Computer Supported Cooperative Work (2009) 18:401–443
DOI 10.1007/s10606-009-9100-4

What Counts as Software Process? Negotiating the Boundary of Software Work Through Artifacts and Conversation

Marisa Leavitt Cohn[1], Susan Elliott Sim[1] & Charlotte P. Lee[2]
[1]*Department of Informatics, University of California, Irvine, CA 92697-3440, USA (Phone: +1-949-8244047; Fax: +1-949-8244056; E-mail: mlcohn@ics.uci.edu; Phone: +1-949-8232373; Fax: +1-949-8244056; E-mail: ses@ics.uci.edu);* [2]*Department of Human Centered Design & Engineering, University of Washington, 423 Sieg Hall, Box 352315, Seattle, WA 98195, USA (Fax: +1-206-5438858; E-mail: cplee@u.washington.edu)*

Abstract. In software development, there is an interplay between Software Process models and Software Process enactments. The former tends to be abstract descriptions or plans. The latter tends to be specific instantiations of some ideal procedure. In this paper, we examine the role of work artifacts and conversations in negotiating between prescriptions from a model and the contingencies that arise in an enactment. A qualitative field study at two Agile software development companies was conducted to investigate the role of artifacts in the software development work and the relationship between these artifacts and the Software Process. Documentation of software requirements is a major concern among software developers and software researchers. Agile software development denotes a different relationship to documentation, one that warrants investigation. Empirical findings are presented which suggest a new understanding of the relationship between artifacts and Software Process. The paper argues that Software Process is a generative system, which participants called "The Conversation," that emerges out of the interplay between Software Process models and Software Process enactments.

Key words: agile software development, requirements documentation, work artifacts, software process models and enactments, collaboration, negotiation

1. Introduction

This paper investigates the role of Software Process and artifacts in the collaborative work of software development at two Agile software development companies. To begin, we would like to share an anecdote from the field.

> We are interviewing a product manager, Brett, at Fast Tools, a company that develops Agile software tools for Agile software developers. The company is a mecca for Agile that in addition to building software tools for Agile software development holds daily training seminars and offers consulting services to companies that are transitioning to Agile. We are asking him to describe to us

> how software is built at Fast Tools, how they gather requirements, what a typical week looks like. To answer this question, Brett pulls up a power point presentation that they use for training which explains Agile processes. Then he pulls over a free-standing white board and writes out a list in two columns. Pointing to the left side he says "These are the sort of things that get emphasized in traditional methodologies. They focus on *process and tools* that define what's supposed to happen, making sure that we *document* everything because if we don't document it we end up with lots of arguments about who said what when, getting people to *sign off* on things because it saves a lot of problems, and *following a plan* because we know if we leave the plan it's not going to work. These are the traditional emphases." Pointing to the right column, he continues "In the Agile world you tend to see more of a focus on *individuals and interactions*, *working software,* and *customer collaboration.*" (Emphasis added to illustrate the words in the list.)

Brett's comment can easily be chalked up as "Agile speak" or as "Agile evangelism" as some of our informants called it. His right column draws directly from the "Agile manifesto" (see Section 5) and aligns with much of Agile software development literature. As illustrated by Brett, adherents of Agile tend to de-emphasize both Software Process and documentation. Yet, in our study of two Agile software development companies we observed quite a lot that could be called Software Process and documentation. During our data analysis, a particular question began to grab our attention. What counts as Software Process? Why do some kinds of artifacts count as software documentation and others not? What role are Software Process and artifacts playing at our research sites?

This paper presents an empirical investigation of these questions. We wish to consider how Software Process and artifacts, as well as the relationship, between them are understood by our participants. We present findings from qualitative fieldwork at two Agile software development companies. We focused our empirical observations and interviews on the explicitation of the Software Process and the role of artifacts. We observed artifacts in use and collected sample artifacts for analysis. We found that artifacts were indeed integral to the accomplishment of the software development work, that these artifacts included those prescribed by Agile software development methods as well as additional artifacts, and that artifacts supported an emergent, bounded, and negotiated Software Process. We found that there was a Software Process that emerged through the performance of the software development work, but which did not encompass all of the work accomplished every day. Rather, the Software Process was something that our informants participated in selectively, negotiating what is inside and outside of the Software Process. And indeed the artifacts they used were critical to the definition of and negotiation of the Software Process.

Software Process has been approached in different ways in research on software development. Two main approaches can be contrasted as emphasizing,

on the one hand, Software Process as something which can be *modeled*, and on the other hand, Software Process as *enacted* in a specific time and place. While our research shares many of the assumptions of the latter perspective, our findings reveal Software Process as something which requires a combination of both perspectives to explicate. In our analysis we argue that the Software Process models and enactments form a generative system. It is through the interplay between Software Process models and enactments that the Software Process and the software system are generated. Artifacts can influence and represent both the Software Process models and Software Process enactments as well as the shifts between these two modes of conceptualizing the Software Process.

Our findings reaffirm the assumptions of research on Software Process enactments by illustrating that Software Process cannot be defined only in terms of the work that goes into creating a working software system. While the development of working software is still central, it is only part of the story. Software process is itself something that the software development team constructs and reproduces over time. Software process is, in other words, one of many accomplishments in the ongoing work of software developers. The Software Process is a collection of effort that goes into making a working system of people and machines. This includes the mechanisms for sharing information about the software and creating ideas about it, the discussions that take place about the software, the use of the system and stories about this use.

Yet the Software Process is not the totality of all enactments of software development work; it is not an all-encompassing description of work practice. Our study results point to a conceptualization of Software Process not as an analytic category with which to scope our investigation, but as a space constructed by the software development team that is bounded and negotiated. Our study participants referred at times to this process as "The Conversation." The Conversation is a bounded space that is collectively constructed through collaboration and negotiation. The work of constructing this bounded space is also what helps them to scope the software system. The Conversation simultaneously constitutes what counts as the Software Process and the software system. It is made up of the software code, the software-in-use, stories about code and its use, conversations, and artifacts. But it is a subset of these things since some activities and artifacts take place outside of The Conversation.

Take for example two instances of hypothetical conversations between colleagues on a software development team. Lisa and Jordan are working on the development of a calendaring software tool for their customer. On one afternoon after completing their work for the current release, they head out for coffee. While they are out they end up discussing the software and come up with some ideas about how to implement a new feature for the system which they bring up at a group meeting on Monday. On another occasion, Lisa bumps into Jordan in the hallway and asks him for some feedback on a particular feature request he had written. She asks Jordan to clarify some assumptions in the feature request, but

Jordan says he had not thought about those assumptions. They find a place to chat and work out the assumptions together, so that Lisa can continue with her work.

We might distinguish these two conversations in a variety of ways: when and where they take place, the fact that one is unprompted and the other an answer to a question. But in our findings the difference that matters is that the first conversation is outside of the Software Process and the latter is inside the Software Process. In the first story, Lisa and Jordan come up with ideas which must be "brought in" (in the terms of our informants) to the Conversation at a later meeting at which other team members are present. In the second story, even though the assumptions about the feature, which will impact how it is implemented, are decided without feedback from the rest of the team, this one-on-one meeting is considered *already* part of the Conversation.

How this distinction between what is inside and outside of the Software Process as Conversation is what this paper is about. How does Software Process come to be constituted as this bounded and negotiated collection of effort? And how do artifacts get used to represent and influence work that takes place inside of The Conversation, outside of The Conversation, and in negotiating the boundary of the Software Process?

2. Background

2.1. Software process models

Software process models are representations of the "sequence of activities, objects, transformations and events that embody strategies for accomplishing software evolution" (Scacchi 2001). These models make explicit an order in which software development ought to take place, the types of artifacts which should be used to design software, and the stages of design such as requirements engineering, architecture design, implementation, quality assurance, and maintenance. While there is a variety of software models, some of which are based on descriptive research, the majority tend to be prescriptive and method-oriented in that they can be employed to support planning and operations of a software development team. Software process, according to these models, is systematic and methodical (Osterweil 1987; Truex et al. 2000) and "context-free" in that it is defined independently of any particular organizational setting (Scacchi 2001).

Software process models prescribe a set of artifacts that will support software development work. Particular software artifacts are often associated with each stage of the Software Process and are refined until they are ready to pass onto the next stage. The Waterfall model is a typical phased description of Software Process and Scrum is a typical iterative and incremental process. These will be described here, as the details are pertinent to our field study. It should be noted that these are normative, high-level sketches, and that the actual implementation in practice can vary greatly.

2.1.1. Waterfall

The Waterfall model is depicted in Figure 1, below. The phases progress in an orderly sequence from Requirements, when the software to be constructed is identified, through to Maintenance, when the software is deployed and evolved. Transitions can be made only between adjacent phases and involve a hand off, either of documents (between the first four phases) or a running software system (between the last three phases). These documents can be large, consisting of hundreds or thousands of pages, depending on the size of the project.

Normally, in Waterfall, there is at least one major document per phase, which serves as the starting point and constraints for the subsequent phases. The requirements phase produces a document describing the problem to be solved, the needs of the stakeholders, and desirable properties of the software to be built. It is often said that the goal is to describe "what" needs to be built, not "how" it should be built. The size and complexity of the requirements depend on the size and formality of the project. It can include prose descriptions, Use Cases, diagrams, tables of relationships, decision trees for business rules, and even sketches for the interface (van Vliet 2008).

In the specification phase, the information in the requirements document is changed from a customer-centric description into a software developer-centric description of what needs to be built. At this point, more technical notations such as UML (Unified Modeling Language) diagrams, formal languages, and logic may be introduced. In the next phase, a design document is produced to describe the structure and high-level details of the software to be built. This document

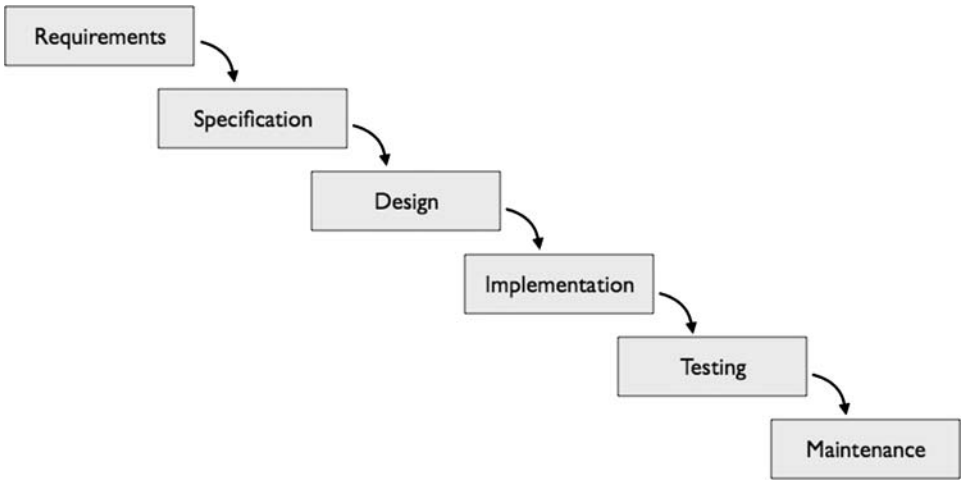

Figure 1. Diagram of the waterfall software development process model.

represents a shift to making decisions about how to build the software and serves as the starting point for the implementation work in the next phase.

Although documentation is not the primary product of the implementation phase, software developers are expected to put comments in their source code, make records of design rationale, and create developer notes for those who will be working on the system later. In the testing phase, the main document is a test plan, which can be as detailed as the initial requirements. During maintenance, records are kept of the changes that are made to the system. Depending on the organization and the criticality of the software system, changes may be tracked by a work ticketing or defect database, or changes may need to be described in detail, so they can be proposed and approved.

2.1.2. Scrum

Scrum is a Software Process that is part of the Agile "family" of Software Process models and programming techniques. The distinguishing feature of Agile methods is an emphasis on adapting to change by working in an incremental and iterative fashion, that is, taking many small steps repeatedly in order to grow a software system. The Agile approach to software development contrasts with phased, sequential processes, as typified by the Waterfall model, which seeks to minimize change through careful study and planning. The two best-known Agile models are Extreme Programming (XP) (Beck 2000), which is more concerned with how software ought to be written, and Scrum (Schwaber and Beedle 2001), which is more concerned with how software projects ought to be managed.

The Scrum model is depicted in Figure 2, below. Rather than dividing the project into phases corresponding to activities, time is divided into increments, called "Sprints" or iterations, where the objective is to produce working software that is another step closer to completion.

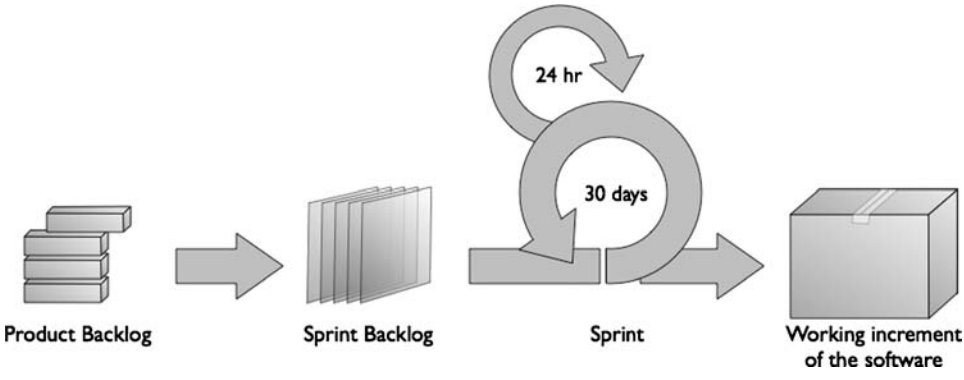

Figure 2. Diagram of the scrum software development process model.

Sprints can vary in length from 1 week to several weeks. In the figure, they are represented as 30 days. Several sprints grouped together is called a "release," and can correspond to a time window (e.g. fiscal quarter) or a logical group of features indicating a level of achievement. A sprint begins with a Sprint Planning meeting in which items from the Product Backlog (called User Stories) are selected and estimated for the upcoming time increment. This set of items becomes the Sprint Backlog. Every 24 h, the team synchronizes their work in a "Daily Stand Up Meeting," which is a short meeting lasting no more than 15 min and is usually held at the beginning of the workday. A sprint concludes with a sprint review meeting to look back on the tactics that were supportive and detrimental to progress.

Iterative and incremental development models have been around for decades (Larman and Basili 2003), but users tended to be isolated, and techniques were invented locally or adopted piecemeal. The current move towards Agile is marked by a large and growing community of users and advocates who are all focused on promoting and applying Agile techniques. The flagship conference in 2007 was sold out with approximately 1,100 attendees, while the Agile 2008 Conference had over 1,600 attendees from around the world. Furthermore, there is a significant cultural and social component to this generation of iterative and incremental development methods, as illustrated by the Manifesto for Agile Software Development (http://agilemanifesto.org/), which states:

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

This declaration was created in 2001 and signed by seventeen luminaries from the professional software development community. It is the single clearest description of Agile, because this phenomenon is best understood as a movement and includes not only programming techniques and project management methods, but also values, culture, and worldview. For instance, XP is defined by four values (communication, simplicity, feedback, and courage), fifteen principles (e.g. Embracing change, Quality work, Travel light), and a set of twelve practices (e.g., Collective Code Ownership, Pair Programming, Simple Design, and Refactoring). Even the organization of the seminal book on XP by Beck (2000) signals a departure from conventional software development. The first third of the

book is devoted to sketching out a new approach to problematizing software development, the second third of the book gives a strategic explanation of XP and how it can work, and only the final third of the book provides guidance on how to implement XP on a project. Robinson and Sharp (2003) observed that in order for this set of twelve interdependent technical practices to function effectively, they need to be supported by values and a community. However, these are not the only technical practices that will give rise to an Agile project or community, leading them to conclude that the twelve practices "both are and are not the most significant thing."

The basic unit of work in Scrum software development is called the User Story (See Figure 3). According to Cohn, a User Story has three parts: a written description of work to be done, conversations with the customer about the work, and the test cases (Cohn 2004). A common format for the written description is "As a <role>, I can <action>, so that <goal>." They are partially a software requirement and partially a to-do item. They should be small, so that they can fit on a 3″×5″ index card (or sticky note) and can be completed within a single sprint. In this formulation, User Stories include both static artifacts (the written description), an interactive aspect (the conversations), and a computational aspect (automated software tests). It is common to post index cards on a Scrum board for everyone to see (See Figure 4). The board is typically divided into columns (e.g. not started, in progress, tested, and accepted) and the placement of the index card indicates the status of the task.

Because one of the Agile principles is to favor working software over comprehensive documentation, lightweight record-keeping techniques are used, including index cards, sticky notes, bulletin boards, and computer code. In this context, it is understandable that adherents of both Agile and non-Agile development processes might think that Agile uses minimal documentation.

There are five primary roles on the Scrum team: Scrum Master, Product Owner, Team Members, Stakeholders, and Users (Schwaber and Beedle 2001). These roles do not necessarily align with conventional job titles, for instance, a

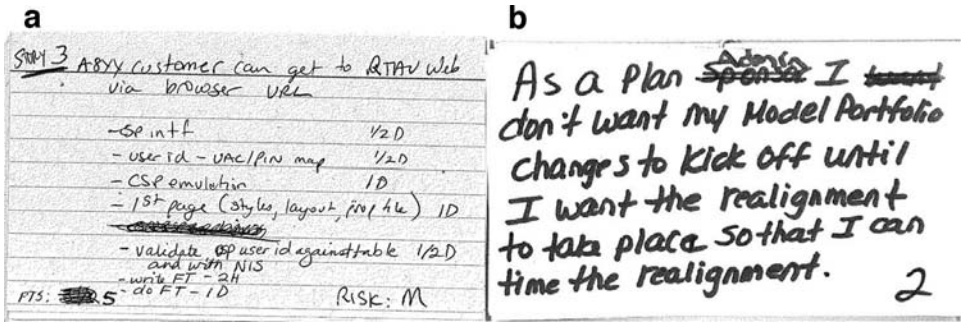

Figure 3. Samples of user story cards from each of our field sites.

Figure 4. The Scrum board at one of our field sites.

software developer could be a Scrum Master or Team Member. We will describe each of the roles and give possible corresponding job titles.

Scrum Master. This person facilitates software development by removing impediments and tracking the progress of the team. Typical tasks for the Scrum Master are convening the Daily Stand Up Meeting, keeping a burn down chart to record the tasks completed, maintaining the Product Backlog, preparing for sprint planning meetings. Project managers are often Scrum Masters, but not always. There may be a technical manager as well as a Scrum Master. A software developer could be a Scrum Master, but a Scrum Master does not necessarily have coding skills. Usually there is one Scrum Master per project, but with very large projects, there can be a Scrum of Scrums with multiple levels of Scrum Masters.

Product Owner. This person represents business concerns and can be literally the person paying for the project or a figurative surrogate for customers. The Product Owner's main responsibilities are writing User Stories and prioritizing the Product Backlog. This role can be fulfilled by a senior manager, a product manager, someone involved in sales and marketing, a business analyst, or a user experience designer.

Team Members. This group of people is engaged in the work of creating the working software. Team members can include software developers, software testers, database analysts, system administrators, technical writers, and multimedia artists. It is up to the team to decide who is included, but membership is usually based on perceptions of the "real work" on a project.

Stakeholders. Anyone with an interest in the software product is a stakeholder, but in practice, stakeholders are limited to those with business interests, such as customers and vendors. They tend to be involved only at release or sprint planning.

Users. A subset of the stakeholders are the Users of the software. Business stakeholders often are not Users. This role is included to remind the team that software should be built for *someone* to use. Individual users are usually not participants in the process like the other roles, but are consulted regularly to obtain feedback. This role can be fulfilled by someone who uses or will use the software, or by a surrogate for them.

Software process, according to these models, is product-oriented and aimed at delivering the software system. All of the activities and artifacts made explicit in these models set the stage for implementation, making incremental progress towards software code. The artifacts in each stage move a step closer to implementation. As the development team moves from requirements documents to design documents to system specifications, the technical language is refined such that the final step of writing of software code can be made error-free. Ultimately, what Software Process gets you is the software system. Whether in phased or iterative and incremental development, the focus is on how artifacts are developed as outputs of a particular stage or iteration of development and used as inputs for a subsequent phase/iteration (See Figure 5). The software process provides the prescription for generating and utilizing a set of artifacts as the means to produce software code.

2.2. Software process enactments

A different set of research on Software Process de-emphasizes Software Process models and focuses instead on enactments—specific instantiations of Software Process (Fuggetta 2000; Feiler and Humphrey 1993). This move has been alternatively dubbed as a shift from prescriptive to descriptive (Dittrich 2002) and methodical to amethodical (Truex et al. 2000). This work highlights that there is more to the work of software development than can be represented in Software

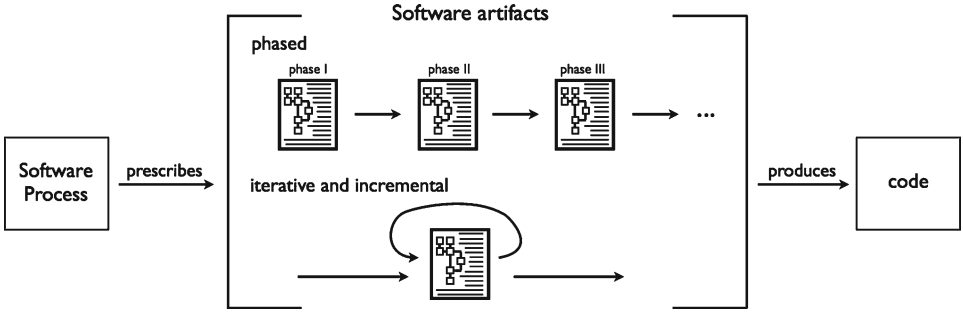

Figure 5. Software process models.

Process models. The research is motivated by the complexity of software development and focuses on process as something accomplished through situated actions, local contingencies, work-arounds, exceptions and deviations from the model. Based on empirical studies, this research claims that models are resources to and accomplishments of the work of software development, and are "undeniably social" and non-instrumental (Dittrich 2002; Fuggetta 2000). According to this view, software process serves as a resource to ongoing software work, along with various prescribed and ad hoc artifacts, and is an accomplishment of software work, rather than strictly a means to produce software code (See Figure 6).

Our research shares this theoretical framing that Software Process is not necessarily a prescription for software development work. Our study results align with evidence from various studies. This research indicates that formal processes and methods are chosen based on social, as well as technical, criteria and at least in part to make the work "organizationally accountable" (Button and Sharrock 1992, 1994). Methods and models "cannot be divorced from their practical use" (Button and Sharrock 1994) and are at times "faked" (Parnas and Clements 1986). Adherence to method is part of the accomplishment of software development and work is often

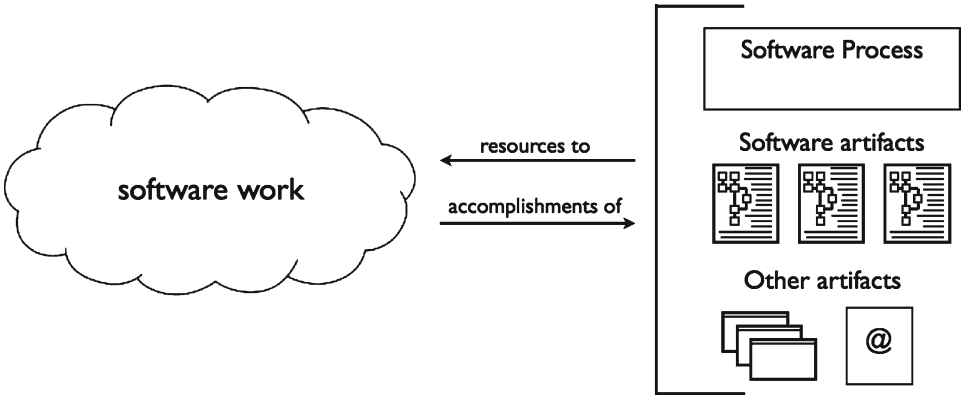

Figure 6. Software process enactments.

performed *as if* in accordance with Software Process models as a way to make the work acceptable and accountable to others. This research demonstrates the importance of organizational context and economic market factors (Fuggetta 2000), organizational roles that bridge between the business and technical worlds of problem and solution, and selective use of codified methods (Sim et al. 2008).

Research emphasizing Software Process enactments takes a broader view of the importance of artifacts in software development work. Studies tend to include treatment of all kinds of artifacts not accounted for in Software Process models such as notes on scrap paper, whiteboard drawings, emails, and programming environments. This research demonstrates that software developers must rely on direct communication across technical boundaries in order to disambiguate specifications or tease out faulty assumptions in documents (Bansler and Bødker 1993). It is not enough to consider those artifacts that document only the "true" and "essential" requirements nor is it simply a matter of improving the artifacts, or developing processes which generate higher quality artifacts. Rather informal communication and supplementary artifacts are needed to work out the meaning of prescribed artifacts (Nørbjerg and Kraft 2002).

The outcome of Software Process from this perspective is not only software but also the work itself, how it is organized, its sociality, culture, and values. The interactions between people and the collaborative effort are described and understood in their own terms. This perspective emphasizes the relationships between people and the conditions of work, as well as the software system as product. In a way, this research takes an all-inclusive perspective of Software Process, within the limitations of the research design, considering all artifacts and enactments which are relevant to the analyst's scope. There is no technical rationality that prescribes what counts as the outcome of the software work, rather it is based on the descriptive aims of the researcher.

3. Software process as a generative system

We conducted a qualitative research study to understand the role of artifacts in the work of Agile software development. We set out to understand:

- What is the Software Process at our field sites?
- What is the role of documentation in the Software Process?
- What is the relationship between the Software Process and artifacts?

During the data analysis a new question emerged: What counts as Software Process and for whom? Or, in other words, who decides what counts as the Software Process? Answering this question is not simply a matter of understanding how and when the formalized Agile Software Process model gets deployed or invoked. Instead, it is a matter of understanding what counts as the Software Process in the day-to-day of software development work.

Our study takes as its starting point the knowledge that Software Process is an accomplishment of software development work and can be discovered only through careful empirical investigation of the context, conditions, and specific enactments of software work. From an analytical standpoint we consider process to be all the work that goes into the making of software. But in trying to understand what Software Process means to our informants, we realized that it was not inclusive of all the work that they perform. We were led to ask, what counts as Software Process *for them*? Who decides what counts as Software Process? The answer we found in the data is everyone, all the time.

Our analysis suggests that Software Process can be viewed as comprising both models and enactments which work together to form a generative system (See Figure 7). Software process as a generative system is made up of patterns of action which people can recognize (models) as well as specific performances of work (enactments). Models and enactments mutually constitute the Software Process which is produced and reproduced through the interplay between the two. Viewing the Software Process in this way means that we can consider the models that we construct as analysts as well as the models that our informants construct of the Software Process. In fact, models are always multiple, since models will differ from one person to another, or from one moment to the next, depending on

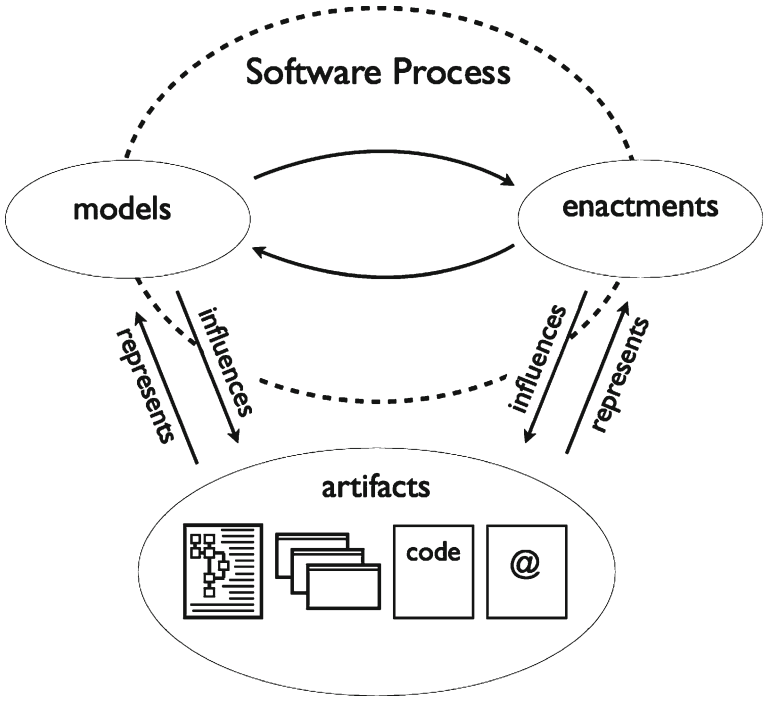

Figure 7. Software process as a generative system.

the point of view. Enactments are also multiple in that even the most codified, standardized, and regulated processes are open to fluctuation and change in practice.

While models and enactments form a generative system, they require different kinds of looking and thinking about the interactions, conversations, and artifacts that make up the collection of effort of software work. There is a tension between these two modes since it is difficult to hold both simultaneously. Any view of software work can shift between these two modes. We can describe specific enactments in specific times and places, or we can think about patterns of action and create stories of typical actions.

Dittrich refers to this tension as an "understanding from within" and a "looking from without" (2002). With this metaphor she draws attention to the balance between the concerns of the researcher who understands from the outside and the concerns of the practitioner who understands from the inside. In fact, we found that our informants too can move between these two modes, understanding the process through specific performances, and reflecting upon their own process by viewing it from the outside. We found that our informants switched between these two modes and that it was the switching between the two that helped to form the boundary of their Software Process which in turn helped them to scope the software system.

Artifacts are everywhere and of many different types. This richness of artifacts in software work grabs our attention as analysts who are trained to think of Software Process in an all-encompassing manner, to look for and consider everything from scraps of paper, to emails, to software code. All of these artifacts are crucial to *our* understanding as software researchers of how Software Process is constituted, shaped, and negotiated. What was most interesting to us was that certain artifacts were used in ways that our study participants considered outside of the Software Process. Artifacts also cross the boundary between the outside and the inside of the process, negotiating the boundary as they cross it. And artifacts exist inside of the Software Process as well (See Figure 8, below).

Artifacts thus represent and influence both models and enactments (See Figure 7). We found that our informants tended to switch their mode of reflecting on artifacts as these artifacts moved from outside to inside the Software Process. When artifacts are outside of the Software Process they are considered more in terms of the ways that they represent and influence *models* of the Software Process and software system. When artifacts are inside of the Software Process they are considered more in terms of the ways that they represent and influence *enactments of* the Software Process and software system.

The outcome of Software Process is the Software Process itself and the work it comprises. The Software Process is generative in that it creates and recreates the Software Process through the interplay between models and enactments. In a way it ceases to make sense to consider the product of the Software Process or to think of the Software Process in terms of its inputs and outputs. The Software Process

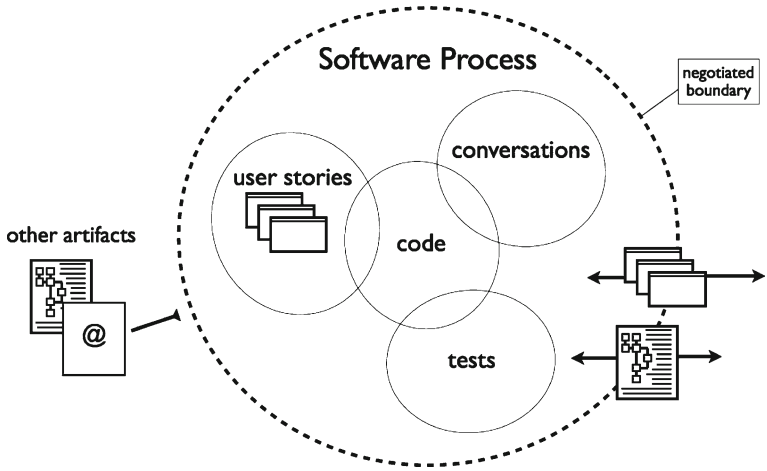

Figure 8. Negotiating the boundary of the software process.

has an outside and an inside, but it is never really done unless the business rationale which brings a team together dissolves. In our analysis of the data we found a blurring of the Software Process and the software system. Rather than the Software Process generating code as its output or product, the software code is an artifact which represents and influences the current enactment of the Software Process, that which is inside The Conversation. Code being written, code in the build, code running "green" through a collection of tests, and code in use, are all part of the Software Process.

4. Research method

In order to gain a detailed and contextually-situated understanding of the role of Software Process and artifacts in software development work, we conducted a week-long qualitative study at two software development companies in Colorado, USA. We chose to study Agile software companies because they present a particularly appropriate setting for the study of artifacts and Software Process. The models of Agile and Waterfall Software Processes described above show that they are organized in significantly different ways, particularly in their prescribed relationships to documentation. According to the models, documents in Waterfall are handed down from phase to phase and once their details are finalized, these documents do not change. Research has shown that in practice, changes to an upstream document can involve a great deal of re-work and effort to propagate the modification into later phases (Boehm 1981). Clearly, in iterative and incremental Software Process models, such as Agile, the approach to documentation needs to be radically different. In Agile, planning and decisions are made as the project progresses and documentation is always in flux. In view of this, it

makes sense that many Agile and non-Agile adherents view Agile as lacking in documentation altogether. We even found that many of our informants differentiate different Software Processes by the kinds of documentation they have[1]. Thus, we think it is particularly appropriate to investigate the relationship of Software Process to documentation further, and in a setting which is organized differently from more traditional settings.

Our research sites were two Agile software development companies. One research site, Easy Retirement, is a software company that provides software to companies so their employees can manage federally-regulated self-directed retirement investment plans. It uses a software-as-service business model. It was founded in 2000 and currently has 26 employees. A second research site, Fast Tools, develops a multi-user Software Process management suite. Fast Tools was founded in 2002. In 2007, it was the fastest growing company in the region. Both companies have adopted Scrum, an iterative and incremental development model, as their software development process model. We identified these sites through an Agile development support community and chose these two in part because of their significant differences. Easy Retirement is a relatively small company that transitioned to Agile in 2005 by hiring an outside consultant for Agile training. While they had fully adopted Agile, the staff liked the change to Agile to varying degrees. Their motivation for adopting Agile was to excel more in their field. Fast Tools, a very large company, was formed to support Agile processes. Its founders are Agile adherents and its staff were well-versed in the Agile rhetoric. For this reason the two companies reflect different motivations for their Software Process and attitudes towards Agile.

We conducted interviews, collected artifacts, and observed work in progress throughout the week, with particular attention to planning meetings during which the team members utilize and create documentary artifacts for the development process. We conducted a total of 15 interviews lasting 30–68 minutes with team members. We chose our interviewees to represent a wide range of positions within the Scrum teams including a company president, a system administrator, a trainer, sales and marketing, software developers, "Product Owners," and "Scrum Masters." We also followed up on recommendations from our informants about particular individuals in the companies we should speak to. In the interviews, we asked our study participants questions about how they gather requirements, what happens in a typical day, and how they know what to build into the software system next.

We observed meetings and work in progress at both companies. We sat in on one sprint-planning meeting at Easy Retirement and a daily stand-up meeting at both companies. At Easy Retirement we observed work in progress with ease, as all of the developers were located in cubicles open to a main area by the scrum board. The Scrum Master, Chief Operations Officer, and Compliance Officer were also located in cubicles or offices facing the open area in front of the scrum board. From this spot we could hear the conversations of the developers and observe

when they gathered for ad hoc meetings with other team members. We did not observe programming directly, because we did not have a view of their terminals, but we observed the interactions between developers and conversations about programming tasks. At Fast Tools, the staff were in the midst of a quarterly review meeting and were not working at their desks. While we could observe only minimal work in progress, we did receive multiple tours of the office by different staff. We conducted our observations from an open work station where some team members gathered to go over customer data.

We collected samples of artifacts by taking photographs of artifacts in use, as well as collecting older artifacts (user story cards) that were no longer in use. We asked our interviewees to explain artifacts in their work environment and asked them to describe the sample artifacts we had collected. At Easy Retirement we collected sample user story cards and observed the story cards in use on the scrum board. For example, we could see whenever a developer came to the scrum board and moved a user story card or took it off the board. At Fast Tools, there were user story cards everywhere, on the walls of every cubicle and office, and even on walls in the hallways. We asked various staff members to explain these sets of artifacts to us.

After completing data collection, we transcribed our field notes and audio recordings of the interviews. The transcripts were analyzed using 'focus coding' (Lofland et al. 2005). Focus coding is a technique that allows pieces of information in the transcript to be coded to highlight points of interest for our study. Following transcription and coding, we cross-tabulated the codes between files to categorize our findings, which are presented in the next section.

5. Research setting

This paper is concerned with Software Process models and enactments and the role of documentation in Agile software development. We will present more in depth findings about the relationship between Software Process and artifacts in the next section but first we want to provide an overview of the Software Process at our fieldsites and the kinds of documentation used. Some of the activities and artifacts we observed aligned with the prescribed Agile Software Process described in the previous section, and some did not. We will also describe a bit about the attitudes that our informants had towards Software Process and documentation. Many of our informants view Agile as light on documentation and process. They see Agile as organized in stark contrast to Waterfall. They also view the user story as Agile's basic unit of work and describe it as made up of a set of artifacts and conversations. These attitudes align with how Agile is normatively portrayed.

Ryan, a software developer at Easy Retirement, worked there before their switch to Agile. He was hesitant about the switch at first. His coworkers suggested that we speak to him to hear a less favorable perspective on Agile,

but we found that he mostly spoke positively about the improvements in the company since the switch. For him this had a lot do with the different role of documentation.

> In the old days we used to do Waterfall, where we spent like three months just building documents. That was not fun. And nobody would get to try out a bit of code for about three months. And then by the time your document was done, your document was out of date, typically. It did not really serve its purpose because when you started doing development it was like, you kind of have this document that kind of sort of shows what you were shooting for. But because things had changed since you designed that, you sort of end with another approach anyways, where you build pieces, do it piece by piece. And you were kind of wasting that time during all this repetition, and those documents never get up to date. So they were not even useful once the project was done, you know. They did not reflect reality any more. That was part of the biggest improvement I saw in development with Agile over Waterfall.

As Ryan notes, documentation in Agile does not serve to "reflect reality" because reality is constantly changing. We heard from many participants that Agile allowed the company and the software to adapt quickly to change. We also heard from many developers this valuing of coding as the most active and engaged kind of work and that Agile allowed them to focus more on coding.

Alexis, at Fast Tools, was relatively new to the company and to Agile. She began in a role of Usability Designer, interviewing customers about their experience and is now transitioning into the Product Owner role, with support from Brett. From her experience with customers, she explained that the switch to Agile is often difficult because of how it changes the relationship to process and documentation.

> I think that the thing that is hardest, is that there is a real cultural shift away from dependence on process and structure and documentation and more towards collaborating. I think for big companies that is difficult because this whole idea that a team can decide for themselves how best they should work is foreign to them, because a lot of companies are very process-intensive.

Here we see the valuing of collaboration over structure, process, and documentation. Alexis also emphasizes that this is a "cultural shift" revealing that the change is more about what gets valued than how things are done. Agile is not just a sequence of activities, but a mindset that, for example, teams can work independently. These values expressed by Ryan and Alexis adhere strongly with those in the Agile Software Process model described in the previous section.

The planning meetings reveal that there are different ways of achieving adherence to the Agile model. For example, we observed various tactics to get people promptly to the daily 15-min stand up meetings. At Fast Tools you pay a dollar to a shared petty fund for every minute you are late to the meeting. At Easy

Retirement, the Scrum Master rings a triangle to gather people to the Scrum Board. Other factors lead to improvisation with the Agile Software Process model. Working with legacy code means working with code that was not developed using Agile methods. At Easy Retirement, this was a huge problem. They contrasted the new and old code, even giving them names and personas[2]. Estimating tasks which dealt with the legacy code would require extra research "into the code" and time to work out the user story.

Transitioning to Agile roles also revealed some interesting tensions since Agile roles do not always map cleanly onto traditional roles. This means that switching to Agile can require interesting translations of work roles. For example, at Easy Retirement they tried out different people as Scrum Master and then settled on Leanne who had formally been a product manager. And at Fast Tools Alexis and Brett were both serving Product Owner, though Brett was from sales and Alexis was a Usability Designer.

In switching to Agile, the team at Easy Retirement saw Agile as giving them a kind of tabula rasa for the Software Process. Leanne said that they tried to create "a blank canvas" of their process during the transition and asked the consultant hired to give them training in Agile methods to "treat them like a lab" for experimentation. Yet, we also saw people who found a middle ground between the old and the new. This of course differed from person to person depending on how much they liked Agile, or how much they liked working collaboratively. According to Leanne, some people did not adjust to Agile and were let go. She explained that some people really like working alone and that Agile requires conversation. Even now Leanne plays the role of enforcing conversation. A developer might come to her with a question about a feature and rather than answer it for them as she would in the past, by going to the user (in-house) and going back with an answer, she will tell them to go talk to the user directly.

As described in the Agile Software Process model, the user story is the basic unit of work and is made up of a set of artifacts and conversations. We found this to be true at the field sites as well both in what we observed and how people describe the user story and how it worked. User stories are the primary way that requirements were captured at our field sites. Our study participants described and interacted with the user stories in ways that align with the Agile definitions of user stories. In our observations and interviews we saw that the user story is not just what is written on the user story card but is actually made up of a set of artifacts: the user story card, the task card, test cases, and code; and conversations: the conversation during which the user story card is written, the conversations that disambiguate the card during implementation including those with the author of the card or with the in-house user whose perspective it inscribes, the conversations about its relevant tests, the conversations about the system-in use, and the conversations about the code which it "touches."

The user story cards were written during the planning meeting during which the entire development team was present, including all software developers, the lead

engineers, chief operations officer, the Scrum Master, and representatives from the sales, business, and marketing team. Different team members would take turns writing user story cards and sometimes would hand off a story card for someone else to finish writing. There was debate at times about what should be on the card, and sometimes a half-written card would be scrapped and started again on a fresh card.

The user story cards include information which links to its author, the point of view of the particular stakeholder, and the person who will take on its implementation tasks. Participants were able to tell us about a user story card from previous releases or projects, telling us who wrote it based on hand-writing, the person who worked on it based on notes added to it later. In showing us stacks of old user story cards, our study participants were able to reflect on the project in different ways. They could note that the velocity for the project was high because lots of cards were finished in each sprint, or that there were a lot of bugs based on the prevalence of pink cards. They might recall who was on a project or some big obstacles they encountered. But individual cards and what was written on them were relatively meaningless. Without being connected to what "we're actually, really working on, people forg[et] what that little card mean[s]." Looking at a recent user story card prompted our study participants to describe the context in which it was written and the conversations taking place around it. When a user story card is written, there was often some debate about what it was and how it should be implemented, but these details were not written on the card.

Greg, a software developer at Easy Retirement, explained that he relies on conversations to fill in the details that are not on the user story cards. "You have people that understand the larger picture too. So you can look at the task and go, well so and so is working on this task at the same time or just before [me]. So you can have some context there where you might be able to use some of what he did." Or as Leanne explains, "There are some stories that everybody understands, but there is usually one or two stories that we will talk about in the sprint planning and we just have a vague understanding. [For one] really complex issue... we went around and around on that." This history that builds up around a user story is important. Greg laughs saying "I will say, well how did it work before and what is the delta between how it worked before and what I need to make it do now?"

This sense of "delta" as small increments of change and working "piece by piece" are the aspects of the Agile process that our informants highlight as advantageous. The user story exists as a kind of documentation that is already a "piece by piece" formulation of work. The user story is just a tiny piece of documentation existing in combination with the work that provides its context. Thus, a piece of code work and a piece of documentation work are crafted together.

Observing meetings reveals just how much negotiation takes place. The user story card plays a central role in this negotiation. At both Easy Retirement and Fast Tools, the meetings we observed had strict turn-taking procedures. At Fast Tools, everyone convened around a conference table for the daily stand-up

meeting (literally standing up around the table behind the conference chairs). They waited for the head of company before beginning the meeting who then identified someone to start first and then they proceeded around the table. At Easy Retirement the marker used to write user stories was used to indicate whose turn it was to speak. Our informants noted that it can be contentious with "each person push[ing] for the work of his or her group." We observed several instances of disagreement about the user story card. Often this was a debate about what is part of the user story or not. "That's a different story," was a common interjection.

6. Results

In order to understand the Software Process and the role of artifacts we asked our study participants questions about how they gather requirements, what happens in a typical day, and how they know what to build into the software system next. For both software development teams, the user stories were central to the work of building the software system. If we asked where they documented requirements, the answer was twofold—the user stories are the main artifact used to document requirements, however user stories do not really document requirements. To our informants, documentation connotes a category into which user stories fit uneasily. We followed up by asking them to explain what a user story is and how it works. We also asked about other artifacts and observed artifacts in use. Sections 6.1–6.3 below present results and analysis in three themes: Living and Still Artifacts, Software Process as Conversation, and Artifacts Inside and Outside.

6.1. Living and still artifacts

We found that our study participants explained their Software Process in terms of Agile's user stories and by contrasting these with the requirements documents found in Waterfall and other traditional Software Processes. They contrasted artifacts which are *still* or *frozen* with those that are *living* and *in play*. They primarily focused on the role of requirements documents and user stories but also discussed test cases and code as artifacts which support the software development process. User stories were the most helpful for keeping the Software Process in play because they have a format which inscribes point of view, are written in front of other team members, are handed off between team members and are moved around the physical space of the office as a way to represent the Software Process in the current iteration. Tests are valued for keeping the software system live because they enforce keeping the system "green" such that code which would make the system fail to run cannot be entered into the code body. And the code itself is considered the best artifact because it *is* the system.

Members of the software development team explained the user stories to us in terms of what user stories are *not*. We heard many stories about how artifacts are used in other Software Processes such as Waterfall, based on past experiences and familiarity with the Waterfall models. Leanne, the Scrum Master at Easy Retirement, had served as office manager before the company switched to Scrum. While they tried out different people as the Scrum Master at first, she quickly settled into the role because, as she explained, she had always been a kind of liaison between the software development and business sides of the company. As Scrum Master she leads the planning meetings, keeping order and time, putting the user story cards on the board, and tracking the progress of the team throughout the week on a hand-drawn chart. She recounts what it was like for her to work with the requirements document before they switched to the Scrum model.

> We had a requirements document about an inch thick and we'd spend three to five days making sure all the developers knew what they are. And the developers went away and no one could interrupt them. No one could ask them to fix something. [The manager] didn't let them and they had to first consult with him and he would decide if it needs to be fixed or not. And [the manager] didn't know the problems as they couldn't contact the engineers. And it would take several months. And by the time they released [the software], the customer didn't get what they wanted anyways as they either had forgotten what was the thing they requested or they were complaining about not having what they wanted. [The developers] were not proud of our company [and] the management team blamed the engineers and vice versa. They really didn't talk with each other and I was in the middle.

Leanne and others emphasized the thickness of the requirements document in non-Agile methods and the time that it took to go through it as well as the ways it did not serve to help different teams communicate, or even seemed a barrier to communication.

Carol is the testing expert at Easy Retirement and is also a well-established author of books on test-driven development. She too described her frustrations with the requirements document before the switch to Scrum.

> We worked so hard to do a very disciplined process. We wrote our requirements document, our design document, we did all the steps and we delivered the product in six months and it was already out of date. The competitors were ahead of us. We needed more discipline and while you are doing that, the marketing will say, 'Well, we really need this feature.' Well, I am sorry, we are doing this project and the requirements are frozen. That is a crazy way to work but it's how people used to work and probably how people still work.

Carol and Leanne and others used these examples as a way to highlight how user stories allow for greater adaptability. Jerry, the owner of Easy Retirement, explained that the reason for the switch to Scrum was the need to "evolve" the

software. He said that at first they went with traditional software development methods but that after building up the base application they needed the software to "react just like the rest of the company does" to change. He saw people, particularly those trained in marketing, as very adaptable and good at responding dynamically to a change in the economy. Scrum allowed the software team to do the same.

Jerry explained:

> Whatever was written on the page and how they [the engineers] interpreted it, was what was developed. I would literally build the entire application in power point and just say when you push this button you are supposed to go to this web page and this is what you are supposed to end with.

In expressing his frustration with the software development team, artifacts stand in for the breakdowns in communication as they did for Carol and Leanne.

Another software engineer told us about the circumstances in which he left his prior job after working for over 6 months on a requirements document that was growing larger and larger. The customer wanted the document to describe the system more and more completely. For him, this was time wasted on trying to build the system on paper when the computational medium is effective because it is not paper. After leaving the company he discovered that 6 years later they still had not begun programming on this project.

Various team members described how they learned to cope with software documentation in the past. One method was simply not to read the documents. As Gordon, a software engineer at Fast Tools noted, "I've worked in a lot of places over the last ten to eleven years now. And in my previous company I never saw a person reading the documentation." The testing expert at Easy Retirement, Carol, explained that she would find work-arounds to negotiate what was in the requirements document. She would take the document back to her office, read through it writing up all her assumptions based on the requirements, and bring it back for verification. This was her way of doing test-driven development in an environment that had not adopted that technique. "Other people" just wait to receive the document and take it as it is, but she "never worked that way." Other participants noted that the best kind of requirements document is one made after the fact, in a sense "faking" the artifact as a requirements document.

The informants differentiate between artifacts which seek to represent the software system, providing a static image, and those that are "living" like the code itself. Gordon makes the distinction in the following:

> In documentation, the second you write it for a living application, it is still, you never ever are able to keep it up to date. On an application that has been around for 30 years and running in the mainframe and you see here is the process of how to deploy it, run it, I think that's good. But documentation like modeling, design... that kind of documentation I feel is completely still.

Artifacts that are static are appropriate for a system that is not changing, but otherwise are not helpful in generating the software system.

In contrast to the stillness of Waterfall documents, our study participants emphasized how user stories capture very little of the system in order to serve as a "trigger" for conversation or to encourage them to "talk instead of write." Sam, the compliance officer, and Product Owner at Easy Retirement, described the user stories as "triggers that help me to ask questions." User stories are often written on index cards during the planning meeting and many people will weigh in on a particular card, negotiating what should be written on it or discussing details of how it will be implemented, or what test will be written for it. The user story serves not to inscribe the details or outcomes of conversations that take place, but just to point to them. Gordon says that he prefers user stories because "the actual approach is it makes people talk about it as opposed to making people write it down." So, even if a question was already answered, if you forget you have to go and ask someone again.

User stories are contingent on context, roles, time, and location. They are interactional and gain significance from the ways in which they are moved and exchanged. Because of their highly contingent meaning, they date quickly and their meaning changes over time. The user story is actually a set of artifacts: the user story card, the task card, test cases, and code. These artifacts are "meaningless without the interplay" between the artifacts and the roles and individuals who use them.

Our study participants told us that the user stories work because they encode points of view on the user story cards. The user story card is commonly written to reflect a particular perspective such as that of the customer or a team member. It is often written from the point of view of the person writing the card, but not always. The User Story format which is invoked frequently, and sometimes used on the card, is "As a <role>, I can <action>, so that <goal>." Developers told us that they enjoyed writing requirements "on behalf of the user or the customer" and business people said that there was increased understanding of what the software development team does and why due to the user stories' inclusion of point of view. Jerry focused on how the user stories increase the ability of the software developers to understand the work of the company. "The user stories get put in the first person. It is a way to help people think 'Alright, I am working in Amy's role and Amy does this every day." The user story enforces a "learning curve" by requiring understanding of others' perspectives, roles, and work processes.

User stories can be tracked by different team members in a variety of documents, but the user story card is always written during meetings at which the entire software development team is present, and with representatives from other parts of the company. The user story card frequently includes information linking it to its author, the perspective it is written from, as well as the person who will be taking on the tasks associated with that story. This could be observed in the ways

that participants discussed specific examples of user story cards with us. They would note the hand-writing as belonging to a particular team member, and initials in the corner marking who would be taking ownership of the story. But more importantly, the stories are identified based on the context in which they were discussed. When a user story card is written, there is often some negotiation or clarification of what work it includes and these details are not typically written on the card; the conversation becomes part of the context for that card.

The software teams discipline themselves to write test cases before programming. The test case is part of the user story, though it is not always written on the user story card. Test cases are written at times by Carol in a wiki page and at times in an automated testing software that runs on top of the software build. Gordon emphasized the importance of test cases for providing a set of requirements for the system. He said that the test cases are probably the "closest thing that we have written down for specification... Because they are written informally in the scenario 'The system should do this,' that is actually the best reference, because it is what it should be doing." Later he continues saying, "I will take a test over documentation any day of the week because it *is* what the system *is* doing. It is going to be more accurate than comments in the code even."

The code too is considered in its role as an artifact supporting the development work on the latest release. The code "*is* a reference," Gordon says, "the code is the code," but it is not as good as test cases for providing requirements. Test cases are a reference "that is always up to sync" and that "you always keep running and turning green." "The code" on the other hand, "is in movement, evolving, so the best information is the current information in the heads of everyone on the team."

The study participants tended to focus on how artifacts support, represent, and influence enactments in Agile software development and how they support, represent and influence Software Process models and models of the system in Waterfall. This marks the two modes of reflection, model and enactment, as a way of distinguishing their Software Process from others. We can see, however, that it is possible to apply the two modes to both sets of artifacts as is demonstrated by the way that informants describe coping with and finding work-arounds for Waterfall requirements documents or in describing patterns or typifications of Agile artifacts in use.

A user story card gains its meaning from its interactions and movements. A user story card is contingent on context, roles, and time of creation. The user story card is an interactional artifact—its movements provide information; it is in flux, changing value as it moves. Its role does not serve to represent requirements but to prompt conversation—it has just enough information to prompt and identify a conversation. The user story's meaning is contingent on the role of the person who wrote it, the perspective it represents. Its meaning is also contingent on the time it was created—as it ages, its meaning degrades. All of these attributes encourage a view of Software Process as it is enacted and how these artifacts support and represent Software Process enactments.

The stories about artifacts in *other* software development models show an emphasis on how the Software Process is prescribed or modeled. In a sense, these are all views of the Waterfall process in hindsight and cannot be taken as accurate depictions of how Waterfall works in practice. The various coping mechanisms described provide a glimpse of how, in practice, these artifacts were made to work and meet their needs. However, what is interesting is the way that these stories define the Waterfall process model in terms of its artifacts and contrast the relationship between the Waterfall process model and its artifacts with the relationship between the Agile Software Process model and its artifacts.

Our informants held the opinion that all artifacts are contingent regardless of the adopted Software Process model. The Waterfall requirements document was contingent on organizational structure, roles and division of labor, it aged over time, and required interpretation and negotiation to acquire meaning. The switch to Scrum in part signified an explicit acknowledgement of this fact within the company.

The artifacts used to support the software development work are "meaningless without the interplay" between the artifacts and the roles and individuals who use them. The user stories can be viewed alternately as a model and an enactment. At times participants focus on how user stories support enactments of Software Process because they are written, discussed, and taken in front of others. At other times they are emphasized for the way that they model a future part of the system in terms of someone's point of view and increase team members' ability to build a model of the work of others.

6.2. Software process as the conversation

In the previous section we discussed the interplay between the artifacts and the conversations that serve as their context, but we have only just begun to peel away the layers of the user stories and the work that they support. According to the study participants, the user stories also encourage a sense of participation in the Software Process. They make being there, witnessing the writing of the artifact, and exchanging it, important. User stories are seen as successful because they require reference to these conversations to be meaningful, so much so that they are faulted for becoming meaningless very quickly as time passes.

In explaining how user stories support this participation in the Software Process, our study participants often describe the Software Process itself as a conversation in which to take part. User stories help to generate, track, and remind participants of the conversation, what and who is a part of it. User stories and other artifacts serve as a kind of collateral for entering into the conversation or remembering what has been said.

One of the particularly interesting features of the user stories is that they are sometimes written prior to the stand up meeting as a way for individuals or a couple of team members to brainstorm ideas for features. These artifacts are brought to the planning meeting, but frequently they are rewritten during the

planning meeting when they are discussed. Different team members bring different artifacts to the planning meeting which they never directly share with others but which inform the conversation and may end up contributing to what is written on a user story card.

Team members referred at times to the Software Process as a conversation or as "The Conversation." They would also speak about "bringing things into" The Conversation when describing user stories, ideas for features, or even ad hoc discussions that took place. We use this term, "The Conversation," from our informants to draw attention to the way that they framed the Software Process as a Conversation. Bringing user stories into The Conversation was sometimes synonymous with bringing them into the software system. Of course ideas for features can be rejected after some debate, but these are then considered not part of The Conversation. In this way the Software Process as The Conversation is conflated at times with the software system.

Everyone at Easy Retirement writes user stories, whereas at Fast Tools the product owners tend to do most of the writing of user stories based on interviews with customers. Leanne, at Easy Retirement, explains how user stories allow people in the company to feel that they are participating in Software Process:

> We encourage everyone to do the stories as everyone has his or her view of how the system works or should work and makes them feel they are part of the product and feel involved. [This applies to] the plan administrator, the sales people. Even the accountant has actually written some stories for the report data that she needs. And this is one of the things the people in this company like about it here is that they feel they are part of the things that are built and the process.

The user stories increase "group ownership of all the tasks" even though some members of the group do not know technically how the task will be achieved. Greg, a software developer at Easy Retirement who was hired within the last year, is still getting used to working at the company. He said:

> [There is a big difference between] sitting down with the requirements on your desk versus actually taking a piece of the requirement in front of everybody else. Now you've opened up the door to conversation about it and now you have perspectives from multiple people and not just yourself and that piece of paper that you think that you are interpreting correctly.

The story cards are also moved around during the sprint. At Fast Tools there are various white boards and tack boards with arrays of user stories taped or pinned up. At Easy Retirement all user stories are on a single Scrum Board. As the team works on the implementation of a user story, the cards progress from columns on the board: "to do," "doing," and "done." Sometimes a developer will take a user story down and will take it to his/her desk while working on it. "You can just look at the board and know how things are going." Leanne explained that

if there are too many cards still in the to do section or if there are a lot of cards of a particular color (pink represents bugs "because programmers hate pink") then you know you might need to push to get through the sprint on time.

Alexis, a product owner at Fast Tools explained that she might come up with user stories before the planning meeting even if things might change during the meeting. During the meeting "we go through each one [of the stories]. The developers ask questions about what you meant by this or that." And participants are able to recall these conversations as the context for the task because "everybody was there in that initial conversation." As one developer put it "One of the strengths [of the user story] is the fact that you are talking about the story in the context of the story... and having everybody there in the room at the same time talking about that task, before any development is actually being done." The fact that the story can be the context for the story is an indication that the story is both something linked to other artifacts and interactions as well as what is written on the user story card.

The code too is an artifact that, while it is the outcome of work, is also something which must be brought into the conversation. During the planning meeting if the code comes up a developer might discuss how to bring it in. Sometimes this means that some research into the code will be needed before a particular user story can be brought into the conversation. During one planning meeting, a developer explained to the rest of the team that a particular task was larger than his teammates expected because it "touched a lot of places in the code," particularly in the legacy code. Ryan explained the role of code in this way:

> I mean to solve it, you really drag out how long it's going to take you to do your estimate [of the user story] because that means before you estimate a story, somebody will have to do some research, you know, actually deep into the software to try to find all these problems that could show up. At times I do not feel comfortable with the estimate I got because I think, ohhh yeah, that part of the system is pretty bad.

In this way, different team members can be seen as more in conversation with particular parts of the code. Greg explained that he is able to talk to other team members if he needs to "get more context" for a task. He frequently goes to Chad, another developer, for help because he is "pretty in tune with the state of the requirements." The requirements are seen as moving "through" the process. Participants from both the software development teams and business teams at both companies often referred to the ongoing "conversation" of the development work. At times developers even referred to the code itself as a conversation or highly intertwined with conversation as Greg does: "There are certain aspects of the system that I can probably be more productive doing something else because I cannot really contribute to this conversation."

The outcome of the Software Process is not just the software system but the Software Process itself. The Software Process is a generative system which is

shaped by the perspectives that team members bring to the Software Process and by the artifacts which enter into, move around within, or stay outside of the Software Process. The perspectives of team members in considering how much they are in tune with or in conversation with various artifacts and parts of the software system help to define the Software Process for them. In speaking about specific enactments they will describe how particular ideas enter into the process or how they participate in parts of the conversation. And in modeling the Software Process by describing patterns of action or typical actions, they will refer to The Conversation as something which everyone takes part in collectively. In a sense, the conversations and The Conversation, map onto Software Process enactments and the Software Process model.

The user stories and other artifacts map out the space of The Conversation as well as support individuals' ability to take part in conversations. As Leanne says, you can see the conversation as it is happening by watching the cards. The user stories as larger than themselves, as encompassing user story cards, other artifacts, parts of the code, test cases which may or may not be written down, and conversations, lead to a picture of artifact, process, code, and conversation as inextricably linked. Indeed there were many instances where our study participants shifted seamlessly from talking about one modality (e.g. artifact) to another (e.g. conversation). In the words of our participants: the code is the documentation; the code is what we do; the tests are what the system is doing; the tests are the intent of what we are building; the user stories are the context but are also made up of the code, the tests, and the conversation; and the ability of any team member to contribute to a User Story is her or his way of contributing to The Conversation.

The role of code as an artifact supporting the Software Process and an outcome of the work leads it to be particularly fraught in terms of how it touches or is brought into the Software Process. There was frequent blurring of the Software Process and the software system in part because of the emphasis on keeping the system live or green. The code is simultaneously model and enactment since the code is part of a process of building up a model of the system and is also the system. The developers frequently describe building a model of the system in terms of how story cards, code, and tests provide a picture of the "intent of what we are building." While the software "build" conventionally refers to compiling and linking the source code, in Scrum, what is not being kept live through conversation is in a sense no longer part of the software "build." It is part of the background, automated. Once you have the build, if the code is never touched again, it is no longer part of the Software Process.

The re-writing of user stories in front of others even when they do not change, also points to the way in which user stories are both models and enactments. As an artifact which a team member brings to the planning meeting it is a model of some intent of what they are building. As something traced out on a card for

others to see it is an enactment of a kind of agreement that this story is now a part of the Software Process, part of The Conversation.

6.3. Artifacts inside and out

In this section, we will present three anecdotes from the data about artifacts that reveal the interplay between Software Process and artifacts.

The first anecdote is about Carol's test wiki which she has developed to maintain a database of test cases. Her wiki contains both the tests which can be automated, with links to their implementations in Fitnesse (a test automation software), and tests which must be performed by a human tester. When we first asked Carol and others how they document requirements for the system, even though they acknowledge that the tests are the closest thing to requirements, they did not mention this very large wiki that contains tests. Carol showed it to us only when we asked her specifically, where do you write the tests. She was completely willing to show it to us when we asked but downplayed its importance.

While the wiki was a kind of historical record or archive of all the tests ever written, it seemed to be more for her own personal use. She was meticulous about keeping it up to date in a way that was reminiscent of her story with the Waterfall requirements document which she pored over looking for assumptions. When we asked whether other team members contribute to the wiki or use the wiki, she consistently said that she was not sure if they did or did not. She said that on occasion someone will ask her a question and rather than answer it she will email them a link to a page in the wiki where it is answered. In these cases she was pretty sure that they read the selection they needed, but that often she had to send a link to the same page again if the question came up again indicating that others were not familiarizing themselves with the wiki extensively.

When we asked other team members about the wiki, their answers were also mostly shrugs. They knew that Carol kept the wiki, but did not know much about it aside from the specific times and circumstances in which she had referred them to it. Carol was even a bit apologetic in her explanation of why she kept the wiki, saying that she did not know why she kept it, and that she was sure it was redundant with other artifacts being used by other team members. She said a few times that they have been meaning to sit down some time, (her and Sam and perhaps others) to consolidate these artifacts, but have not gotten around to it.

As for how Carol uses the artifact, she said that she uses it to keep a complete picture of all the tests for herself. She also mentioned times when she used it as collateral evidence in conversations because she can find the email in her email archive where she sent a link to a team member in response to a question. Thus it is a way to keep others accountable to doing disciplined test-driven development and to the decisions they make together about tests.

A second anecdote is about Sam the compliance officer and an artifact he uses to keep track of user stories. He has two artifacts, one that is a spreadsheet of user

stories with dates when they were written, added to a release, and completed. Another is a word document template that he uses to create a document for each user story. He showed us how he goes through after each planning meeting and enters in all the user stories for the new sprint and at the end of the sprint goes through and checks them off as completed.

Sam was similarly reticent about these artifacts. He was reluctant to show them to us only because he did not think we would care about them. They were just artifacts he used for himself, he said. After sharing them with us, he admitted that he had actually designed these artifacts as a way to keep track of features before they switched to Scrum but had continued to use them after the switch.

Yet, the switch to Scrum had changed the way that he used these artifacts.

> I even have these little things like sign-offs (laughs) at the bottom. You know, like I'll put everything in there, go to that story owner, have them read it, [and ask] "Does this capture 100% of what you want? If so, sign here." Like a contract kind of thing. I have never gone to that extent, but when I put [the template] together I had thought of that. It might be a more formal thing that some organizations would need. I put it on there, but I never really use it.

The artifact had a place for a signature as if the user story were a kind of contract for work. However, he had not used the signature line since switching to Scrum. Sam said that he made this artifact for himself, for his needs as compliance officer, yet it turned out that he would sit down each sprint, before the planning meeting with Carol and maybe a product owner, and brainstorm user stories for the sprint.

The final story is about how Jerry, the company owner, relies upon user story estimates to negotiate with customers and potential industry partners. He too sees the user story as sort of mini-contract for work.

> If we have an existing business requirement, you fully explain our process to anybody who said that they want something [a new feature]. And I told them, if you want it to be prioritized, here is what it means to me. So it is number 5 under 122 others, (laughs). You want it to be in the top five? Tell me you will pay me 25 thousand dollars, otherwise just understand that there is no return on the investment. [I use] the user points to assign a value to it. So if it is a 5 point theme it is going to cost just around 15 thousand dollars and so you are able to give that kind of information to somebody and say OK, so it is going to cost me 15 grand and the best case scenario is over three years and makes me 60 and that means y equals x. I've got these three things that we are doing that you are interested in that can make us both exponentially more money, so why would I prioritize that above those things.

A similar kind of negotiation can take place internally on a smaller scale. Leanne said that sometimes the business people are "asking for the world" because they care more about sales than anything else. They might not understand that a request is very big, for example if it requires changes to the

legacy code. Leanne would go to Jerry and explain that it is a “very expensive” feature, relying on estimates to convince him.

These moments reveal that software work can take place outside of the Software Process and that artifacts can support work both inside and outside of the Software Process. It also gives a window into how each team member uses artifacts to negotiate the boundary of the Software Process. Each team member builds up his or her own model of the Software Process and the software system. Carol's dismissal of her wiki, Sam’s marginalization of his user story sign-offs, and Jerry’s reliance on the user stories to project the value of features, demonstrate the way that they each negotiate the boundary of the Software Process on an individual basis.

These also demonstrate in each case the interplay between the model and the enactment since the artifacts support both modes. For Carol and Sam, telling us about what they do in specific cases with these artifacts was an afterthought or apology. Jerry on the other hand works more explicitly with negotiating the boundary of the software system by using the user story to account for models of the system.

These artifacts could be viewed as deviations from or work-arounds to the Software Process model, but our analysis is that these artifacts exist outside of the Software Process, defining and negotiating its boundary. This is achieved in part because the Software Process is bounded by keeping artifacts that represent and influence models of the Software Process or system on the outside and artifacts that represent and influence enactments on the inside. Carol and Sam’s artifacts in particular seem designed to support more of a view of the Software Process as model from the outside. Sam's lines for sign-offs, though never used, are an indication that the document serves as a representation of work. Signatures are one of those odd moments where an enactment of signing actually designates the artifact as a model. Carol’s wiki too resembles some of the specifications documents which aim to provide a complete model of the system to be built. Indeed with her wiki someone could readily recreate the software system.

The fact that Jerry, Sam, and Carol all use the artifacts as a form of collateral reveals how these artifacts can enter into particular enactments of Software Process. While Sam's spreadsheet primarily serves as a picture of the Software Process and software system, he also uses it to track conversations as a reminder for or form of evidence of a decision.

As Gordon at Fast Tools put it, there are many artifacts which “radiate out from the code.” Certain artifacts are central to the Software Process such as user story cards, code, and tests, others are peripheral to the Software Process such as these improvised and personal artifacts, and others move across the boundary of Software Process such as user stories. Yet there are no smooth transitions of artifacts across this boundary. Instead there is a kind of tension, or a pressure that the artifacts exert on the boundary from within and without.

7. Discussion

7.1. Artifacts

We have shown in our analysis that artifacts serve to represent and influence both Software Process models and enactments. The Software Process is a generative system which relies upon both points of view. Our study participants move between these two points of view as they move from inside to outside the Software Process. And artifacts are integral to their ability to shift perspectives, to work within and outside of the Software Process, and particularly to negotiate the boundary of the Software Process.

Research on boundary objects (Star and Griesemer 1989) has investigated how artifacts can move between different communities of practice allowing teams to collaborate across technical differences in practice and language. Boundary objects are flexible enough to mean different things to different sets of people, yet robust enough to be identifiable to everyone who uses them. User stories appear to be similar to boundary objects in that their meanings change depending on the context and that some amount of standardization is needed for user stories to be recognizable to different team members. However, we also found that user stories were being used to define and negotiate boundaries on a daily basis. Rather than moving easily across boundaries, they constituted the boundary. Additionally, while their meaning changes depending on context, the user stories help to accomplish context as something shared with the entire team, breaking down technical differences.

Our informants emphasized the variety of perspectives which were encoded into the user stories, how these perspectives provided context, translated the activities of others, and prompted conversation. User stories call attention to the tension between different technical roles and their intentions for the software system, rather than alleviating these tensions. User stories may be an example of "conscription devices" (Lee 2007, citing Henderson 1999), artifacts that "enlist group participation" and are "adjusted through group interaction." However, they did more than encourage participation. Lee's notion of "boundary-negotiating artifacts" is perhaps the most fitting category for the user story. The boundary-negotiating artifacts are artifacts that test and establish boundaries, practices, and standards. Unlike boundary objects they cross boundaries roughly, if at all, and are sometimes used to negotiate the boundaries themselves. Artifacts such as Sam's sign-offs and Carol's wiki may be examples of "self-explanation artifacts," a particular type of boundary-negotiating artifact, that are created and used privately, but are sometimes indirectly presented to others (Lee 2007).

Bertelsen has conceptualized artifacts as part of project to develop a design epistemology. Design artifacts, like boundary objects, "mediate across heterogeneity" (Bertelsen 2000). This broadens the perspective from artifacts which move

across boundaries to those that exist in the heterogeneous spaces between people with different technical roles, skills, or experiences. Bertelsen suggests that design artifacts move between construction and representation. Our findings align with his notes towards an epistemology showing that models and enactments, like construction and representation, are two modes which exist in tension and which together form a generative system. User stories reveal this tension between "the possible and the existing" in that they support both models of the software system and enactments of software work. However we have focused on the ways that user stories mediate the Software Process itself. The fact that the Software Process and the software system became conflated at our field sites suggests that artifacts can support not only a confrontation between the possible and the existing, but also a blurring between the two (an assemblage of the possible and the existing).

Our project contributes to the understanding of this space where the model meets enactment, what Bertelsen has called a heterogeneous space, or what Lee has called a boundary space. While their work has aimed to theorize the types of artifacts that exist in this space, we hope to add to the understanding of how boundary work is accomplished. The boundary work of software work takes place at the edges of the Software Process.

7.2. Models vs enactments

Our model of Software Process as a generative system draws heavily on the work of Feldman and Pentland on organizational routines. They provide a model of the organizational routine as a generative system which emerges out of the interplay between the ostensive and performative modes of reflecting on the routine. They define the ostensive as referring to abstract patterns, generalizations, or "theories." These are, in commonsense terms, the ideas we have about the world based on a collecting up of experiences or a narrative or script of typical actions (Pentland and Feldman 2005). They define the performative as referring to "specific actions taken by specific people at specific times" and specific locations (Pentland and Feldman 2005). These actions "are carried out against the background of rules and expectations, but the particular course of action we choose are always, to some extent, novel." This theoretical concept, founded in structuration (Giddens 1979) and practice theory (Bourdieu 1977), draws also upon Latour's definition of the "ostensive" and "performative" modes (Latour 1986).

Others have referred to this distinction between model and enactment as prescriptive and descriptive, method and amethod (Truex et al. 2000). While we have used the distinction between model and enactment, the distinction between ostensive and performative has some advantages. For one thing, models are really only one kind of ostensive reflection. Our study participants' metaphor of the Software Process as a Conversation was an ostensive description, though perhaps

not a model. Bertelsen and Naur reflect on the role that theories and ideas play in design work and software development, which are also examples of ostensive reflections.

The ostensive and performative, because they are adjectives rather than nouns, emphasize that these are modes of looking, not mutually exclusive types. The ostensive and performative are indeed *modes* of looking and reflecting. Like multistability in Gestalt theory, they are difficult to hold simultaneously but are not mutually exclusive. Any activity or interaction can be viewed in either mode, foregrounding and backgrounding certain aspects of the activity. This leads to the tendency for research to focus almost entirely on one or the other, as noted by Truex et al. (2000), and for studies of the ostensive to marginalize the performative as valuable knowledge and vice versa. But in fact it is possible to think about how the two modes interrelate and how moves back and forth are common throughout work activity.

The ostensive/performative distinction has advantages over others such as subjective/objective and structure/agency because it focuses on the collective way in which performances are achieved and on the "ability of both participants and observers" of a particular work activity to generate the ostensive from the performative, or in other words to make both ostensive and performative accounts and move between them (Pentland and Feldman 2005). This responds directly to Dittrich's call for a consideration of how practitioners and researchers can move between an "understanding from within" to a "looking from without" (2002). Our study participants, like us, can move between being a participant and an observer of the Software Process and software system.

Bertelsen also points out that theories too can be design artifacts drawing on the work of Peter Naur. Naur presents an argument for "programming as theory building" in *Computing: A Human Activity* (1992). He argues that software programs are not the central product of software development, nor can the knowledge needed to build software by fully contained in the software, its documentation, or its specification artifacts. He makes a move to reclaim the importance of theory. He claims "the building of the program is the same as the building of the theory of it by and in the team of programmers." Yet he says "in building the theory there can be no particular sequence of actions." We agree, and would add that theories are a kind of model or ostensive aspect of the system and that artifacts can support these theories. Like Bertelsen (2000) and Naur (1992) we are seeking a way to make room for the interplay between theories of the system and its enactments.

We can consider the artifacts and activities of software development through both the ostensive and performative modes. The user story illustrates how an artifact can influence and support both ostensive and performative aspects of software development work. Alexis, Sam, and others told us how they would write up user stories on their own before planning meetings. These artifacts for them were a way of brainstorming and coming up with ideas for the system. The

user stories are thus helping to represent the system as it will be in the future, supporting an ostensive view of the software system. However, during the planning meeting these user stories written out on cards or excel sheets were not directly shared with the rest of the team. Instead, the emphasis was on writing the cards in front of others. In this setting even the gestures of writing become important as enactments. The movements of the card also provide meaning during the planning meeting and afterwards. For example, one software developer might write a user story on an index card in front of the rest of the team, with others chiming in about what should be on the card. He then hands it to someone who will be working on that user story, who writes up a bunch of task cards. The cards are then placed on the magnetic scrum board and moved each time that the "user story" is worked on as it goes from "to do" to "done." We can foreground the performative mode by focusing on these specific interactions. When a software developer takes a card back to his desk while working on programming the user story, it may be that the card is then supporting an ostensive mode of reflecting on the software system.

We can identify moments were artifacts shift between the two modes such as the moment of when a document gets signed off. The writing of the signature foregrounds the performance, but the signature becomes a mark on the artifact which allows it to ostensively represent the system being built. It is the signature that allows the artifact to move from one mode to the other. We saw this in the case of Sam's user story artifacts where he tracked user stories and had a place for sign off. The fact that he never used these sign-offs was an indication that their role of respresenting an ostensive view of the system never reached full closure. Because Agile artifacts lack sign-offs, Easy Retirement also had difficulty with auditing companies. Legal and regulatory compliance are important to their work since they are providing financial service software. The auditors had a difficult time dealing with their process because there were no documents with sign-offs. "They were very upset that we didn't have any signatures confirming what we were going to make in the next sprint." As a result, Easy Retirement came up with a new document called a Software Release Document, which had the release date on which it was tested, a build number, and a signature.

The Conversation also reveals this shifting between the two modes. Our participants switched back and forth frequently between describing The Conversation and describing conversations. The Conversation is an ostensive conceptualization of the Software Process. It generalizes many enactments and activities into a coherent whole. While it is clear that what takes place inside of the Software Process is many conversations, which not all members are always privy to, the view that these together form one single Conversation, is an ostensive move.

The software code itself shifts between these two modes. The code is an artifact which simultaneously supports both modes of viewing the software system. The code artifact helps the development team to think about the intended

system, or to build up a pattern of actions which then influence the user stories. In this capacity it supports an ostensive view of the software system, what the team intends it to be. But the code artifact is also a performance of a collection of actions. The code, in combination with automated tests running "green," in the "build," and in use, supports a series of enactments.

We also found that our informants moved between these two modes in such a way that helped to negotiate the boundary of the Software Process. They tended to foreground the ostensive mode when describing activities and artifacts outside of the Software Process and to foreground the performative when describing activities and artifacts inside the Software Process. They marginalized artifacts that were outside of the Software Process and that helped them to build their own ostensive view of the system. They also marginalized the artifacts used in non-Agile processes by foregrounding their ostensive aspects.

We found that our study participants move back and forth between the ostensive and performative mode in their descriptions of the Software Process and their use of artifacts. These moves between the two modes accomplish several things. First, they support a contrast between the companies' current Software Process and other Software Processes. Second, a privileging of the performative mode and marginalizing of the ostensive mode helps to highlight their Software Process as superior to other processes. Thirdly, the privileging of the performative mode and marginalizing of the ostensive mode helps to stabilize a boundary between what counts as inside of and outside of the Software Process as Conversation. Artifacts which are reflected upon in the ostensive mode are downplayed and considered outside of the Conversation. Those artifacts which enter the Conversation are used to emphasize the performative aspects of the work.

Based on our observations we also found that the user stories as artifacts support this privileging of the performative mode because they represent very little of the ostensive aspects of the software product (only short ostensive narrative fragments). The User Stories support the negotiation of the boundary of the Conversation process through this performative mode which helps to construct a shared collective ostensive vision of the software system. The performative mode foregrounds the ways in which User Stories are written in front of the whole team, handed off among members, and moved and circulated through the implementation process. User Stories are able to cross the boundary between what is outside of and inside of the Process Conversation and thus negotiate its boundary.

Inside of the Software Process we found that there was a blurring of the Software Process and the software system. Additionally, our informants blended together the user story artifacts and activities in ways that made them seem indivisible. The artifacts that support activities inside of the Software Process are the user story cards, test cases, and code. These artifacts were woven together with each other and with conversations in such a way that it became very difficult

for our study participants to tease them apart. Rather than describe the Software Process ostensively, they frequently made recourse to specific instances where the artifacts and conversations came into interaction.

Our study participants negotiate this boundary by moving from the ostensive the performative modes of reflection on the Software Process and the software system. They foreground the ostensive for that which is outside of the Software Process and the performative for what is inside. They constitute the boundary through the interplay between the two modes. As they switch from viewing an artifact in one way to viewing it in the other, they maintain and negotiate the boundary. User stories move across this boundary, but at the same time must be written from scratch in order to enter the Software Process. Code exists both as output and input to each iteration and can be viewed simultaneously from both modes as either something which is read and documented to support an ostensive model of the software system or as something which is woven into the user story as part of conversations.

7.3. The negotiated space

We found that our study participants negotiate the boundary of the Software Process in part by switching from the ostensive mode to the performative mode in their reflection on artifacts (See Figure 7). While artifacts such as the user story cards or wikis, when outside of the process are considered for their ability to represent ostensive aspects of the software system, once they are inside the Software Process they are valued for their ability to represent enactments (See Figure 8). This helps to negotiate what is inside of the Software Process and what is without. What is being negotiated is the boundary of the Software Process, the Software Process itself, and the software system.

While the boundary object is a "pioneering concept" that has been "a useful placeholder for explaining that artifacts 'live' in the space between collaborating communities of practice," it is limited to providing different communities of practice a "means of translation" between them since the boundary object can "inhabit multiple worlds simultaneously." This concept is limited, however, in that it does not help to understand sites where the boundary between communities of practice are still being negotiated or, in our study, where a collective boundary of what is the Software Process are under constant negotiation.

The software system itself is under constant negotiation. As Naur points out, the nature of developing software systems involves a constant negotiation between problems in the world and problems in the computer. Lee also suggests that "perhaps boundary negotiating is part of a process by which methods are developed." In developing software systems, methods are not only a means to write computer code, code is itself a set of methods. It makes sense then, that artifacts would be found to support constant negotiation rather than stabilization.

Bertelsen describes the activities of design as heterogeneous. In our study we found that the Software Process is heterogeneous in that those participating in it bring different perspectives and sets of technical expertise. It is also heterogeneous in that what exists inside of the Software Process is a mixture of the possible and the existing. Rather than drawing a clear line between designs for the software system yet-to-be-built and that which is already built, inside of the Software Process these two modes blend together. User stories are crafted together with code. The user story itself is heterogeneous and is the basic unit of work and of the software system.

Naur describes the relationship between the programmer and the software as that which keeps it "live" through the programmer's theories about the code. It is the ostensive aspects which keep software's performance live. We found that user stories provide a way for not only programmers but the entire team to keep the software system live. The Software Process as a generative system allows a group of people to collaboratively keep the software system "live."

Our interest has been to understand boundary-making and pushing rather than how standards and collaboration can dissolve boundaries. In our analysis we have claimed that Software Process is generated by the interplay between models and enactments, and depends on the choices of participants to decide what counts as Software Process and what does not. Artifacts are a part of these choices and of the negotiation of the boundary of the Software Process. As Lee states "artifacts can serve to establish and destabilize protocols themselves... artifacts can be used to push boundaries rather than merely sailing across them" (Lee 2007).

Our analysis of the Software Process helps to fill the gap between research on Software Process models and enactments. We hope that this conceptualization of the Software Process as a generative system can help to reveal the interplay between research on models and enactments. It may also help to consider our roles as software researchers and what it means for us to study and intervene in the making of Software Process. Rather than viewing research as an effort to model or describe, it can be understood as part of the process of boundary negotiating. We can aim to ask what does it mean to make and use methods? (Truex et al. 2000; Dittrich 2002).

7.4. Implications for practice

While the study of enactments has focused on where the abstract models meet the real world in terms of their gaps, omissions, and inadequacies, this research points to the ways that negotiating the Software Process itself can be an entry point to participation and collaboration. Our study suggests that both research and practice can be informed by the generative interplay between what have tended to be two distinct modes of understanding Software Process. It suggests that the two camps of academic software research, the prescriptive and descriptive, or method and amethod (Truex et al. 2000), might have a space for dialog. It also means that

practitioners might reflect on their own practice in a different way. Software process is something which developers can engage in to varying degrees and these choices can be seen as agentic moves which help to constitute the software process rather than impede it. Rather than being apologetic about duplicative artifacts or nonstandard work practice (as our informants were at times), our results suggest that elimination of duplication is not necessarily desirable, nor is bringing all activities into the fold of Process.

Relevant to both research and practice, we wish to add to the discussion of what it means to make and use methods. Methods are something that both researchers and practitioners reflect upon and perform, thus placing us on more equal footing. At Dittrich has suggested, this might mean entering into collaborative method development. Or rather it might mean that there are roles to be played by both researchers and practitioners to broaden the boundary of process as a means for increased participation. Rather than seeing code as at the center of the software development process, we can see conversations at the center. Rather than finding *ways in to* code, we can find *ways out for* code. We can ask, how does code move out into the stories and conversations, rather than asking how we get the conversations about use integrated into code.

8. Conclusion: the boundary work of software work

In this paper we have posed the question of What counts as Software Process? We ask this question to draw attention to the ways in which both researchers and practitioners actively make choices about what counts as Software Process. The question of what counts as Software Process then leads us to consider how these choices are made and why. If we as researchers can move from the "understanding from within" to the "looking from without," and if practitioners can do the same (which indeed they can), then what is happening as we make this shift, and what does it accomplish? Given that Software Process is not a prescription for practice, what is accomplished by invoking and describing an ostensive Software Process and what relationship does this have to particular performances of Software Process?

We found that what counts as Software Process, for our informants, is a collection of software work which is bounded. The Software Process provides a generative system through which the software team scopes and defines its work, not as the outcome of Software Process but through the negotiation of the Software Process itself. Software Process is not a prescriptive model that is periodically accessed to make work accountable, nor does it encompass all of the work of the software development team. Rather, the Software Process comprises many models and ostensive reflections on the Software Process, ideas and theories about what their process is. And the Software Process is also a bounded set of practices that comprise the software work, made up of conversations and artifacts. Software Process is not only models that prescribe software methods or

an analytical category for the sake of description, but a bit of both, or the gray space between.

The gray space is defined by the way that these two modes of thinking about Software Process form a generative system. The Software Process is constituted by the interplay between these two modes, because models constrain and enable enactments, and enactments create and recreate models. As software developers switch between the ostensive and performative modes of reflecting on their work and on the artifacts that they use, these switches accomplish much of the work of maintaining and negotiating the boundary of the Software Process.

We have proposed a different picture of the Software Process. Rather than just model or enactment, it is a generative system that involves both models and enactments. Software process is generated through the interplay between the ostensive and performative. Software developers move between the two modes, and the choice to highlight the ostensive or performative is not made once-and-for-all, but is made again and again. It is a generative process, but not one that is unbounded as found in the research on Software Process enactments. There are agentic decisions to choose what is included in the Software Process and what is excluded, which means this boundary is under constant negotiation.

This negotiation takes place through the conversations and artifacts of the Software Process. Conversations and artifacts exist inside of the Software Process, outside of the Software Process, and constitute the boundary of the Software Process itself. The ostensive and performative modes helped to define these different sets of artifacts and conversations. Artifacts outside of the process tend to be viewed in the ostensive mode, for their capabilities in representing the software system, these artifacts are used as a kind of collateral to negotiate and push the boundary, but do not in fact cross the boundary. Artifacts such as user stories seem to cross the boundary through a switch between the ostensive and performative modes. Inside of the Software Process the performative mode is privileged such that artifacts are meaningless without the links that tie them to conversations.

Code, tests, and user story cards, come together to form an indivisible unit of work that is a mixture of the possible and the existing, a combination of the system-being-built and the system-in-use. Code, rather than being the center towards which all work is oriented, moves out into the Software Process through these conversations and artifacts. Rather than positioning design artifacts or code as the central means by which access is gained to software development, the boundaries are where much of the participation takes place as team members decide individually and collectively what counts as Software Process. We have sought to understand not how ideas get *into* code but how code *moves out* into the Software Process.

By asking what counts as the Software Process we have opened up the possibility of considering the boundary work of Software Process as the locus for knowledge work in software development. It is at the boundaries where methods are made, negotiated, and brought into the software system. The Software Process

is crafted *together with* code instead of *for* code. Software code and the Software Process are crafted together through the interplay of artifacts and conversations, models and enactments, at the boundary of the Software Process.

Acknowledgements

We are indebted to our field sites and the participants who consented to be observed and interviewed. Many, many thanks to Rosalva Gallardo-Valencia who was instrumental in the data collection and transcription. Thanks also to Anahita Fazl who helped with the transcription. Yvonne Dittrich provided, reviewed and guided our research direction. This research has been funded in part by the National Science Foundation (Award IIS-0712994) and the Agile Alliance Academic Research Program.

Note

1. We found that many of our study participants distinguished Agile from other processes based on the role of documentation. They compared and contrasted different Software Processes in terms of documentation. For example, Brett, a Product Owner at Fast Tools gave a rundown of various processes: "Some software teams work collaboratively and iteratively but produce tons of documentation" doing "ten iterations of documentation with no software." RUP (Rational Unified Process) is "lighter weight than Waterfall, you can redefine it, but it is still focused on... process and tools and documents." In RUP, "you can choose between ten different suites of documents and tailor them to your specification." And in Extreme Programming, "the XP person will say I don't care [about documentation] it is whatever you want," so there is "no guidance about documents." Agile actually has a lot of guidance about documents in terms of what a user story should look like and how it should be used. But user stories tend not to be considered documentation in the same way.
2. The legacy code was called Whitney for Whitney Houston to denote that it was "all washed up" and the new code was called Ghidrah and was represented by a three-headed dragon figurine who "eats the legacy code" a reversal of saving the damsel in distress from the dragon.

References

Bansler, J. P., & Bødker, K. (1993). A reappraisal of structured analysis: design in an organizational context. *ACM Transactions on Information Systems, 11*(2), 165–193.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley Professional.

Bertelsen, O. (2000). Design artefacts: towards a design-oriented epistemology. Scandinavian Journal of Information Systems, 12, 15–27.

Boehm, B. (1981). *Software engineering economics*. Englewood Cliffs: Prentice-Hall.

Bourdieu, P. (1977). *Outline of a theory of practice*. Cambridge: Cambridge University Press.

Button, G., & Sharrock, W. (1992). *The mundane work of writing and reading computer programs*. Cambridge: Rank Xerox, EuroPARC.

Button, G., & Sharrock, W. (1994). Occasioned practices in the work of software engineers. In M. Jirotka & J. Goguen (Eds.), *Requirements engineering: Social and technical issues*. London: Academic Press Professional.

Cohn, M. (2004). *User stories applied: For agile software development*. Boston: Addison-Wesley Professional.

Dittrich, Y. (2002). Doing empirical research on software development: Finding a path between understanding, intervention, and method development. In Y. Dittrich, C. Floyd, & R. Klischewski (Eds.), *Social thinking-software practice*. Cambridge: MIT.

Feiler, P. H., & Humphrey, W. S. (1993). Software process development and enactment: Concepts and definitions. In *Second International Conference on Continuous Software Process Improvement* (pp. 28–40).

Fuggetta, A. (2000). Software process: A roadmap. In *Proceedings of the conference on The Future of Software Engineering* (pp. 25–34).

Giddens, A. (1979). Chapter 2: Agency, structure. In *Central problems in social theory: action, structure, and contradiction in social analysis*. University of California Press.

Larman, C., & Basili, V. R. (2003). Iterative and incremental software development. *IEEE Software, 36*(6), 47–56.

Latour, B. (1986). The powers of association. In *Power, action and belief: A new sociology of knowledge* (vol. 32, pp. 264–280).

Lee, C. P. (2007). Boundary negotiating artifacts: unbinding the Routine of boundary objects and embracing chaos in collaborative work. *Journal of Computer Supported Cooperative Work, 18* (3), 307–339.

Lofland, J., Snow, D. A., Anderson, L., & Lofland, L. H. (2005). *Analyzing social settings: A guide to qualitative observation and analysis* (4th ed.). Belmont: Wadsworth.

Naur, P. (1992). *Computing: A human activity*. New York: ACM.

Nørbjerg, J., & Kraft, P. (2002). Software practice is social practice. In Y. Dittrich, C. Floyd & R. Klischewski (Eds.), *Social thinking—software practice*. Cambridge: MIT.

Osterweil, L. (1987). Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering* (pp. 2–13).

Parnas, D. L., & Clements, P. C. (1986). A rational design process: how and why to fake it. *IEEE Transactions on Software Engineering, 12*(2), 251–257.

Pentland, B. T., & Feldman, M. S. (2005). Organizational routines as a unit of analysis. *Industrial and Corporate Change, 14*, 793–815.

Robinson, H., & Sharp, H. (2003). *XP culture: Why the twelve practices both are and are not the most significant thing*. Salt Lake City: Agile Development Conference.

Scacchi, W. (2001). Process models in software engineering. In J. Marciniak (Ed.), *Encyclopedia of software engineering* (2nd ed.). New York: Wiley.

Schwaber, K., & Beedle, M. (2001). *Agile software development with SCRUM*. Englewood Cliffs: Prentice Hall.

Sim, S. E., Alspaugh, T. A., & Al-Ani, B. (2008). Marginal notes on amethodical requirements engineering: What experts learned from experience. In *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference* (pp. 105–114).

Star, S. L., & Griesemer, J. R. (1989). Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's Museum of Vertebrate Zoology 1907–39. *Social Studies of Science, 19*, 387–420.

Truex, D., Baskerville, R., & Travis, J. (2000). Amethodical systems development: the deferred meaning of systems development methods. *Accounting, Management and Information Technologies, 10*(1), 53–79.

van Vliet, H. (2008). *Software engineering: Principles and practice* (3rd ed.). New York: Wiley.