Discursive Co-development of Agile Systems and Agile Methods

Richard Baskerville, Jan Pries-Heje

Discursive Co-Development of Agile Systems and Agile Methods

Richard Baskerville

Georgia State University, USA
Baskerville@acm.org

Jan Pries-Heje

Roskilde University, Denmark
janph@ruc.dk

Abstract. Agile methods continue their growth in popularity. This spreading usage increases the need for adapting agile approaches to specific organizations. Hence, we investigate how system developers engage in the evolution of both agile systems and agile methods in practice. We study adaptation of the agile method *Scrum* in six organizations. Based on this study we design a framework explaining how agile methods, and in particular Scrum, are constantly articulated and re-articulated when diffused in practice. This framework includes a two-by-two dimensional grouping that includes three classes of fragments: Objects, Organization, and Process. The fourth class involves a discursive articulation that occurs on the same logical plane as the fragments. Unlike method engineering, the discourse is an inseparable part of the methodology itself, not a separate "meta" method.

Keywords: Agile Methods; Discourse; Adaptation of Methods; Technological Rules; Scrum.

1 Introduction

Agile development approaches assume continuous method evolution, but sometimes offer little guidance how to go about managing this evolution. There is considerable work on the construction of information systems development methods, but this work largely has an engineering orientation of most value to plan-based development. There is a need for more work specifically on the alternatives to such engineering approaches for developing and redeveloping agile methods on the fly. Agile methods involve frequent, fast, and continuous change. On-the-fly redevelopment of agile methods also involves frequent, fast, and continuous change. The research reported here indicates that such evolution of agile methods involves the engagement of a discursive mode of method development.

adfa, p. 1, 2011.

Our research question regards “How do system developers over time engage in the evolution of both agile systems and agile methods in practice?” The research reported below indicates the presence of a co-development discourse within which both the methodologies and the systems that the methodologies create are evolving. This methodological evolution progresses through a process of fragmentation, articulation, and re-articulation continuously throughout a software development project.

2 Agile development methods

Agile information systems development has been adopted successfully by practitioners partly or completely replacing more linear or plan-driven approaches. Agile is a success in systems development, and acknowledged as such by both the practitioner and researcher community [1]. Agile IS development can be defined either as an approach focusing on delivering something useful at high speed [2] or as an approach that can adapt to a continuously changing target or requirements [3]. Agile development is organized as iterations, repeating the same activities in short cycles [4]. In 2001 the agile manifesto provided a statement of values and principles (agilemanifesto.org) that has spawned many agile methods. Conboy [3] defined agility as the intersection of speed, change, learning, and customer value. He used these characteristics to propose a taxonomy and an instrument for assessing the agility of a particular practice. This assessment operated across agile, plan-driven, or in-house methods. The agility of a methodology is: “the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment” [3, p. 340].

2.1 Scrum

Our cases below use Scrum, a frequent example of agile methodology. Scrum originated as The Rugby Approach [5]. This approach used small cross-functional teams produce better results. The Rugby theme arose from concepts like “Scrum” and “Sprint” that referred to the game to describe how work was carried out by a team in this approach. We have read and categorized literature on Scrum [6]. As a result we identified four objects (abbreviated “OB”), three types of organization (abbreviated “OR”), and five types of process (abbreviated “PR”).

Scrum is iterative as with other agile approaches to IS development. The iteration in Scrum is called a Sprint. Work is organized in short iterations no more than 2-4 weeks long (PR-1). The team is a self-organizing team of equals (OR-1) The user or customer is deeply integrated in the development and has representation through the Product Owner role (OR-2). This role is defined as a user or customer with power and ability to make decisions. The Product Owner defines the desired functionality in a new IT system and originates it as a User Story (OB-1). The Product Owner then prioritizes User Stories in a Product Backlog (OB-2). The functionality from a User Story with the highest priority is broken down into tasks in a Sprint Planning Meeting (PR-2)

on the first day of a Sprint. This breakdown of User Stories into tasks is a collaborative effort involving the participation of the project team as well as the Product Owner. Planning Poker (PR2.1) is a mechanism for ensuring the right level of breakdown. All participants estimate the tasks using a number of pre-defined estimates. The allowable estimates are “Less than one day for one developer”, “One day”, “Two days”, “Three days”, and “Too big”. If the group agrees a task is “Too big” then the task needs to be broken down further. If the group agrees on “Less than one day for one developer” then the task needs to be merged with another task.

The Sprint continues after the planning meeting. Each day the project team meets in a Daily Stand-up Meeting (PR-3) that takes place in front of a Scrum Board (OB-3). A Scrum Master (OR-3) is responsible for the meeting, which should take no more than 15 minutes. In the meeting, each team member chooses one or more of the tasks. During the meeting every team member answers three standardised questions. (1) What did you do yesterday? (2) What are you doing today? (3) What problems did you encounter?

The Scrum Board (OB-3) is usually a whiteboard with four columns. The columns are titled “Ready”, “In-progress”, “Done”, and “Done-Done”. *Ready* details the task breakdown and Planning Poker estimates. *In-Progress* indicates tasks that a team member has chosen the task and work is underway. Upon task completion, this team member moves moved the task to *Done*. After this, a different team member may quality-assure the task, and afterwards the task moves to *Done-Done*. After this, the task is registered on a Burn-Down Chart (OB-4) that depicts “expected” versus “realized” production.

During the Daily Stand-up meeting (PR-3) team members record their answers to the first two of the three standardised questions by moving tasks from column to column on the Scrum Board. For example the answer to, “What did you do yesterday?” could move a task from *In-Progress* to *Done*. The answer to, “What are you doing today?” can move a task from *Ready* to *In-Progress*. Team members usually write their names on the card associated with the task card as a self-commitment made apparent through the Scrum Board.

After the Sprint iteration completes, the product is demonstrated to the Product Owner in a Sprint Review Meeting (PR-4). First, the Product Owner reflects on the value of the deliverable. Second, the team conducts a learning Retrospective (PR-5). In this retrospective, the team considers what succeeded and failed in the Sprint. Finally, the team determines one or two changes to implement or at least pursue in the next Sprint.

3 Research Method

The research reported in this paper operated from a staged research design (two stages). The first stage involved a multiple case study with six cases. Originally the cases were chosen to study the diffusion and adoption of Scrum. But an early finding of the studies was that all the case companies had evolved agile methods. Hence the focus in this paper is to develop a framework that helps explain how developers evolve agile

methods during projects. We chose six cases in order to build a *replication logic* for contrasting results across the cases [7]. The specific agile methodology was held constant (Scrum) to reduce the ontological noise that would arise from differing perspectives of alternative agile methods. The different cases were selected to provide diversity in organizations using a common agile approach. Thereby the replication logic captures shifts in a more cohesive agility viewpoint across diverse settings. Table 1 provides details of the six organizations represented in our cases. The second stage created an evaluation engagement in which 25 practicing software developers received an orientation to the framework and were given the opportunity to evaluate its use in appraising the agile method adaptation processes in their own organizations.

Table 1. The six companies studied. Pseudonym names used to preserve anonymity.

Name (Pseudonym)	Characteristics	Roles and Number of Subjects
GlobeRiver	Develops engineering products. 500 employees in R&D function world-wide when interviewed	3 people interviewed: Danish and Indian Scrum master, and Danish Facilitator
SuperSystem	Develops software for the military, the banking industry, hospitals, etc. Approx. 400 employees when interviewed.	4 people interviewed: a Lead developer, a Scrum master, manager, and person in charge of implementing Scrum
DareYou	An off- and online gaming company; works with several suppliers located in different places	2 people interviewed: The Project manager and the Product owner.
ShipSoft	A software house producing software for international production. 150 employees	20 people interviewed. Observation over three periods of time including full week
PubliContract	Public organization that contracted a private software house.	All people in Scrum team interviewed. Three product owners interviewed.
InterFin	International financial organization with IT development in Europe and India.	Scrum team followed for a full project (one year in length). Scrum in Europe and India.

In the six stage-one-cases we collected data using techniques appropriate for each setting. For example, the Interfin case involved interviews and observation, both in India and Europe, over the course of the project. In general, analytic induction (inductive reasoning about a social phenomenon) was used for data analysis [8]. This approach involved inducing concepts, so-called “social laws”, from a deep analysis of empirically isolated instances [9]. The research question drove this analytic inductive reasoning. It was an analytical process whereby data was coded when found in the six cases and

when it was related to the diffusion and evolution of agile methods [10]. The analysis revealed that the phenomena in the case could be classified as either objects or organization. An alternative classification also surfaced in which the phenomena could be classified as either static concepts/processes, or as elements that were dynamic and ever changing. There were 12 concepts in this classification scheme that were labeled method fragments (a term adopted from the method engineering literature). In concert with the analytic induction, we followed an interrogatory data analysis process described by Pascale [11]. In our implementation, the interrogatives included, "Under what contexts does this adaptation of Scrum arise?" "Under what circumstances may we find exceptions to this general adaptation rule?" The results were coded as technological rules [12].

4 Fragmentation and articulation

Agile development meets the need for systems development where a setting is subjected to continual change. Planned methodologies (which invoke mechanistic organization) are less suitable in such settings. But agile approaches (which involve organic organization) fit such settings well [13]. Agile methodologies, like other kinds of methodologies share certain general characteristics. Methodologies are organized collections of concepts, beliefs, values, and normative principles that are supported by material resources [14]. Methodologies often adopt a particular perspective intended for a particular application domain involving particular prerequisites and activities. Because methodologies are rarely used in the exact way described [15, 16], method fragments become cobbled together with novel elements that comprise a situated methodology unique to its setting. For example, in agile development often embodies an ad-hoc mix of fragments intuitively assembled from Scrum, XP, etc, [17].

A method fragment is a concept, notation, tool, technique, etc. that has been coherently separated from the overall methodology. It is lifted from its original methodological framework and used in a different one. The use of "method fragments" or "method chunks" is a central principle in method engineering [18]. These method fragments can have varying levels of abstraction and granularity [19]. Method fragments are a concept best known within the frameworks and processes of method engineering. Such frameworks engineer IS development methods by assembling them from an inventory of components: methodologies, method fragments, and innovations [20]. Computer-based method engineering uses formal models to enable the rapid development of computer-aided systems analysis and software engineering (CASA/CASE) tools that provide a unique methodology for each unique development settings [21].

This discourse involving organizations, settings, developers, and their method fragments should not be mistaken for the several variations of agile engineering and agile method engineering. Such variations include obtaining feedback from users of the subject methods, or method engineering of agile methods [22], or meta-meta-method engineering or the representation of agile ways to semantically capture the domain knowledge [23]. A key discovery found in our data is the participation of the fragments themselves (along with developers, settings, etc.) in the process. Because of

its discursive character, *articulation* engenders an emergent form of methodology evolution that subsumes its complexity into a holistic and reflective social discourse. It does not deal with complexity through reduction, as an engineering approach might, but rather approaches it as a conversation between people, their problems, and their tools.

Method engineering provides a strategy for method adaptation. Situated method engineering is a strategy for continual integration of fragments such that methods can adapt as developers learn about their changing environment [24]. This work has also been used as an approach to software process improvement for agile methods and for object-oriented methods [25]. One objective has been the rational transformation of unique methodologies using fragment assemblies [26]. The approach provides a form of meta-method for configuring off-the-shelf methods componentization [27].

Our agile cases seemed to choose a mode of fragmentation and articulation that differed from the method engineering mode. They used a discursive perspective instead of an analytical design perspective. These were evolving agile methodologies as a discourse shared between organizations, settings, developers, and their method fragments. Fragmentation is retained because it is common beyond engineering. Method fragmentation is found in business process management [28], security method adaptation [29], self-organizing systems [30], requirements traceability [31], and for requirements elicitation [32]. Articulation has a discursive character that subsumes complexity into a holistic and reflective social discourse. Unlike an engineering approach that uses reduction to eliminate complexity, it approaches adaptation as a conversation between people, their problems, and their tools.

5 Co-developing systems and methods

The process of co-development in the cases above can be expressed in terms of method fragments, technological rules, and articulation. We analyzed the major method fragments in the Scrum cases seeking an understanding of how these fragments were adopted or adapted in the cases. This analysis revealed two central dimensions that characterized the fragments.

5.1 Dimensions of agile method fragmentation

As discussed above, the analysis provided two central dimensions in the fragmentation and articulation of evolving agile methodology. One dimension distinguishes static fragmentation versus dynamic fragmentation as distinguishing characteristics of the fragments. This was more of a criteria for articulation or re-articulation of the method than it was a factor of the criteria for fragment use [33]. Static fragments are often used or reused without changing the fragment internally. Dynamic fragments are often used or reused only after internal modifications or adaptations. These dynamic fragments were often themselves re-articulated in innovative ways. Dynamic fragments required internal changes before the next re-use in the method.

The second dimension distinguished actor fragments versus artifact fragments. In actor fragments, human autonomy was somehow featured in the fragment. Actor

fragments tended to be loosely articulated; imbued with a permissive spirit giving people the latitude to re-arrange their behaviors during the development project. In contrast, artifact fragments suppressed human autonomy; imbued with a restrictive spirit that limited changes in individual behavior during the project. Together, these two dimensions define four classes of fragments: Process, Objects, Organization, and Articulation. See Figure 1. Each of these classes is described below.

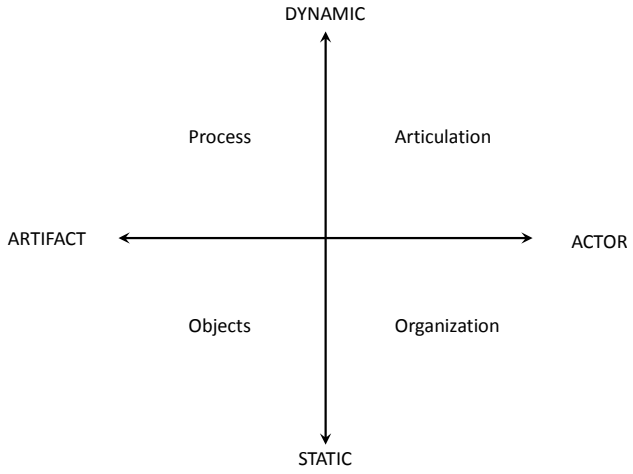

Fig. 1.
Dimensions of agile fragmentation and reassembly.

6 Scrum Fragment Technological Rules

For the purpose of expressing the method fragments in Scrum, we adopt the notion of technological rules. Van Aken's design rules operate in the following manner. In a management vision of design science, there are two possible outputs: artefacts or interventions, and there may be three kinds of design in a professional episode: The object-design defines the artefact or intervention. The realization-design is the plan for implementing the artefact or intervention. The process-design is the plan for the design process itself. In this sense designing is similar to developing prescriptive knowledge. Van Aken suggests expressing a design in the form of technological rules: 'A technological rule follows the logic of "if you want to achieve Y in situation Z, then perform action X". The core of the rule is this X, a general solution concept for a type of field problem' [12, p. 23]. Van Aken emphasises that technological rules need grounding [12, p. 25]: 'Without grounding, the use of technological rules degenerates to mere

"instrumentalism", that is, to a working with theoretically ungrounded rules of thumb'. The rules need to be grounded in a way acceptable from a social science perspective.

Based on our analysis of the six cases we were able to formulate the following technological rules for the specific parts of Scrum. In each rule, we note the way in which each rule is empirically grounded in the data collected and the analysis and coding of data from the six cases. Note that the version presented below is the refined version that resulted from an evaluation described later in this paper.

6.1 Objects

Object fragments are Static Artifacts. This means they are frequently used for (re)assembly in new variants of situated Scrum without changing (re-articulating) the fragment. These fragments marginalize human autonomy in the sense that these involve structures that do not provide much variance according to the individual actors in the setting. These are listed here along with examples of the technological rules that inhabit each fragment.

Object: User Stories

OB-1: If you want to express requirements for new systems in a simple user-oriented form that is easier to communicate to and with end users than "classic" rigorous requirements (e.g. following the IEEE standard)
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then write User Stories on index cards in your projects
Grounding: Used at SuperSystem

OB1.1: If you want to involve users more in projects
- In a situation where you have decided to express requirements as user stories
- - then let users write or participate in writing user stories
Grounding: In DareYou the customer had written the User Stories on how to play a specific game.

Object: Product Backlog documentation

OB-2: If you want to have a dynamic list of functionality where it is easy to add new or subtract "old" functions
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then maintain a list called Product Backlog and let user update list dynamically and prioritize list regularly; i.e. before each sprint
Grounding: All companies maintained a product backlog list of wished-for functionality

OB2.1: If you need to have more documentation, e.g because it is required by law or regulation or because you need to maintain the resulting systems for years after
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then you need to decide specifically what additional documentation is required
Grounding: Avoiding unnecessary documentation is a general principle of agile methods. (Cf. the agile Manifesto)

Object: Scrum Board

OB-3: If you need a visible coordination mechanism for project teams where it is easy to progress and whether anyone in the team needs help or have problems finishing tasks assigned
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then use a Scrum Board in your projects
Grounding: All cases except DareYou used a Scrum Board as a visible coordination mechanism

OB3.1: If Scrum-team is located in different locations
- In a situation where you have decided to use the Scrum Board object
- - then use an electronic Scrum Board that can be seen in all locations simultaneously. Possibly coupled with a video-link so people at all locations can see Scrum Board and each other at the same time
Grounding: GlobeRiver and ShipSoft did this with good results

OB3.2: If Scrum-team needs to focus more on increased quality
- In a situation where you have decided to use the Scrum Board object
- - then consider assigning quality or tester role to person in the team and/or add a *Done Done* column to Scrum Board
Grounding: PubliContract and some projects in ShipSoft did this

Object: Burn-Down Chart

OB-4: If you want to have simple visible mechanism for follow-up in projects where it is possible in one glance (on the chart) to see how close or far you are from having achieved the work planned
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then implement a Burn Down Chart in your projects
- Alternatively use more traditional follow-up techniques such as Earned Value
Grounding: ShipSoft had Burn-Down charts for every project. SuperSystem also maintained Burn-Down charts in every project

6.2 Organization

These fragments are Static Actors. This means they are frequently used for (re)assembly without necessarily rearticulating the fragment itself. Roles such as Scrum Master and Product owner are common, and seldom changes. Beyond this role, however, these fragments do privilege human autonomy in the sense that the actors have much latitude in how they enact this role. These organization fragments are listed along with the technological rules that inhabit each fragment.

Organization: Self-organizing team of equals

OR-1: If you have a team of experienced professionals, with more or less the same level of competence, in a culture where hierarchy is not desired
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then consider organizing the team as a self-organizing teams of equals without a project manager to assign tasks
Grounding: InterFin showed that this may be hard in a culture with high power-distance (in the Hoftstede sense)

OR1.1: If you want to want to use Scrum in a team where the team members have different (or very uneven) competences
- In a situation where you have decided to use self-organizing teams of equals
- - then you may need to assign specialist roles to different team members. You need to adapt the *process*-elements (PR-1 to PR-5) to allow for non-equals.; and you may consider having a traditional project manager
Grounding: InterFin had a test specialist in a Scrum team. SuperSystems also used specialist roles as part of their Scrum adaptation

OR1.2: If you have a larger team than 8-10 people
- In a situation where you have decided to use self-organizing teams of equals
- - then you may organize a number of Scrum-teams each with a Scrum Master, and then the Scrum Masters can meet (daily) in a Scrum-of-Scrum
Grounding: We saw this done in both DareYou and InterFin with good results

Organization: Product Owner role

OR-2: If you need a decision making ability in relation to all user- or functionality-oriented issues (e.g. to make firm decisions on what functionality is included and excluded)
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then you should have a highly decisive customer take on the role as Product Owner
Grounding: DareYou for example had a manager from the customer site in the Product Owner role.

OR2.1: If you cannot assign the role of product owner to one person but have many stakeholders that want to be heard and to be part of the decision making
- In a situation where you need the decision making ability of the product owner
- - then organize a product owner forum and name a chief product owner who can make the final decision when disputing views arise among stakeholders
Grounding: Exactly the solution chosen in PubliContract where they see it as very beneficial and a way to preserve the effectiveness of the product owner role

OR2.2: If you want to have a product owner
- In a situation where you do not have access to customers (e.g. because you are doing product development)
- - then find a person with a good market understanding to fill the role as product owner
Grounding:SuperSystem, Globeriver, Interfin and ShipSoft were all doing this

Organization: Scrum Master role

OR-3: If you want to have a person specifically responsible for ensuring that agile process is followed
- In a situation where you are considering the use of agile methods; specifically Scrum, and you have decided to have a self-organizing team of equals (OR-1)
- - then have one person in each team take on the role as Scrum Master
- Alternatively just use existing Project Manager role
Grounding: Found as described in SuperSystem,

OR3.1: If you want to maintain *both* a Project Manager and a Scrum Master role *and* not enacted by same person
- In a situation where you have decided to have Scrum Master role enacted
- - then you need to negotiate responsibilities for the two roles and the interface between them
Grounding: PubliContract did exactly this. In DareYou the customer was also the project Manager. In InterFin the Project Manager was placed above two Scrum masters that each had a team of their own

6.3 Process

These fragments are Dynamic Artifacts. This means they are more often modified, adapted, and rearticulated as the method evolves. Nevertheless, these fragments do not afford much latitude to the individual actor in changing their behavior within the process. They are listed here along with examples of the technological rules that inhabit each fragment.

Process: Organize work in short iterations

PR-1: If you want to organize work in small iterations to deliver something of value *fast*
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then use *Sprints*
Grounding: All six companies did this. The shortest iterations we saw were one week (ShipSoft). The longest was eight weeks

Process: Sprint Planning Meeting

PR-2: If you want to start the iteration with a planning meeting where work breakdown and estimation takes place
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then have a one day Sprint Planning meeting on the first day of the iteration with the development team and the product owner present
Grounding: All six companies did this. In a few instances the product owner was not present which caused delays and indecisiveness due to the lack of needed information on what the user actually wanted

PR2.1: If you need estimates for tasks fast
- In a situation where you have decided to use agile method and Sprint Planning meetings
- - then use Planning Poker to come up with estimates
- Alternatively you can use any other estimation techniques
Grounding: Several companies used Planning Poker i.e. DareYou and ShipSoft

Process: Daily Stand-up Meeting

PR-3: If you have meetings in teams that take up too much time and you want to have shorter and more effective meetings without long discussions of the agenda and/or specific problems
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then use daily stand-up meeting lasting no longer that 15 minutes and with a standard agenda: (1) What have you been doing? (2) What are you doing now? (3) Problems encountered?
Grounding: Five out of six companies did this. In one ShipSoft project they were even standing in front of PC screen when doing daily meeting between Denmark and India. In one project in InterFin they were not standing up for their daily meeting because they were in an open office environment where it bothered other projects when they were standing. And in DareYou the daily meeting was conducted on phone with same standardised agenda but sitting down

PR3.1: If you want to use short stand-up meetings
- In a situation where you do not have full-time resources (people)
- - then organise the stand-up meeting weekly, bi-weekly, every 2nd day or the like
Grounding: In GroundRiver they did not have meetings every day due to part-time resources. Instead they had a weekly meeting between the people working in India and the people from Europe (a project manager and a facilitator)

Process: Demo of value at the end

PR-4: If the functionality that is developed in a sprint can be put into production immediately and you want customers or end users to see what they are getting out of each sprint (e.g. because you know that is likely to increase their satisfaction with the development)
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then demonstrate that you have developed something of value at the end of a sprint
Grounding: PubliContract and DareYou did this

PR4.1: If you want to adapt to an existing release schedule
- In a situation where you uses agile methods in combination with more traditional schedule
- - then demonstrate value but do not release
Grounding: InterFin did this to fit Scrum with traditional mainframe-oriented release plan every 3-4 months

Process: Retrospective at the end

PR-5: If you want to capture learning and put lessons learned into use quickly
- In a situation where you are considering the use of agile methods; specifically Scrum
- - then carry out a retrospective at the end of a sprint (iteration)
Grounding: All six companies had adopted adapted this practice

7 Articulation

The articulation group is too poorly structured for expression using technological rules. The articulation of fragments is itself a fragment because it is the on-the-fly, discursive process where developers assemble the fragments into a working methodology. While similar to the method engineering notions of design rationale or design model, it was not a "meta" process or a "meta" design. In agile development, the discourse about the adaptation and evolution of the methodology is part of the normal development conversation. Articulation fragments were distinctly dynamic actors. Fragments in this group often specified criteria for articulation or re-articulation of the method. It was dynamic because the articulation fragments changed internally on each use in the method. Articulation fragments are actor fragments that privilege human autonomy. They allow people to adjust their future behaviors as the development project evolved.

A more complete framing of this method articulation is the theory nexus [34, 35]. A theory nexus encompasses the interaction between theories and designed artifacts. It helps frame the process that results when multiple theories overdetermine the design for an object that, in fact, represents a setting where the design outcome is at least partly the result of the use of the object. In our cases, the theories are embodied in the technological rules. Agile methodologies are designed and re-designed on-the-fly, in

concert with the use of the methodology, creating a theory nexus as technological rules and fragments-in-use are combined, separated, and recombined. Within the theory nexus, a discourse is present. This discourse articulates and rearticulates the dynamic objects in the presence of the static objects. In the six cases, such discourse episodes were embodied by each Sprint. Only experience with the methodology can determine the exact effects of an evolving methodology in relation to underlying technological rules.

The nexus is a discourse, a complex conversation that extends across (1) a deductive view of the relationship between fragmentation and methodology, (2) a reciprocal relationship between the articulation and re-articulation of technological rules; and (3) the evolutionary iterations of methodological framing (Carroll and Kellogg 1989). A nexus binds method fragments with realities and shapes a momentary version of a methodology. This moment immediately initiates a new episode in the discourse (a Sprint in the cases). The fragments, the participants, the methodology, the setting, and the problem are engaged in this discourse. Each re-articulation of the methodology results in a new momentary version that necessarily precipitates a new episode (the next Sprint). These re-articulation episodes within the nexus persist throughout the life of an agile development project. There is not a methodology, but a succession of different methodologies that momentarily provide structure through regularities that are present only for unique episodes.

8 Evaluating the co-development framework

To evaluate the framework, we engaged with 25 software practitioners from mainly engineering-oriented companies, each of whom represented different software development companies and each with extensive experience from companies that had adapted Scrum or were considering doing so. After having been introduced to the framework and the technological rules they were asked to fill out a questionnaire with the purpose of improving the framework. Thirteen of the 25 participants decided voluntarily to participate. Six of the 13 could immediately use the framework to evaluate the Method-System Co-Development activities in their own organizations whereas the seven others reflected on past situations in which they adapted Scrum or imagine such a future situation. The results indicate that the framework was easy to learn and very helpful in their own practice. It was clear from the responses that there was substantial interest in further development of the framework for future use. Further the evaluation pointed to four things that have been changed in the version of the framework presented in this paper.

First, many of the technological rules were formulated too briefly (e.g., using a phrase such as “if you want to …”). For version 2 (presented above) we considerably elaborated the technological rules with a focus on the benefits to be achieved from adapting the object, organization or process.

Second, it was stated in the evaluation that the framework was “…not detailed enough for implementation. An example was the relationship between Scrum Master and Project Manager”. For version 2 (above), we have added more details and we have

made it clear that there are relationships between some of the fragments by adding a numbering system for easy reference and by adding references from fragments to other fragments. See for example "OR1.1" where the technological rule now includes a reference to "PR-1 to PR-5".

Third, it was pointed out that the technological rules on the product owner role were too rudimentary. It didn't include the "hard things" such as "product owner availability" and "what to do if the product owner was only interested in final results and not in partial results after each sprint?". To cope with this comment we added several statements to the technological rule on product owner (see OR-2 above).

Fourth, one evaluator pointed out that he did not believe in the "supermarket approach" that we were using and that he believed there was a minimum level of Scrum elements necessary to regard the approach as Scrum. Nevertheless, the six cases we originally analyzed clearly show that companies in practice do use a supermarket approach, taking some Scrum fragments into use and eschewing others. But our study of the six cases also showed that at least six-seven fragments of the 12 identified were taken into use in all six cases. This effect suggests that there is indeed a minimum of at least 50% of the fragments that are intuitively taken into use.

Finally, the evaluation confirmed that for none of the six companies had adapted Scrum as a one-shot event. They were all continuously adapting Scrum in a discursive process as we have presented in this paper.

9 Conclusion

Among the limitations in the work above, the approach used cases representing one instance of agile methodology (Scrum). While observationally consistent, it limits the confidence that the study findings will generalize across all agile methodologies. We also studied instances of Scrum projects, which limited observations of any longitudinal evolution from project-to-project.

The adaptation of agile methods is a special case of an adoption process where users purposefully adopt parts of the methodology and discard other parts. In this sense our study has implications for future studies of adoption in which a technology (such as a tool, method, or process) grows more adoptable by promoting its re-articulation through discursive usage.

The main contribution of this paper is the framework explaining how Agile methods, and in particular Scrum, are constantly articulated and re-articulated when diffused in practice. This framework includes a two dimensional groupings that include three classes of fragments: Objects, Organization, and Process. The fourth class involves a discursive articulation that occurs on the same logical plane as the fragments. Unlike method engineering, the discourse is an inseparable part of the methodology itself, not a separate "meta" method. Agile method adaptation is a functional part of routine development practice.

There is practical value in the nexus and the technological rules. They have a demonstrated prescriptive design value useful for many development managers employing agile methods.

10 References

1. Dybå, T. and T. Dingsøyr, *Empirical studies of agile software development: A systematic review*. Information & Software Technology, 2008. **50**(9-10): p. 833-859.
2. Ågerfalk, P.J., B. Fitzgerald, and S. Slaughter, *State of the Art and Research Challenges.* Information Systems Research, 2009. **20**(3): p. 317-318.
3. Conboy, K., *Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development.* Information Systems Research, 2009. **20**(3): p. 329-354.
4. Austin, R.D. and L. Devin, *Research Commentary--Weighing the Benefits and Costs of Flexibility in Making Software: Toward a Contingency Theory of the Determinants of Development Process Design.* Information Systems Research, 2009. **20**(3): p. 462-a-477.
5. Takeuchi, H. and I. Nonaka, *The New New Product Development Game*. Harvard Business Review, 1986(January-February).
6. Sutherland, J. and K. Schwaber, *The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework*, 2010, SCRUM Training Institute.
7. Yin, R.K., *Case Study Research: Design and Methods* 4th ed2008, Thousand Oaks, Calif.: Sage.
8. Ragin, C.C., *Constructing Social Research: The Unity and Diversity of Method*1994: Pine Forge Press.
9. Znaniecki, F., *The method of sociology*1934, New York: Farrar & Rinehart.
10. Miles, M.B. and A.M. Huberman, *Qualitative Data Analysis: A Sourcebook of New Methods*. 2nd ed1994, Newbury Park, Calif: Sage.
11. Pascale, C.-M., *Cartographies of Knowledge: Exploring Qualitative Epistemologies*2010: Sage.
12. van Aken, J.E., *Management Research as a Design Science: Articulating the Research Products of Mode 2 Knowledge Production in Management.* British Journal of Management, 2005. **16**(1): p. 19-36.
13. Burns, T. and G.M. Stalker, *The Management of Innovation*1961, London: Tavistock.
14. Lyytinen, K., *A Taxanomic Perspective of Information Systems Development. Theoretical Constructs and Recommendations*, in *Critical Issues in Information Systems Research*, R. Boland and R. Hirschheim, Editors. 1987, Wiley: New York.
15. Bansler, J. and K. Bødker, *A reappraisal of structured analysis: Design in an organizational context*,. ACM Transactions on Information Systems, 1993. **11**(2): p. 165-193.
16. Truex, D.P., R. Baskerville, and J. Travis, *Amethodical Systems Development: The Deferred Meaning of Systems Development Methods.* Accounting, Management and Information Technology, 2000. **10**(1): p. 53-79.
17. Esfahani, H.C., et al., *Situational Evaluation of Method Fragments: An Evidence-Based Goal-Oriented Approach*, in *Advanced Information Systems Engineering*2010, Springer Berlin / Heidelberg. p. 424-438.
18. Rolland, C. and N. Prakash, *A proposal for context-specific method engineering*, in *Method Engineering: Principles of method construction and support*, S. Brinkkemper, K. Lyytinen, and R. Welke, Editors. 1996, Chapman & Hall: London. p. 191-208.
19. Tan, W.-K. and C.-H. Tan, *Teaching Information Systems Development via Process Variants.* Journal of Information Systems Education, 2010. **21**(2): p. 159-172.
20. Brinkkemper, S., K. Lyttinen, and R. Welke, eds. *Method Engineering*. 1996, Chapman & Hall: London.

21. Odell, J.J., *A primer to method engineering*, in *Method Engineering: Principles of method construction and tool support*, S. Brinkkkemper, K. Lyytinen, and R. Welke, Editors. 1996, Chapman & Hall: London. p. 1-7.
22. Schapiro, S.B. and M.H. Henry. *Engineering agile systems through architectural modularity*. in *IEEE International Systems Conference (SysCon)*. 2012.
23. Berki, E., *Formal metamodelling and agile method engineering in metaCASE and CAME tool environments*, in *Proceedings of The 1st South-East European Workshop on Formal Methods* K. Tigka and P. Kefalas, Editors. 2004, SEERC: Thessaloniki. p. 170-188.
24. Van Slooten, K., *Situated method engineering.* Information Resources Management Journal, 1996. **9**(3): p. 24-31.
25. Henderson-Sellers, B. and M.K. Serour, *Creating a Dual-Agility Method: The Value of Method Engineering.* Journal of Database Management, 2005. **16**(4): p. 1-23.
26. Karlsson, F. and K. Wistrand, *Combining method engineering with activity theory: theoretical grounding of the method component concept.* European Journal of Information Systems, 2006. **15**(1): p. 82-90.
27. Karlsson, F. and P.J. Agerfalk, *Method configuration: adapting to situational characteristics while creating reusable assets.* Information and Software Technology, 2004. **46**(9): p. 619-633.
28. Ravesteyn, P., *A Context Dependent Implementation Method for Business Process Management Systems.* Communications of the IIMA, 2009. **9**(1): p. 31-45.
29. Low, G., H. Mouratidisb, and B. Henderson-Sellers, *Using a Situational Method Engineering Approach to Identify Reusable Method Fragments from the Secure TROPOS Methodology.* Journal of Object Technology, 2010. **9**(4): p. 91–125.
30. Puviani, M., et al., *A Method Fragments Approach to Methodologies for Engineering Self-Organizing Systems.* ACM Transactions on Autonomous and Adaptive Systems, 2012. **7**(3): p. Article 33.
31. Domges, R. and K. Pohl, *Adapting traceability environments to project-specific needs.* Association for Computing Machinery. Communications of the ACM, 1998. **41**(12): p. 54-62.
32. Haumer, P., K. Pohl, and K. Weidenhaupt, *Requirements elicitation and validation with real world scenes.* IEEE Transactions on Software Engineering, 1998. **24**(12): p. 1036-1054.
33. Aydin, M.N., et al., *On the Adaptation of an Agile Information Systems Development Method.* Journal of Database Management, 2005. **16**(4): p. 24-40.
34. Pries-Heje, J. and R. Baskerville, *The design theory nexus.* MIS Quarterly, 2008. **32**(4): p. 731-755.
35. Carroll, J.M. and W.A. Kellogg, *Artifact as theory-nexus: hermeneutics meets theory-based design.* CM SIGCHI Bulletin--Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind, 1989. **20**(SI): p. 7-14.