Collaborative Events and Shared Artefacts: Agile Interaction Designers and Developers Working Toward Common Aims

Judith M. Brown, Gitte Lindgaard, and Robert Biddle

Department of Psychology and School of Computer Science
Carleton University
Ottawa, Canada

Emails: mmjbrown@connect.carleton.ca, gitte_lindgaard or robert_biddle@carleton.ca

Abstract—Agile processes emphasize collaboration. We were interested in studying collaboration in agile teams including interaction designers, since the integration of user interaction design processes and software development processes is still an open issue. This study focused on designer and developer collaborations in the early stages of project work at four workplaces. We found designer-developer collaborations were extensive and we developed a categorization scheme of collaboration forms and artefacts that support this relationship. While some designer-developer collaborations were directed towards planning, which has been extensively researched, a larger part was directed towards realignment work. This latter type of collaborative work took three basic forms: scheduled, impromptu, and chats. Regardless of the form of collaboration, designer-developer interactions were mediated by twelve categories of artefacts. These artefacts helped designers and developers to determine, more specifically, what to create. We discuss the implications of our observations on alignment work for theory and practice.

Keywords—empirical, qualitative, software, collaboration, agile, interaction designer, developer, artefacts, artifacts;

I. INTRODUCTION

Agile processes emphasize collaboration more than traditional methods [1]. Collaborations or interactions, which are an essential element of collaborations, are mentioned directly in two of the four values listed in the agile manifesto [2] and in the other two values, collaborative techniques are applied to achieve the value [3]–[5].

Collaborating is a direct way for individuals to engage with others to achieve a joint purpose. Collaborative events are not meetings, which are agenda-driven, controlled by a single individual, and unusual events in agile teams. Collaborating can be contrasted with simply interacting, communicating where there is no joint purpose, or to other forms of interaction that lean more toward having a joint purpose, such as cooperating, or coordinating one’s activities with others [6]. Collaborations are complex events that happen through talk, through artefact use, through gestures, through various electronic media, or through combinations of these channels of communication. The value of collaboration to software project success in general is an active research area [1], [7], [8], with much of the research studying the positive impacts of collaboration.

We were interested in studying agile projects involving interaction designers, since the integration of user interaction design processes and software development processes is still an open issue [9]–[11]. The study focused on collaborations that occurred in the early stages of project work, a messy and influential stage. When projects are just starting, collaborations between the customer/business team [12], [13] and the rest of the agile team are especially important so that the team as a whole can build understanding, develop ideas about what the software should be from an end-user, business and technological perspective, and begin the process of bringing the software into existence. To find out how this work is accomplished, we studied one member of the customer/business team (an interaction designer) and one member of the rest of the team (a software developer).

Our focus was collaborative events that involved at least one designer and one developer, which we aimed to characterize. We also focused on the artefacts used in these direct face-to-face collaborations, as a number of studies have shown that artefacts are heavily used in this stage of the work [8], [14]. We aimed to identify these artefacts, describe their general purpose, and their context of use.

II. DESIGN OF THE STUDY

Four teams, self-identifying as agile, at four organizations that were opportunistically sampled [15], participated in the study. Jaba was a medium-sized Canadian organization with 40 employees developing learning games for the Canadian market.1 Talia was a global organization of 15,000 employees developing communication products for a world wide market. Their team used Scrum. Quadrow was a service organization of 1,000 individuals developing software products for the Canadian market. Their team applied XP. Lastly, Edgeworks was an organization of 1000 individuals developing business products for a global market. Their team was using agile practices for the first time. Three teams had 1 designer and several developers. Jaba had several designers and 2 developers. All had a business

1Organization and participant names are fictitious. Images have been modified and blurred to hide identifying information

87
representative. The products being created included a feature for an established product, a web application, a web portal, and a standalone product. In all cases, a well-designed user interface was perceived to be critical to business value. Two of the four organizations studied were developing software “from scratch” (Jaba and Quadrow), one was redeveloping a user interface from scratch (Talia), and one was developing a new feature in an established product (Edgeworks). All the products were successfully completed.

Two teams were novice and two were mature. Planning events included the designer(s), developer(s) and business person(s). These occurred daily at Talia and Quadrow and weekly at Jaba. Team iterations ranged from daily at Jaba to every 2 weeks at Quadrow to every 4 weeks at Talia. Testing was a daily practice at Jaba, extensively practiced at Quadrow, and was being set up at Talia. Only Jaba had a co-located team. At the other organizations disciplines were co-located because most people worked on multiple projects.

For each software project, we collected data from one designer, one developer and, wherever possible, one of their managers, coaches or team leaders. At three of the organizations we collected demographic data, observed, videotaped, photographed, and interviewed. When collecting data at these organizations we first followed the designer for several days, then the developer. At the fourth organization we collected the same information through artefact-mediated interviews, but we were not able to observe and video. These data are summarized in Figure 1. To collect the data we used standard data collection techniques [16]–[19].

Six designers and five developers were interviewed for this study. The years of experience for the developer ranged from 3 to 22 years. The developers ranged in experience from 9 months to 7 years.

A socio-cultural perspective called cultural-historical activity theory [19]–[22] provided the framework for the study. In this framework purposeful, artefact-mediated interactions are understood as the cognitive acts of purposeful agents [23] that occur to sustain a motivated group activity, which is viewed as a system. From this viewpoint, studying mediating interactions in software creation work is akin to studying group cognition directly.

In cultural-historical psychology, mediating artefacts are also understood as an important repository of culture [24], in this case the culture of agile software practices. The framework predicts that teams will leverage agile culture by using artefacts that are similar to the artefacts of other agile teams. The framework also draws attention to the transformation of this culture through innovative actions and team interactions. This was especially evident in this study because the place of the interface designer on agile teams is ill-defined and, we observed, was negotiated on a per project basis. Cultural practices can also be found in patterned ways of working in agile teams (such as patterns of collaboration), which, like artefacts, are both leveraged and transformed in practice. We looked at both of these repositories of agile culture, by creating classification schemes and shedding some light on how, as a system, they contribute to the joint cognitive work of designers and developers.

In preparation for analyzing our data, we first embedded our photos and video snapshots in interview transcripts or meeting notes to be able to examine artefacts (visual, tactile, and audible) in their context of use. We used grounded theory [25] to analyze the data, applying this well-established method for qualitative analysis once to identify various types of collaborations (ways of working), and once to identify the types of artefacts used in these collaborations. Grounded theory is a method that allows researchers to analyze large amounts of integrated data from many different sources to identify patterns in the data and produce theory. At the heart of the grounded theory method is the method of constant comparison. Analysts create carefully defined codes and then higher-level categories that they constantly refine, combine, or break up, as they test emerging theories about the data. The method keeps the analyst close to the data and therefore the categories or theories that emerge are a true reflection of the data that was collected. The constant comparative method was used in this research to create categorization schemes as part of a larger study that aims to create a model, i.e., theory, of collaborative software processes that will incorporate these categories. We used Atlas.ti to support the analysis process that produced the categories. This involved coding transcripts of all meetings and photos of hundreds of artefacts, and grouping these into meaningful categories, which were created during the analysis process. Atlas.ti helped us to code, compare, count, and find data.

### III. Three Forms of Collaborative Work

We found agile teams are jointly motivated by project goals (creating a suitable software product) and various project-specific outcomes. We also observed that collaborative work, much more than individual work, served to clarify product and project aims, and that it regularly shifted and realigned the activities of individuals whose work was intertwined, as was the case for designers and developers.
We found collaborations between designers and developers were ubiquitous at all four organizations and in all forms of collaboration, artefacts were heavily used. We observed designers and developers interacting in scheduled collaborations in prearranged locations. These collaborations usually involved other team members—sometimes even the entire team. Because we also observed deskwork, we noticed that it was frequently punctuated by chats with colleagues or by impromptu collaborations involving several colleagues. Impromptu collaborations happened when two or more colleagues quickly arranged to “work through” an issue in a readily available meeting place. Collaborations through chats were much like impromptu or scheduled collaborations because the topic, the matters discussed, and the artefacts that were used were much the same, but chats were less prolonged and they usually involved only two individuals.

Seventy-five collaborative events were observed in total and of these 39 (52%) involved at least one designer and one developer as shown in Figure 2, albeit broken down into three categories of collaboration. There was a significant amount of variation between agile teams with respect to differing proportions of scheduled collaborations, impromptu collaborations and chats at the various sites. For example, Talia stands out in its exclusive use of scheduled collaboration, and chats were much like impromptu or scheduled collaborations because the topic, the matters discussed, and the artefacts that were used were much the same, but chats were less prolonged and they usually involved only two individuals.

We observed that creating software involves planning and then enacting plans. While there is a great emphasis on planning and development work in the agile literature, we found that 36 of the 39 (92%) designer-developer collaborative events were about re/aligning individual work or individual understandings to meet product and project aims. We see this as an important part of enacting plans and emphasize alignment work here.

Next we describe each collaborative event type in detail.

**Scheduled collaborations** had a prearranged meeting place, a prearranged list of attendees, and a declared purpose. People playing various roles attended this type of collaborative event, which usually included the designer(s) and developer(s). The most structured collaborative events were regularly scheduled project standups in which the team systematically checked the status of the project work. Quadrow organized daily stand-ups, as did Talia, although we did not observe these.

The more common scheduled collaborative events concerning alignment work was less structured and took different forms at different organizations. At Jaba, the lead game designer used these events for communicating vision and direction, and soliciting responses. Talia arranged large events with many stakeholders to discuss specific aspects of the project, and the designer took advantage of these events to present new design ideas. Quadrow held scheduled collaborative events with clients, in which the clients took the lead and designers and developers responded.

**Impromptu collaborations** were opportunistic gatherings of designers and developers in a purposefully designed space that typically spanned half an hour or more. These collaborations were purposeful, and geared toward the work of creating the software, which advanced when the team became aligned. An alternative to a face-to-face impromptu collaboration was an email chain, which occurred only occasionally.

Jaba held mostly impromptu collaborations, or “working sessions” where team members offered their opinions on the software product under development, and the people who attended genuinely wanted to be there. These collaborations were about “touching base”, making misunderstandings visible, “airing” ideas and problems, setting and resetting the direction for the project, and re-motivating the team.

The Quadrow team used impromptu collaborations to explain the design, and “talk around it.” Mason, a designer, used impromptu collaborations to explain “what the team [especially developers] should do” and to test the fit between stories recorded in JIRA “tickets” and mock-ups, which he says helped clarify the project scope. Muriel, also a Quadrow designer, said that impromptu events supported discussions about how something was going to be implemented, and were useful for getting input from the developers that might result in changes to the design. Wil, the developer at Quadrow, said that impromptu collaborations were usually

<table>
<thead>
<tr>
<th>Collaboration Type</th>
<th>Talia</th>
<th>Jaba</th>
<th>Quadrow</th>
<th>Total/ Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduled:</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>– Avg. attendees</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>4.6</td>
</tr>
<tr>
<td>– Involves both</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>6.2</td>
</tr>
<tr>
<td>– Aligning work</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3.9</td>
</tr>
<tr>
<td>Impromptu:</td>
<td>0</td>
<td>11</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>– Avg. attendees</td>
<td>n/a</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>– Involves both</td>
<td>n/a</td>
<td>4</td>
<td>1</td>
<td>2.3</td>
</tr>
<tr>
<td>– Aligning work</td>
<td>n/a</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Work-related Chat:</td>
<td>0</td>
<td>28</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>– Avg. attendees</td>
<td>n/a</td>
<td>2.5</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>– Involves both</td>
<td>n/a</td>
<td>13</td>
<td>5</td>
<td>46%</td>
</tr>
<tr>
<td>– Aligning work</td>
<td>n/a</td>
<td>13</td>
<td>5</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 2. Collaborative events across organizations. Designers and developers were present at most collaborative events, many involving other team members. The majority of collaborations were about aligning work.
were informal, worked-focused, brief discussions that typically lasted about 15 minutes. They were an unstructured way of collaborating that was usually a spur of the moment, often desk-to-desk interaction between two or three people. Chats were also heavily mediated by artefacts. We saw many designer-developer chats at Jaba and Quadrow. At Talia and Edgeworks, the designer and developer, who worked on separate floors, accomplished their chats by exchanging emails.

**Chats** were informal, worked-focused, brief discussions that typically lasted about 15 minutes. They were an unstructured way of collaborating that was usually a spur of the moment, often desk-to-desk interaction between two or three people. Chats were also heavily mediated by artefacts. We saw many designer-developer chats at Jaba and Quadrow. At Talia and Edgeworks, the designer and developer, who worked on separate floors, accomplished their chats by exchanging emails.

**IV. TWELVE CATEGORIES OF ARTEFACTS**

We used Cole’s [24] definition of artefact and found that artefact use was ubiquitous in designer-developer interactions, even when we only considered artefacts that were part of software culture.

…an artifact [sic] is an aspect of the material world that has been modified over the history of its incorporation in goal directed human action [or activity]. By virtue of the changes wrought in the process of their creation and use, artefacts are simultaneously ideal (conceptual) and material. … The properties of artefacts apply with equal force whether one is considering language/speech or the more usually noted forms of artifacts such as tables and knives which constitute material culture. [24]

We saw many artefacts in their visual and tactile forms, but we were also sensitive to more conceptual artefacts embedded in designer-developer talk. For example, designers and developers frequently recounted stories of user interactions. As well as being sensitive to various material forms of artefacts, we were also sensitive to an artefact’s conceptual form, i.e. its meaning. We especially focused on an artefact’s purpose. For example, over the course of 60 days, one team proposed three alternative models for their software including a “content management system”, “network monitoring system”, and “service manager”. These models were meaningful to both the designer and developer because they served to direct and constrain their work. The model helped the designer know how many types of screens to design, and it helped the developer determine the software’s structure. The joint work determined the best model.

The artefacts we saw being used in collaborative events were very accessible in that they were easy for the designer and developer (and other team members) to use and to understand them. Collectively, these artefacts primarily helped the designer and developer (and other members of the team) determine in much greater detail what they should create. The actual building of software artefacts happened when designers and developers worked alone at their desks. Working alone, developers typically wrote program code while designers generated interface proxies with tools that, for two teams, also generated code for the user interface. Collaborative work was essential for realigning this independent work and exploring future work in detail. Collaborative events were generally triggered because of tensions on the project. Designers and developers would use artefacts to resolve these initial tensions and then continue collaborating to reveal and resolve other tensions. Mostly they addressed project and product tensions by simulating software interactions. Typically designers and developers brought artefacts to support their collaborations (usually to illustrate a tension) and took them back to their workplaces, either directly or through memory of the artefact, to support their individual work. Our artefact classification scheme is summarised in Figure 3. The first four categories capture ubiquitous artefacts and the next eight capture equally important, but less ubiquitous artefacts. Below we review each artefact category, presenting typical examples and describing what was achieved through designer-developer interactions with that artefact.

**Design ideas** were concrete suggestions for evolving the interface or application. There were three categories: positive ideas, which were suggestions for shaping or enhancing the software interface; negative ideas, which were suggestions for eliminating something from the software interface; and constraining ideas/rules, which were a suggestion that would constrain the future work on the software interface.

Many examples of design ideas were found in the talk of the designers and developers. As developer Jason said, “I think talking to someone will always be a better way of communicating obscure and abstract ideas,” but sketching was also valuable when words failed: “…if it’s going to take …more than 3 sentences to describe [then I would draw it] …” Wil, also a developer, described how they used sketching to convey high-level ideas at Quadrow:

**Interviewer:** When do you start sketching?

Wil: I don’t know, sometimes, a picture is …far easier for people to understand, [it] makes it clearer. It’s always at the high level—right?—[the] relationship between objects, or relationships between pages, like navigation, or …or what’s gonna be displayed on the page, things like that. …Sometimes you wanna just …brainstorm through it to see if it makes sense. And then, once you’ve …decided on a particular path then, yeah, …everybody lines up on the same page …

An example of a positive design idea is “… we need a new type called …, the ‘are you feeling lucky’ type” (Mason). An example of a negative idea is “We’re going to have to remove this facilitator idea. As much depth as it would give us, we’ve got to get rid of it” (Owen). An example of a design idea that is a constraint is “we should model it the way the consumer would expect it to be.” (Mason).
Artefact Category | Description
-----------------|--------------------------------------------------
Design Idea      | Suggestions for evolving the interface or application.
Design Question   | A potentially challenging inquiry about the interface or application.
Interface Proxy   | Stands in for some aspect of the interactive product, e.g., look, feel, navigation, or flow.
Theory           | A narrative construct that binds together a character, action, and sometimes a setting.
Computational Proxy | A stand-in for the computational back end of an application.
Context Proxy    | A stand-in for the context of the interface or application.
Exemplar         | Another artefact that is like the application being designed in some way.
Framework        | A software structure or digital artefact that supports the development of other software.
Model            | A mental model of knowledge, a thing, or a process or a representation of such.
Plan             | A statement about intended future work.
Software under development | An intermediate version of the product; may be limited in form or contain placeholders.
Stipulations     | A comprehensive description of something, e.g., business needs or design ideas.

Figure 3. Artefact Categorization Scheme for Collaborative Events. All were used in every project. The first four artefacts were ubiquitous within all projects. The others were also important, but less ubiquitous within a project.

Design questions were inquiries that prompted reflection or challenged an assumption, gap, or capability. Not all queries were coded as design questions. Not coded were types of questions that occurred in all kinds of talk, such as questions that confirmed the state of a task. Some questions that were more design-specific were also eliminated such as questions about the wider process, questions that were an idea posed as a question, or questions that were a rule posed as a question. Design questions were used in collaborations to test ideas or models. They were used to find the limits of ideas or models and helped to clarify them. Questions were usually spoken, but occasionally a question was posed with an artefact, as in one case at Edgeworks where a developer created an image to ask a designer how a feature, as designed, would support multi-lingualism.

Examples of design questions from the Jaba agile team include: “Isn’t that part of the game? Isn’t it that you should be able to keep track of everything?”; “The other three areas, do they fit well [with the decisions we’re making here]?” “You’re teaching them to tie actions to employees, is that right [meaning, what we want]?”

Interface proxies stood in for the object of the activity (the interactive software) and supported reflection on the interface. Multiple interface proxies were regularly used by designers and developers in meetings, each one emphasizing a different aspect of the interface (e.g., visual elements, the dynamic interaction, and so on) or each one representing an alternative design solution. Examples of interface proxies are shown in Figure 4 to give a sense of the diversity within this category.

When asked about interface proxies, one designer said,

Well, basically, it’s a focal point for people to discuss. So you’re basically creating something, it might not be the perfect thing, or you might think it is the perfect thing, and then everybody looks at it and goes like: “Oh my God, that is [exactly] what I’m talking about,” or, “I’m not talking about that at all,” … So it’s a way that we can propose new things, … and it’s a concrete thing to look at and they can … actually make comments on it or about it. (Mason)

Damian, the developer on the same team as Mason, saw product proxies as useful because they incorporated Mason’s concepts/ideas of end-user interaction.

Interviewer: You say the designer’s concept diagrams are the most useful. Why?
Damian: It’s … the interaction … it’s more sequence-like. He would present, [then] you get the concept, [then] you get to say [what you think]. Once he’s done, then it’s your turn to get more information, or [to] question some of the concepts. … I don’t think
Stories were narratives that had a main character, an action, and sometimes a setting. They were concrete in at least one of these aspects. They were primarily used to simulate interactions. Stories varied a lot in their richness, but were generally fairly short. An example of a typical story is: “So let’s say that I’m at the very beginning [of the game] and I’m going to [select the] create [button]. Ok? So, I pick one of the things for the actors [in the game] to create.” (Owen)

Usually the main character in a story was a person, but sometimes the product, the application or the calculating engine was the main character in a story, such as this story from Talia “So, if your manager [a software package] managed more than one object type, if there happened to be one, then . . . “

At Quadrow we encountered stories in tabular form that were supplied by clients as test input for the consumer calculator the team was designing. The tables supplied about 20 cases, each case representing a different scenario for the calculator. Precise values for a half dozen variables (one variable per column) were entered in the columns of the tables. The rows represented real-life cases the clients expected the calculator to handle. The cases were used to determine the scope of the calculator, and to identify cases that needed special handling.

Mason explained that stories helped to cope with the complexity of the project work at Quadrow:

Mason: …we need to create, …a document where we have these [a whole bunch of scenarios] in it. And we can run those against the mockups. Which was what we were kinda coming out with because . . . there were too many permutations, . . . it was getting out of hand.

Designers and developers also used stories to motivate the team and to raise concerns.

Computational proxies were an unusual artefact seen in two workplaces. This artefact stood in for the computation engine to be designed, which would be part of the back end of the software application. It supported a discussion about the behaviour of the application, and was only required when this behaviour was complex and would impact the user interface design. Examples are shown in Figure 5. Part a) shows a drawing created at Jaba with paper, pencil and ruler. The team used this game board to play their envisioned game with a penny and dice. The purpose was to explore the variables in the game and to determine what variables to use and how the variables should be used together to compute a final score. They were looking for a variable set that would help them to create a fun game that promoted learning.

Part b) of Figure 5 shows a table drawn on a whiteboard that was created at Quadrow to simulate a computation. The computational engine at Quadrow was very complex. Because some cases could not easily be computed, work with the computational engine helped the designer and developer determine a realistic scope for their application. Quadrow also had a spreadsheet that they programmed to simulate their computational engine (not shown).

Context proxies were designed artefacts that supported designers and developers to access the product context environment directly or indirectly to support an exploration of that context or environment or to simulate user interactions in context. Designers and developers indirectly referred to the context by using documentation or visualizations, but sometimes they could directly access parts of the actual product context. For example, the Quadrow team who were designing for Web users brought the context for their application into the meeting room simply by browsing to sites where their applications would be incorporated. Talia used the previous version of their software running on a server, and had it populated with “toy data” so the designers and developers could “play with it.”

Exemplars were concrete artefacts that were like the application being designed in some way. They usually represented a positive example of something to be emulated, although the occasional negative exemplars appeared too. They helped to set the direction of the work. Negative examples were often the previous versions of products, as was the case at Talia where the team explored the previous product in order to create a list of elements that should change. Exemplars were typically other software products, often the organization’s other products. As developer Jason at Jaba says, “You’re just leveraging what you’ve done quite well [in previous gaming products] to create a new game.”

Sean, the Talia designer, described how each person on the team was influenced by different exemplars:

When we get into what the . . . form’s going to look like or what the tree’s going to look like, we’re going to be proposing stuff that we [the designers] like [which is generally more simple designs]. We have those axes to
grind, like the bag of hammers that we have that we carry around with us and ... we’re gonna throw some hammers on the table and say, “Do it this way.” And they’re [the developers are] going to go, “Well that’s not the way YYY looks” and ... then you’ll get the reverse argument [the argument for a more complex design].

Generally, each team used only a couple of exemplars to guide their work, but these were very influential. However, occasionally exemplars were used a lot. For instance, seven different exemplars were used in one scheduled collaborative event at Jaba when a key interactive element of the game was being designed.

**Frameworks** were a software structure (e.g., code) or digital artefact (e.g., an XML schema or a digital image) that supported, and was the starting point for, the development of other software. As a digital artefact, frameworks were amorphous. Frameworks ensured important aims were met in the completed software and were used by both designers and developers to ensure standards were achieved.

Framework examples were extremely varied and were generally just referred to during collaborative events. Because frameworks were frequently part of the fabric of the work, and assimilated into the culture of the workplace, they were also only rarely mentioned, despite their importance.

Frameworks were observed in all workplaces. Developer Damian’s software platform for the user interface at Talia ensured the entire interface employed many of the designer’s ideas, and that the software-to-be was maintainable. The software interface to the server back end in the Talia project also acted as a framework for Talia’s designer because this was the designer’s source for a clear specification of interface objects. Developers Mason and Muriel at Quadrow used CSS style sheets to ensure accessibility standards were met in their Web software because this meant the standards became a part of that software. Frank, the designer at Edgeworks, considered that each software iteration in their agile-like process provided a stable foundation for future development, so this software was a framework to him. He also regularly built software for the developers that produced standard layouts for screens to ensure compliance.

Digital artefacts provided a solid foundation for, and constraints on, the work of the team, both of which were invaluable in advancing the work. Designers and developers both used them to ensure the work of the other party met important criteria.

**Models** were an abstract artefact that designers and developers used to guide the development of the application. These could capture a rationale, general knowledge, or a process. Models often influenced the overall product and represented a direction or a position to take.

Examples included a domain model, such as the model of the problem domain that the designers were attempting to create at Quadrow that would limit the scope of their work to “something reasonable.” Other examples included a specific gaming model and a specific learning model that would provide the foundation for the game development at Jaba. Talia used models to identify the type of software being developed (e.g., a content management system), which helped them to determine the types of windows to design and the software’s structure. Models were usually mentioned in designer-developer talk, but sometimes they appeared as diagrams or sketches. They often came from outside sources, such as academia, and were powerful influencers on the software’s direction. Figure 6 provides two examples.

**Plans** were statements about intended future work that clarified intentions. Plans sometimes captured important timeframes for the completion of the work, and sometimes people were assigned to tasks. The occasional plan was very complex, but most plans were very simple, and were in the form of a to-do list. Some plans are shown in Figure 7.

Although plans did appear from time to time, intricate plans did not figure strongly in collaborative events we characterize as alignment work. Many to-do lists were generated at the end of scheduled or impromptu meetings.
The software under development supported reflections on the user interface and also the status of the team’s work, hence we distinguish this type of artefact from product proxies. Traditionally, software products are generated from source code, but especially in cases where software has a significant user interface, the software under development is generated not only from code, but also from input files such as XML files or other types of content files that specify text, images, video, or audio. Designers and developers used the software under development to explore how the software would be experienced by an end user. It was rare for this type of artefact to be used in the early stages of design, but an example is shown in Figure 8.

The two organizations developing software “from scratch” made efforts to produce very realistic prototypes of their product so it was “as if” their prototype was the software under development. At Jaba, the developer recreated a prototype of the game under development on a daily basis. These prototypes, especially the first one, had a strong motivational effect on the team. The designers depended on the developer to integrate their contributions (art, graphics, and dialogue) into the prototype, which allowed them to respond, realign, and continue their work. Quadrow developed a realistic prototype for their Web application, but in Quadrow’s case, the designer, and not the developer, was constantly reproducing it. For both Jaba and Quadrow, the flow of interaction and the integration of many elements with each other were important considerations in the work, which could be evaluated best by examining the software under development. Although rarely a part of collaborative work in early stages, once the software was coming into existence, many references were made to it to check its status. Producing this artefact was always very challenging.

Stipulations were usually elaborated in document format and made details clear. Stipulations were comprehensive descriptions of some aspect of the design such as the requirements for software or a detailed written description of the user interface. Stipulations were consciously developed to be a shareable artefact and generally had a set format. Stipulations gave both broad and detailed directions for the work of the team or for part of the team.

The alignment work of designers and developers was joint work. Design ideas helped to advance the work, but we were surprised at the equally important role of design questions, which helped to draw attention to important issues or conflicts. Interface proxies were commonly used to express design ideas. Stories were used to knit varied elements together typically including end users, their purpose, their circumstances, their problems, and their interactions with the proposed interface. These artefacts were ubiquitous in all forms of collaboration.

Computational proxies were used when the computations in the back end of the application would have an impact on the design of the user interface, as happened for the team who wanted to experience the strategic aspects of the game play (from game start to game win) before beginning the design of their game. Context proxies stood in for the context in which the software would be used. Frameworks were very important artefacts that had a very large impact on the team’s work in that they enforced standards. Models and exemplars were highly influential in shaping the general form of the interface. They supported and constrained it and ensured certain attributes in the software, such as accessibility. The software under development only occasionally mediated designer-developer collaborations, but we hypothesized this was because the agile teams we studied were in the very early stages of development. Planning artefacts helped to organize the work, but were not consulted extensively when designers and developers were enacting those plans. Stipulation documents helped to clarify and break up the work.

V. CONTRIBUTION OF THE RESEARCH

This research contributes to a body of research from a variety of perspectives including distributed cognition, situated action, cognitive psychology, and activity theory, all of which have been used to theorize about the work of teams that create technology where the user interface component is a significant factor [6], [26]. Much of this research has emphasized the role of designers as mediators between the more technology-focused team members and the end user community. While acknowledging this as an important perspective, our study has emphasized designers and developers as collaborators. As such, we have contributed a genuinely collaborative model of designer-developer collaborations. In this model, the nature of the challenging, collaborative work is emphasized, and the focus is the accomplishment of this work, rather than the resolution of interdisciplinary tensions. This approach is similar to the work of others who have theorized that software creation work in general is co-constructive and collaborative [27]–[29], but our work is empirical, more explicit, and has a stronger emphasis on group cognition.
A significant amount of research in the last decade focuses specifically on artefacts used in software creation work, an important direction suggested by Kuutti in 1996 [30]. We have tried to find a middle ground between work on artefact use that is very general and primarily theoretical, such as work on boundary objects and bridging elements [31], [32] and work that is more grounded, but focused on only one type of artefact [26], [33]. Our work is most similar to Paay et al.’s, [34] who studied shared artefact use in a software team that included an ethnographer. We have created a picture of a system of artefacts used in agile software creation activity as foundational work for theorizing real software creation activity, building on studies conducted in labs or through workshops [35], [36].

This study presented a categorization scheme of collaboration forms and artefacts found in meetings where designers and developers interacted. The design of the study followed standards for social validity, subjectivity, reflexivity, adequacy and integrity of the data and analysis, and criteria for meaningfulness as outlined by Morrow [37]. This study has validity for similar agile teams, especially when considered relative to theoretical studies, case studies, lab studies, or studies of student teams.

Using cultural-historical psychology, we theorized both collaborative events and artefacts found in meetings where designers and developers interacted. The design of the study followed standards for social validity, subjectivity, reflexivity, adequacy and integrity of the data and analysis, and criteria for meaningfulness as outlined by Morrow [37]. This study has validity for similar agile teams, especially when considered relative to theoretical studies, case studies, lab studies, or studies of student teams.

Using cultural-historical psychology, we theorized both collaborative events and artefacts found in meetings where designers and developers interacted. Our work identifying these elements is also potentially transformative. We focused on the joint work of interaction designers and developers in particular to focus attention on customer-developer collaborations. We found that a collaborative relationship between designers and developers was extensive in all teams. We found that some designer-developer collaborations were directed towards planning, but a larger part was directed towards alignment work, and we focused on this latter type of collaboration. The study showed that collaborative work took three basic forms: scheduled, impromptu and chats and that designers and developers on each team seemed to use all of these collaborative methods, but the percentage of each form of collaboration differed in each of the workplaces. Regardless of the form of collaboration, all designer-developer collaborations were mediated by many artefacts. These artefacts helped to determine, more specifically, how the team would meet product and project aims. Although the specific artefacts differed from one organization to another, all organizations were remarkably consistent in that designer-developer collaborations were supported by artefacts that fell into the twelve artefact categories described in this paper.

VI. CONCLUSION

Knowing the various forms of designer-developer collaboration and the artefacts that mediate these collaborations can help agile teams reflect on their practice, and especially on collaborative work which frequently entails setting the direction of the project and aligning work. For example, we found that Talia had a weak context proxy and we could see that this lack of alignment/understanding of the context impaired their work, which was evident when the designer and developer discussed a login screen design that the business person informed them would in no way meet security requirements. We also found that most teams were very casual about their use of exemplars and models, often selecting these without much of a search, conscious deliberation, or debate, despite their highly influential nature. This work may also help to develop new collaborative tools because it provides a detailed description of the context of collaborative multidisciplinary work. We believe what we’ve written about designer-developer interactions also applies to team interactions. It paints a picture of workplaces that fluidly distribute their collaborative work across a variety of practices including scheduled and impromptu collaborations and chats. This would need to be taken into account in the design of, for instance, collaborative and multi-touch tabletop systems. Such systems would need to be designed so that artefacts could be used freely in all three types of collaborative situations, since we found that similar work is accomplished in all three collaborative forms.

The collaboration between interaction designers and software developers is critical to the success of many agile projects, but little has been known about how this collaboration works. Our contribution has been to examine actual practice and highlight the collaborative events and collaborative artefacts that we saw being successfully used, painting a holistic picture of this work. We have proposed a tentative model and suggested that practitioners could use the categorization schemes for retrospectives. We plan to build on this work to create a more detailed model of the collaborative work of designers and developers.

ACKNOWLEDGMENT

Thanks to Natural Sciences and Engineering Research Council of Canada (NSERC) for SurfNet funding.

REFERENCES

[5] H. Sharp, H. Robinson, and M. Petre, “The role of physical artefacts in agile software development:


