Design Methods as Discourse on Practice

Marisa Leavitt Cohn, Susan Elliott Sim, Paul Dourish
Department of Informatics, University of California, Irvine
Irvine, CA 92697-3440
{mlcohn, ses, jpd}@ics.uci.edu

ABSTRACT
In this paper, we present a view of design methods as discourse on practice. We consider how the deployment of a particular set of design methods enables and constrains not only practical action but also discursive action within the design practice. A case study of agile software development methods illustrates the ways that methods establish conditions for who can speak in the design process and how. We indentify three main kinds of discourse work performed in the invoking of design methods. These are the establishing of ontologies, the authorizing of voices, and the legitimizing of practices. We then discuss implications of this view on methods for CSCW research on the relationship between methods and practice as well as implications for participation in the design process.

Categories and Subject Descriptors
K.4.3 [**Computer and Society**]: Organizational Impacts – *Computer-supported collaborative work*

General Terms
Management, Measurement, Documentation, Performance, Design, Human Factors, Standardization, Theory, Verification.

Keywords
Methods, Discourse, Authority, Voice, Design, Participation.

1. INTRODUCTION
What do methods do? This has long been a question in CSCW in considering the relationship between methods and action. Methods can be seen as resources to practical action, as ways of regulating organizational work, or to narrate the performances of human-machine configurations [25]. But methods also tell us about roles, objects, and subjects that exist in the universe of a particular approach to design. Methodological commitments may be secondary to the accomplishment of work, but they also establish a particular discourse in which action unfolds. The question of who has voice in the design process is in part a question of how methods shape the discourse – fixing relationships between designers and stakeholders and shaping who can speak and how.

This paper explores this question of who can speak in the design process and the modes of authority and legitimation that are deployed through methods. We consider first the tradition in CSCW of considering the relationship of methods to practice and then discuss a case study of software development methods as an example of organizational design methods and collaborative design work. Using a lens of discourse we identify the kinds of discursive work that are accomplished by practitioners in their deployment of agile methods and consider implications for both CSCW research and design practice.

GROUP'10, November 7–10, 2010, Sanibel Island, Florida, USA.

2. METHODS IN CSCW
The relationship between methods and practice has long been a concern of CSCW research on systems design [1,2,5,7,14,15,17,24,25,26]. This is in part because systems design is oriented to the methodical and ordered aspects of work that are available to computerization. The paradox of systems design is that it relies upon the articulation of methods for work but those very methods are never a complete or accurate description of work practice. Methods are a way to talk about practice in an idealized form in order to support practices of accounting, due process, or to share generalizations about practice with other audiences. Methods, rather than being accurate descriptions of enacted work, are resources to and outcomes of practical action [14,25]. Even for the scientist or social scientist, methods comprise a particular story about research activities that foreground certain actions and background others. In design, as in science, local conditions and contingencies arise which require deviation from method.

Much of the early work in CSCW on methods focuses on this issue of the gap between methods and practice [2,14,25]. Designers of organizational workflow or information systems encountered representations of the organizational structure of work, organizational routines, and methods for getting work done. These might be found in diagrams, forms, workflows, or narrative texts describing work practice. Suchman in particular critiques the assumption in systems design that methods are stepwise specifications for action which are then realized by actors and that the successful accomplishment of work depends upon the adequacy and completeness of these specifications [25]. Rather "making procedures work" is an accomplishment of practical action. Methods are always an incomplete representation of action and can be resources to action such as when a person consults the standard procedure as they work, or can be used to support accounts for work after-the-fact. While the meaning of "procedure" has a definite technical meaning to systems developers, it has a "softer" meaning for the accomplishment of organizational work. This discrepancy can become problematic if organizational methods are taken as accurate descriptions of work practice, leading to inappropriate system requirements. Once designed, systems can impose a set of methods upon work. Organizational methods, once coded into methods to be carried out by computers-in-use, may be less available to the flexible and situated nature of work.

A second concern regarding the gap between methods and practice is related to issues of voice and authority. Suchman notes that treating methodical accounts of work as a means for the realization of work resembles a Tayloristic approach to managing and controlling work [25]. While she suggests that the assumption that methods determine practice is faulty, she nonetheless points to the ways that methods can be imposed as a way to construct authority over the actions of others as part of a system of regulation and control, however incomplete and imperfect. Gerson and Star raise a similar concern about the use of methodical representations of work as system requirements [14]. They say that such representations are always incomplete in part because they represent particular viewpoints about work while "screening out" others. Viewpoints about work are always multiple in real-world systems. Design methods that codify only expert and explicit organizational knowledge will neglect voices and perspectives such as that of "the computer room managers who are the only ones who can coax the system into functioning, or the crack secretary who 'really runs the place.'" Generating system requirements from articulated organizational methods will leave out work practices such as the work that goes into articulating those methods. They recommend the adoption of methods from the sociology of science that have been successful in describing and analyzing how tacit knowledge is incorporated into scientific facts as a way to capture the multi-voiced effort of organizational work. They also call for design methods that are integrated into the context of ongoing use and maintenance [14].

This article reflects more upon this second concern raised in CSCW research on the relationship between methods and practice. While it is important to consider the ways that gaps between methods and practice can lead to systems that inhibit and constrain practical action, we would like to reinvigorate the question of what it means to represent particular voices in the design process? What consequences do methods have for who can speak in the design process and how?

To answer these questions we draw upon an empirical case study of agile software development practices as an illustrative example of design methods. Agile methods are in many ways the antithesis to the design methods critiqued by Suchman and Gerson and Star [14,25]. Agile as a professional movement seems almost to respond to their call for design methods that integrate design with use and acknowledge the multiple and fragmented viewpoints on systems design. Agile methods construct no notion of a single authority over the design process such as a 'Systems Architect,' nor any singular authoritative document that represents the requirements for the system. A case study of agile methods thus offers a prime opportunity to reexamine questions about the relationship between methods and practice in systems design and how methods have consequences for the representation of voice and construction of authority in the design process.

3. AGILE METHODS

While we will refer throughout this paper to "agile methods", "Agile" is actually a "family" of software process models, methods, and techniques. These include Extreme Programming [3] and Scrum [21]. The distinguishing feature of agile methods is their emphasis on adapting to change and taking an incremental and iterative approach to design and development, rather than a phased and sequential approach to minimize change through careful planning. In agile methods, "design" is not a phase of the development cycle but is considered to be happening throughout development. These kinds of methods originate in the late 1990s as a reaction to the more "rationalized, engineering-based" approaches to software development [11 citing 20]. The software development community expressed frustration with changing expectations leading to wasted coding effort. Extreme Programming in particular was a move towards "programmer-centric" computing, emphasizing collocated programming teams and pair-programming and less documentation writing. In many ways the agile methodology is likened to a social movement. Agile methods are seen to have "divided the software development community into opposing camps of traditionalists and agilists" [20]. One contributor to the Agile Manifesto, written in 2001 by contributors to Extreme Programming, Scrum, Crystal, and other agile methods, narrates agile's history saying, "A bigger gathering of organizational anarchists would be hard to find" [4].

Agile methods coincide with calls within CSCW for iterative design, collocation of designers and stakeholders, integration of design with use implementation and maintenance, and evolving systems over time. Since the agile process is iterative and incremental there are many more occasions in which individuals have the opportunity to voice their perspective on what the system should be designed to do or how it should perform. However, this does not mean that everyone has a voice or that all voices are the same. Rather, it means that it is all the more important to understand the nuances through which methods establish conditions for possible voices to emerge in the design process. The main instrument to discipline who can say what in the design process is not located in the representational artifact of the requirements document and the deliberative process that goes into articulating that document. Instead, there is an ongoing process of articulation and rearticulation of system requirements. Agile thus offers an ideal case study to reconsider CSCW concerns.

Many of the agile methods we will be discussing in this paper are derived from the Scrum software process model. We will be discussing some concepts like the "User Story" which are common to many agile processes, as well as some that are particular to Scrum like the Scrum Master, the daily stand-up meeting, or the Scrum Board. While organizations often pick and choose which methods to use from any model and often mix methods from multiple models, the focus of the paper will be to consider the relationship between the methods chosen by the people in our case study and their practices.

Scrum is a software process that is in the agile "family" and is depicted in Figure 1 below [10,21]. In Scrum, rather than dividing the software project into phases in which certain activities such as design are accomplished, the project is divided into time increments, called "Sprints" or iterations. The duration of the sprint varies from company to company, but is typically between 1 and 4 weeks. In the figure the sprint is represented as 30 days. During the sprint there is a daily "stand-up" meeting at which all team members are present including software developers, heads of teams, and usually a "product owner" who is a representative from a customer company or in many cases an internal proxy for the customer such as a customer service representative. The daily stand-up meeting is depicted by the 24-hour cycle. Several sprints can be grouped together into a release that often coincides with a meaningful time period like a fiscal quarter or a set of features being completed. In many companies the software is being updated for the user at the end of every sprint, in others at the end of a release. Either way, new software updates are being released

on the order of every few weeks to months, rather than years as is common in traditional phased development.

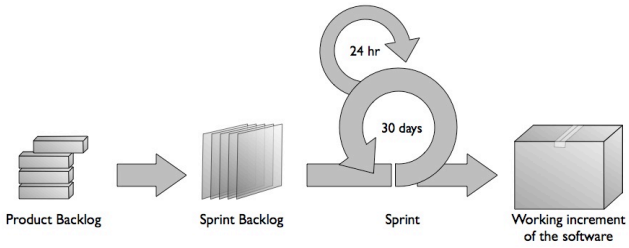

Figure 1: The Scrum Process (based on model [10])

A typical Scrum project begins with a meeting at which many ideas for system features and requirements are brainstormed and archived in a "Product Backlog" in the form of "User Stories." User Stories are like system requirements written in a narrative text form with the format: "As a <role>, I can <action>, so that <goal>." According to [10], a User Story is multi-modal and exists in three parts – the written description of work to be done, conversations about the work, and test cases. The written description of the User Story should be small enough to fit on a 3" x 5" index card or sticky note and must be able to be completed in a single sprint. In principle, the user stories can be written by anyone in the organization but should emphasize the "user's" point of view. This however can mean that the point of view can be that of a programmer, CEO, customer service representative or anyone internal to the company who interacts with the software, so long as it is written from their perspective as a user.

At the start of each sprint the team gathers for a spring planning meeting at which they draw user stories from the product backlog and generate new user stories and decide which should be completed in the given sprint. This is determined in part by priority and in part by the "sprint velocity" which is a number of "User Story points" that a team will complete in a sprint. Every user story is estimated with a number of points for the general "size" of the requirement. This set of user stories becomes the sprint backlog. At the daily stand-up meeting the team meets for about 15 min to give quick updates. The sprint concludes with a review meeting to look back on the practices of the sprint, what worked and what did not.

During the sprint, user story cards are often placed on the Scrum Board for others to see. The board is typically divided into columns such as "not started," "in progress," "tested," and "accepted." The cards can be placed on the board and moved from one column to the next to indicate progress. Teams often have a way to indicate user stories where a bug or issue is impeding progress such as a pink colored card or column. Software developers also often take the user story cards down from the board while they are working on them.

There are five primary roles on a Scrum team: Scrum Master, Product Owner, Team Members, Stakeholders, and Users [21]. The Scrum Master facilitates the software development process by tracking the team progress and helping to remove impediments. The 'Scrum Master' maintains the product backlog, prepares for and convenes the daily stand-up meeting, and tracks the completion of tasks for each sprint in a "burn-down" chart which shows the number of user story points left in the sprint. The 'Product Owner' represents business concerns. This person could be the person paying for the project or a figurative surrogate for customers. The 'Product Owner' should write and prioritize user stories. 'Team members' includes everyone working to build the software including developers, testers database analysts, system administrators, and technical writers. 'Stakeholders' refers to anyone with an interest in the software product, but in practices this tends to be business interests. 'Users' is a subset of stakeholders of the software. Individual users are not usually involved in the development process but are often consulted to gain feedback. The User role might be fulfilled by someone who will use the software or a surrogate.

4. METHODS AS DISCOURSE

CSCW research tends to emphasize the consequences of methods for practical action. When organizational methods are computerized and imposed without an adequate understanding of the organizational practice, there can be a gap that is consequential for the accomplishment of work at a practical level. This imposition can radically alter the kinds of practical action that will be required to make the methods work for local conditions. The fact that work will get done regardless of which method is adopted, however, tends to lead to the tacit implication that methods do not matter much at all since they have little impact on the ultimate outcome of work. Whether an organization adopts iterative or plan-based methods, software will be developed and claims that a particular method guarantees certain product qualities are suspect. Methods provide a way of accounting for work to others and thus deploying any set of methods will do so long as they provide a shared language.

But this shared language, as a kind of discourse on practice, has consequences with regards to voice, authority, and participation in the design process. The methods deployed in a design project have consequences for discursive action as well as practical action. Methods enable and encourage certain kinds of discourse, establishing the conditions for who can speak and how. Methods inhibit or preclude other ways of speaking. Methods establish certain conditions for possible discursive actions – who can speak about what and when. This is not about consequences for the shape of the outcome of work but rather about who can participate and who can say what in the design process.

Design methods are productive of certain roles and ontologies – certain subjects and objects. For example, requirements engineering methods construct the "requirement" as an object of the design process along with associated artifacts that represent requirements. In agile methods, the "requirement" is replaced by the "user story" object. Both the requirement and the user story are available to the practical accomplishment of getting software developed, but each sets up a different set of conditions of what is speak-able and by whom in the design process.

Foucault addresses this quality of methods in his work on governmentality [12]. By governmentality, Foucault refers to methods of government and the ways that they articulate a rationale for governing over subjects and objects – what he calls governmental reason or governmental rationality. He aims to consider governmentality without calling its methods "primary" or "original" and "already given." Instead he considers how methods are a reflection on and rationalization of practice. Governmentality is not simply the articulation of a mode of managing and regulating the state; it calls a particular notion of

the state into being. In his analysis of governmentality, he considers the social construction of the concept of "sovereignty" how governmental method "fixes the definition and respective positions of the governed and the governors… in relation to each other." Likewise design methods call certain concepts, such as the "system" into being and fix relations between the designer and the designed or between system and use. Technological systems, like the state, are "a specific and discontinuous reality" that is expressed through the discourse of method, setting up the conditions for its material realization [12].

This approach to methods aligns with existing CSCW work on methods and practice but draws attention to method as discourse rather than resource or outcome. It focuses on the ways that methods are a discourse on practice. Methods establish the discursive conditions in which practice takes place in the sense that they fix certain relations and concepts that can be taken up in practice. Discourse does not determine practice but it does shape who can speak about what and how. Considering methods as discourse emphasizes the ability of organizational groups to narrate work and systems rather than the ability for organizational groups to accomplish a particular end product.

In this article, when we speak of methods we mean methods as stories about sequences of actions already carried out or to be carried out. They can be used to talk about practice in an idealized form for purposes of accounting, due process, or to share generalizations about practice with other audiences. But they are also idealized forms through which people think about and talk about their own practice. They form a discourse on practice by shaping a set of conditions for possibility in the design process – possibilities for speaking and representing voice. This article is an inquiry into the conditions of possibility established by agile methods.

This definition, and its conflations between stories about actions that are performed at or away from the computer interface, actions that might be individual or collective, and stories that might address organizational structure (e.g. "every day our team gathers for a group meeting to discuss progress") or specific organizational routines (e.g. "I pull up a list of reports on the screen, sorted by date, then print the most recent report"), aligns with the way the term "method" was used by our informants. To our study participants, method is many things and can be used interchangeably with terms like "technique" and "process." Methods thus exist at many levels, telling stories about the behavior of the organization, of specific teams, of individuals, and of the system. The term is even used at the level of software code, specific methods in the software which perform certain steps of action. This is not a meaningless conflation of the term. Rather, much of the work of producing software is translating methods for social and practical action into more technical methods that can be written in code and performed by computers. How these transformations are accomplished collaboratively by the group is what this paper aims to illustrate. The discursive work of methods in this case of software development is relevant for thinking about the role of methods in design practice more broadly.

Gerson and Star note that there is a "proliferation of 'methodologies'" in response to the problematic gap between methods and practice [14]. This is also the view of Truex et al in their deconstruction of software development [26]. They note that "method" is the privileged text for software. Methods proliferate in part because systems design is oriented to method. Methods are accounts and representations of work that highlight what is methodical and screen out what is amethodical. Systems designers are oriented to method and the job of software developers is often to read for and write methods can be translated into code-able methods. In a way, systems design invokes a process of proliferating methods to bridge some gap between current practice and future practice. While other terms like "procedure" or "plan" have been used to talk about methodical accounts of organizational work, we chose the term method because of its privileged role in systems design and software and to draw attention to the, at times productive, conflations that occur between methods at various levels (design, organization, code).

Methods that have been realized computationally have a way of imposing themselves on human actors and can be used as instruments of control. As Suchman suggests, methods can be used to construct authority over a process. Whether that authority is realized has little to do with whether the methods are enacted by people but more to do with the ways that they can be imposed upon and used to discipline people. While agile methods are not as heavy-handed and overt as those which Suchman critiques, methods are still part of the construction of authority in the design process through the ways that they legitimize certain kinds of subjects and objects and ways of speaking about design activities and objects. Methods still play a role in setting up the conditions for possible voices to emerge in the design process. The following sections draw on our case study of agile methods to explore various ways that methods set up these discursive conditions.

5. OUR CASE STUDY AND METHODS

This article draws upon a case study of agile software development methods. Our case study brings together ethnographic fieldwork and interpretative analysis of agile discourse. We have conducted ethnographic research across various sites including agile trainings, workshops, and networking events for agile practitioners in California including official certification trainings offered by the Agile Alliance. We have interviewed a total of 18 agile practitioners in five companies. We also conducted a week-long observational study of agile work practice at two companies near Denver, Colorado in which we observed planning meetings and ongoing work practice and spoke with team members in a range of roles from software programmers and engineers to customer service representatives, managers and CEOs. We have also included in our case study, analysis of prescriptive texts including blogs, textbooks, and articles about agile methods. The two companies who participated in our observational field study include one, FastTools, that develops software tools and training workshops for agile software development and another, Easy Retirement[·], that develops retirement planning software tools for companies and individuals to manage retirement plans and packages. Both of these companies utilize agile methods from the Scrum process model. Many of the examples we draw upon to illustrate our exploration of methods come from these observations of the two companies.

Our observational field study of agile software development methods in practice was initially designed to investigate how agile methods and artifacts support the articulation of system requirements and the implementation of those requirements. In analyzing our data for these research questions about the role of

[·] Names of organizations and individuals have been changed.

documentation and artifacts in the articulation of system requirements, we realized that some of the most compelling stories about our data dropped out. We realized that methods have been considered for the ways that they support group work of coordination, articulation, and negotiation, in systems design. But we had observed methods accomplishing other kinds of work for collaborative group work. This motivated us to analyze agile methods for the discursive work they support as well. In analyzing our data we have employed interpretive methods drawn from narrative and literary analysis. Close readings of methods, artifacts, and stories are being used in the emerging interdisciplinary field of Software Studies, which combines approaches from media studies, comparative literature and sociology of science [13].

In agile software development processes, design takes place in an iterative process. Agile places value on collocation of users and designers and the iterative design of the software system and organizational processes. It is therefore a unique case of software development in which the ways that methods narrate their work overlaps significantly with the ways that they narrate the software system itself. In more highly phased and architected systems, in which large requirements documents are drafted, the narration of what the system is precedes the work to implement the system. Different teams of actors tend to perform the work of creating a story about the system and its behavior and the work of identifying assumptions and translating the requirements in programming languages and tests. By placing much of this work in a coterminous iteration of approximately two weeks, performed by collocated teams, the ways that methods shape the ways that the system and the work to build and use it are narrated is greatly exposed. In the following section we discuss themes within discursive work. We aim to develop these themes in ways that will be relevant to other design contexts as well.

6. DISCURSIVE METHOD WORK

It is easy to identify the ways that "agile" imagined as an overarching methodology is deployed as part of discourse at the level of the software profession. One of the most common narratives we heard in our field study was that of contrasting agile to other methodologies like "waterfall". Many of our informants described what agile was in terms of what it was not, identifying agile in contrast to the values and especially the documenting techniques of other methodologies. Agility often is offered as an alternative counter-narrative to software processes driven by large amounts of documentation. It articulates its origins within a movement to renew the celebration of computer programming as a craft by allowing programmers to "do what they do best" – to write code not documentation. In a way it is a programmer-centered design process that arrives at a broader valuing of users through this first step to empower programmers to focus on programming rather than on specifying requirements. By being thrifty on documentation and design artifacts like architecture diagrams, it reinforces the idea of the process as counter to highly architected and phased software processes. Agile methods are increasingly popular in industry, but this is in part due to those who in the past could not say they had any formal process claiming to adopt Agile to legitimize their process. Some of our informants explained the agile process to us in terms taken straight from the Agile Manifesto's stated values: "individuals and interactions over process, working software over documentation, customer collaboration over negotiation, responding to change over following a plan."

At this level, where methodologies pit against other methodologies, it is easy to see discourse at work and ideologies being formed. However, one of our findings is that much of the same work gets done both before and after adopting agile methods. For the tester, for example, when working in previous "Waterfall" processes she would find a workaround to the phased approach to development. Rather than wait for the requirements to be implemented before starting her work, she would take the requirements document and pore over it for assumptions and going back to the engineers who wrote it to get edits and refinements, doing her best to incorporate a tester's perspective early in the process. As she put it, "other people will wait… but I never worked that way." In agile methods, on the other hand, "test-driven development" is the norm, but the tester had other kinds of workarounds to make up for agile methods emphasis on light documentation. She kept a personal wiki of all user stories and their associated tests.

This confirms the skepticism in CSCW towards methods, which claims that the accomplishment of practical action will proceed regardless of the method being used and that methods do not determine practice. At the same time, however, the tester talked at length about the qualitative ways that her work is now different from what it is like to interact with documents to the kinds of relationships she has with people. In this section we identify and discuss three themes within the discursive work we observed in the deployment of agile methods.

6.1 Establishing Ontologies

Methods enable people to speak about particular objects and subjects as part of design practice. This element of discourse is called ontology. Ontology refers to the "study of the most general kinds that exist in the universe" [16]. A discourse constitutes an ontology of the objects and subjects that can be talked about, that are considered elements in the bounded universe of the discursive community or discipline. In requirements engineering, for example, requirements are a part of this discipline's ontology. Requirements can be talked about in terms of their qualities and how they are elicited or engineered, but are assumed to exist as part of the natural universe of software development. To Foucault ontologies are not only a set of objects that we construct, but also a way of positioning subjects and making ourselves subjects of knowledge production [16]. The "user" is another good example of an ontological subject. Identifying a person as a user is a first step towards gaining knowledge about use.

One of the primary ways that agile methods establish an ontology is through the method of writing "user stories." The "user story" is an ontological object of agile software development that does not exist in other methods. It is often compared to and contrasted with other forms of requirements gathering tools like scenarios [8] or use cases [9]. In our case study of agile methods we have observed several debates about the differences between a user story and a use case. An illustration that these concepts are ontological is that the ideas are incommensurable. There is no satisfactory way to differentiate the use case from the user story across the divide between users of agile and other methods. For example, 'Tom' might ask 'Jerry' what makes a user story different from a use case but every answer Jerry provides leads Tom to say "that sounds like a use case."

One can see the discursive quality of the user story as an agile method in the ways it is used to differentiate agile from other methodologies. The user story was considered by participants to

be the antithesis of the large requirements document found in "Waterfall" methods. The user story is described in prescriptive texts as multi-modal in that it exists in conversations, written documents, and software code [21]. While the User Story exists in these multiple places it is often written on an index card in the format: As a <person in a role>, I can <carry out an action>, so that <rationale or motivation for action. > An example of a User Story might be: As a customer service representative, I can pull up a list of customers by last name, so that I can locate duplicates. It is a fragmentary narrative text.

In our observations and interviews with agile practitioners, we found that the user story indeed carried multiple meanings and could be located in multiple activities, places, and artifacts. For our informants, the user story connotes multiple ideas at once. The user story is both a story about a particular set of actions taking place in the company and a way to refer to those activities directly. Practitioners refer to the user story at times as something taking place inside of the organization, as something that people are playing out, much the way that actors might be enacting a story from the inside. A person might say that the user story is taking place, and that certain individuals are a part of it. At other times the user story is used to refer to an account of activities, a story told in a particular time and place such as at a group meeting.

The agile method of "user story" enables people to talk about their work and the software system in ways that elide distinctions between system requirements, ongoing work, and the software code itself. The user story is able to be many things at once. It is a story about the performance of the software system and about work that is taking place with the software in the workplace. At many times we heard the story talked about in the same way you might talk about a feature – as something that is performed by the technical system. At other times the story was something happening in the workspace, being played out by specific actors supported by computers. People are *in* the stories, part of the stories, in tune with the stories. A story is said to be currently taking place, perhaps because of some new priority or change in the organization.

These elisions between a story as an account of practice and as the practice itself aligns with other findings about the use of narrative in organizational settings. Boje [6] suggests that at times the idea of a story refers to a set of utterances (such as in this case the story which is told at a meeting and written down on an index card) or as the "real story behind the story," a specific ostensive set of actions that took place. By establishing an ontology in which stories are an object of the design process, this elision between actual and desired system performance becomes possible. One would never say that a requirement is taking place in the organization in addition to being an articulation of possible action.

The user story also establishes the ontological category of "user" as part of the discourse on practice. In the groups we observed, the end-user of the system was often represented by a proxy in the company such as a user experience designer who conducted interviews with users. However, the user story still establishes certain conditions for the kinds of relationships that it can present. Often the user story inscribes the perspective of someone in the company such as a customer service representative, but it does so by casting this person as a user of the system. This at times makes it difficult to prioritize requirements that are not user-facing but have to do with optimizing the performance of the system or enabling better bug handling.

The user story exists in excess of any particular artifact that may be considered part of it. The user story was at times referred to as an ongoing conversation within the team, or as a set of actions that take place with the software system in a particular time and place. A user story might be said to exist in the sense that the work it narrates exists within the company and the software, even though it is somewhat aspirational since the story articulates desired performances with the system. In this way the user story is part of everyday practice but is also within the realm of design, sitting at some distance to current practice.

The user story as a kind of sociotechnical artifact carries with it certain ways of voicing what the software will do. It points to the way this story exists ostensibly within the space and cannot be fully written down. The user story is a confusing object when compared to the traditional "requirement." It confounded one of our collaborators, a software engineer, who kept asking, "where are the requirements?!" The user story differs ontologically from the requirements document in several ways. The fact that it is a story emphasizes its telling rather than just the requirement as a text that is written down. It is also understood in a colloquial way as a story that the organization and team members are living through, with people existing inside of the story. And it comes along with metaphors that do not apply to the requirement, such as being in touch with or in tune with a story, or being part of one story more than another. The method of writing user stories does not authorize the requirements engineer in the same way as the requirements document to write the requirements document as the definitive representation of the user's needs and values. Instead, the user story is a unit of the sociotechnical assemblage in which the engineer as well as the user is implicated. However, it is still important to address the question of who gets to have an authoritative voice in an agile organization about what gets coded and implemented in the software.

6.2 Authorizing Voice

Methods authorize different people to speak in different ways in the design process. Methods condition the possible ways to voice values for the design process. For example, methods shape whether customer service representatives, requirements engineers, ethnographers, or designers are authorized to articulate system requirements, features, or priorities. This is in part a question of who can speak for or serve as surrogates for whom (e.g. who can speak on behalf of users or customers) and also who can speak for what (e.g. who can speak authoritatively about particular artifacts or lines of code) and in what ways.

Authority suggests that it isn't only a matter of who can speak in the design process, but the power that certain kinds of voices or utterances have in practice. While anyone can speak the utterances "I now pronounce you married" only certain officiating authorities can speak these in a way that performs the enactment of marriage [18]. Technical writers are able for example to write about the system but are not authorized in most design methods to voice design changes to the system. The work of the technical writer is considered after-the-fact of the knowledge work that goes into producing software even if that is not the case.

The method of writing user stories from a particular point of view can authorize people in those roles to speak about a user story. If a user story says, "As a human resources representative…" then it might be possible for the developer working on coding that story to speak to the human resources representative directly. As one informant put it "I read that and I know it is 'Sally', and this is

what 'Sally' does every day." This enables 'Sally' to have a voice in the software process and to have authority over a particular story. It also enables a supervisor to tell a developer "don't ask me, go ask 'Sally', she sits 10 feet away" whereas in prior to adopting agile methods this kind of feedback loop happened but was not authorized. This is not to say that methods can preclude people from speaking to each other, but having authority to speak can qualitatively change the experience and can have consequences for the visibility, accountability, and responsibility of individuals.

While in some cases the user story inscribes the perspective of collocated team members in other cases the perspectives invoked in the user stories are fictional personas or proxies for real people. Many users of the software are not in-house and cannot be consulted. Still, agile methods enable particular roles like that of the "product owner" to stand in for the user. The product owner is authorized to speak on behalf of the user. While a focal actor in a user story might be in-house and able to re-narrate the story if the details were unclear or have changed, the owner of the story is authorized to see that the story is achieved. The product owner decides ultimately which stories are worth implementing and the acceptance criteria for whether a story is complete. The user story owner is the developer who may not be writing all of the code for that story but who is authorized to say, for example, when that story is done. The metaphor of ownership is present in many agile methods from the use of the role of product owner to provide a singular voice for the product or the calling out of a particular developer as the owner of a user story. Stories inscribe perspectives in such a way that they authorize a person to re-narrate parts of that story. But the owner of a story is authorized to say when a story is "done".

The Scrum Master and the Scrum task board are part of Scrum methods in which the contrast between discursive and practical action can be illustrated. The Scrum Master role is most similar to a technical lead or managerial role in other methods. However, this role is given a new title in Scrum in part because of what it can do discursively. The Scrum Master is so called because s/he has mastered the Scrum process and is authorized to ensure that the team is adhering to the process. A distinction is made between the Scrum Master and a technical lead in that the Scrum Master does not own the requirements or own the outcome of work, but instead "owns the process". Leeann, the Scrum Master at Easy Retirement, had many of the same responsibilities in her previous role as office manager. She had always been the person covering the distance between the business and engineering sides of the company. She ensured that there was communication for example to get clarifications from the CEO about a particular story, or to justify a different approach that the developers wanted to take to a solution. Her role as Scrum Master, and being in charge of the scrum task board, authorize her to do the same work in new ways. As part of her Scrum Master role she summons the team to the task board for the daily stand-up meeting and makes sure that the methods for the meeting are adhered to. She ensures that only one person speaks at a time and that debates are kept to a minimum.

The Scrum task board also conditions the realm of possible ways to speak at group meetings. At stand-up meetings, Leeann is at the front of the room so that she can move user story cards around on the task board and track progress on the "burn-down chart" – a hand-plotted graph of work left to complete by the end of the sprint. While anyone can write on the cards, the task board sets up conditions within the space such that it is easiest for the team to gather facing the board at a distance to see it. The task board provides a kind of visualization of the Scrum process with all the user stories as elements within it. As the owner of the process, the Scrum Master is in charge of this visualization and this places her somewhat outside of the team as an observer of the process.

Authorization can also be seen in the ways that certain material artifacts are designated as authoritative. When a manager signs off on a document or a contract, the document is seen to change to a more official and authoritative state. In the traditional phased methods of building comprehensive requirements document, this document is often signed off at each stage before being handed off to the next team. The documents serve multiple purposes; they provide requirements to the development team for implementation purposes, and they can be used after implementation to validate the software. The requirements document is enabled to serve as a proxy for the multiple voices that articulated the requirements in part through the document's signatories. Agile does away with documents with many signatures on them, but at the same time creates new ways to authorize work. The user story was described at times as a mini-contract for work. Writing narrative fragments on an index card in front of other people, taking that card in front of other people, were ways that authors of stories were inscribed and also ways that certain people were authorized to transform those stories into code.

6.3 Legitimizing Practices

Methods can legitimize certain practices and artifacts over others. This means that while we may see the same sets of artifacts being used in different design practices, which of these artifacts are highlighted, emphasized, and legitimized is part of the discourse on practice. Legitimation is often about which practices are considered to be inside of the design process versus outside, or what kinds of practice are recognized as in accord with method versus amethodical. When a practice is done through a workaround or out of order it can still be integral to the accomplishment of work, but can be deemed amethodical and in some cases can be seen as a threat to stability in the organization.

Carol, a tester, spoke to us about how she used to deal with large requirements documents. In phased development methods, testing happens after implementation, but rather than waiting she would get access to the document early and go through it making as many assumptions as she could, sending it back with questions, and helping to finalize it before it was passed along to the developers. In a sense she managed to insert a testing perspective early into a non-test-driven process. In the current organization, testing is a legitimate part of ongoing development practice. Some of the developers do not consistently adhere to test-driven development and Carol can reprimand them to write tests. Developers often write tests just after or along side of their programming tasks, and this is part of how they practically accomplish their work, but it also gets pointed out at the planning meeting as the less legitimate practice.

Agile methods also legitimize certain artifacts in new ways. The role of tests is one such artifact that takes on a new role as part of "design." Testing shifts from being outside of design to being inside of design. And tests themselves are legitimated as part of the expression of requirements rather than their validation. Code too is an artifact that is now legitimate for different purposes. In agile, code is considered a form of documentation. Code documents design decisions, it documents a user story (so in a

way it is a documentation of requirements). This is a discursive shift from seeing code as the product of requirements and the requirements document as a proper way to validate the code's performance. We heard from informants repeatedly that the code is its own documentation and is the best documentation because it is always up-to-date. The tests were legitimated not only by the tester but also by developers, even those who were somewhat neglectful of writing tests all the time, saying for example that the tests "are what the system should be doing".

This legitimization of code and tests as integral to requirements documentation and design is a dramatic shift from the viewpoint that code is product and tests are product validation. Informants would tell us "there is no documentation other than the tests and the code itself." At the same time, however, we also observed the use of other documentary artifacts to keep track of requirements and tests in an archival and comprehensive way. These documents included a wiki with all user stories and associated tests kept by Carol, the tester, and an excel sheet with all user stories and who is working on them kept by the product owner, Sam, who was formerly a compliance officer. However, there was also significant evidence that these artifacts were not viewed as legitimate. Carol and Sam were not forthcoming at first about these documents, and then were reluctant to share them with us. Eventually they shared them but also expressed some regret that these documents were redundant with other efforts or admitted that they were vestigial artifacts from prior methods. Sam even laughed explaining to us that his template for tracking each user story had a signature line at the bottom since he used to require signatures on requirements documents. He had left the signature line there but never used it since the team switched to agile.

Sam showed us a document of new story ideas he put together with Carol before every sprint planning meeting. He brings this document to the meeting to aid his memory but both he and Carol, when sharing these user stories, write them anew on an index card in front of the rest of the team. The documents he kept support Sam in the practical accomplishment of remembering user story ideas, tracking responsibilities, taking notes during discussions so that he can keep people accountable to decisions. In his role as a compliance officer this is important because he is the intermediary between the company and auditors. Carol eventually showed us her wiki where she kept all the test cases written for each user story. We asked her if others used the wiki or contributed to it and she said that she didn't know, but didn't think so. She said that she would often respond to the developers' questions about a user story by emailing them a link to a page in the wiki. She too expressed concern that her wiki wasn't standardized within the organization and that people might be doing duplicate work. This is not to say that her wiki was not very supportive of the accomplishment of hers and others' work. Clearly the wiki served practical purposes. What is interesting here is the way that such a wiki is considered illegitimate in the face of agile methods.

The user stories on the other hand were not only shared publicly during meetings, but are written and taken in front of the whole team. They are also posted on the Scrum task board where everyone can interact with them and view their movement across t he board. The agile method of writing and thinking about requirements in terms of stories favors a collective and oral mode of collaborative work. User story cards were talked about as lively and in movement and opposed to static and frozen documentation. Yet the work-arounds of Sam and Carol for keeping track of user stories and tests are reciprocal to the kinds of work-arounds they contrived in "Waterfall" methods. Yet what is significantly different is which of these artifacts and practices were kept private or made more public, and what agile methods enable for discursive as well as practical action.

6.4 Translating Materials and Ends

A large part of the work of designing systems is the translating of ends into material form. By translation, we mean not only the practical accomplishment of transforming the state of materials such as found in the building of hardware, wiring of circuits, or programming of software interfaces, but also the discursive connections that must be made during design in order for it to proceed. Values that are identified for design are often located in social and cultural discourse and must be translated into technical discourse in order for them to become viable for implementation and compelling to designers. This can take place through various discursive modes such as correspondence and substitution.

For example part of the process of translating values into design through requirements engineering is the construction of a model about the world of use that can be used to establish correspondence between models at the level of the system design and the external world [1,24]. This correspondence is established and is then tethered heuristically to the design process. In some design methods a model of correspondence is adopted from the scientific discourse such as when user studies are conducted to provide a science of social or ergonomic action. On other design methods correspondence is understood more as a social contract for purposes of validation. The written requirements document an event in which certain utterances of social ends are articulated so that these can be adhered to in the design process and then used to validate the outcome. It is clear that in some design methods this kind of discursive correspondence is not constructed at all, such as in critical design work (see for example [22]).

Other kinds of translations occur in order to allow for the substitution of one object for another or enabling certain subjects to speak on behalf of others or particular objects. One can think of this as the task of the translator who can, for example, speak on behalf of the use context to the audience of engineers, or who can speak on behalf of the system and code in order to translate technical ends to stakeholders. Translation is always a two way street in that to translate from the social to the technical posits translatability between the social and technical. Translation work is not just a matter of the technologist or designer getting values into design; it is also a process of translating technical values into the social world [15].

Methods set the conditions for this kind of translation work by establishing ontologies, authorizing voices, and legitimizing certain practices. One of the ways we can see this happening is in the way that the user story establishes an ontology of the story in which notions of inside and outside are elided. One of the ways that requirements engineering methods enable translation is through the clear demarcation of what is outside of the system design and performance for the sake of correspondence [27]. The requirements document, once written and vetted, can be used as a substitute for the ends that it articulates. With the user story, no such substitution is made. Rather, the user story acts as a kind of link between people performing activities of use and design, maintaining that link until it is no longer needed. The user story also dissolves distinctions of correspondence because it elides demarcations of inside and outside. There are times that the user

story is identified as something existing in the natural world outside of the system, but it can quickly be elided into a notion of a story as a performance-with-the-system. Agile methods do not provide a clear ontological object that represents or models an outside of the system for purposes of correspondence or validation. Instead there is a constant back and forth between the narration of practice that is inside or outside of the story.

As another example, the relationship between code and use is re-imagined in the discourse of agile methods. Code is not articulated as internal to the system and use occurring only at general user interfaces. Instead code is an artifact in the work setting, one way of documenting stories about use (albeit in language translated for computers to receive as input). Discourse is in part about fixing of relationships for the sake of production of knowledge. In agile, use is not discursively tied to code as something that can validate its performances (such as in a user study). Instead, a web of artifacts including the tests "probe[s] what the code really does." "Where the requirements can say whatever they want… the executable tests, there is no arguing with them. If you put these inputs in them you get this output out, that is a fact." In this quote you can see discourse at work in the fixing of relations between objects and the setting up of the conditions for what can be known as a fact.

The legitimation of code as documentation and the idea of a user story existing as a multi-modal artifact set up conditions in which it is not possible to speak about use as a way to validate code. Use of the latest system release is the starting point for a new narration of the user story, a continual feedback loop. In this way the "code is its own reward" for both the developer and the user. Code-in-use gives everyone something to tell new stories about. Our informants emphasize, as in the following quote, the role of interpretation as an impediment to translation work in other methods, whereas the role of interpretation drops out of the discourse in agile. *"When you are 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 in that piece of paper that you think that you are interpreting correctly."* In agile, instead of interpretation of a single story as text, a story is told and retold. "There are times that… the business people ask for something and the engineers don't understand why it is necessary and I have to explain the reasons to them, fill in the whole story… and then take it back to the business."

7. IMPLICATIONS

In this section we discuss some of the implications of this work. As a community, if we look at methods in discursive terms different kinds of group work are highlighted. This has implications for the ways we might consider the organization of design work. The findings regarding the discursive work of methods suggest that CSCW consider aspects of design group work in new ways with regard to the transforming of materials to fit end goals and the narrating of design work and design outcomes. The findings also show that the discursive work of deploying methods has consequences for the broadening of participation in design practice.

7.1 Narrating design work

Considering the discursive work of methods draws attention to the ways that people narrate technological systems as part of practices of both design and use. Many design methods, and CSCW work on design practice, emphasize the ability of designers to describe and represent the system that stakeholders want and value (for example [2]). In phased design, the reliance on authoritative comprehensive documents pushes issues of representation to the foreground. However, in design processes where design and use are happening continually, it may be more relevant to ask about how the system is narrated and how particular voices are able to emerge and become focal or authorial in this narrative process.

Our findings about discourse work suggest as well that the narrating of the organizational work of design is mutually constitutive and inextricably linked to the narrating of the system. Method work involves system users (in-house) and developers to narrate methodical stories about computer-supported work, both design and use work. Narrating the ways that this work comes together in the organization and in the system are difficult to demarcate. Technologists and users of the system must find a way to interface their methods and knit together their narrative accounts of work with the stories that are translated into the software system.

Work on technomethodology suggests that the system interface can provide the user with accounts of the system's behavior [7]. This suggests that the primary resource available for the user to make account of the system is the system itself. Increasingly, however, systems which bring together users and designers within an organizational or institutional framework. Web-based technologies allow designers to capture data about user behavior to analyze and users have access to twitter feeds of developers. Just as the user is a "scenic feature" of design work [23], so too can the programmer or designer be a "scenic feature" of use. In this way, the larger discourse of design and systems development, as located in popular culture and media becomes relevant to these relationships between designers and systems stakeholders.

Discourse, therefore, is increasingly relevant to the understanding of technology design and as this paper has demonstrated discourse is not only the context in which systems are built. Discourse is more than a way of talking about a system, a set of utterances taking place around the system; it can constitute the ongoing relationships in the human-computer assemblage. This discourse has consequences for what it means to narrate a system, and who can have a voice in the narrative.

7.2 Broadening the scope of participation

The approach taken in this paper also has implications for participation in design processes. While Participatory Design (PD) literature has mostly been concerned with the involvement of users in the design process [19], our paper has drawn attention to a broader concern with who in the design process can speak and how. This ranges from the user who is being represented by customer service personnel to the coder who may work on the system yet not be able to make decisions about what gets coded.

Methods discursively shape the scope of design, what is inside and relevant to scope versus outside and irrelevant to scope. This does not mean that the work that takes place out of scope is not integral to the performance of the system. On the contrary, even so-called "use" is a form of labor that enables the system to perform; yet this labor is certainly precluded from the scope of design by most design methods. What kinds of work get called design or not design, coding or not coding has consequences for how visible or authoritative certain kinds of labor become.

Participatory Design research sets forth guidelines and methods for practitioners to adopt in order to enable and encourage participation in the design process. In this way, PD implicitly critiques methods that do not promote participation and suggests that methods have consequences for participation. In this work we have suggested that all methods, including PD methods, establish a discourse on practice that sets relationships and populates the design process with objects. With the lens of discourse the concern becomes not only how methods can be used to get people involved, but also the ways that these methods establish conditions about who can speak in the design process and how.

8. CONCLUSIONS AND FUTURE WORK

Truex et al call methods an "exclusionary conceptual framework" that "conceal" multiple voices and meanings [26]. Agre too suggests that design techniques are often "a method for designing artifacts and a thematics for narrating [their] operation" [1]. In this way, methods, for systems design, are not innocuous. Rather they are consequential to the kinds of voices and ideas that are included or excluded from design process. Future work is needed to understand the specific mechanisms through which voices are authorized in design practice and to consider comparatively and critically the various modes of authority and legitimation that arise in different design methods. While our claims are particular to the discourse of agile and Scrum, we hope the case study demonstrates that an examination of methods from a discursive perspective is possible and valuable. Additionally, it is clear from the ways that "Waterfall" methods become one way of talking about agile methods demonstrates that methods have discursive interplay within the broader software profession and within the field of design. Future studies that speak across multiple organizations and methods could shed more light on this interplay and its consequences.

9. ACKNOWLEDGMENTS

Many thanks to Rosalva Gallardo-Valencia for her contribution to the field study and data analysis.

10. REFERENCES

[1] Agre, P. Toward a Critical Technical Practice: Lessons Learned in Trying to Reform AI. In G.C. Bowker, L. Gasser, S.L. Star and B. Turner, Bridging the Great Divide: Social Science, Technical Systems, and Cooperative Work. Erlbaum, 1997, 1-17.

[2] Anderson, R.J. Representations and Requirements: The Value of Ethnography in System Design. Human-Computer Interaction 9, 2 (1994), 151-182.

[3] Beck, K. Extreme programming explained: embrace change. Addison-Wesley Professional, 2000.

[4] Beedle, M., Bennekum, A.V., Cockburn, A., et al. Manifesto for Agile Software Development. http://agilemanifesto.org/.

[5] Bertelsen, O. Design Artefacts: Towards a design-oriented epistemology. Scandinavian Journal of Information Systems 12, (2000), 15–27.

[6] Boje, D. The storytelling organization: A study of story performance in an office-supply firm. Administrative Science Quarterly, (1991).

[7] Button, G. and Dourish, P. Technomethodology: paradoxes and possibilities. Conference on Human Factors in Computing Systems, (1996).

[8] Carroll, J. Five reasons for scenario-based design. Interacting with Computers 13, 1 (2000), 43-60.

[9] Cockburn, A. Crystal clear a human-powered methodology for small teams. (2004).

[10] Cohn, M. User stories applied: For agile software development. Addison-Wesley Professional, 2004.

[11] Dybå, T. and Dingsøyr, T. Empirical studies of agile software development: A systematic review. Information and Software Technology 50, 9-10 (2008), 833–859.

[12] Foucault, M. Lecture: 10 January 1979. In The Birth of Biopolitics. 1979.

[13] Fuller, M. Software studies: a lexicon. The MIT Press, 2008.

[14] Gerson, E.M. and Star, S.L. Analyzing due process in the workplace. Proceedings of the third ACM-SIGOIS conference on Office automation systems 4, 3 (1986), 70-78.

[15] Grudin, J. The computer reaches out: the historical continuity of interface design. ACM Press, New York, New York, USA, 1990.

[16] Hacking, I. Historical ontology. Harvard Univ Pr, 2004.

[17] Holtzblatt, K. and Beyer, H. Requirements Gathering: The Human Factor. Communications of the ACM 38, 5 (1995), 30-32.

[18] Lukes, S. Power. New York University Press, 1986.

[19] Muller, M.J. and Kuhn, S. Participatory design. Communications of the ACM 36, 6 (1993), 24-28.

[20] Nerur, S., Mahapatra, R., and Mangalaraj, G. Challenges of migrating to agile methodologies. Communications of the ACM 48, 5 (2005), 78.

[21] Schwaber, K. and Beedle, M. Agile software development with Scrum. Pearson Education International, 2008.

[22] Sengers, P., Boehner, K., David, S., and Kaye, J. Reflective design. Proceedings of the 4th decennial conference on Critical computing: between sense and sensibility, ACM (2005), 58.

[23] Sharrock, W. and Anderson, B. The user as a scenic feature of the design space. Design Studies 15, 1 (1994), 5-18.

[24] Suchman, L. Human-machine reconfigurations: Plans and situated actions. Cambridge Univ Pr, 2007.

[25] Suchman, L.A. Office procedure as practical action: models of work and system design. ACM Transactions on Information Systems 1, 4 (1983), 320-328.

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

[27] Woolgar, S. Configuring the User: the case of usability trials. In J. Law, A Sociology of Monsters: Essays on Power, Technology and Domination. The Sociological Review Routledge, London, 1991, 57-99.