplined. However, many have alternate conceptions of agile methodologies that may have unstated political rationales. Developers often view agile processes as an attempt to micromanage them because of the much more frequent interactions. Also, many project managers are reluctant to surrender the feeling of control that Gantt charts and other plan-driven process artifacts give them [76]. Corporate users seem to be only interested in those aspects of agile development that address their particular problems [77].

As we alluded to in the introduction, evidence from the field suggests that most organizations are using a more blended approach to their agile adoption, in that they are adopting those principles that align well with pre-existing processes and company culture [15][16]. This is a very intuitive and practical approach. Some researchers have even developed a conceptual framework to guide more effective agile method tailoring [78]. However, our analysis clearly shows that each of the twelve XP principles disciplines a different relationship, with many reinforcing the effect of others to attain synergies. By not adopting most, if not all, of the principles we contend that the overall disciplnary effect may be weakened. Each of the developer-customer-code relationships we described needs to be disciplined if the agile method is to work effectively and attain a highly productive rhythmic tempo.

In order for agile development to be more faithfully appropriated, employees and managers will have to understand that agile methods exude a different form of discipline. For instance, project managers will have to focus less on schedules, allowing customers to have more say in the direction of development and the assignment of programmers to tasks. Instead, they will have to concentrate more on other activities like maintaining a temporal and spatial rhythm in the development process. This involves giving up control on outcome and focusing more on managing process. Once the principles are more faithfully adopted, the process needs to be trusted more.

As our field research mainly involved the study of an XP development process, further research on different forms of agile software development methods, and the inter-action between their actor entities is needed. In this way, we can better understand how to balance, and not trade, discipline with agility.

10. References

[1] Lee, G., and W. Xia. 2010. "Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility". *MIS Quarterly* 34(1): 87-114.

[2] Chen, Y., Y. Wang, S. Nevo, J. Jin, L. Wang, and W. Chow. 2014. "IT capability and organizational performance: the roles of business process agility and environmental factors". *European Journal of Information Systems* 23(3): 326-342.

[3] Sambamurthy, V., A. Bharadwaj, and V. Grover. 2003. "Shaping Agility through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms". *MIS Quarterly* 27(2): 237-263.
[4] Lindstrom, L., and R. Jeffries. 2004. "Extreme Programming and Agile Software Development Methodologies". *Information Systems Management* 21(3): 41-52.
[5] Gibbs, W. 1994. "Software's Chronic Crisis". *Scientific American* (Sep): 86-96.
[6] Highsmith, J. 2002. *Agile Software Development Ecosystems*. Addison-Wesley.
[7] Highsmith J., and A. Cockburn. 2001. "Agile Software Development: The Business of Innovation". *Computer* 34(9): 120-122
[8] Williams, L., and A. Cockburn. 2003. "Agile Software Development: Its about feedback and change". *Computer*, June.
[9] Cockburn, A., 2007. *Agile Software Development: The Cooperative Game*. Addison-Wesley.
[10] Version One. 2014. "8th Annual State of Agile Survey". Retrieved Jan. 7, 2015 from Version One Web site: http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf
[11] Cao, L., K. Mohan, P. Xu, and B. Ramesh. 2009. "A framework for adapting agile methodologies". *European Journal of Information Systems* 18: 332-343
[12] Leonardi, P., and S. Barley. 2010. "What's Under Construction Here? Social Action, Materiality, and Power in Constructivist Studies of Technology and Organizing". *The Academy of Management Annals* 4 (1): 1-51.
[13] DeSanctis, G., and M. Poole. 1994. "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory". *Organization Science* 5 (2): 121-147.
[14] Saeed, A., and P. Tingling. 2013. "Extreme Programming beyond Adoption: A Longitudinal Case Study of a Software Start-up". *International Journal of Business and Management Invention* 2(7): 89-95
[15] West, D., and T. Grant. 2010. "Agile Development: Mainstream Adoption has Changed Agility". *Forrester Research Inc*. January 20. http://pmshow2012.programmedevelopment.com/public/uploads/files/forrester_agile_development_mainstream_adoption_has_changed_agility.pdf.
[16] Diebold, P., and M. Dahlem. 2014. "Agile practices in practice: a mapping study". In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* 30. ACM.
[17] Vinekar, V., and C. Huntley. 2010. "Agility vs. Maturity: Is there really a tradeoff?". *IEEE Computer* 43.5
[18] Georgiadou, E. 2003. "Software Process and Product Improvement: A Historical Perspective". *Cybernetics and Systems Analysis* 39(1): 125-142.
[19] Brooks, F., 1987. "No Silver Bullet: essence and accidents of software engineering". *Computer* 20(4): 10-19.
[20] Lehman, M. 1989. "Uncertainty in computer applications and its control through the engineering of software". *Journal of Software Maintenance: Research and Practice* (1): 3-27.
[21] Pressman, R. 2001. *Software Engineering: A Practitioner's Approach*. McGraw Hill.
[22] Robinson, H., P. Hall, F. Hovenden and J. Rachel. 1998. "Postmodern Software Development." *The Computer Journal* 41(6): 363-375.
[23] Germain, E., and Robillard, P. (2005). "Engineering-based process and agile methodologies for software development: a comparative case study". *The Journal of Systems and Software* (75): 17-27.
[24] Cusumano, M., and Selby R. 1997. "How Microsoft builds software". *Communications of the ACM* 40(6): 53-61.
[25] Berry, D., 2002. "The Inevitable Pain of Software Development: Why there is no silver bullet". In *Proceedings of the 2002 Radical Innovations of Software and Systems Engineering in the Future*, 50-74.
[26] Ackoff, R., 1967. "Management Misinformation Systems". *Management Science* 14 (4): 147-156.
[27] Paulk, M., Curtis, B., Chrissis, M. and Weber, C. (February, 1993). Capability Maturity Model for Software, Version 1.1. Retrieved on Jan. 7, 2015 from Software Engineering Institute Web site: http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html.
[28] Herbsleb J., D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk. 1997. "Software quality and the capability maturity model". *Communications of the ACM* 40(6): 30-40
[29] Brooks, F., 1974. *The Mythical Man-month*. Reading,MA: Addison-Wesley.
[30] Schneider, J., and L. Johnston. 2005. "Extreme Programming - helpful or harmful in educating undergraduates?" *The Journal of Systems and Software* 74: 121-132.
[31] Gupta, S., R. Soni, A. Jolly, and A. Rana. 2012. "Optimized Approach to Software Release Planning with Volatile Requirements". *European Scientific Journal 8*(23): 13-21
[32] Duggan, E. 2004. "Silver Pellets for Improving Software Quality". *Information Resources Management Journal* 17(2): 1-21.
[33] Conboy, K. 2009. "Agility from first principles: reconstructing the concept of agility in information systems development". *Information Systems Research* 20: 329–354.
[34] Dingsoyr, T., S. Nerur, V. Balijepally, and N. Brede Moe. 2012. "A Decade of Agile Methodologies: Towards explaining Agile Software Development". *Journal of Systems and Software* 85(6): 1213-1221.
[35] Goldman, S., R. Nagel, and K. Preiss. 1995. *Agile Competitors and Virtual Organizations*. New York: Van Nostrand Reinhold.
[36] Chin, G., 2004. *Agile project management: how to succeed in the face of changing project requirements.* AMACOM.
[37] Maruping, L., V. Venkatesh, and R. Agarwal. 2009. "A control theory perspective on agile methodology use and changing user requirements". *Information Systems Research 20*(3): 377-399.
[38] Leffingwell, D. 2010. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise.* Addison-Wesley Professional.
[39] Beck, K., 1999. "Embracing Change with Extreme Programming". *Computer* 32 (10): 70-77.
[40] Low, J., J. Johnson, P. Hall,F. Hovenden, J. Rachel, H. Robinson, and S. Woolgar. 1996. "Read this and change the way you feel about software engineering". *Information and Software Technology* 38: 77-87.
[41] Beath, C., N. Berente, N. Gallivan, and K. Lyytinen. 2013. "Expanding the Frontiers of Information Systems Research: Introduction to the Special Issue". *Journal of the Association of Information Systems* 14 (4-5): i-xvi.
[42] Sharp, H., and H. Robinson. 2004. "An Ethnographic Study of XP Practice". *Empirical Software Engineering* 9: 353-375.

[43] Martin, A., R. Biddle, and J. Noble. 2004. "The XP customer role in practice: three studies". *In proceedings of ADC 2004*, Salt Lake City, June.
[45] Ewusi-Mensah, K. 1997. "Critical Issues in Abandoned Information Systems Development Projects". *Communications of the ACM* 40(9): 74-80.
[46] Kavanagh, D., and L. Araujo. 1995. "Chronigami: Folding and unfolding time". *Accounting, Management and Information Technologies 5*(2): 103-121.
[47] Crossan M., E. Cunha, D. Vera, and J. Cunha. 2005. "Time and Organizational Improvisation". *Academy of Management Review* 30(1): 129-145
[48] Whipp, R. 1994. "A Time to Be Concerned: A position paper on time and management". *Time & Society* 3(1): 99-116.
[49] Orlikowski, W. and J. Yates. 2002. "It's about time: Temporal structuring in organizations". *Organization Science* 13(6): 684-700.
[50] Nowotny, H. 1992. "Time and social theory". *Time & Society* 1(3): 421-454.
[51] Lee, H., and E. Whitley. 2002. "Time and Information Technology: Temporal Impacts on Individuals, Organizations and Society". *The Information Society* 18: 235-240.
[52] Giddens, A. 1984. *The Constitution of Society: Outline of the Theory of Structuration.* Berkeley: University of California Press.
[53] Chomsky, N., 1986. *Knowledge of language : its nature, origins, and use.* New York: Praeger.
[54] Fowler, M. 2001. "Variations on a Theme of XP". Accessed January 5 2015 from martinfowler.com: http://martinfowler.com/articles/xpVariation.html.
[55] Meso, P., and R. Jain. 2006. "Agile software development: adaptive systems principles and best practices". *Information Systems Management 23*(3), 19-30.
[56] Cockburn, A., and J. Highsmith. 2001. "Agile Software Development: The People Factor". *Computer* 34 (11): 131-133.
[57] Augustine, S., B. Payne, F. Sencindiver, and S. Woodcock. 2005. "Agile project management: steering from the edges". *Communications of the ACM* 48 (12): 85-89.
[58] Foucault, M. 1977. *Discipline and Punish: The Birth of the Prison.* New York: Vintage Books.
[59] Zuboff, S. 1988. *In The Age of The Smart Machine*. Basic Books.
[60] Kallinikos, J., H. Hasselbladh, and A. Marton. 2013. "Governing social practice". *Theory and society* 42(4): 395-421.
[61] Yin, R. 1994. *Case Study Research: Design and Methods*, Thousand Oaks, CA, Sage Publications.
[62] Ellis, C., T. Adams, and A. Bochner. 2011. "Autoethnography: An Overview". *Forum:: Qualitative Social Research* 12(1).
[63] Maso, I. 2001. "Phenomenology and ethnography". In *Handbook of ethnography,* edited by Atkinson P., A.
Coffey, S. Delamont, J. Lofland, and L. Lofland, 136-144, Thousand Oaks, CA, Sage.
[64] Beck, K., 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Pub.
[65] Tedlock, B. 1991. "From participant observation to the observation of participation: The mergence of narrative ethnography". *Journal of Anthropological Research 47*(1): 69-94
[66] McConnell, S. 1996. *Rapid Development.* Microsoft Press.
[67] Erickson, J., K. Lyytinen, and K. Siau. 2005. "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research". *Journal of Database Management* 16(4): 88-100.
[68] Williams, L., and R. Kessler. 2000. "All I really need to know about pair programming I learned in kindergarten". *Communications of the ACM* 43(5): 108-114.
[69] Saige, C., and N. Berente. 2016. "Pair Programming vs. Solo Programming: What do we know after 15 years of research? *HICSS*, 2016, 2016 49th Hawaii International Conference on System Sciences (HICSS), pp. 5398-5406, doi:10.1109/HICSS.2016.667
[70] Sahay, S. 1998. "Implementing GIS technology in India: some issues of time and space". *Accounting, Management and Information Technology* (8): 147-188.
[71] Jeffries, R., A. Anderson, and C. Hendrickson. 2000. *Extreme Programming Installed.* Addison Wesley.
[72] Fowler, M. 2006. Continuous Integration. Accessed January 5 2015 from martinfowler.com: http://www.martinfowler.com/articles/continuousIntegration.html.
[73] Jeffries, R. 1999. "Extreme Testing". *Software Testing & Quality Engineering* (1:2): 23-26
[74] Boehm, B., and R. Turner. 2003. *Balancing Agility and Discipline: A Guide to the Perplexed.* Addison Wesley.
[75] Booch, G., 2001. "Developing the Future". *Communications of the ACM* 44(3): 119-121.
[76] Cohn, M., and D. Ford, D. 2003. "Intoducing an Agile Process to an Organization". *IEEE Computer* (June): 74-78.
[77] Sliwa, C. 2002. "Users Warm up to Agile Programming". *Computerworld*, March 18.
[78] Conboy, K and B. Fitzgerald. 2010. "Method and developer characteristics for effective agile method tailoring: a study of expert opinion". *ACM Transactions on Software Engineering Methodology,* 20 (1): 1-27