Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems

Nolwenn Maudet, Germán Leiva, Michel Beaudouin-Lafon, Wendy E. Mackay

Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems

Nolwenn Maudet[1,2] **Germán Leiva**[1,2] **Michel Beaudouin-Lafon**[1,2] **Wendy E. Mackay**[2,1]
[1]LRI, Université Paris-Sud, CNRS
Université Paris-Saclay
F-91405 Orsay, France
[2]Inria
Université Paris-Saclay
F-91405 Orsay, France
{maudet, leiva, mbl, mackay}@lri.fr

ABSTRACT
Professional interaction designers and software developers have different trainings and skills, yet they need to closely collaborate to create interactive systems. We conducted three studies to understand the mismatches between their processes, tools and representations. Based on 16 interviews, we found that current practices induce unnecessary rework and cause discrepancies between the original design and the implementation. We identified three key *design breakdowns* where designers omitted critical details, ignored the presence of edge cases or disregarded technical limitations. We next observed two face-to-face meetings between designers and developers. We found that early involvement of the developer helped to mitigate potential design breakdowns but new ones emerged as the project unfolded. Finally, we ran a participatory design session with two designer/developer pairs. Both pairs had difficulty representing and communicating pre-existing interactions. Creating complete interaction descriptions required iterating from individual examples to rule-based representations. We conclude with implications for designing collaborative tools that facilitate the designer's ability to express and the developer's ability to implement complex interactive systems.

Author Keywords
Designer-Developer Collaboration; Design Breakdowns; Shared Artifacts; Collaborative Prototyping Tools.

ACM Classification Keywords
H.5.3. Information Interfaces and Presentation (e.g. HCI): Computer-supported cooperative work

INTRODUCTION
Interactive software development in the 1970s and 1980s involved traditional software engineering processes, with limited interdisciplinary collaboration. By the 1990s, the advent of full color, high-resolution displays enabled high-quality interactive graphics, with a corresponding need for professional designers. However, integrating designers' and developers' work practices has proven difficult, often leading to friction between them [10].

Designers and developers of interactive systems have different backgrounds and skills, and focus on different aspects of the design process [18]. Designers are trained to communicate visually: They use graphical editors, e.g. Adobe *Illustrator* and *Photoshop*, to create "static design documents" [24] such as wireframes and mockups. They prioritize visual appearance and interaction behavior [6] over the rules and data structures that govern the software. By contrast, developers are trained to work with abstractions: They use text editors and Integrated Development Environments (IDEs) to create functional systems. They prioritize the translation of design documents into implementable formats over the details of visual design and user interaction.

Designers and developers often work independently, requiring a hand-off phase for turning static design documents into working code. Although a number of collaborative prototyping tools have emerged that attempt to bridge this gap[1], the transition between design and implementation is poorly understood.

Our work seeks to uncover the problems faced by designers and developers as they collaborate, with a particular emphasis on the representation, communication and interpretation of interactive systems. Our overall goal is to better inform the design of collaborative prototyping tools and facilitate the transition from design to implementation.

After reviewing related research, we describe the results of three studies that examine the collaboration problems of professional designers and developers. In the first study, we gathered 25 stories from 16 designers and developers. We analyze the collected data and generate a classification of the recurring issues. In the second study, we observed how a group of designers and developers worked around these breakdowns during face-to-face meetings. In the last study, we conducted a participatory design session with two designer-developer pairs to explore how they represent and communicate interaction. We conclude with implications for the design of collaborative tools that facilitate how designers and developers represent, interpret and communicate interactive system designs.

[1]See, e.g., http://www.prototypingtools.co

CSCW'17, February 25-March 1, 2017, Portland, OR, USA.

ACM 978-1-4503-4335-0/17/03...$15.00
DOI: http://dx.doi.org/10.1145/2998181.2998190

RELATED WORK

Studying the designer-developer collaboration process

Although user-centered design methods for interactive systems emerged in the 1980s [25], it took a long time for them to be integrated into software engineering processes. By the mid-1990s, Poltrock and Grudin [28] found that large corporations' "development practices blocked the successful application of accepted principles of interface design".

Agile methodologies[2] were created to address a number of software engineering issues, with the emphasis on delivering working software as quickly as possible. However, Ferreira et al. [10] showed how this approach is often at odds with the user-centered design tradition of iterating the design from the user's perspective, before development begins. Silva [8] identified three roles undertaken by designers during a project, and found that these role changes may complicate the designer's collaboration with developers. In their literature review, Salah et al. [29] survey the challenges of integrating agile methodologies into user-centered design practices. They show the need for a "shared understanding of the design vision": developers must understand what they are expected to implement as soon as possible.

In this context, Brown et al. [4] analyze two major aspects of the collaboration process: collaboration events and artifacts. Their study of collaboration events shows that designers and developers constantly perform "interactional alignment work" [31] and that the collaboration process is "patterned around the use of artifacts" [5].

Understanding the role of artifacts

Star and Griesemer [30] introduced the concept of *boundary objects*, which coordinate collaborative work within communities of practice. Lee [16] distinguishes between *boundary objects* designed to "satisfy the information needs of the collaborating parties" and *boundary negotiating artifacts* designed to push the boundaries in complex, non-routine projects that lack standardized objects for collaboration.

In the context of designing and developing interactive systems, the most common *boundary objects* are *design artifacts*. For example, Newman [24] analyzed the specificities of intermediate artifacts such as sitemaps, storyboards, mockups and prototypes. On the other hand, few studies focus on collaboration with respect to design artifacts as *boundary negotiating artifacts* between designers and developers. Brown et al. [4] established twelve categories of artifacts used for collaboration, including "design ideas", "stories" and "interface proxies". We are particularly interested in "interface proxies" because they serve as "a focal point for people to discuss".

Myers et al. [23] surveyed more than 200 designers to understand how they address interaction in their design practices. They found that design documents focus primarily on the visual design: designers find it much easier to communicate visual appearance than interaction behavior to designers. Ozcen et al. [26] concur, noting that designers "struggle to have a conversation with the material" when creating refined interactive systems. Park et al. [27] conducted a laboratory study that shows the differences between how designers and developers use text to represent interaction. They found that programmers "use more verbose descriptions" while "designer's experience with tools like Photoshop and PowerPoint influences their natural expression of behaviors".

Overall, the literature suggests that designers have difficulty communicating the design of interaction behavior to developers. However, the causes for these problems remain unclear. We need to better understand how designers currently represent interaction behavior to developers, as well as which aspects of this representation hinders collaboration.

Tools to bridge the gap between design and development

Several authors propose extending engineering representations to encompass interaction design. For example, UML[3] diagrams can be extended to describe interactive behaviors [7] and serve as a communication tool between designers and developers. Inspired by the popularity of software design patterns [13], Borchers [3] proposed the creation of catalogs of interaction design patterns, and Wiemann [34] proposed using them as the Lingua Franca [9] between designers and developers. Today, developers often use the design patterns defined by software vendors, such as Apple's iOS Human-Interface Guidelines [4] and Google's Material Design [5]. Although these offer a useful starting point for understanding the conventions of a given platform, they cannot solve all representation issues, especially when creating new interactions.

Another approach is to bridge the design activity of prototyping with the development activity of coding. As early as 1995, Myers [22] advocated prototyping tools that support a wide range of interactions. Landay et al. [15] proposed an electronic sketching approach and envisioned "a future in which the user interface code will be generated by user interface designers using tools". To support *evolutionary prototypes* [2] that gradually evolve into the final system, DENIM [17] lets designers create sketches in the early design phase and later augment them with drawings that express interactivity. Similarly, DEMAIS [1] uses storyboard annotations to produce working examples early in the design process.

Outside the research community, practitioners acknowledge the gap between design artifacts and their subsequent implementation. Within the past few years, over 40 commercial prototyping tools have appeared[6], and a 2015 survey of 4,000 designers found that 53% use a prototyping tool [33]. However, these tools do not result in final implementations: designers still lack tools that link their representations to developer's implementations. Our goal is to inspire the next generation of prototyping tools by identifying and analyzing the problems caused by the limitations and misinterpretation of design artifacts and proposing strategies to overcome them.

[2] http://www.agilemanifesto.org

[3] http://www.uml.org

[4] http://developer.apple.com/go/?id=app-review-ios-hig

[5] http://www.google.com/design/spec/material-design

[6] http://www.cooper.com/prototyping-tools

STUDY ONE: DESIGNERS & DEVELOPERS INTERVIEWS

The goal of the first study was to examine the existing practices of professional designers and developers from a wide variety of settings. We were particularly interested in how:

- *designers* represent and communicate a design;
- *developers* interpret the design; and
- *designers and developers* identify and overcome breakdowns that appear during the process.

METHOD

We conducted critical incident interviews [19] about recent design projects, in order to obtain specific, detailed stories of their successes and failures. We were particularly interested in their problems representing and communicating interaction with each other. We also looked for recurring patterns across work settings, cultures and types of projects.

Participants

We recruited 16 professional designers and developers (7 women, ages 24-46) from France (8), Sweden (3), Argentina (2), the USA, Canada and China, who create web sites, mobile applications or interactive installations. Their work environments include: digital agency (6), design studio (4), start-up (2), freelance (2), and software factory (1).

Participants $P1_{ds}$-$P8_{ds}$ are designers (ds), self-described as UX Designer, Visual Designer, Interaction Designer, or Graphic Designer. Participants $P9_{dv}$–$P16_{dv}$ are developers (dv), self-described as Mobile Developer, Web Developer, Front-End Developer, or Creative Coder. Their experience in collaborating across disciplines, i.e. from designer to developer or from developer to designer, ranges from 1.5 to 20 years (mean 8). Half of them typically collaborate remotely, none have worked with each other. All participants reported that they follow agile methodology.

Procedure

We interviewed participants in their studio or office for approximately 90 minutes. We asked designers to choose recent projects in which they collaborated with a developer, and asked developers to choose recent projects in which they collaborated with a designer. For each project, we asked them to show us their tools and the specific artifacts they created, and to describe, step-by-step, the details of how they communicated the design or implementation. We probed for both successful and unsuccessful collaboration examples.

Data Collection

We collected 25 stories (one or two per participant) from different projects. During the interviews, we recorded audio and video of the participants manipulating the artifacts they created. We also photographed the final products they produced and took notes.

Analysis

We analyzed the 25 stories using a Grounded Theory [32] approach. We studied the projects with a particular focus on breakdowns related to creating or interpreting the design documents. We first selected examples that formed natural categories, looking for higher-level concepts that emerged from the details of each projects. We iterated and mapped each story to one or more categories. We illustrated each project with a modified version of a StoryPortrait [14] to facilitate the analysis (see Figure 1). A StoryPortrait includes a photograph of the interactive system or a key artifact created during the project, as well as a timeline to show the successive steps of the collaboration, including the participant's quotes and drawings.

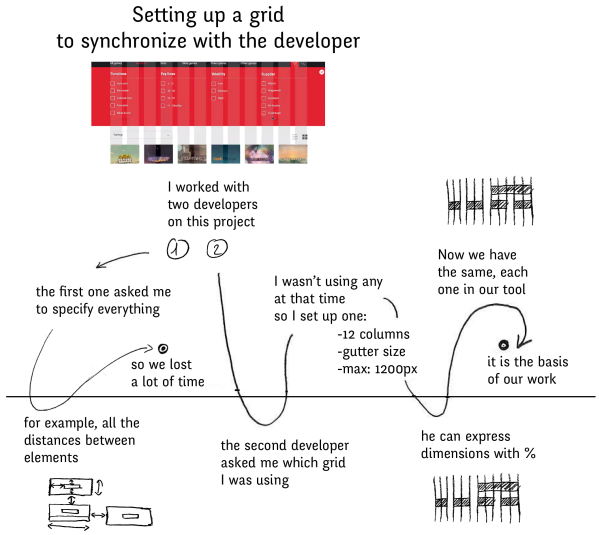

Figure 1. Each StoryPortrait includes a photograph of a key artifact, as well as a timeline to show the successive steps of the collaboration, including participant's quotes and drawings. Here $P5_{ds}$ explains how she created a grid system to collaborate more easily with the developer.

RESULTS AND DISCUSSION

Reworking and Redundancy

Designers produce multiple design documents

All designers create multiple documents to communicate different aspects of their designs. Designers create extra design documents when the original design documents lack specificity or lead to confusion. Unfortunately, much of the information in these additional documents is redundant. We found that designers spent time recreating the same information across separate documents. For example, $P2_{ds}$ created five documents to communicate the design of a small application: *UxPin* "for sharing mockups"; *Pixate* "for detailed animations that cannot be expressed with words"; *InVision* "for interactive mockups with basic interactions and annotations for non-obvious features"; *Photoshop* because "these developers are used to work with .psd files"; and *Illustrator*, "the software we actually use to produce the screens." She also used email to communicate additional design details to the development team. Even with all these documents, this designer was unable to clearly communicate the design.

Designers use several media to represent interactions

Although all designers use *images* of "screens" to represent the visual design of the interface, these are insufficient to accurately describe user interaction. Designers resort to other

formats, each with different trade-offs, to communicate their ideas. Most common is *text*, used extensively by all designers. For example, $P6_{ds}$ briefly described in an email how the user moves between screens: "from the login screen, you slide to the next screen". Text comments and annotations are easy to produce, but rarely sufficiently explicit or complete, leaving details open to interpretation. Less common is *video* (24% of the projects) which makes it possible to visualize custom animations, but is expensive, time-consuming and does not fully communicate the user experience. Finally, designers occasionally create *interactive mockups* (12% of the projects) using the built-in set of interactions in the tool of choice. These communicate how the interaction should feel, but only when the tool has the right set of pre-defined interaction types.

All but one designer created custom *"guidelines"* or *"specifications"*. For example, $P7_{ds}$ created a video to communicate the design of an "in progress icon" animation. When the developer was unable to recreate the design from the video, $P7_{ds}$ created an additional file that "extracts the useful information [from the video] and represents it on a timeline".

Developers recreate design documents

The most common activities mentioned by the developer include interpreting the design documents and recreating them with developer tools. For example, $P9_{dv}$ received an informal text description of a custom animation, but had to ask for a visual representation in order to fully understand the design.

We were surprised by the amount of time that developers spent recreating design documents. Some developers came up with interesting strategies to increase their productivity. For example, when developing a mobile radio application, $P14_{dv}$ inserted the provided image as the background of her corresponding view in the IDE's Interface Builder. She then positioned her components on top, to recreate the designer's composition. She could then "figure out the [layout] constraints" of the screen to make it responsive, such as determining that some elements were center aligned. $P11_{dv}$ created a similar setup with two monitors. To implement the visual design, he places the mockups on the smaller screen to assess them: "I measure by eye rather than being pixel perfect."

Developers misinterpret designs

Many design decisions are lost, as developers struggle to interpret and implement the designer's original intent. In fact, none of the initial implementations were exactly as the original design. $P1_{ds}$ felt that the developer "used our design as an inspiration, then he made many design decisions that he did not have to take". Similarly, $P3_{ds}$ provided a video that showed the developer how to vary a text-box color according to the background picture. He later realized that the developer had only partially implemented his idea by sampling a single pixel, instead of generating an average color based on several pixels.

During the implementation phase, designers create *correction* documents to show the location of the mismatches and what should be modified. For example, to correct a vertical misalignment, $P3_{ds}$ created a video. He first traced a segmented line to highlight the misalignment and then animated the correct repositioning of the elements. In the context of a real state website project, $P6_{ds}$ discovered several visual mismatches including wrong margins, colors and fonts. He decided to modify the CSS and correct the mistakes by himself, using the web-browser developer tools. Because these changes were local to $P6_{ds}$'s browser, he screencaptured the new website's look and added some annotations linking the modified CSS code to the visual result. The developer then recreated all of these steps with his own tools.

Strategies to avoid rework and redundancy

We found cases of rework and redundancy in all the interviews, but only two developers and one designer explicitly mentioned strategies to avoid them. $P14_{dv}$ had a simple solution: "Designers should just create their interfaces directly in Xcode", referencing the Interface Builder within the IDE.

$P5_{ds}$ designed a complex casino website with many similar UI components. To avoid recreating them each time, she "was inspired by the developers' way of working": she created a modular styleguide that served as a shared visual library. She could then copy modules from the styleguide to create each new screen, gradually adding new modules or missing information such as the color of the hyperlinks, as requested by the developers.

$P12_{dv}$ began with mockups and specifications for a web-based interactive advertisement builder. He used *Flash* to create the architecture of the interface, writing the code "so that the designer could easily touch it". The developer encouraged the designer to directly manipulate the code to fine-tune look and feel details, such as modifying the images or changing the duration, delay and type of each animation. This strategy allowed $P12_{dv}$ to avoid misunderstandings and unnecessary back and forths.

Design Breakdowns

We use the term *design breakdown* to describe an impediment that must be fixed before the design can be implemented. We identified three recurrent types of *design breakdowns* related exclusively to the collaboration between designers and developers (Figure 2). These categories emerged from the most common issues encountered in the 25 analyzed projects.

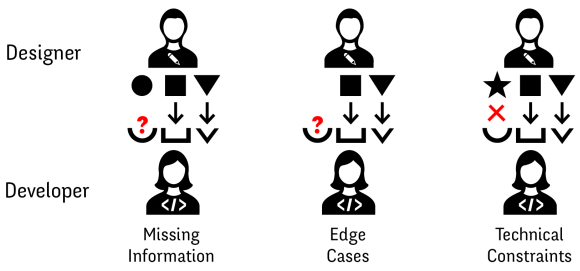

Figure 2. Key *design breakdowns* between designers and developers: *Missing information*: Designers do not communicate necessary details. *Edge Cases*: Designers do not consider certain problematic situations. *Technical constraints*: Designers are not aware of technical limitations.

Missing Information

The first type of breakdown occurs when the designer makes a decision without communicating it to the developer. Two designers and four developers reported cases of *missing information*. For example, $P9_{dv}$ received an interactive mockup of a webpage. He could not determine whether the page's calendar widget was interactive or simply the output of another interaction. $P9_{dv}$ also lacked the design rationale: "What did they create that calendar for?" Similarly, $P13_{dv}$ received only static mockups for a sports application, and could not determine how to move from one screen to another. $P14_{dv}$ created a "design specification file" for the designer with missing information from the original design files.

Designers found it difficult to represent and communicate dynamic behavior to the developers. For example, $P8_{ds}$ wanted to create an animation of a blossoming flower but did not know how to represent her idea in *After Effects*. She ended up drawing a few sketches and then sat next to the developer as they worked out how to implement her idea directly in code. In two cases, the designers avoided mentioning interaction at all, relying on the developer to create off-the-shelf interactions. This supports Myers et al.'s [23] argument that designers find interactions hard to represent. For example, $P6_{ds}$ provided static design documents without representing some transitions, even if they were simple: "I let the developer pick the interaction between the screens, since they are very basic."

Edge Cases

The second type of design breakdown is missing edge cases, when designers focus on typical scenarios and do not consider extreme or problematic situations. Developers are trained to think about edge cases; designers are not. All developers reported that designers omit important edge cases from their design documents, and that they had to decide how to handle these situations themselves.

$P13_{dv}$ received only mockups to develop a sport application. Because the designer had only specified the "sunshine cases", $P13_{dv}$ had to make design decisions for each of the different edge cases. For example, the client required him to include advertisements, so he modified the original design to accommodate the ads. Similarly, $P16_{dv}$ prepared a responsive grid for a cruise company website. The original mockup only featured the desktop version of the website. $P16_{dv}$ did not know how to handle large elements that did not fit within the width of the screen of the mobile version: "Should the rectangle be transformed into a square or should it take a full row?" For $P16_{dv}$, designers usually "don't take into account the dynamic nature of the data".

Responsive websites make it particularly difficult for designers to consider all possible layout cases. For example, $P11_{dv}$ explained how designers of a responsive website had specified the element widths based on a percentage of the screen, but did not consider what the maximum width should be, forcing $P11_{dv}$ to make the decision using his "designer's eye".

Some designers overcome these issues with design guidelines. For example, $P4_{ds}$ created a 16-page specification with annotated wireframes to explain the sign-up functionally of a website. She reported that "specifications make me think of all the states and exceptions". She also used the guideline to capture and communicate the rationale for her design decisions. Similarly, $P3_{ds}$ created a spreadsheet to help him think and "explain the rules of the game and the limits" for each website element.

Technical Constraints

The third type of design breakdown is the designer's lack of awareness of technical limitations, either in general or with respect to the developer's skills. Five designers and four developers reported breakdowns due to such misunderstandings, which created additional work for the developer. For example, $P13_{dv}$ received a design for an iPad application that called for horizontal scrolling when in portrait orientation. But $P13_{dv}$ "could not recycle his code from the landscape version to create it". He had to reimplement it from scratch, since it had already been approved by the client.

This type of misunderstanding leads developers to modify the design themselves. For example, $P11_{dv}$ created a responsive website for a start-up. The designers created a desktop and a mobile version of their design but "did not realize that they had modified the behavior between the two versions and I would have had to develop two different source codes". Instead, he decided to redesign the layout to make it feasible as a "simple responsive website".

Not being aware of technical constraints is also a problem for designers. For example, when working on a complex website, the developer first told $P6_{ds}$ that "everything was possible". $P6_{ds}$ soon discovered that the developer was unable to implement many elements with his tools, even though they had already been validated with the client. $P6_{ds}$ said: "He should have said it earlier, we would have adapted our design." Instead, they were forced to redesign the project several times to accommodate the developer's limitations.

Collaboration is usually smoother when the designer is aware of the developer's constraints and possibilities. For example, $P5_{ds}$ worked on a project with two different developers. The first asked her to specify all the dimensions, such as the distances among all the elements on the screen, "so we lost a lot of time". The second developer asked for a grid specification, which she created with 12 columns, a gutter size and a maximum size of 1200px. "Now we have the same, each one in our own tool." The grid allowed the developer to express dimensions in percentages, sparing $P5_{ds}$ the need to make additional annotations and saving a great deal of time (Figure 1).

Late developer involvement

Even though all participants claimed to use agile methodology, only five of the 25 projects (two remote and three co-located) included face-to-face sessions between designers and developers dedicated to co-design the initial interaction. For example, $P4_{ds}$ had an idea for a custom navigation rule and invited all the designers and developers to help design it. The developer was able to implement the resulting navigation behavior without additional instructions or documents: "Nothing was written down, we only had the screens."

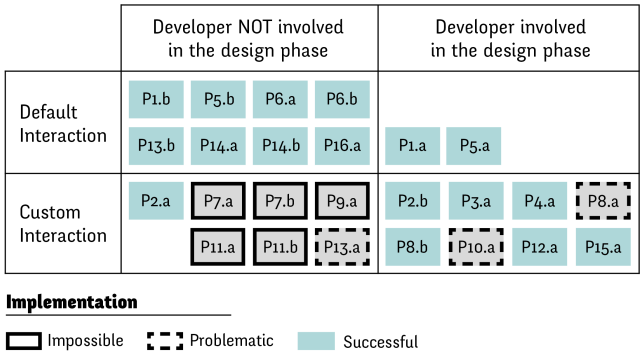

Figure 3. Relationship between interaction complexity and developer involvement. Lack of developer involvement in the early phase of custom interaction design is correlated with problematic or impossible implementation.

Other similar examples suggest that involving developers during the design phase makes it easier to create complex interactions (Figure 3). In such cases, developers gain an understanding of the desired interaction during the meeting and designers need not fully represent it in their design documents.

Developers were most likely to be called in for the design phase when the project included custom interactions. In most of these cases (6/8), developers successfully implemented the desired custom interaction, as in the aforementioned example of $P3_{ds}$'s flower animation. However, when the project required a custom interaction and the developer was not involved at the design stage, most developers were not able to subsequently implement the proposed interaction (5/7). For example, $P7_{ds}$ reported that the developer "just did not implement" the custom transition he had proposed. One of the remaining cases was still problematic: $P13_{dv}$ was frustrated with the proposed interaction: "I could not recycle my code, but as the design had already been validated by the client I still had to implement it. I lost a lot of time."

In summary, we identify three main types of issues when designers and developers collaborate on the creation of interactive systems: reworking and redundancy, *design breakdowns* and late developer involvement. We found reworking and redundancies in both designers and developer practices. Designers struggle to represent interaction with their current tools and use multiple design documents to communicate different aspects of their design. Developers spend an excessive amount of time recreating the designer's documents and correcting their misinterpretations. During the implementation phase, developers face three types of *design breakdowns* — missing information, edge cases and technical constraints — that undermine the collaboration process. Our data also suggests that projects requiring custom interaction should involve developers early during the design phase.

STUDY TWO: CASE STUDY

Motivation

Study one identified three types of breakdowns that occur between designers and developers in a wide range of contexts. To further understand these breakdowns and how they are addressed, we conducted a longitudinal study of a team of designers and one developer. We observed $P1_{ds}$ from Study One and his team during the entire duration of a one-month project, a responsive website for a crowd-sourced directory of companies. We were interested in whether *design breakdowns* still appear when a developer is involved early in the project, and, if so, which strategies are used to avoid or mitigate these breakdowns.

Participants

We studied three designers and one developer (ages 24-25, one woman). This was the first time that this group of designers had collaborated with this particular developer.

Procedure

We observed the two face-to-face design meetings that involved all the designers and the developer. The first two-hour meeting (Figure 4) focused on the design of the website. The second meeting lasted an hour and focused on implementation. We also interviewed the designers separately, prior to the second meeting, to learn more about their design tools.

Data Collection

We video recorded both meetings and took notes. We took pictures of collaborative actions, i.e. exchanges between the designers and the developer, and their manipulation of artifacts such as drawings, notes and software. We also received copies of the emails exchanged during the project.

Analysis

We used Chronoviz [11] to annotate relevant, interesting events during the meetings. Two coders marked and analyzed the times when a participant asked a question, or when a designer sought confirmation from a developer or vice versa. We correlated these marks to the *design breakdowns* classification from Study One.

Results and Discussion

The email exchanges focused primarily on discussions with the client about requirements and validating design decisions. Since these activities are beyond the scope of this paper, we focus our analysis solely on the two face-to-face meetings.

First Meeting - Accounting for design breakdowns

The main benefit of the early face-to-face meeting was to let participants seek validation from each other and to avoid potential problems. We identified examples of avoiding *missing information*, considering *edge cases*, and clarifying *technical constraints*.

In order to avoid *missing information* (12 occurrences), the developer often encouraged the designers to specify concrete details about their design ideas. For example, when the designers proposed a button related to the advanced search feature of the website, the developer demanded greater precision: "Is it going to have radio buttons or checkboxes?" Similarly, when a designer suggested that "there should be two sharing buttons", the developer immediately sought concrete details: "Ok, but what exactly do you want me to share... the URL of their website, the URL of our website or a Facebook link?" The

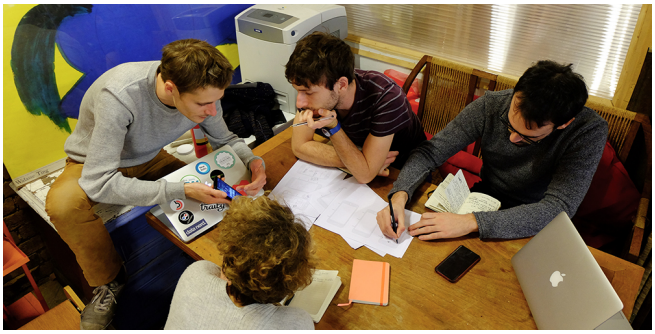

Figure 4. First meeting. The developer shows an example of an existing interaction to the designers while one designer represents it on paper.

developer required these details in order to translate the design idea into specific elements that he could implement.

The mere presence of the developer pushed the designers to be more explicit about certain design issues. For example, when designing the "company card" that would be displayed on the search page, one of the designers realized that he needed to specify the title's maximum number of characters to maintain the visual consistency.

The developer also pushed the designers to think about *edge cases* (5 occurrences). For example, when designing the category system for filtering companies, he asked the designers: "Can a project exist without a category?" This insight led one designer to come up with a different strategy: He proposed an "other" category that groups together previously uncategorized companies. Similarly, when the designers proposed adding a gesture for deselecting a category on the mobile version, the developer asked them to consider how this design decision would affect the desktop version of the website. Given the developer's warning, the designers decided to skip the feature: "Based on what you just said, I think we should not let the user select different filters."

Designers often sought validation, confirmation or information about *technical constraints* (17 occurrences). This echoes the "considering implementability" category observed by Brown et al. [5]. First, they were able to confirm with the developer the feasibility of their design. For example, one designer asked the developer: "Is it possible to have a swipe gesture on a mobile website?" In order to make informed decisions, they asked the developer about the complexity of implementing certain designs. When the designers proposed a search feature for companies, the developer asked them to specify exactly what should be searchable. The designers idea was to search within all company-related information, including their descriptions. The developer replied: "Everything is possible... but if you really want to make a search inside the description, it will be a bit more complex." He suggested only looking up names and tags, but with an autocompletion feature. The designers agreed.

Second Meeting - Fixing design breakdowns

Even though they were able to handle many design breakdowns during the first meeting, new ones appeared during the implementation process. The developer found new *edge cases* (4 occurrences). For example, he noticed that a company card with multiple subcategories would occlude the company's name: "There is a risk that it overlaps with the title, I think it should be redesigned." The designer responded: "Maybe we can put three dots and display the extra ones only on [mouse] hover."

The developer also requested *missing information* that he could not infer (8 occurrences). For example, he could not understand why the subcategory was not displayed on the company card. The designer had thought about this, but did not communicate it to him: "I did not explain it in my screens but here we are actually within the housing category. In fact, the housing icon should be highlighted in the upper menu." In this case, the developer had interpreted the highlight as a hover state, and not as a selected state. The developer also asked: "What exactly is clickable on the item [company] card? Is it only the title and image or the whole card?" Another problem was that the developer could not understand the purpose of a cross in the corner of the company card: "This little cross here, what should happen when I press it? Is it a back button?"

In a few cases, designers asked for more details about decisions made by the developer. For example, when reviewing the search feature, the designer asked for a clarification: "In which order are the items shown when they are displayed as results?" Designers also questioned some of the developer's decisions: "Why do we need pagination? Is it because of the heavy loading [time of the HTTP] request?" The developer nodded as the designer proposed an alternative: "We should put the maximum number of items on the page without loading problems."

Vocabulary mismatch between designers and developers

In both meetings, differences in the vocabulary used by designers and developers led to miscommunication. Sometimes, designers and developers used different terms for the same concept. For example, during the second meeting the developer talked about a "fixed" element using CSS terminology. The designer, who tried to take the user's perspective, referred to the same object as a "moving" element, an element that follows the scroll. It took some time for them to discover that they were talking about the same behavior.

We observed several strategies for overcoming these issues (5 occurrences). Developers and designers tried to bridge the vocabulary gap by adopting each other's terminology. For example, when discussing whether an item should appear in several categories, one designer started using mathematical concepts when communicating with the developer: "Is it the union or the intersection of these two categories?" The developer also reformulated the original design idea in terms of UI widgets: "It is either a radio button or a checkbox."

On several occasions, designers and developers looked up specific interaction techniques on a particular website or found examples from a mobile application on a smartphone to show the others. This "communication-by-example" helped them verify that they were talking about the same interaction technique.

In summary, we found that both designers and developers actively try to mitigate *design breakdowns* when meeting face-

to-face. Involving the developer at the beginning of the design process helped the team reduce the amount of *missing information* in the design documents, discover and handle *edge cases* and set clear *technical constraints* for the scope of the design. Even so, new *design breakdowns* occurred during the implementation phase and had to be solved collaboratively by the team. Vocabulary mismatches also created several collaboration issues, especially when discussing interactive behavior.

PARTICIPATORY DESIGN WORKSHOP

Motivation

Study Two suggests that early involvement by the developer can reduce some, but not all breakdowns. The third study attempts to tease apart problems that arise from generating design ideas and those that arise from an inability to successfully represent the interaction itself. We conducted a participatory design workshop that combines multiple methods [20], to complement the interviews from Study One and the longitudinal study from Study Two.

First, we are interested in whether *design breakdowns* are simply a natural result of the creation process, or if they are also by-products of the limitations in the representations used to describe interactive systems. Second, since these representations are traditionally the product of designers, we wanted to elicit new kinds of representations by asking designers and developers to create them together.

Participants

We recruited two designers and two developers (all men, ages 24-33). Two of them (one designer and one developer) participated in Study One and were invited due to their interest in this research. The other two professionals were recommended by participants from the previous studies. The developers had not previously worked with the designers. They had 1.5 to 10 years of experience collaborating across disciplines. Besides the four active participants, the authors attended the workshop: two as observers and two as participant-observers.

Procedure

The workshop lasted three hours and featured two activities designed to examine how designers represent and communicate *existing* interaction behaviors. To make the interactions more challenging to describe, we avoided standard widgets such as buttons, or standard interactions such as mouse clicks. Instead, we chose novel, unfamiliar interaction techniques from two mobile applications that rely heavily on continuous gestures. Participants were given the opportunity to explore these techniques for themselves on a mobile device we provided. The techniques included:

- Interaction 1: The *Clear* to-do list mobile app [7] uses a spread gesture to progressively indicate the creation of a new item between two existing items. Lifting the fingers creates the item.
- Interaction 2: The *Paper* note-taking mobile application [8] uses a lasso technique to select an area of the canvas to be cut, which can then be moved with a pan gesture. While moving, a tap with another finger outside the selection copies it at that particular location.
- Interaction 3: The *Paper* app uses a lasso selection to specify an area to fill with a color selected by tapping on a color swatch. When the lasso crosses itself, the area is colored with the so-called even-odd winding rule, leading to unexpected results.

Activity 1 - Warm-up interaction game (1h)

Designers and developers were divided into two pairs, grouped by roles. Designers received *Interaction 1* and developers received *Interaction 2*. We made sure that none of the participants knew these interactions. We asked the designers to describe the interaction as they would ideally communicate it and asked the developers how they would receive it. We instructed them to give as complete a description as possible, and gave them access to all the tools and means they use in their daily work practices. When participants were satisfied with their representation, they gave the resulting artifacts to the other pair. Each pair then tried to describe what they understood from the representation. We also asked them to try to find the interaction in the real application. Afterwards, participants discussed the issues encountered as they created and interpreted the representations.

Activity 2 - Communicating an interaction (2h)

The two designer-developer pairs received *Interaction 3*. We asked each pair to come up with strategies or new representations that fully communicate the original interaction. We asked them to create representations that satisfy both members of the pair.

Data Collection

We collected all artifacts created by the participants: sketches, diagrams, text descriptions, paper prototypes and stop-motion videos. We took photographs and videos as they manipulated these artifacts, and took notes during the discussions.

Results and Discussion

Current representations do not encourage completeness

Participants took approximately 15 minutes each to create and be satisfied with their representations in Activity 1. Even so, it was clear that, even though they were given a fully functioning interaction, the four proposed representations were incomplete. This suggests that some design breakdowns are a by-product of inadequate representations.

The designers relied primarily on visual representations based on drawings and annotations. Developers felt that these were effective in communicating the overall idea, but left too many unanswered questions for correct implementation. For example, the designers did not communicate certain types of feedback, such as the gesture spread threshold or the animated transition that placed new items at the top of the list. During the discussion, one of the developers explained that a picture requires more translation steps than a text description for implementation: “If I receive a picture, I first need to translate it into text and then I need to translate it into code.”

[7] http://www.realmacsoftware.com/clear/

[8] http://www.fiftythree.com/paper

Developers relied mainly on text, including programming vocabulary, e.g., conditionals and loops, complemented by a few visual elements. Text descriptions provided specific information for the implementation but did not clearly convey the look and feel of the interaction to the designers. For example, when trying to represent *Interaction 2*, developers did not communicate the increased opacity outside the selection and the flash effect when pasting the copied area.

Strategies for creating complete representations

During Activity 2, the two pairs explored seven different strategies for fully representing and communicating the interaction. We describe two of the most promising: Pair 1 decomposed *interaction 3* using examples from other applications: the lasso tool from Photoshop combined with the paint bucket from Illustrator. The designer from Pair 1 proposed recording videos to demonstrate the use of these tools and combine them. He argued that this strategy would avoid misunderstandings as well as provide a complete description of the interaction. The developer from Pair 1 proposed a shared "lexicon" describing the objects of the program, their characteristics and the tools that can interact with them. Pair 1 thought that a common vocabulary would facilitate the discussion about how to extract common components.

In order to reach a shared and complete representation, both pairs refined their representations through multiple iterations. They started with a visual example, and then added rules and annotations to produce a more complete description. For example, the designer from Pair 1 drew a snapshot of the interaction at four points in time: "(1) I touch, (2) I move (3) I release (4) The tool creates the closed shape." (Figure 5a). Next, the developers and designers collaborated to gradually generalize the description of the interaction. The developer, inspired by the designer's representation and his knowledge of "flow programming", drew a diagram representing the different components of the interaction (Figure 5b): finger, shape, line and closed shape. Based on the developer's representation, the designer built a new representation that combined the strengths of both proposals (Figure 5c). He color-coded a visual example and mapped each graphical element to a detailed programming-oriented description: "a straight line between the point of origin and the current finger position [...] a curve that records the complete path of the finger from the origin to the current finger position [...]".

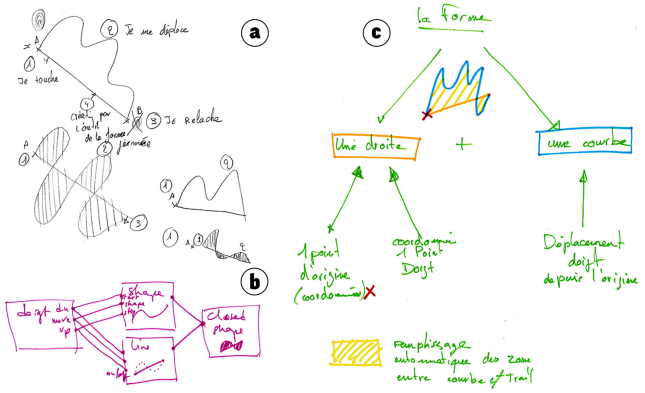

Figure 5. (a) Designer drew a snapshot of the interaction at four points in time. (b) Developer created a diagram connecting primitive graphical elements and functions with user inputs. (c) Designer merged the two representations with an example.

In summary, we observed that even when provided with an existing and complete interaction, both designers' and developers' representations suffered from *missing information* and *edge cases*. This suggests that current representations are limited and may result in *design breakdowns*. To create complete representations, designers and developers gradually added rules on top of concrete examples.

DISCUSSION AND IMPLICATIONS FOR DESIGN

We identified three types of *design breakdowns* that occur, regardless of the distribution of the team, their experience or the project domain. Designers sometimes create representations with *missing information*; they often fail to consider problematic *edge cases*; and they may be unaware of *technical constraints*. Our studies show that breakdowns can result in severe mismatches between the system originally envisioned by the designers and its final implementation. To avoid such consequences, we should help designers and developers identify these breakdowns and solve them as early as possible.

What causes these breakdowns? Clearly, some designers and developers lack adequate skills. However, the ubiquity of breakdowns across a wide variety of professionals, on diverse projects in different settings, makes it unlikely that this is the only factor. What about differences in training and background knowledge? For example, several *technical constraint* breakdowns occurred when designers did not understand the limitations of the current software platform. Moffett [21] suggests including programming concepts as an integral part of designer's training. Conversely, offering developers basic design training would help them fill in *missing information* by applying relevant design principles.

Miscommunication is another key factor. Like Ozcen et al. [26] and Park et al. [27], we identified numerous occasions where designers struggle to communicate interaction. Study Two shows that this can be mitigated somewhat by involving developers early in the design process, as advocated by Salah et al. [29].

Study Three indicates that a key source of design breakdowns and subsequent rework is the lack of true *boundary objects* for representing interaction that can be shared by designers and developers. Designer tools generate isolated representations that must be updated manually if the actual system changes. Designers are also responsible for manually keeping the design consistent across multiple design documents, often with differing formats. By contrast, developer tools require precise and exhaustive representations that can be compiled and executed, and manipulate representations directly connected with the actual system, ensuring that they are always up-to-date.

We believe that tools for the design and implementation of interactive systems must accommodate the skills, values and practices of both designers and developers, with better integration between them. Based on our findings, we suggest four key implications for the design of tools to support collaboration between designers and developers.

Multiple synchronized views

Today's design and development tools operate in isolation: changes in a designer's tool such as *Sketch* are not reflected in the developer's tool, such as *XCode*. Worse, changes in one design document are not automatically reflected in the others. Some designers and developers address this by linking their documents via cloud-based file syncing or by referencing external resources from the code. Also, several IDEs support multiple views: UI code can be opened with a text editor or an interface builder and manipulating either tool modifies the underlying code file. Unfortunately, these solutions are tool-specific and ad-hoc, and do not reduce the amount of reworking across design documents and development tools.

We argue that interaction design tools should support syncing capabilities and automatic transformations among representations. This raises a key challenge, which is how to represent an interaction from one tool, if there is no corresponding representation in the other tool. For example, the source code may not provide a dynamic representation of a particular interaction in the design tool. Further work should focus on finding ways to connect designer-friendly views, e.g., timelines, videos, diagrams, to the corresponding source code.

Refactoring the user interface

Some graphical tools include "symbols" or "smart objects" that are referenced across documents instead of being copied. Designers can modify these smart objects and see the changes reflected wherever they are used. While this encourages modularity it also requires planning: "smart objects" must be created before being used. However, the fluid nature of design can make pre-planning difficult: new ideas or constraints may appear and clients often change their minds.

We argue that interaction design tools should support *refactoring*: "The process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure" [12]. Allowing designers to extract similar visual elements and interaction patterns to create reusable modules or components would reduce the amount of reworking and enhance consistency in the user interface. These new modules could be added to existing design pattern libraries, thus increasing support for novel forms of interaction.

Achieving complete interaction descriptions

Designers and developers in the participatory design workshop started with specific, concrete examples, and then, through multiple iterations, generalized it step by step to create a more abstract representation. This suggests that designers and developers would benefit from formalisms that both provide complete descriptions of interactions, but can also be represented with specific examples.

We argue that interaction design tools should provide simple, high-level interaction abstractions, e.g. swipe or drag&drop, but also allow the designer to manipulate lower-level descriptions when necessary, e.g. mouse up and down. Being able to transition from specific examples to higher-level abstractions would help designers create more complete descriptions of each interaction and help avoid *missing information*.

Exploring design possibilities

Our results suggest that designers are less likely than developers to specify *edge cases*, even though their early identification can avoid significant problems later. We argue that designer's tools should help designers discover edge cases.

First, the tools should show how the design reacts to different-sized data sets, e.g., small, medium and large, within the constraints of the domain (similar to the concept of load testing in software development). Second, the tools should support the simulation of typical situations in different contexts, e.g., mobile vs. stationary, connected vs. disconnected, or bright vs. low light conditions. Consistency checks run under each condition could then highlight potential design problems.

CONCLUSION AND FUTURE WORK

This paper studied how designers communicate and developers interpret interactive system designs. First, we conducted 16 interviews with professional designers and developers. We showed that the current workflow induces unnecessary rework. Designers create a multitude of redundant design documents and developers must recreate them with their own tools. This process often introduces mismatches with the original design.

We identified three key *design breakdowns*: *missing information*, when designers do not communicate a specific detail; *edge cases*, when designers do not think about a particular case; and *technical constraints*, when designers are not aware of developer's technical limitations. The interviews also showed that when developers are not involved in the initial design of custom interactions, the implementation tends to be problematic or even impossible.

To further understand how designers and developers address these breakdowns, we conducted a longitudinal case study. We found that even if the early involvement of the developer mitigated the occurrences of *design breakdowns*, new ones appeared in subsequent meetings.

We then ran a participatory design workshop to study the role played by the representations of interactions in breakdowns. We observed that the limitations of the representations used to communicate interaction result in missing information and edge cases. These results suggest that communicating and representing interactions require an iterative process, from individual concrete examples to rule-based representations.

Finally, we presented four strategies for creating better collaboration tools that take *design breakdowns* into account and reduce reworking: synchronized views, user interface refactoring, complete interaction descriptions and support for design exploration. We plan to explore these approaches by designing new tools that help bridge the gap between designers and developers.

ACKNOWLEDGMENTS

This research was supported by European Research Council grant n° 321135 CREATIV: Creating Co-Adaptive Human-Computer Partnerships. Our thanks to all the participants for their rich insights and the fruitful discussions. We also thank the ex)**situ** team and especially Carla Griggio for her thoughtful comments and suggestions.

REFERENCES

1. Brian P Bailey, Joseph A Konstan, John, and John V Carlis. 2001. Supporting Multimedia Designers: Towards More Effective Design Tools. In *Proc. Multimedia Modeling: Modeling Mutlimedia Information and Systems (MMM2001)*. 267–286.
2. Michel Beaudouin-Lafon and Wendy E Mackay. 2003. Prototyping tools and techniques. In *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*. 1017–1039.
3. Jan Borchers. 2001. *A Pattern Approach to Interaction Design*. John Wiley & Sons, Inc.
4. Judith M. Brown, Gitte Lindgaard, and Robert Biddle. 2011. Collaborative Events and Shared Artefacts: Agile Interaction Designers and Developers Working Toward Common Aims. In *2011 AGILE Conference*. IEEE, 87–96.
5. Judith M. Brown, Gitte Lindgaard, and Robert Biddle. 2012. Joint implicit alignment work of interaction designers and software developers. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction Making Sense Through Design - NordiCHI '12*. ACM Press, New York, New York, USA, 693.
6. Alan Cooper, Robert Reimann, and David Cronin. 2007. *About Face 3: The Essentials of Interaction Design*. Wiley. 610 pages.
7. Paulo Pinheiro da Silva and Norman W. Paton. 2000. *UMLi: the unified modeling language for interactive applications*. Ph.D. Dissertation. University of Manchester.
8. Tiago da Silva, Milene Selbach Silveira, and Frank Maurer. 2013. Ten Lessons Learned from Integrating Interaction Design and Agile Development. In *2013 Agile Conference*. IEEE, 42–49.
9. Thomas Erickson. 2000. Lingua Francas for design. In *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. ACM Press, New York, New York, USA, 357–368.
10. Jennifer Ferreira, Helen Sharp, and Hugh Robinson. 2011. User Experience Design and Agile Development: Managing Cooperation Through Articulation Work. *Softw. Pract. Exper.* 41, 9 (Aug. 2011), 963–974.
11. Adam Fouse, Nadir Weibel, Edwin Hutchins, and James D. Hollan. 2011. ChronoViz: a system for supporting navigation of time-coded data. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*. ACM Press, New York, New York, USA, 299.
12. Martin Fowler and Kent Beck. 1999. *Refactoring: Improving the Design of Existing Code*. 431 pages.
13. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.
14. Ghita Jalal, Nolwenn Maudet, and Wendy E Mackay. 2015. Color Portraits: From Color Picking to Interacting with Color. In *Proceedings of the ACM CHI'15 Conference on Human Factors in Computing Systems*, Vol. 1. 4207–4216.
15. James A. Landay and Brad A. Myers. 1995. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*. ACM Press, New York, New York, USA, 43–50.
16. Charlotte P. Lee. 2007. Boundary Negotiating Artifacts: Unbinding the Routine of Boundary Objects and Embracing Chaos in Collaborative Work. *Computer Supported Cooperative Work (CSCW)* 16, 3 (apr 2007), 307–339.
17. James Lin, Mark W. Newman, Jason I. Hong, and James A. Landay. 2000. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '00*. ACM Press, New York, New York, USA, 510–517.
18. Jonas Löwgren. 1995. Applying Design Methodology to Software Development. In *Proceedings of the 1st Conference on Designing Interactive Systems: Processes, Practices, Methods, & Techniques (DIS '95)*. ACM, New York, NY, USA, 87–95.
19. Wendy Mackay. 2002. Using video to support interaction design, In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. *DVD Tutorial, Inria and ACM/SIGCHI* (2002).
20. Wendy E. Mackay and Anne-Laure Fayard. 1997. HCI, natural science and design: a framework for triangulation across disciplines. In *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '97*. ACM Press, New York, New York, USA, 223–234.
21. Jack Moffett. 2014. *Bridging UX and Web Development: Better Results through Team Integration*. Elsevier Science. 224 pages.
22. Brad Myers. 1995. User interface software tools. *ACM Transactions on Computer-Human Interaction* 2, 1 (mar 1995), 64–103.
23. Brad Myers, Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew Ko. 2008. How designers design and program interactive behaviors. *Proceedings - 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008* (2008), 177–184.
24. Mark W. Newman and James A. Landay. 2000. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. ACM Press, New York, New York, USA, 263–274.

25. Donald A. Norman and Stephen W. Draper. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.
26. Fatih Kursat Ozenc, Miso Kim, John Zimmerman, Stephen Oney, and Brad Myers. 2010. How to support designers in getting hold of the immaterial material of software. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, New York, New York, USA, 2513.
27. Sun Young Park, Brad Myers, and Andrew J. Ko. 2008. Designers' natural descriptions of interactive behaviors. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 185–188.
28. Steven E. Poltrock and Jonathan Grudin. 1994. Organizational Obstacles to Interface Design and Development: Two Participant-observer Studies. *ACM Trans. Comput.-Hum. Interact.* 1, 1 (March 1994), 52–80.
29. Dina Salah, Richard F. Paige, and Paul Cairns. 2014. A systematic literature review for agile development processes and user centred design integration. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. ACM Press, New York, New York, USA, 1–10.
30. Susan Leigh Star and James R. Griesemer. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science* 19, 3 (1989), 387–420.
31. Anselm Strauss. 1988. The Articulation of Project Work: An Organizational Process. *The Sociological Quarterly* 29, 2 (1988), 163–178.
32. Anselm Strauss and Juliet M. Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications. 312 pages.
33. Khoi Vinh. 2015. Design Tools Survey | The Tools Designers Are Using Today. (2015). `http://tools.subtraction.com/index.html`
34. Meike Wiemann. 2016. *Patterns as a tool for collaboration: A case study of collaboration between designers and developers through user interface pattern libraries.* Ph.D. Dissertation. Universitet Umeå.