Sketching and Conceptions of Software Design

David Socha
Computing and Software Systems
University of Washington Bothell
Bothell, USA
socha@uw.edu

Josh Tenenberg
Institute of Technology
University of Washington Tacoma
Tacoma, USA
jtenenbg@uw.edu

***Abstract*—In this paper, we describe a study of sketching and design within a software organization in which hundreds of hours of video of development activity in situ were captured and analyzed. We use the study as a basis from which to question how researcher conceptions of software design—what it is, when and where it occurs, and how it is accounted—affect the way in which design is empirically studied. When researcher conceptions of design substantially differ from the actual design practices of those who are studied, researchers are at risk of seeing only what they are looking for and in this way miss the very design practices carried out by software developers in their quotidian work that the researchers were hoping to characterize.**

***Index Terms*—Software design, sketching, UML, inscriptions**

I. Introduction

When we research software design, what is the object of our study? We ask this question in the most literal sense: what is it that we actually examine? What artifacts and/or behavior, in what settings? For some, these questions seem self-answering: if one wants to study software design, then of course, one studies software design.

As obvious and tautalogous as this question might appear, we ask it in order to make visible the presuppositions that researchers have about design. The reason why it matters is that as researchers, these presuppositions that we have about what design *is* may prefigure the results that we obtain in our empirical investigations. In short, we may only find what we are looking for and in this way miss the very design practices carried out by software developers in their quotidian work that we were hoping to characterize. This is in fact, what we came to discover about our own preconceptions related to the empirical study of design that we undertook in a software development organization.

We undertook this study with the purpose of understanding software design practice [1]. We were interested in characterizing software design in practice, particularly the ways in which software developers use sketches and diagrams that represent computational behavior and/or structure, a focus of considerable interest in the software community [2]–[5].

Imagine our surprise on entering the field and finding that there was little sketching activity occurring in the software development organization we were studying! If they weren't sketching, what were they doing? And where and how did design happen if they were doing little sketching? We changed the focus of our data collection and analysis, turning from the sketchpad and whiteboard to the *pairing stations*—the computers where software developers did most of their work as pairs sitting side by side. Collecting hundreds of hours of video of these sessions over several months, along with videos of the standup meetings that preceded pairing sessions, videos of other meetings, hundreds of photographs, and dozens of hours of observation and ethnographic interviews, we set out to investigate how these software developers produced their work. Whereas we had originally wondered where all of the design activity was occurring if there was little sketching and diagramming, only after immersion in this data for many months did we begin to see what had been hiding in plain sight the entire time: design was happening everywhere, all the time. Our very conceptions of design had prevented us from seeing the design that was there.

This paper, then, is a cautionary tale about how as researchers our conceptions of software development are embedded within every aspect of our empirical studies, and how we can begin to overcome these conceptions, not only to learn more about how software is actually constructed, but to challenge the very way in which we conceptualize the enterprise.

II. Our Presuppositions

In a prior paper, we proposed a research study to investigate how groups of professional software developers create and use diagrams and diagramming in their authentic work, i.e. "in the wild"' [1]. Drawing from theories of situated and distributed cognition, we conjectured that the design activities of professionals in situ would differ substantially from the work reported in studies based upon self-reports of students or of professionals working in laboratory settings. Our plan was to instrument a particular location in the organization under study at which we assumed sketching "normally" occurs with video cameras so as to capture not only the "what" of sketching but also the "how." And immediately after these sketching sessions we would interview the participants, mediated by playback of the video recordings, to capture the "why" of these sketching sessions. We believed that fine-grained analysis of audio-visual recordings of in situ work would illuminate important aspects of software design not available in other studies such as the Studying Professional Software Design workshop [4], [6]. As this paper shows, our beliefs about analysing in situ work was correct in practice, but not for the reasons we hypothesized. This paper reports on the results of this study, and reflects on

Accepted for publication at CHASE 2015

the relationship between conceptions of design and how design research is carried out.

III. Organizational Context

Our data was collected at a 9 year-old software development company in the Seattle area that employs approximately 50 people. In 2011, the company was acquired by a non-US parent organization that has continued to let the company operate largely independently. The company's product is a software system that helps friends and family share information. It has over 13 million users, includes a significant backend Software-as-a-Service (SaaS) component, and has both a web-based version and client versions for Macintosh, Windows, iPhone, and iPad.

The founders of this company came from a technical background and designed the company's processes and practices based upon a small set of values and principles intended to address the question of: "How to operate a small software team and make it its best?" Their goal was to optimize for a self-organizing team of 5-14 people, instead of trying to create practices that scale to larger groups. All 50 members of the organization work in a single office with an open floor plan (see Fig. 1).

Since the founding of this organization, the software developers in it have used a mix of extreme programming [7] and Scrum [8] practices, which they continually experiment with and adapt. The developer stations (see Fig. 1), the physical locations at which the software developers work, each consist of a workstation configured to support pair programming. These are concentrated in a central part of the office, so that each developer is able to directly see, hear, and interact with the other developers nearby, thereby providing "radical collocation" [9].

While many agile teams do a daily standup, this organization does three standups (which they refer to as "huddles") per day: one huddle before each of the three daily 2-hour-long pair programming sessions. The intent is to provide shorter feedback cycles among the developers to help them better do their complex work. These huddles are done in the huddle area located between the whiteboard walls and the developer stations. This huddle space was intentionally located directly between the developer stations and a large whiteboard wall (labeled W and N in Fig. 1) in order to facilitate the interplay between the activity at the developer stations, the activity in the huddles, and the mediating artifacts on the whiteboard wall [10].

Whiteboards have played a key role in these practices since the company's founding. In their first location, they often used 2-4 whiteboards to cover a wall. When they moved to their current location, over which they had more control, they painted seven entire walls with whiteboard paint and purchased a half dozen small (3'x4') portable whiteboards. These floor-to-ceiling whiteboard walls and the smaller portable whiteboards have become increasingly intertwined with the company's development practices, and are used for a wide variety of purposes, including sketching and diagramming by the software developers.

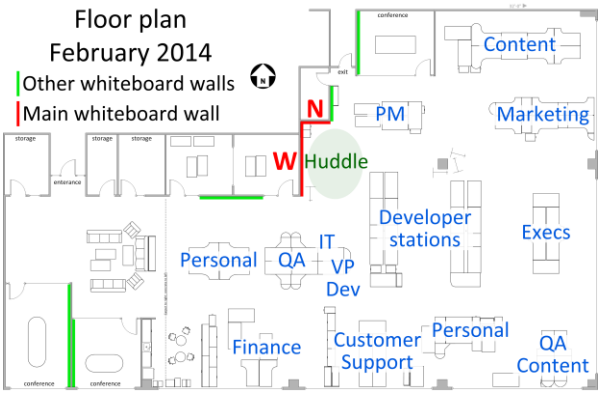

Fig. 1: Office floor plan

IV. Data Collection

Data collection began on October 17, 2012 as the organization's VP of Engineering led the first author on an initial tour of the site to better understand the setting and organizational context in order to plan how to collect audio-video recordings of sketching sessions. While the tour revealed many whiteboard surfaces in this organization, the VP indicated that developers only sketched on whiteboards a few times per week across the entire team. Furthermore, those sketching sessions were rarely planned in advance, and were usually in a conference room or on a portable whiteboard that they wheeled to a convenient location for a discussion, such as the couch area near the main entrance. And the sketching sessions often were brief, lasting anywhere from a few minutes to an hour or so. In other words, to record sketching sessions we would have to be on site continuously and would only get a few sessions per week.

Momentarily destabilized by learning that sketching and diagramming were rare events, we sought instead the "interactional hot-spots," following the advice of Jordan and Henderson [11]. We thus shifted our research gaze from studying "*sketching* in the wild" to "*collaboration* in the wild." This collaboration was primarily located at the developer stations and at the huddle area, and so we focused the bulk of our data collection at these locations.

Based upon this reframe of our research study, the first author iterated on the video collection, progressively improving the recording setup and increasing the amount of data collected. On November 19, 2012 he collected six short videos using a handheld digital SLR, for a total of 54 minutes of video. On November 27, 2012 he collected 6:18 hours of video using two cameras. On January 24, 2013 he used four video cameras to record 6.5 hours of activity at two developer stations.

Then in February 2014, the first author returned with 9 wide-angle GoPro cameras and six audio recorders to collect an extensive dataset over an 11 day period. This dataset includes 380 hours of video from the developer stations, 17 huddles, and other meetings; time-lapse images of the entire room over the 11 day period; screen recordings from some of the developer workstations; and additional photographs. Although most of these cameras and audio recorders were at fixed locations, two of the video cameras were hand-operated, one by the first

author, and another by the software developers themselves, who would bring the camera with them as they changed location.

V. Data Analysis

Having abandoned sketching as a research focus, we carried out analyses of the multiple ways in which this organization uses the whiteboard to structure its work, as well as the ways in which developer pairs create and sustain awareness while working side by side, reported elsewhere [10], [12]. It was only in doing those analyses, however, that we began to notice small amounts of sketching activity at the whiteboard. We realized that, given our extensive data collection, we had the means available for gaining insight into how much sketching activity was occurring, where it was occurring, and how it was being used within this organization.

In answering "how much," we were immediately confronted with the issue of how to "count" in a way that is "accountable" to our research community. We defined sketching activity as any time a developer was making marks on a surface (whiteboard, paper, iPad), pointing to such surfaces, or orienting their body to these surfaces while in conversation. Although counting even such things as bullet lists written on the whiteboard (a frequent occurrence during huddles) risks overcounting the amount of sketching, we did not want to rule out this activity, since some researchers have considered it to be sketching (e.g. "to do lists" are taken as sketching in [13]). We did not, however, count any of the time that individuals or pairs worked on the computer with keyboard and mouse, since we were told that the developers never used sketching/diagramming software, nor did we ever observe such use. What this means is that there might have been instances in which we counted particular activity as sketching, such as writing a list on the whiteboard, when a similar list typed at the keyboard of the computer would not have been counted.

To assess the amount of sketching activity done by this group of software developers, we focused on the video from two of the 11 days of data: February 19 and 21, 2014. These two days contained most of the observed sketching activity, including several sketching sessions at the huddle area, at the couches, and in a conference room. Thus, the results reported here may again overestimate the amount of sketching done in this organization.

In order to more quickly browse the dozens of video files to identify sketching, we created "thumbnail" images composed of one frame from every 5 seconds of each video (see Fig. 2 for a fragment of one of our thumbnail files). In the example provided, one can see the point at which a green-shirted software developer sits at the developer station. We were able to quickly scroll through these files of thumbnails, exploiting our (human) visual processing to identify those points when sketching might be occurring. We could then examine these places more carefully (both in the thumbnail files and in the associated video files) to develop our counts.

Sketching activity observed over a contiguous sequence of frames was accounted for as the duration between the first and last of those frames. Sketching activity observed in a single isolated frame was accounted for as 5 seconds of sketching activity. Using the thumbnails may miss some sketching activity, and in fact viewing one of the videos did reveal several momentary sketching events that were not observed in that video's thumbnails.

To determine the proportion of time spent in sketching activity, we similarly accounted for the amount of "active" video: video in which people were at the video's location, e.g., at the pairing station shown in Fig. 2. Only about half of the video was "active", since the developers were doing their work with no instructions from us about where to work and there were more developer workstations than pairs of developers.

Table 1 shows the duration of "active" time and sketching activity for these two days, categorized by 5 loci of work: during huddles, during post-huddle discussions, at pairing stations, at couches, and in conference rooms.

The "location %" column of sketching activity shows the percent of that location's active time during which sketching was observed. The "total %" column shows this location's sketching activity as percent of the total time across all locations. During the 39:45 "active" hours, we identified 6:02 hours (15%) of sketching.

Fifteen percent of the time may appear to be a considerable percentage devoted to sketching. But a finer-grained analysis reveals a different story. First, and importantly, the types of sketching differed by location. Huddles consisted of 5% of this video, and included sketching 24% of the time. This sketching activity consisted almost exclusively of a bookkeeping activity in which developers used the "parking lot" section of the whiteboard wall to inscribe brief notes of topics to discuss at the end of the huddle [10]. This is not the type of sketching activity commonly associated with "software design".

Post-huddle discussions comprise 7% of this video, and 6% of the total sketching time. Most of this time (82%) involved sketching on the whiteboard during design discussions.

Fig. 2: Portion of thumbnail of video

Table 1: Amount of "active" time and sketching activity on Feb 19 and 21

	"Active"		Sketching activity		
Location	**hh:mm**	**%**	**hh:mm**	**location %**	**total %**
Huddle	02:02	5	00:29	24	1
Post huddle	02:58	7	02:26	82	6
Pairing station	31:08	78	00:22	1	1
Couches	00:39	2	00:39	100	2
Conference room	02:59	8	02:05	70	5
TOTAL	**39:45**	**100**	**06:02**	**15**	**15**

Pairing stations consisted of 78% of the video time analyzed and 1% of the total sketching time. Most of this sketching, however, was a 19-minute session during which a solo developer worked on his own "Personal Shield", part of a team building activity. The remainder of the sketching time was four sessions in which a developer briefly wrote on a sketchpad, for a total of 2:54 minutes. Thus, only 2:54 minutes (0.16%) of the 31:08 hours of pairing station video had sketching that might have been related to software design.

The video from the couches consisted of 2% (39 minutes) of the video that was analyzed. It was from a single design discussion between two developers sitting in separate couches. During this time, one of the developers was continuously making inscriptions on his iPad, which was visible only to him. In addition to the sketching, both developers made extensive use of "air sketches" enacted via hand and arm gestures. Our definition of "sketching" is restricted to making or referring to inscriptions or marks on a physical surface, and thus did not include this "air sketching".

Conference rooms consisted of 8% (2:59 hours) of the video, 70% of which involved making inscriptions or referring to inscriptions. This was the continuation of the design discussion between the two developers that had begun at the couches. During this time, the two developers spent only a few minutes actually making marks on the whiteboard. The rest of the sketching time was referring to these inscriptions, or one of the developers making marks on his iPad, which, once again, were not visible to the other developer, except for one brief moment. The whiteboard sketch that was drawn, from the most extensive sketching session during the two days analysed, was also remarkably simple, shown in Fig. 3.

At the same time, in analysing this conference room episode in more detail, we came to see that it complicates prior notions of sketching and accounting for the duration and uses of sketches. "Sketching" involves not only making marks on a surface, since this only accounted for a small percentage of the time taken during this session. The software developers also spent considerable time *augmenting* the sketch with words, gestures, and deictics. They also spent time during this session oriented to the sketch, but rather than augmenting what was drawn, they spoke about *alternatives in relation to* what they had sketched. And finally, there was a considerable part of this session, in which the developers sat nearby the sketch, but never oriented their bodies toward it nor made any verbal reference to it. Which of these activities, then, is sketching? In counting all of it, we drew an analytical boundary that is somewhat artificial, a point we return to in the Discussion.

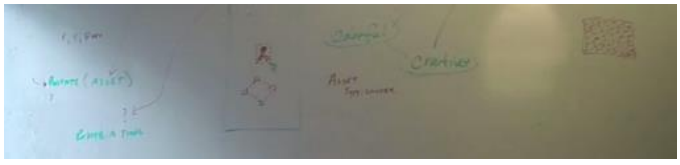

Fig. 3: Sketching from longest sketching session of Feb 19 and 21. The two ovals and lines in the top middle were there before this session began.

VI. Discussion

What conceptions of design are made visible in the study reported above? When we speak of "conceptions," we do not only mean those explicit ideas "in mind." We mean as well the inchoate, enacted conceptions as embedded in the way in which we carried out the study and the participants carry out their everyday work. In the earliest published description of our research, we discuss not design, but "the situated use of sketches and diagrams by expert software practitioners in their everyday activities in the workplace" [1]. We presupposed that these sketches were created as part of a design process, and hence there was no need to explicitly link "sketch" to "design." What we did not assume, and what was the very object of our research study, was the specific nature of the semiotic marks that software developers make on media such as whiteboards. Were they UML? Were they boxes and arrows? Were they bullet lists? Were they something else entirely? In addition, we did not assume that the meaning was "in" the sketches created, but rather, following Roth, that sketches "in everyday settings … become apparently fused to the things or contexts that they describe. … The graph is relevant together with the world [that the creator] inhabits together with other people and objects that surround them" [14]. Thus, in going to the workplace to observe and record software development "in the wild," we hoped to overcome what we saw as limitations in the research on software design in which software designers were studied in contrived (laboratory) settings.

In wanting to instrument a conference room or specific place within the organization that we studied, however, we presupposed that design *qua* sketching happened primarily (only?) in a special place at regular times. We assumed, as do Baltes and Diehl [13, p. 530], that "[s]ketches and diagrams play an important role in the daily work of software developers." Thus, capturing this activity should be unproblematic: we simply go to the special place of daily sketching activity and turn on our video recorders; what could be simpler? As we recount above, however, our research participants quickly informed us that, though, yes, they occasionally do some sketching, that it was not where the action was. The *interactional hot spots*, "sites of activity for which videotaping promises to be productive" [11, p. 43], were at the pairing stations and huddle area, and so this is where we focused our attention. As a result, sketching dropped from our analytic gaze.

Several months later, when given the opportunity to return for further data collection, we "over sampled," by placing cameras and microphones in as many places as our participants would allow and for which we had resources. Our focus was again on the pairing stations, but we captured data in many other locations: the huddle area, conference rooms, the seating area. It was only on seeing the most fragmentary glimpses of sketching, bits of ephemera that disappeared almost as quickly as they erupted—a box hastily drawn on the whiteboard wall by a pair of software developers after a huddle, a list jotted on a notepad by a pair of developers at a pairing station—that sketching re-emerged for us as an object of study. For what these small glimpses made salient was the almost complete

absence of sketching. Was this really the case? And if so, then what did it mean for how these software developers *do* design?

As the above analysis indicates, all evidence suggests that sketching, of the sort that involves anything more elaborate than a bullet list or a couple of boxes and arrows, is a rare occurrence at this organization. But this does not mean that this organization does not do *design* as far as they conceive it. Design and sketching are not equivalent, nor does one imply the other. The developers at this organization deliberately structure their work process and division of labor so that all developers work at all levels of detail; there is no division of labor between "designers" and "coders," each of whom specializes in a different level of detail. As one of the principals in the organization elaborates in response to a question about doing "design" in a traditional, UML-style fashion: "we were an agile shop. And we didn't want to work that way because we didn't think it was productive. And so yeah, that's a very sort of waterfall-style approach. The architect sits on high, figures everything out beforehand, maybe doing sketches or who knows what, and then passes the design off. But we didn't do that." This organization did not do that because what this "waterfall-style" division of labor implies is that there are distinct "phases" of software development in which design happens at a particular time by particular people, resulting in an explicitly represented design artifact whose meaning is discernible to someone else charged with writing code consistent with it.

Rather, the software developers whom we studied viewed design as distinctly and deliberately not limited to a particular phase or particular people, a bounded temporal event within a predefined software lifecycle. Rather, the developers saw themselves as doing design work continuously and everywhere. "So the thing is the way I do it – the only way I think that's reasonable to do it is to go constantly back and forth, try not to figure out all of the design beforehand, but only try to figure out some of it, like maybe even just think of some of it in your head basically, and then go and start typing, right? And then refine it and continue back and forth and back and forth and back and forth, so almost constantly."

One interpretation of our data, then, is that although there was little sketching of the kind described in most prior studies of software sketching and design, there was continuous *designing*. The hours and hours in which pairs sit together at the pairing stations looking at the code were never simply "implementation" as distinct from "design," but were a constant back-and-forth between coding and designing.

Designing was always there, a constant presence, and yet (until recently) invisible to us in its ubiquity. When the paucity of sketching became noticeable to us, we began to reconsider the conceptions of software design we had tacitly embedded within our research design, an assumption of a near-equivalence between sketching and designing; designing happening in particular places at particular times by particular people. Only as a result of extensive video capture of the development activity and subsequent analysis did we begin to see how our initial research design embedded a tacit assumption of design that carried with it vestiges of a waterfall model that we ourselves had long ago abandoned. In looking at the pairing stations, the huddle areas, and the larger patterns of interaction within the developer space, this extensive video capture allowed us to trace the ways in which this organization structures its development activities. As a result, we were able to see the enactment of Agile practices of continuous and iterative design through the deliberate structuring of this organization's software development labor and work processes.

A. Researcher Conceptions Of Design

What then, are the implications of this study on the current research discourse concerning software design and sketching? In particular, how do *researcher conceptions of design* figure into the ways in which empirical researchers structure their studies of design?

The study of sketching and diagramming by software developers has received considerable interest by empirical researchers over the last decade. In some of this research, the connection between design and sketching is explicit, justified by software development being characterized as a design discipline, and as such it follows that sketches are important. For instance, Cherubini et al. begin their research report concerning how and why software developers use sketches with: "Diagrams are important tools in every design and engineering discipline" [5, p. 557]. Similarly, Walny et al begin a report of their study on sketching in software development with: "Visualization through sketching and diagramming plays an important role in the design process in various domains, including architecture, design, and engineering" [15, p. 1]. For others, sketches are necessarily used in software design because of the complexity of the relations between the computational units: "Software design is a highly visual activity, where diagrams are used for brainstorming, grounding, and communicating ideas and decisions [6]. This is particularly true for the object-oriented (OO) paradigm, which involves large numbers of entities and complex relations between them" [16, p. 261]. Others use the term "modeling" instead of design: "This empirical study complements and resonates with other studies of UML use in industry, finding (as others do) that practitioners take a broad view of what constitutes 'modeling'" [2, p. 11]. And for some researchers, the connection between sketching and design is implied: "Over the past years, studies have shown the importance of sketches and diagrams in software development" [13, p. 530].

What all of these conceptions of the relationship between sketching and software development have in common is the relative equivalence of design activity and sketching, that one implies the other. The following syllogism thus captures the essential argument that these researchers make: 1) (all) design disciplines use sketches as essential representations for design, 2) software development is a design discipline, therefore 3) software developers use sketches as essential representations for design. If we study sketching, then by virtue of its use in design disciplines in general, we will be studying design in software development. And if we are to study design, then by virtue of the importance of sketching for these disciplines, we will need to study sketching. Design qua sketching is distinctly *not* coding: these are distinct activities, using distinct notations.

The problem with researchers presupposing that design is sketching is design, is that in those organizations who carry out an enactment of Agile design similar to the organization described above, the vast majority of design activity will be overlooked. It will simply not be accounted as design, but will instead be seen as "coding" or "implementation" or "pair programming" if it is considered at all. Take the survey by Baltes and Diehl [13, p. 533], for instance, that asks software developers "When did you create your last sketch or diagram," "How many persons contributed to the sketch/diagram" and similar. If given to software developers from an organization like the one that we studied, such a survey, although perhaps characterizing the few designs that are created, will miss the lion's share of the design activity, at least as far as how the participants themselves construe it.

Design as sketching, as a phase of activity distinct from coding, is sometimes so embedded within a research design that it goes unremarked by researchers. Or, if it is discussed, it is simply to name the phase, for example as *initial* or *early* design activity [4], [6]. For instance, consider the empirical study protocol described by Petre et al. [4] that served as the basis for the NSF-sponsored workshop Studying Professional Software Design in 2010 attended by 54 design researchers and resulting in a special issue of *Design Studies* [4] and *IEEE Software* [6]. This protocol was developed to answer the research question: "What do software designers do when they design software" [4, p. 536]? Video recordings were analyzed of three pairs of software developers who were given a design prompt, and "each pair … worked together at a whiteboard for two hours" [6, p. 29]. "Furthermore, it asked that the designers consider how to model the software system, as well as how users would interact with the system" [4, p. 536]. The design prompt itself explicitly states that "[t]he result of this session should be: *the ability to present your design to a team of software developers who will be tasked with actually implementing it* [emphasis in original]" [4, p. 544].

In placing the participants *at the whiteboard*, the research design naturalizes the whiteboard as a site of design activity, in contrast to, for instance, the pairing stations at which most of the design activity occurred at the organization that we studied. The research setting itself then, in the very way in which the researchers were physically arranged and given particular materials, presupposes design as a sketching activity, or at least an activity in which inscriptions are to be recorded on the whiteboard. Further, by explicitly telling the research participants to *model a system that will be implemented by others*, the researchers are presupposing a conception of design in which a "design" is a model that is handed off in toto, "thrown over the wall" from one group of software developers to another. Design is a distinct phase, with a clear beginning and end, which takes place at the whiteboard, and results in a design representation (a "model") that is then used to guide the coding phase. In waterfall fashion, labor is divided by specialization and the software process is organized as a set of phases in an assembly-line fashion. This is not the conception of design that we observed. At the organization that we studied, the designers are the implementers; there is no distinction between designers and developers. One result is that the details brought into the design discussions span multiple levels of abstraction from objects in the existing codebase to possible alternative architectural designs to experiences from using competitors' systems to ideas for creating strategic competitive business advantages. They range from existing implementation details to business design–all based upon years of shared history. And designs do not have to be externally represented for anyone but the software developers themselves to use; they are not handed off.

The other complexity that lab-based and similar research protocols gloss concerns the boundary between design and non-design activity. For if design is a distinct phase, when does it begin and end? In the activity that we studied, there are no researchers external to the setting designating an arbitrary start and end time for the activities observed within the setting. Rather, the software developers simply go about their work. When they carry out their inscriptional activity, there is no one there to mark a boundary. Before they move to a whiteboard, they might be talking about a particular problem or concern, and at some point, the pair decides to move to the whiteboard. They make a few marks on the board, continue talking, make more marks, talk some more. Some of the marks look box-like, some are labels, some are lines and arcs. They move away from the board, continue talking about the inscriptions, making a hand-shape that gesturally mirrors an inscription on the board in order to index the discussion that occurred in and around when that inscription was written. If design is everywhere and all the time, then such activity is non-problematic, for boundaries do not need to be precisely determined in order to do such things as determine the ratio of time spent in one activity as compared to another. But if design is conceived as an activity distinct from coding, from planning, from determining requirements, then what are the boundaries between these activities? When does "sketching" or "design" activity begin and when does it end? Is it only at those moments when marks are made on the board? When a box or arrow or freehand drawing is made but not the labels? When the participants are at the location at which the sketch is made but not when they move away from it?

These definitional questions are not simply academic, since for research designs involving surveys and interviews, the very interpretation of the questions asked are determined each time anew by the respondent. And unfortunately, such interpretations are invisible to the researcher. For instance, in a survey study by Baltes et al. [13, p. 533] the respondents were asked "When did you create your last sketch or diagram" and "How much effective work time went into the creation and revision of the sketch/diagram up to now?" Does a "to do" list count as a "sketch"? And if so, how much time is to be accounted as "work time" in its "creation" and "revision"? Such matters of interpretation are left to the respondents to construe in their own way, with responses aggregated together using standard statistical analysis methods. How then are we as researchers to construe these aggregated results? How then have the respondents made these distinctions? And to what range of settings do these results apply?

These problems of interpretation and construal do not disappear, however, even if the researchers are the only ones making these determinations. In the study presented above, in order to make any claims about the amount of time in which design sketches and diagrams are used within the organization we studied, we had to make a number of choices for purposes of accounting. What inscriptions, in what media, do we count as a sketch or diagram? When does the sketching and diagramming activity begin and end? Although we document these choices above, providing our rationale for our accounting scheme, there are nonetheless several somewhat arbitrary choices that we have made purely for purposes of drawing sharp analytic boundaries. To some extent, we draw these boundaries, recognizing their artificiality, in service of an argument in which we abandon these very boundaries.

VII. Conclusion

There is no neutral way to study software design; empirical researchers always take a position. Not only are researchers "located" in a physical space with the individuals whom they study, they are similarly located within a conceptual space shared with a research community. How researchers conceptualize their objects of inquiry determine the questions they ask, the methods they use in answering them, and the interpretations that they give to the data they collect. It is into this conceptual space, the space in which our very notions of what software design *is*, that we have placed this paper. The story of the particular study of software design that we undertook within a software organization might be titled "Sin and Redemption." Our sin, venial and perhaps unavoidable, was to have a particular conception of software design around which we planned our data collection, which turned out to be inconsistent with the design practices of the developers whom we were studying. One can hardly enter the field without any preconceptions about what one intends to study. Our redemption was first in going to the field at all (rather than working in the lab), in watching and listening to the software developers to find their "interactional hot-spots," and "over sampling" so that we had extensive data across the organization over time and space. It was only in noticing the small amount of non-trivial sketching that we recognized how our original plan for data collection embedded a "waterfall-style" conception of design. In this conception, design is viewed as a particular phase, neatly delineated from other development activities such as coding, testing, and requirements gathering, a conception that conflicted with the enacted practices of the software developers under scrutiny.

If research on software development is going to provide deeper insights into how software development is and could be practiced, then it is important that we not only seek what we hope to find. We must also look beyond the narrow compass of our own preconceptions to see the conceptions and practices of those whom we study.

Acknowledgment

We thank our study participants who have been so kind to extend to us the level of trust that is critical for this type of research. This work was partially funded by a 2012-2013 Worthington Distinguished Scholar award, and a UW Bothell CSS Graduate Research award to the first author from the University of Washington, Bothell. Thanks to Natalie Jolly for critical reading and commentary of early drafts.

References

[1] D. Socha and J. Tenenberg, "Sketching software in the wild," in Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), 2013, pp. 1237–1240.

[2] M. Petre, "UML in practice," in 35th International Conference on Software Engineering (ICSE 2013), 2013.

[3] A. Baker and A. van der Hoek, "Ideas, subjects, and cycles as lenses for understanding the software design process," Des. Stud., vol. 31, no. 6, pp. 590–613, Nov. 2010.

[4] M. Petre, A. van der Hoek, and A. Baker, "Editorial," Des. Stud., vol. 31, no. 6, pp. 533–544, Nov. 2010.

[5] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07, 2007, vol. 1, pp. 557–566.

[6] A. Baker, A. van der Hoek, H. Ossher, and M. Petre, "Guest editors' introduction: studying professional software design," IEEE Softw., vol. 29, no. 1, pp. 28–33, Jan. 2012.

[7] K. Beck, Extreme Programming Explained: Embrace Change. Reading, MA: Addison-Wesley Professional, 1999.

[8] K. Schwaber and M. Beedle, Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice Hall, 2002.

[9] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson, "How does radical collocation help a team succeed?," in CSCW'00, 2000, pp. 339–346.

[10] D. Socha, T. Frever, and C. Zhang, "Using a large whiteboard wall to support software development teams," in Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS'15), 2015.

[11] B. B. Jordan and A. Henderson, "Interaction analysis: foundations and practice," J. Learn. Sci., vol. 4, no. 1, pp. 39–103, 1995.

[12] J. Tenenberg, W.-M. Roth, and D. Socha, "From I-awareness to we-awareness in CSCW," Comput. Support. Coop. Work. (in press).

[13] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 530–541.

[14] W.-M. Roth, Toward an Anthropology of Graphing: Semiotic and Activity-Theoretic Perspectives. Kluwer Academic Publishers, 2003.

[15] J. Walny, J. Haber, M. Dork, J. Sillito, and S. Carpendale, "Follow that sketch: lifecycles of diagrams and sketches in software development," in 2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2011, pp. 1–8.

[16] U. Dekel and J. Herbsleb, "Notation and representation in collaborative object-oriented design: an observational study," in OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, 2007, pp. 261–280