How Software Designers Interact with Sketches at the Whiteboard

Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek

Abstract—Whiteboard sketches play a crucial role in software development, helping to support groups of designers in reasoning about a software design problem at hand. However, little is known about these sketches and how they support design 'in the moment', particularly in terms of the relationships among sketches, visual syntactic elements within sketches, and reasoning activities. To address this gap, we analyzed 14 hours of design activity by 8 pairs of professional software designers, manually coding over 4000 events capturing the introduction of visual syntactic elements into sketches, focus transitions between sketches, and reasoning activities. Our findings indicate that sketches serve as a rich medium for supporting design conversations. Designers often use general-purpose notations. Designers introduce new syntactic elements to record aspects of the design, or re-purpose sketches as the design develops. Designers constantly shift focus between sketches, using groups of sketches together that contain complementary information. Finally, sketches play an important role in supporting several types of reasoning activities (mental simulation, review of progress, consideration of alternatives). But these activities often leave no trace and rarely lead to sketch creation. We discuss the implications of these and other findings for the practice of software design at the whiteboard and for the creation of new electronic software design sketching tools.

Index Terms— Interaction styles, systems analysis and design, user-centered design

1 Introduction

Sketches play an important role in any design process. They serve as an extension of a designer's own memory [37], help them reason through complex tasks [54], and support them in picturing and evolving hypothetical ideas and abstract concepts [17]. Since sketches are explicit and externalized representations [28], they assist designers in such tasks as brainstorming [51], evaluating ideas [7], and collaborating with others [8, 58].

Unsurprisingly, sketches play an important role in the software design process as well. A typical software project results in numerous, diverse sketches [8, 41]. Figure 1 presents two representative images taken from two software companies: a whiteboard containing a sketch of a potential solution to a design problem (left) and a whiteboard wall containing the collective sketches from many design meetings (right).

These kinds of whiteboard sketches, and the design processes that they support, have seen an increase in interest from the software engineering research community in the past several years. Building upon work that looks at software designers 'in action' more generally (e.g., [13, 19, 21, 41, 60]), the design work of software designers at the whiteboard has been examined from a range of perspectives, including idea generation [2], design notations [6, 33, 42, 56], decision making [59-60], epistemic uncertainty [4], and collaboration [32].

While these previous studies make important contributions, our understanding of how software design sketches are actually produced, and used, at the whiteboard is still limited. We have a sense of the kinds of sketches software designers produce and, at a high level, the types of activities the sketches support (e.g., understanding code, idea externalization, design review, comparison of alternatives). We do not know, however, precisely how the sketches come about and evolve, nor how they support designers in navigating their design problems.

To address this gap, we conducted the first study to examine how sketches evolve and support reasoning activities on a moment-to-moment basis. Drawing on data from design sessions from 8 pairs of professional software designers, we coded 14 hours of videos for 4238 events across 155 sketches, capturing moment-to-moment data of how designers introduced types of visual syntactic elements, shifted their focus between sketches, and used sketches to support reasoning activities. We then analyzed this data to answer several research questions:

- *Types of Sketches*. What types of sketches do designers create? Do the sketches created vary in relation to the approach to design taken? How syntactically complex are sketches? How do sketches evolve?
- *Focus*. How long do designers focus on individual sketches? How do designers make reference to sketches? How do designers shift their attention between sketches?
- *Reasoning*. How do designers use sketches to under-

- *N. Mangano is the co-founder of Molimur, 27562 Escuna, Mission Viejo, CA 92692. Email: nick@molimur.com*
- *T. D. LaToza, and A. van der Hoek are with the Department of Informatics, Donald Bren School Information and Computer Sciences, University of California, Irvine, Irvine, CA 92697 E-mail: {tlatoza, andre}@ ics.uci.edu*
- *M. Petre is with the Faculty of Mathematics and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom. Email: m.petre@open.ac.uk.*

Figure 1. Two examples of software design sketches on whiteboards.

stand and advance the state of the design at hand? Which types of reasoning activities do the sketches support? Are the outcomes of design discussions always recorded in the sketches supporting these discussions?

The result of our analysis is a rich account of the nature of sketching at the whiteboard, revealing the relationship between sketching and the design process it supports. While we report on many results that help shed light on how software designers work, several findings stand out. First, we found that, while all designers created domain, requirements, and systems sketches, the number of sketches they created depended strongly on the approach to design taken. Second, we found that designers rapidly shifted their focus over small groups of sketches, with each sketch playing an important and distinct role within the design conversation. As a result, designers rarely worked with only a single sketch for more than 30 seconds. Third, we found that sketches served an important role in helping designers discuss alternatives, review their progress, and perform mental simulations. Over 80% of the reasoning activities we identified made use of sketches.

The remainder of this paper is organized as follows: Section 2 presents background material on sketching, both generally and as studied in software development. Section 3 introduces our study design and Section 4 the coding scheme used in our data anlysis. Section 5 presents our observations and results. Sections 6 and 7 discuss our threats to validity and findings, respectively, and section 8 provides a brief conclusion.

2 Background

Designers at work, with their tendency to sketch, have received a great amount of attention over the years. Early work focused on characterizing the activity of design. For non-routine, ill-structured problems, Simon and Newell characterized design as a process leading to a good enough solution to the problem in the context at hand [37, 48]. Some models of design attempted to make design repeatable by delineating the solution space in which alternative solutions are weighed and explored [49]. Other work demonstrated that software design is a learning activity, and that exploring the problem is as important as exploring the solution [11]. Later research found that software designers approach design using an opportunistic strategy, in which designers work with designs different levels of abstraction simultaneously, and dive selectively into detailed design when it suits them [21], while showing a relative absence of alternatives. Another study found that designers explore solutions both vertically, by moving between levels of abstraction, and laterally, by utilizing different strategies to solve the same problem [17].

Other work has examined the role of sketching in the design process and the types of sketches that designers produce. Ferguson categorized the sketches that designers produce as *thinking*, *talking*, and *communication* sketches [15]. Suwa found that sketches serve to help thinking activities by rapidly externalizing thoughts onto paper [52]. Schön described the process of reviewing and thinking over the sketches that a person has created as having a "reflective conversation" with the material [45]. Gero and Suwa found that such reflective conversations over the visual details in one's sketch led to "unexpected discoveries" which stemmed from unintended details when drawn [51]. Goldschmidt called the process of reinterpreting sketches beyond their original intended depiction as "seeing as", and noted that professionals are more adept at the act of reinterpretation [18].

Software design, in particular, has seen a similar exploration of design and sketching. Several studies have found that informal sketched prototypes receive more feedback than formal prototypes [23, 46, 57]. Mynatt demonstrated that office workers can identify chunks of categorized contents in their own writing on the whiteboard [35]. Damm *et al*. found that software developers begin with informal sketches and refine them into more formal notations [12]. Newman *et al*. observed that website developers switch between levels of abstraction in designing websites [38].

Recently, there have been an increasing number of

studies investigating software design. LaToza *et al*. found that professional software developers consider whiteboards and paper the most effective means of designing [29]. Cherubini *et al*. found that whiteboards frequently serve as the medium of choice for working through important design decisions, but also that most drawn sketches are transient and often discarded [8]. Dekel and Herbsleb found that the knowledge contained in sketches is highly dependent on the designer's ability to reconstruct the meaning of sketches, and that notations such as UML serve as idioms and are not followed strictly [14]. Petre found that designers juxtapose sketches and deliberately change formalisms to highlight different aspects of a problem, and use provisionality to allow a dialog with incomplete ideas [41]. Yatani *et al.* found that, while there were important barriers to using diagrams in a distributed development team, diagrams still played an important role in ad-hoc meetings, designing, onboarding, project awareness, and documentation [58].

Recent work examined reasoning and decision-making qualities used over sketches. Ball and Christensen demonstrated that mental simulations allow designers to discuss incomplete ideas and reduce uncertainty [3]. Baker and van der Hoek showed that designers often examine two ideas simultaneously, and rotate between pairs of ideas [1]. Zannier and Maurer found that designers are more critical of designs when evaluating them all at once, and less critical when evaluating ideas serially [60]. Tang *et al.* observed that software designers often make a decision without justifying it and move forward [53]. Christiaans and Almendra noted that software design is more complex than product design, because decisions in software must involve knowledge of the user interface and structure, and thus decisions are chained [9].

Using data from three of the eight design sessions reported in this paper, attendees of the Studying Professional Software Design workshop derived a variety of models of the design process [55]. Petre analyzed sketching at the whiteboard from the perspective of 'cognitive dimensions of notations' and highlighted both the importance of gesture in whiteboard sessions and how, as the *role* of a sketch changed over the course of a design discussion, the *type* of representation evolved [43]. Ossher et al. found that designers used multiple representations of concerns which became increasingly detailed and formal through the design conversations [40]. Nakakoji and Yamamoto found that designers used 'designing' and 'drawing' as synonyms, used drawings to establish shared understanding, used a variety of notations, reappropriated sketches for new purposes, redrew sketches, and sometimes did not write down important concepts [36].

Based on these studies of design, many tools have been produced for supporting design. These tools range from generic whiteboard sketching tools, such as Flatland [34], DENIM [38], Post-Brainstorm [16], Designer's OutPost [26], and Range [25], to tools more closely tied to software design, such as SILK [27], Knight [12], SUMLOW [20], InkKit [10], and Calico [30]. A comprehensive summary is presented by Johnson et al. [24]. Based on an extensive review of the literature, Moody proposes a set of principles for designing software design notations for cognitive effectiveness [33].

Our study builds on this work – focusing specifically on how sketching at the whiteboard supports informal software design – by collecting and analyzing moment-to-moment data on the introduction of new types of visual syntactic elements, focus transitions between sketches, and reasoning activities.

3 METHOD

To investigate software design sketching at the whiteboard, we conducted an observational study of professional software designers. To recruit participants, we approached alumni of UC Irvine with a request for access to professional software designers who excelled at software design. This request led to contacts at a number of different organizations, which, through discussions and follow-up, resulted in recruitment of the eighteen professional software designers who participated in the study. The professionals came from seven different organizations. As design at the whiteboard often occurs in collaborative settings involving multiple designers [8, 30], such as ad-hoc meetings, we organized participants into pairs. Each pair belonged to the same organization (two organizations provided two pairs of designers).

We administered a uniform design prompt to all pairs, videotaped each pair at work during their design session, interviewed the participants afterward, and, once all videos were collected, analyzed the whole set of videos. We excluded one video from the analysis due to periodically obstructed views and poor audio quality, yielding a total of eight pairs that formed the basis of the results presented here.

The task instructions, coding guide, and raw data from our study are publicly available.[1]

Participants. All those recruited were viewed as expert designers by their peers. Designers' expertise ranged across a variety of domains (including health care, document workflow, and photo imaging software) and professional specializations (including system analyst, software architecture, UI development, and software development). All worked directly with software and reported that they employ whiteboards in their work. Members of each pair were familiar with each other, although they did not necessarily work together in their organizations at the time of the study. The professional experience of the participants ranged from 2 years to 26 years. Most designers had a number of years of experience, with an average of 16.5 years. While we were initially surprised that a professional with only 2 years of experience would be viewed as an expert, a subsequent conversation with his peers confirmed that he was. 15 of the participants were

[1] http://sdcl.ics.uci.edu/study-materials-and-data/

Designer	Pair	Exp. (years)	Professional specialization
1	A	26	UI development
2	A	20	Interaction design
3	B	15	Software architecture
4	B	11	Software architecture and UI development
5	C	2	Software architecture
6	C	24	Software architecture
7	D	21	Project manager, software architect
8	D	26	Software architecture
9	E	12	Software developer
10	E	10	System analyst
11	F	8	Software developer
12	F	6	Software developer
13	G	25	Software developer
14	G	25	Software developer
15	H	23	Software developer
16	H	10	Software developer

Table 1. For each designer who participated, the pair to which they belonged, their years of professional experience, and their professional specialization. Pairs C and D belonged to the same organization, as did pairs B and G.

male and 1 was female. Table 1 lists each designer's pair, professional experience, and specialization.

Task. Each pair was provided with a written prompt asking them to design an educational traffic flow simulation program to be used by a professor in a civil engineering course to teach students traffic light patterns. The prompt, two pages in length, described a set of open-ended goals and requirements, including offering students the ability to: (1) create visual maps of roads, (2) specify the behavior of lights at intersections, (3) simulate traffic flow, and (4) change parameters of the simulation (e.g., traffic density). Pairs were asked to produce a design at the level of detail necessary to present "to a group of software developers who will be tasked with implementing it".

Setting. The study was conducted at participants' places of work, with a constrained setup that was replicated as closely as possible at each site. We asked each pair to work at a whiteboard approximately 8ft wide by 4ft tall. Participants were asked not to write on the prompt or other paper in order to capture as much of their work as possible with the particular camera setup we used.

All sessions lasted approximately two and a half hours. This time was divided into four segments:

(1) approximately one hour and fifty minutes for the design activity itself (mean = 1 hour and 47 minutes, min = 1 hour, max = 2 hours and 4 minutes),
(2) a ten minute break,
(3) ten minutes for a summary, and
(4) a twenty minute semi-structured exit interview.

At the start of the first segment, participants were given the design prompt and asked to work on a design that addressed this prompt. After the break, they had ten minutes to briefly recap, explain, and motivate their design. In the exit interview, we asked participants to summarize how they went about their design, what their professional background was, and whether they felt the design process they followed was different from how they normally proceeded and, if so, how (see Section 6 for a discussion of these differences and how they affect the validity of our results).

All sessions were video recorded, with one camera directed at the whiteboard and one positioned to capture a broader view of the participants. Three of the videos are available by request[2]. Our analysis focused primarily on the design activity portion of the videos. A total of 14.26 hours of design at the whiteboard was studied and dissected.

4 Coding Scheme

We created three coding schemes focusing on three aspects of design at the whiteboard:

(1) What types of sketches and visual syntactic elements do software designers use during design at the whiteboard?
(2) How do software designers focus on and transition among the sketches they produce?
(3) What reasoning activities do software designers use their sketches to support?

Each coding scheme captured *events* that happened and identified them as actions or activities. Some events are self-contained *actions* that happen within a moment (e.g., proposing a design alternative). Others describe *activities* that occur over a period of time (e.g., a review of progress). Hence activities are delineated by pairs of events, identifying the beginning and end of the activity. All events are labeled with information characterizing the event and with appropriate timestamps.

The coding schemes were developed using an iterative, inductive approach, comparable to open coding [50]. First, we chose three sessions that we viewed as representative of the eight sessions. We then examined portions of these sessions to determine how to segment the sketches and activities meaningfully, as well as to identify the relevant concepts and categories that emerged. We operationalized this into coding schemes, articulating the coding criteria in detail. After several iterations, we finalized the coding schemes. Two authors then independently coded a fourth session, yielding a Cohen's kappa inter-rater agreement of .7 and percentage agreements of

[2] http://www.ics.uci.edu/design-workshop/videos.html

Sketches

sketch type – derived from (1) graphical symbols and compositional rules and (2) designers' verbal references.
e.g., map, list, table, GUI, ER diagram

sketch domain – aspect of the design modeled
requirements – requirements repeating or extending the requirements provided in the prompt
domain – modeling of the problem described in the prompt
system – the architecture or implementation of the software system
user interface – the interface of the systems as the user sees and interacts with it

visual syntactic element types – derived from visual elements and compositional rules
e.g., arrows, text labels, title, rectangle, relationship

Focus fixations and transitions

type of momentary reference – focus on a sketch for than 3 seconds in which no edits were made
quick glance – gazing at a sketch without pointing
point – with finger
split focus - pointing and glancing at different sketches

relationship between origin & destination sketch
abstraction – sketches providing an overview or more detailed view of a portion of a sketch
alternative – sketches providing competing solutions
point-in-time – sketches depicting changes over time, phases, or conditions

Reasoning activities

type – cognitive processes evident through speech, gaze, or gesture with a specific purpose or goal
mental simulation – talking through a mental model to execute and advance to a new state
review of progress – review of the design to take stock of what has been done
suggested design alternative – presentation of a solution that competes with or replaces an existing solution

use of sketches
no sketches used – designers did not gaze at or physically reference the whiteboard
used existing sketches – gestured over existing sketches without drawing
edited existing sketches – drew on, annotated, or edited existing sketches
created new sketch

Figure 2. An overview of the coding schemes, listing the three aspects of the design process analyzed and the dimensions (italics) and possible values for each dimension.

80.0%, 82.4%, and 78.6% for the sketches, focus fixations and transitions, and reasoning activity categories, respectively. This indicates an acceptable inter-rater reliability in an analysis of 13% (1) of the design sessions. The remaining seven sessions were then coded by a single author.

Figure 2 summarizes the coding schemes. In the remainder of this section, we describe each coding scheme in turn.

4.1 Sketches

Observations of designers have found that, rather than viewing whiteboard content as a single entity, people segment content on the whiteboard into distinct *sketches* [34]. We define a sketch, for our purposes, as a segment of the whiteboard containing content:

(1) related to a single, central topic (an "invariate" [5]), with
(2) similar visual features,
(3) proximity,
(4) continuity, and
(5) a similar initial creation time.

Sketches may be primarily textual (e.g., a list), primarily pictorial (e.g., a drawing), or contain elements of both (e.g., a diagram).

The first coding scheme examined the sketches designers created, identifying events in which sketches were created or erased or new visual syntactic elements (see below for a precise definition) were added. To segment whiteboard content into sketches, we performed two passes. In the first pass, we examined snapshots of the video at 5-minute intervals, examining if new sketches had been introduced. Each new sketch that appeared in

the snapshots was numbered from left-to-right, top-to-bottom. In the second pass, we added to and revised our set of sketches for each session by watching the videos, and identifying sketches that shared a central topic in conversation and visual focus by the designers. For example, a sketch consisting of a user interface for creating maps was revised into two different sketches when we observed that the designers divided their attention between a map and its input panel (center left of Figure 3).

Sketch type. We labeled each sketch with a *type* based on two factors:

(1) the set of graphical symbols and compositional rules characterizing the sketch, and
(2) designers' verbal references to the sketch.

The first factor draws heavily on Moody's definition of "visual syntax" [33], which he defines as the combination of visual vocabulary (graphical symbols) and visual grammar (compositional rules). Our usage differs in that our sketch types emerged from the data and from existing sketch types such as UML, rather than being taken from a fixed and abstract set of graphical symbols and compositional rules. In situations where the first factor alone left the type of sketch ambiguous, we also considered how the designers referred to the sketch. For example, to differentiate sketches with both ER diagram and class diagram visual syntactic elements, we relied on how designers themselves referred to the sketch. In some situations, sketches evolved in place from one sketch type to another. In these cases, we used a combination of key visual syntax elements and how designers referred to the sketch to determine when the sketch type changed.

Sketch domain. Sketches also varied in terms of which aspect of design each modeled (e.g., a list of requirements vs. a list of software components). Through our iterative analysis of sketch content, we arrived at four *domains* and used these domains to categorize each sketch. These four domains were:

(1) *requirements* repeating or extending the requirements provided in the prompt,
(2) the architecture or implementation of the software *system*,
(3) the *user interface* of the system as the user sees and interacts with it, and
(4) the *problem domain* modeling the problem described in the prompt.

Visual syntactic element types. We identified each instance in which a new type of *visual syntactic element (VSE)* was first added to a sketch. A visual syntactic element refers to a component of a sketch that is uniquely identified through a combination of its visual elements and its arrangement in relationship to other components (i.e., compositional rules). VSEs included text, graphical symbols (such as boxes or arrows), drawing elements (such as shading), and drawn objects (such as cars). To categorize visual elements of VSEs, we used Bertin's six retinal variables, a set of atomic building blocks for any visual representation: *shape*, *size*, *color*, *brightness*, *orientation*, *texture* [5]. For example, a box-and-arrow sketch contains two VSEs – boxes and arrows – differing in the shape retinal variable, and a list title is distinct from a list item because of differences in arrangement. In coding the types of VSEs, we iteratively built a set of VSE types, recording each new VSE type we identified during coding. Each VSE type was recorded with a picture used to identify subsequent occurrences. To reduce the coding required, we did not record additional instances of the use of a VSE within a sketch after recording its initial introduction.

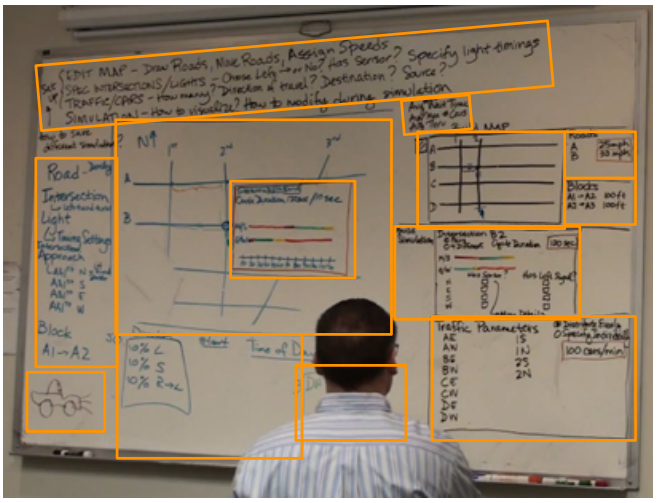


Figure 3. Whiteboard content was segmented into distinct sketches.

4.2 Focus fixations and transitions

The second coding scheme examined software designers' *focus and transitions* between sketches. At each moment of the design sessions, we tracked each pair's *focus point*, which we coded as a specific whiteboard sketch, the prompt, or none. We determined the focus point based on factors such as the person speaking, where their gaze was directed (based on the two recorded camera angles), the location of their pen, and the topic of the active discussion. While both designers nearly always had the same focus point, in situations in which they differed, we coded the pair's focus point as belonging to the designer who was speaking, or, if neither was speaking, to the designer who was writing. We ignored all activity conducted off-camera (which was usually re-reading portions of the prompt).

In our open coding, we observed that designers often referred to adjacent sketches in short bursts, but did not make any changes to that adjacent sketch. We define a *momentary reference* as an instance in which a sketch has a continuous focus duration shorter than 3 seconds, during which no changes were made. We distinguished three behaviours signalling designers' reference to adjacent sketches:

(1) *quick glance*,
(2) *point* with a finger, or
(3) *split focus* (pointing and glancing at different

sketches).

Intervals with no focus point—when the designers did not gaze at the whiteboard or prompt for a period of 15 seconds or longer—were identified as *inactive*. Inactive intervals could contain momentary references; only establishing a new focus point with a 3 second gaze or edit ended inactive intervals. For example, if the designers spent two minutes talking without using the whiteboard, pointed at a sketch for a brief second, and then returned to talking, this was coded as a single inactive interval with a momentary reference to a sketch.

Whenever designers' focus shifted between sketches on the whiteboard, we noted the origin and destination sketches, including their sketch type and domain (focus transitions to or from focus points other than whiteboard sketches were not labeled). In addition, we identified and coded three additional dimensions of the relationship between the source and destination sketches:

Abstraction. Sketches with an *abstraction* relationship represented a concept at differing levels of abstraction, either by providing a more detailed view of the entire sketch (e.g., a class diagram with only class names vs. a class diagram with class names, methods names, relationships, cardinality) or by providing a more detailed view of a portion of the sketch (e.g., a list and map, where the map depicts a concrete realization of a list element).

Alternative. Sketches with an *alternative* relationship described different potential solutions. We coded sketches as alternatives if the designers explicitly referred to the content in the sketches as competing solutions and used the sketches to discuss and compare the solutions at least once.

Point-in-time. Sketches with a *point-in-time* relationship were visually similar sketches depicting changes over time, phases, or conditions.

4.3 Reasoning activities

The third coding scheme examined the *reasoning activities* supported by sketches. We define reasoning activities as evident cognitive processes (as reflected by speech) with a specific purpose or goal. To code reasoning activities, we primarily used speech, but we also used gaze and gesture when it helped to confirm or clarify designers' intent. We chose to focus on coding reasoning activities that:

(1) involved lower-level goals over short periods of time,
(2) had well-defined boundaries, and
(3) could be consistently distinguished by both coders.

Thus, we did not include *designing* as a reasoning activity, as it spanned much of the session and did not have well-defined boundaries. This yielded three reasoning activities:

Mental simulation. A *mental simulation* refers to instances in which the designers talk through an execution of a mental model for the purpose of evaluating a design idea, allowing designers to turn "uncertainly into approximate answers" by performing thought experiments [3]. We build on the definition used by Ball and Christensen [3] in which the key feature is that the talk-through involves a simulation 'run' that alters the designer's mental representation to a new state. Similar to Olson *et al.*'s definition of "walkthrough" [39], we include instances such as simulating the intended user interacting with the design, following the movement of data within the system, or tracing state changes within the system.

Review of progress. A *review of progress* includes all moments when the designers review their design for the purpose of taking stock of what they have done. Our definition builds on what Olson *et al.* call "summary" [39], in which the designers momentarily take a step back from the design to consider the progress they have made, the goals they have yet to meet, or the tasks they have yet to complete [27].

Suggested design alternative. A *suggested design alternative* includes instances in which a designer presents a solution that competes with or replaces an existing solution. Unlike prior work that examined the structure of design rationale in detail [39], we did not build a comprehensive design rationale graph, but rather recorded instances where designers verbally presented a new design alternative that contrasted an existing design idea. We recorded an event whenever an alternative was *first* mentioned; when designers quickly mentioned alternatives in rapid succession, these were coded as separate alternatives.

Mental simulation and review of progress were coded as a pair of events, signifying the beginning and end of the activity. As we found that the discussion of distinct alternatives often overlapped and was difficult to distinguish, we tracked only the moment in which a design alternative was introduced, and we coded these as a single event. For each instantaneous event or event signifying the beginning of an activity, we noted both the type of reasoning activity and the role sketches served in supporting the activity:

(1) No sketches were used – designers did not gaze at or physically reference the whiteboard.
(2) Designers gestured over existing sketches without drawing.
(3) Designers drew on, annotated, or edited existing sketches.

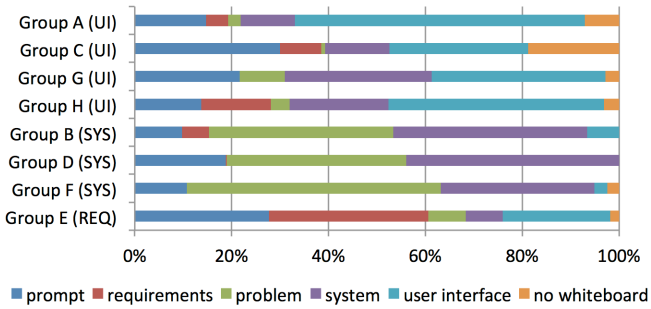


Figure 4. The fraction of session time each pair spent examining the design prompt, working with sketches of each domain, or not working at the whiteboard.

(4) Designers created a new sketch.

5 RESULTS

We first report our results on the *sketches* that designers produced. We then report how designers focused on and moved their attention among sketches. Finally, we report on the reasoning activities of the designers and how sketches supported these activities. Throughout, we report both quantitiatve data from our coding scheme and qualitative observations explaining this data. Numbers characterizing the behavior of designers are reported as an average across all pairs with a standard deviation (e.g., 23 ± 7%).

5.1 Sketches

In the following sections, we first examine how designers' choice of approach to design influenced the types of sketches they drew. We then examine the types of sketches designers created and the types of visual syntactic elements used within these sketches. Finally, we examine how designers annotated sketches and how sketches evolved through the design sessions.

5.1.1 *Approaches to Design*

While most pairs addressed the requirements, system, and user interface of the design, pairs tended to focus their efforts on a single aspect, using it to drive the design of the other aspects. This choice of focus led to three approaches to design that we observed, each of which was reflected in the time designers spent with sketches of the corresponding domains (Figure 4, Figure 9):

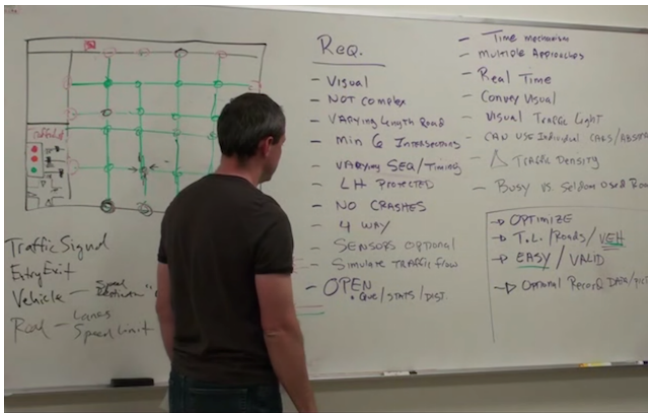
- *user interface-driven:* pairs A, C, G, and H (user interface sketches: 42 ± 13%),
- *system-driven:* pairs B, D, and F (system sketches: 42 ± 8%), and
- *requirements-driven:* pair E (requirements sketches: 33%).

These patterns were also evident in the quantity of sketches designers produced (Table 2). Each pair's approach to design was partially influenced by the designers' job descriptions: the only pair with a software analyst (E) was the only pair to focus on requirements, and the only pair with an interaction designer (A) focused on user interface design. Pairs with software architects and software developers were split between focusing on the user interface and the system.

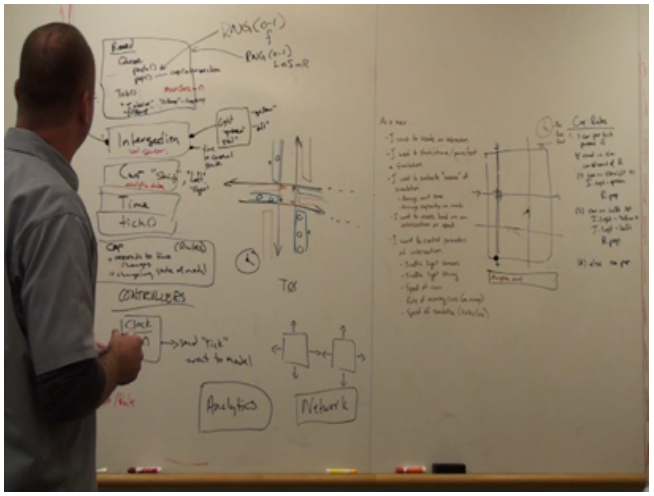
User-interface-driven User-interface-driven pairs (A, C, G, H) generally asked themselves: "How can users use our system to accomplish their goals?", focusing on front-end sketches and discerning how to get input from the user and display a result. These pairs relied heavily on user interface sketches to structure and organize their work (Figure 3, Figure 5a).

Reflecting the traffic simulator problem domain, the user-interface-driven pairs made frequent use of maps (30 ± 6% of session time) to brainstorm user interactions and support discussion of the problem. Maps also served as a hub: pairs frequently shifted attention to a map before shifting to a sketch of a different sketch type.

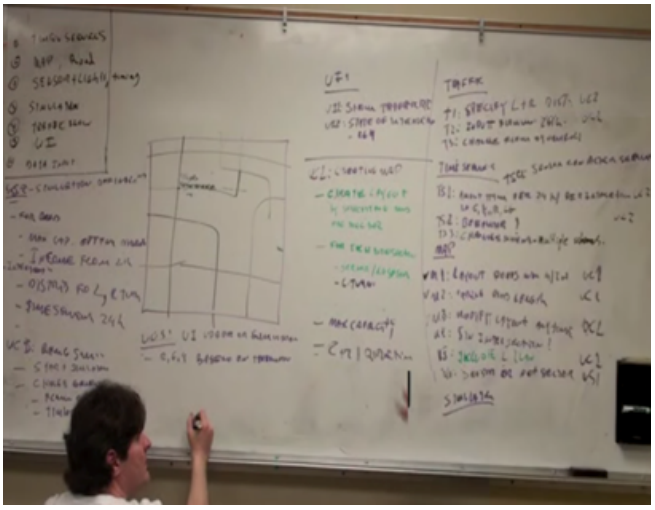
Lists were the second most common sketch type for user-interface-driven pairs (19 ± 9% of session time). Lists helped designers to record potential classes they discov-



(a) User interface-driven (pair H)



(b) System-driven (pair B)



(c) Requirements-driven (pair E)

Figure 5. Each pair focused their efforts on a specific aspect of the design.

ered while designing the user interface. Lists also served an important role in recording requirements, which the designers periodically referenced to verify that they were being met. Toward the end of the sessions, three of the UI pairs finally considered the software architecture in detail, either by creating an ER diagram (A and H) or by making lists (C) describing the structure of a map class. They stated that they did this to satisfy the prompt's requirements that they describe the system in sufficient detail for software developers to implement it.

System-driven In contrast to user-interface-driven pairs, system-driven pairs (B, D, F) focused first on the system design, designing classes and their interactions. In general, these pairs asked themselves: "How does the system work?" As seen in Figure 5b, these pairs split their time between using software sketches to document and discuss the classes in their system design (as in the left of Figure 5b) and using problem sketches to discuss and understand how their system worked (e.g., the map in the center of Figure 5b).

All three system-driven pairs created sketches listing nouns and sketches representing a map with an intersection symbol. The left side of Figure 5b presents an evolved representation of two examples of these sketches, where the list of nouns became the software architecture alongside the intersection sketch. Beyond this basic setup, system-driven pairs proceeded with their own variations. One pair (B) picked a class from their design, defining it in detail. Late in the session, two of the pairs (B, D) drew a second class diagram in order to define classes and their relationships in much greater detail.

While working with system sketches, designers pointed to classes, marked up elements with annotations and arrows, and explained classes verbally to their partner. Lists were sometimes used to document design decisions, such as explicit instructions on how a particular component behaved (far right of Figure 5b), or to record user stories (center of Figure 5b). As these designers dug more deeply into the system design, they created sketches allowing them to explore the system and problem in more detail, such as line graphs and code. These sketches helped them explore 'proofs of concept' of their proposed solutions.

Requirements-driven The requirements-driven pair (E) focused first on listing requirements, posing the question: "What is it that we need the system to do?" They intentionally avoided designing the system through much of the session, later reporting that "*we wanted to be clear about what we're building, not how we're going to build it*", only considering the system and user interface late in the session.

Lists played a central role, helping the pair to direct their thinking within the design session, reason about parts of the system, and document design decisions. After looking at the prompt, they began with a list of requirements (upper left corner of Figure 5c), and then broke each out into separate, more detailed lists of requirements, maintaining a one-to-one relationship between items in the initial list and the title of each new list. They then laid out a set of use cases (right side of Figure 5c), creating a map to help them generate use cases by enumerating ways in which the user would interact with the map. As they went through the use cases, they annotated each requirement, identifying how it mapped to use cases. As they addressed requirements, they checked off the corresponding requirements. At the end of the session, they erased a large section of the board to create an ER diagram.

	Number of sketches (*by pair*)									Avg. # VSE types
	A	B	C	D	E	F	G	H	Avg.	
(by sketch type)										
map	3	9	6	3	7	19	5	6	7.3	5.3
list	5	3	6	3	15	7	2	5	5.8	4.0
table	11	-	-	-	-	-	4	1	2.0	3.7
GUI	2	1	1	-	2	5	-	2	1.6	4.5
ER diagram	1	-	-	1	1	-	-	3	0.8	6.7
class diagram	-	2	1	-	-	-	-	-	0.4	11.8
code	-	1	-	-	-	2	-	-	0.4	5.0
drawing	1	-	-	-	-	-	-	-	0.1	1.0
traffic signal	-	-	-	-	-	-	2	1	0.4	4.8
line graph	-	1	-	-	1	-	-	-	0.3	4.5
array structure	-	1	-	-	-	-	-	-	0.1	3.0
(by domain)										
problem	3	6	1	3	5	23	5	6	6.6	
user interface	16	5	6	0	5	3	6	4	5.5	
system	2	6	5	3	1	9	2	5	4.3	
requirements	2	1	2	1	15	1	0	3	3.3	
Total sketches	23	18	14	7	26	36	13	18	19.4	

Table 2. The sketches designers drew by sketch type and by domain and their average number of distinct types of visual syntactic elements (not listed for domains, as the number of VSEs varied greatly by sketch type).

5.1.2 *Types of Sketches and Visual Syntactic Elements*

Across all sessions, designers used a total of 11 sketch types (listed in Table 2). Designers often created ad-hoc sketch types reflecting the domain, including the most frequent sketch type – maps. Designers also created system diagrams with defined notations (e.g., class diagram, ER diagram), sketches with ad-hoc notations (e.g., array

structure, GUI), and general-purpose sketches (lists, tables, line graphs, drawings). Designers created an average of 19.4 (± 8.9) sketches per session, each with an average of 4.8 types of visual syntactic elements. Table 3 lists the most frequent types of visual syntactic elements for the most frequent types of sketches.

Despite the traditional focus on software design notations for describing system structure, most of the sketches did not concern the design of the system. Categorising sketches by domain, pairs created an average of 6.5 (± 6.9) problem sketches, 6.4 (± 4.4) user interface sketches, 4.1 (± 2.6) system sketches, and 3.6 (± 5.1) requirements sketches. Even when using system sketches, designers more often chose to sketch using the syntactically simple lists over more elaborate and complete notations such as ER diagrams (0.8 ± 1.0 per pair) or class diagrams (0.4 ± 0.7 per pair). For example, designers often simply made lists of classes and methods rather than use boxes with a title to denote classes as in class diagrams.

Reflecting the traffic problem domain, *maps* were the most common type of sketch created (7.3 ± 5.4 per pair) and were used by all pairs. Designers used maps to reason about traffic in two different ways: understanding how intersections work, and designing user interfaces. Software-driven pairs often used problem domain maps to figure out how cars moved through intersections. They asked questions such as how does a car decide to make a left turn, do cars instantaneously move from one intersection to the next, and what component of the system controls a behavior. To answer these questions, designers sometimes went to a very low level of abstraction, drawing individual cars in their depictions of maps (Figure 5b).

In contrast, user-interface-driven pairs instead used maps depicting the user interface. These pairs generally reasoned about traffic at the level of traffic flows, considering scenarios such as that on Mondays and Wednesdays people went out to lunch, changing the traffic density. As a result, the maps these designers drew were often at a higher level (Figure 3, Figure 5a).

In supporting these uses, the VSE types designers used varied between road-level views and map-level views. Road-level views often used either lines demarcating the edges of roads (parallel lines – 33% of maps, 7 pairs) or intersecting roads (intersection – 30%, 5 pairs), and represented traffic with indicated individual cars, intersections, and traffic patterns with rectangles (37%, 7 pairs), circles (28%, 6 pairs), or dotted lines (17%, 5 pairs). Map-level views used lines to represent individual roads (single line – 30%, 6 pairs) or sets of roads (grid – 14%, 5 pairs), indicating flows of traffic with arrows with lines (47%, 7 pairs).

Lists were the second most common sketch type (5.9 ± 4.1 sketches per pair) and were used by all pairs. Designers used lists to record requirements, components, use cases, and component behavior. Most lists were syntactically simple and contained few types of VSEs (average 3.7 ± 1.6), most commonly first-order elements (92%, 8 pairs) and titles (76%, 8 pairs). Designers often did not expand items in a list: only 36% contained second-order elements. Instead, designers created new lists. Parentheses were sometimes used to express provisional information, often next to titles or first order elements. Designers very rarely reordered or rearranged lists. Most were not numerically numbered, and for those that were, the order served only to make items quickly referenceable or to provide a count of items. This suggests that designers rarely worked with lists that encoded meaning in their order. Lists often depended implicitly on other sketches, but designers rarely made these dependencies explicit; only 20% contained arrows explicitly showing relationships from elements in the list to other sketches. Few lists referenced sketches by name.

Tables were most often used to represent aspects of the user interface, typically as a mechanism for inputting data (e.g., Figure 3). Tables varied in the ways in which designers created them. Some tables were created bottom-up, beginning with values in the table, and only later (if ever) annotated with row headers. For example, when designing traffic intersections, designers wrote down row values into the table, and later added the headers for these values. Other tables were created top-down; designers drew the row headers first and then typically included additional syntactic elements such as column headers and grid lines.

GUIs were often created after interface maps, helping designers to determine how the map content might be manipulated by a user. For example, designers would mimic moving sliders, then imagine how the map interface would update as a result. GUIs most often contained a bounding box (53%, 5 pairs), check box (46%, 3 pairs), and title (39%, 3 pairs).

Entity-relationship (ER) and class diagrams contained a larger set of core notations shared by most sketches (Table 3); this likely reflects ER and class diagram's status as well-known notations with conventions on how to draw them. Relationships between entities were ubiquitous, and designers often explicitly differentiated the type of relationship they denoted. For example, in ER diagrams, designers used relationships with no head or tail (83%, 3 pairs), relationships with triangle heads (67%, 4 pairs), and solid dots (50%, 3 pairs). Once made, decisions on relationship type were never changed. Designers also tended to use ER notations (e.g., dots and hollow circles to represent containment and referencing), even when working with class diagrams, which have a different formal representation defined in the UML (e.g., dashed arrows and diamond heads).

Designers sometimes explicitly avoided notations they considered to be too detailed for their purposes. For example, upon noticing that their partner had created a second-order item in an ER diagram, one designer (E) interrupted, noting that: "we don't have time to go into that level of detail". Class diagrams were used by only two pairs (B, C). To explain the run-time behavior of components, designers used class diagrams rather than sequence

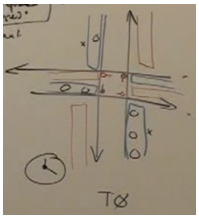
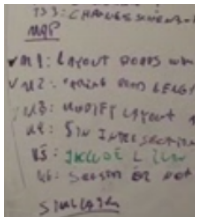
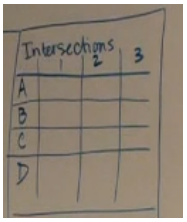
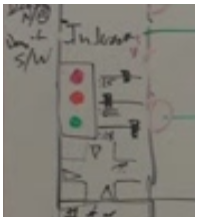
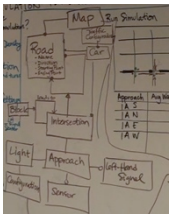
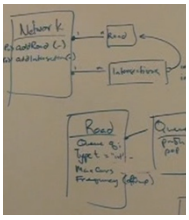
					
Map **(8 pairs)** **7.3 per pair**	**List** **(8 pairs)** **5.8 per pair**	**Table** **(3 pairs)** **2.0 per pair**	**GUI** **(6 pairs)** **1.6 per pair**	**ER diagram** **(4 pairs)** **0.8 per pair**	**Class diagram** **(2 pairs)** **0.4 per pair**
1. Arrows (47%, pairs=7) 2. Text labels (38%, pairs=6) 3. Rectangle (37%, pairs=7) 4. Parallel lines (33%, pairs=7) 5. Single line (30%, pairs=6) 6. Intersection (30%, pairs=5) 7. Bounding box (30%, pairs=6) 8. Circle (28%, pairs=6) 9. Notch on line (18%, pairs=5) 10. Dotted line (17%, pairs=5)	1. First-order element (92%, pairs=8) 2. Title (76%,pairs=8) 3. Second-order element (36%, pairs=7) 4. Item circled (24%, pairs=5) 5. Arrow (20%, pairs=5) 6. Parenthesis annotation (18%, pairs=3) 7. Floating element (9%, pairs=3) 8. Underline (8%, pairs=2) 9. Question mark (4%, pairs=2) 10. Star symbol (4%, pairs=2)	1. Row header (88%, pairs=3) 2. Bounding box (50%, pairs=3) 3. Column header (50%, pairs=2) 4. Grid lines (50%, pairs=2) 5. Title (50%, pairs=1) 6. Cell items (25%, pairs=2) 7. "etc." symbol (25%, pairs=2) 8. "X" mark in cell (13%, pairs=1) 9. Cell highlight with different color (13%, pairs=1) 10. Free-floating label (13%, pairs=1)	1. Bounding box (53%, pairs=5) 2. Check box (46%, pairs=3) 3. Title (39%, pairs=3) 4. Slider (39%, pairs=4) 5. Squiggly line representing etc. (23%, pairs=2) 6. Timeline black and white (17%, pairs=2) 7. Timeline colored sections (13%, pairs=1) 8. Rectangular button (8%, pairs=2) 9. Triangular button (8%, pairs=1) 10. Red/green/yellow circles combo (8%, pair=2)	1. Element name (100%, pairs=4) 2. Bounding box (83%, pairs=3) 3. Relationship (no head or tail) (83%, pairs=3) 4. Relationship (triangle head) (67%, pairs=4) 5. Relationship (solid dot) (50%, pairs=3) 6. Member items (50%, pairs=3) 7. Cardinality (33%, pairs=2) 8. Title box (33%, pairs=2) 9. Dotted line (33%, pairs=2) 10. Bounding circle (33%, pairs=1)	1. Bounding box (100%, pairs=2) 2. Freeform text descr. (100%, pairs=2) 3. Relationship ("v" arrowhead) (100%, pairs=2) 4. Cardinality (67%, pairs=1) 5. Class name (67%, pairs=2) 6. Relationship (solid dot head) (67%, pairs=1) 7. Explicit function arguments (67%, pairs=1) 8. Informal function parameters (67%, pairs=2) 9. Field names using code syntax (67%, pairs=1) 10. Title box (67%, pairs=1)

Table 3. The six most frequent sketch types and their ten most frequent VSEs (the remaining sketch types were used too infrequently to identify recurring visual syntactic elements). Each sketch type is listed with an example image (above), the average number of sketches per pair (n), and the number of pairs that used the sketch type. Each VSE is listed with the percentage of sketches that used it and the number of pairs which used it.

diagrams. It is interesting to observe that this allowed them to explore many distinct scenarios instead of just one. However, in the exit interviews of three of the software pairs (B, D, F), they stated that they would normally create sequence diagrams as the "next steps".

5.1.3 *Annotating sketches*

Designers often annotated their sketches, using visual syntactic elements to guide attention, support decision-making, record provisional information, or reference other sketches. As designers moved between sketches, they sometimes annotated sketches for emphasis or to guide attention. For example, one designer (H) walking through an explanation said "especially this" and underlined an element. Emphasis marks did not reflect any observable pattern: designers used a broad range of VSE types, including an underline, a circle, a star, a large "X", emboldened dots, and vertical bars. Once made, these marks were not subsequently used or referenced but persisted for the life of the sketch. Designers sometimes used VSEs to explicitly refer to other sketches, typically with declarative labels referencing other sketches or with arrows.

Annotations helped to support decision-making. Several pairs used check marks on lists, maps, class diagrams and ER diagrams. One pair (H) discussed a requirement that involved many parts of the system. To address this issue, they drew question marks next to all event objects, and systematically visited the question marks and placed either a check mark or large "X" to signify whether the object did or did not address a requirement. Another pair (E) checked off requirements in a list as they verified that they were addressed.

Designers also annotated sketches to record provisional information. For example, designers who could not settle on a name for a list wrote a second name in parentheses next to the official title. This behavior occurred across many pairs (A, B, C, E, F) and several representation types, especially ER and class diagrams. Other pairs used slashes in lists to indicate alternative names.

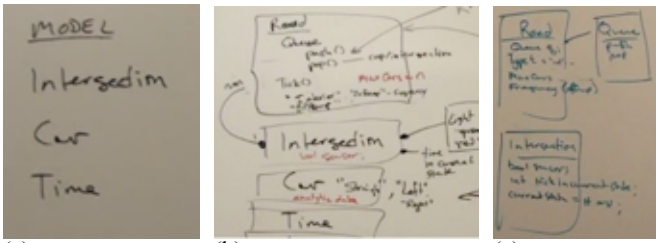
5.1.4 *Evolving sketches*

Designers continually invented new types of visual syntactic elements to capture and externalize concepts they discussed while designing. Figure 6 plots the growth of

VSE types in sketches over time. For example, one pair (B) wished to determine how users were interacting with maps and drew a UI map sketch (right side of Figure 5b). They first drew a box representing the whole area and then drew lines to represent the user adding streets to the map. They then proposed: 'what if every time two streets intersected would create an intersection?', and drew an intersection VSE where two lines met (mark where lines meet in Figure 3b). Next, they decided that cars would be generated and enter from off the map – represented as an "arrow" VSE – and be terminated when they exited the map – represented as a solid dot VSE. Overall, the longer designers spent at a sketch, the more VSE types they created ($r = .62 \pm .36$).

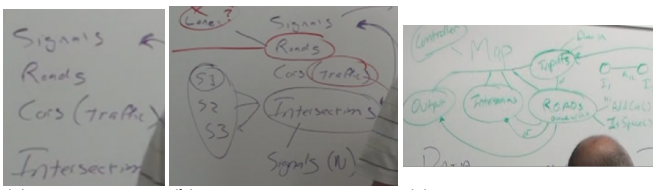
As designs progressed and designers had more concepts to describe, designers sometimes externalized concepts on the whiteboard by borrowing syntactic elements from other notations. For example, designers often began sketches as a list, perhaps reflecting their simplicity. As the discussion continued, designers sometimes borrowed syntactic elements from other types of sketches. For example, lists that represented software entities sometimes began to borrow syntax from class and ER diagrams such as arrows, cardinality, and bounding boxes (Figure 7a and 7b; Figure 8a and 8b). Lists representing requirements documents gained labels on first order elements or gained arrows to represent dependencies between requirements. As a result, some lists began to evolve into sketches of another type (e.g. Figures 7 and 8).

In some situations, designers chose to evolve their sketches, often in situations where aesthetic presentation was not a priority. In other cases, sketches began their transition in place before being redrawn (Figure 7b and 7c; Figure 8b and 8c). Redrawing sketches was often triggered by a shift from using a sketch to help think through the design to using a sketch to document the design. In



(a) (b) (c)

Figure 7. Content (a) initially created as a list (b) evolved in place to a class diagram, and (c) later was redrawn as a new class diagram (pair B).



(a) (b) (c)

Figure 8. Content (a) initially created as a list (b) evolved by borrowing syntactic elements, and (c) later was redrawn as a new ER diagram (pair D).

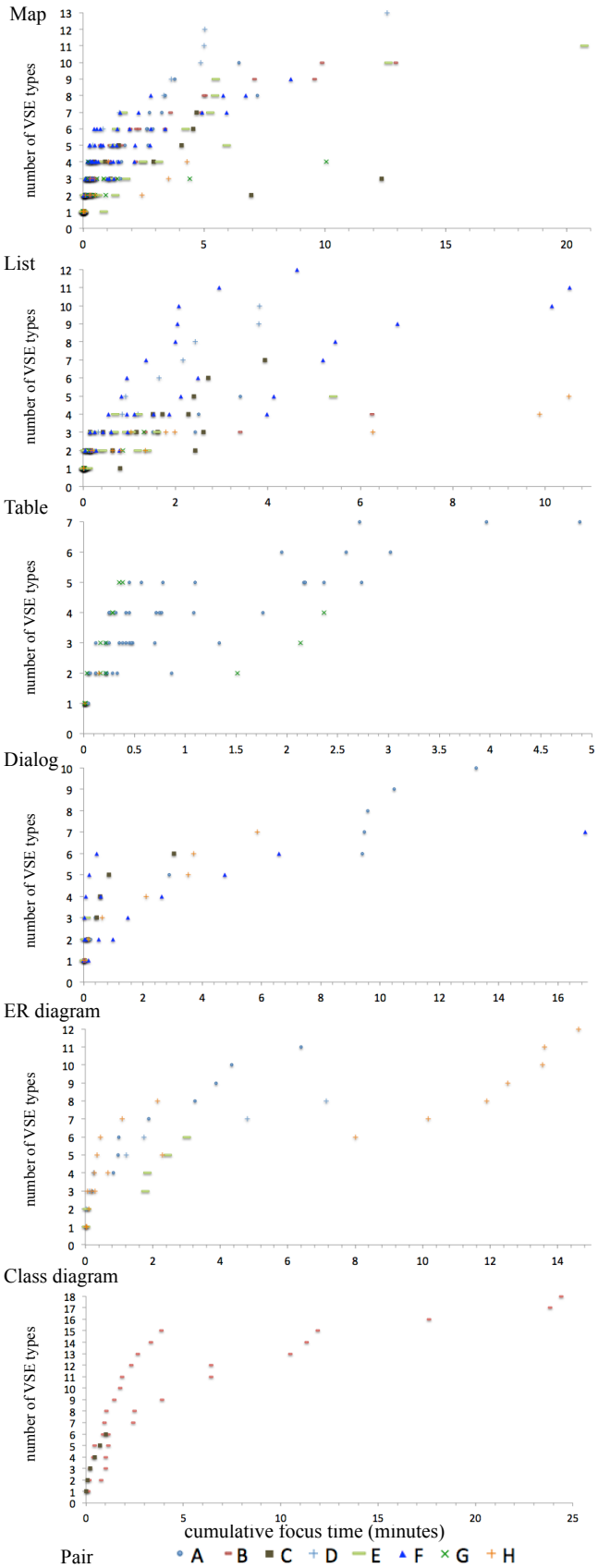


Figure 6. The number of VSE types in a sketch as a function of time. Each data point depicts the number of VSE types in an individual sketch when it has had focus for the corresponding time.

some cases designers also left the prior sketch. For example, toward the end of the session, when designers wished to record more of the design knowledge in their heads, pairs B and H redrew the sketch from scratch in much greater detail. As they drew the new sketch, they left the old sketch intact, allowing them to refer to the old sketch as they drew the new one.

Pairs varied greatly in how often they erased existing sketches, ranging from erasing 0% to 81% of their sketches. Overall, pairs erased 41 ± 31% of their sketches. Pairs that created more sketches erased more, as they needed the space to create the additional setkches.

5.2 Focus and transitions

Throughout the design sessions, pairs shifted their focus among sketches on the whiteboard. Figure 9 depicts the focus periods of all sketches. Designers sometimes had a single, core sketch that dominated their design session. For example, pair B spent much of their session interacting with a class diagram (top red line in Figure 9B) and pair H interacted extensively with a map (top blue line in Figure 9H). But, designers more often worked with groups of related sketches; and even pairs B and H used a wide range of additional sketches in support of their core sketch.

Designers often rapidly shifted between related sketches that each served a separate purpose in supporting the current design activity. For example, when enumerating items in a list, pairs shifted their focus to a map to discuss how cars moved, to a table to understand how different categories of elements in the map interacted with each other, and finally to a class diagram to record new aspects of the design. Overall, transitions between sketches were very frequent, leading to short focus periods in which pairs worked with a single sketch (Figures 9 and 10). Designers rarely (12 ± 5% of total focus periods) focused for more than 30 seconds at a single sketch, and they often spent less than 10 seconds at a sketch (66 ± 9%).

As designers worked with a group of sketches, they often shifted back to a sketch with which they had recently worked (Figure 11). In some cases, pairs 'ping-ponged' back and forth between a single pair of sketches (e.g., top two sketches of pairs B and D in Figure 9). Average focus periods were short across all sketch types – there were no types of sketches that designers did *not* use in close collaboration with other sketches (Figure 9).

To examine the typical size of the groups of sketches with which designers worked, we counted the number of sketches which received the designers' focus within windows of 30 seconds, 1 minute, and 5 minutes (averaged across all windows in 1 second increments; e.g., windows of 0 – 30, 1 – 31, etc.). Overall, pairs used an average of 1.5 (± .9) sketches every 30 seconds, 2.0 (± 1.2) sketches every minute, and 3.9 (± 2.0) sketches every 5 minutes (Figure 12).

Focus transitions were often to sketches of different types (53 ± 17%) or domains (43 ± 16%) and sometimes crossed levels of abstraction (27 ± 13%). For example, sys

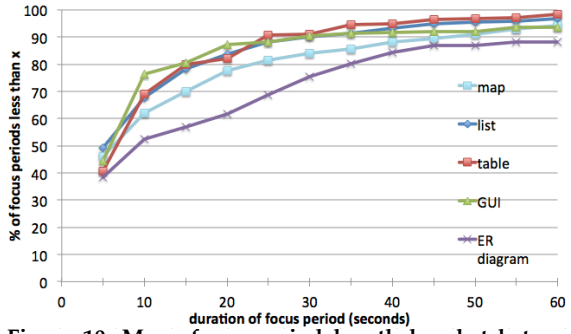


Figure 10. Mean focus period length by sketch type (averaged across pairs).

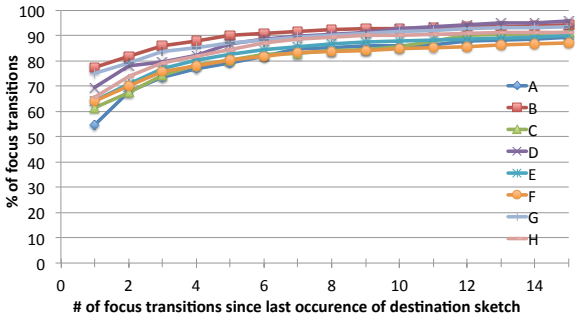


Figure 11. The number of focus transitions since the sketch last had focus, as a percentage of total focus transitions (reported by pair).

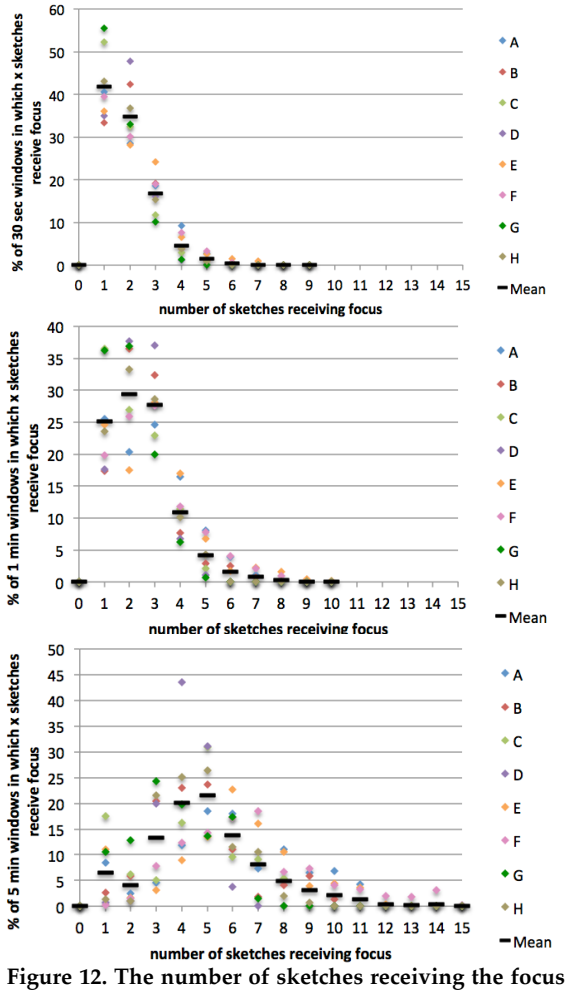


Figure 12. The number of sketches receiving the focus within windows of 30 sec, 1 min, and 5 min.

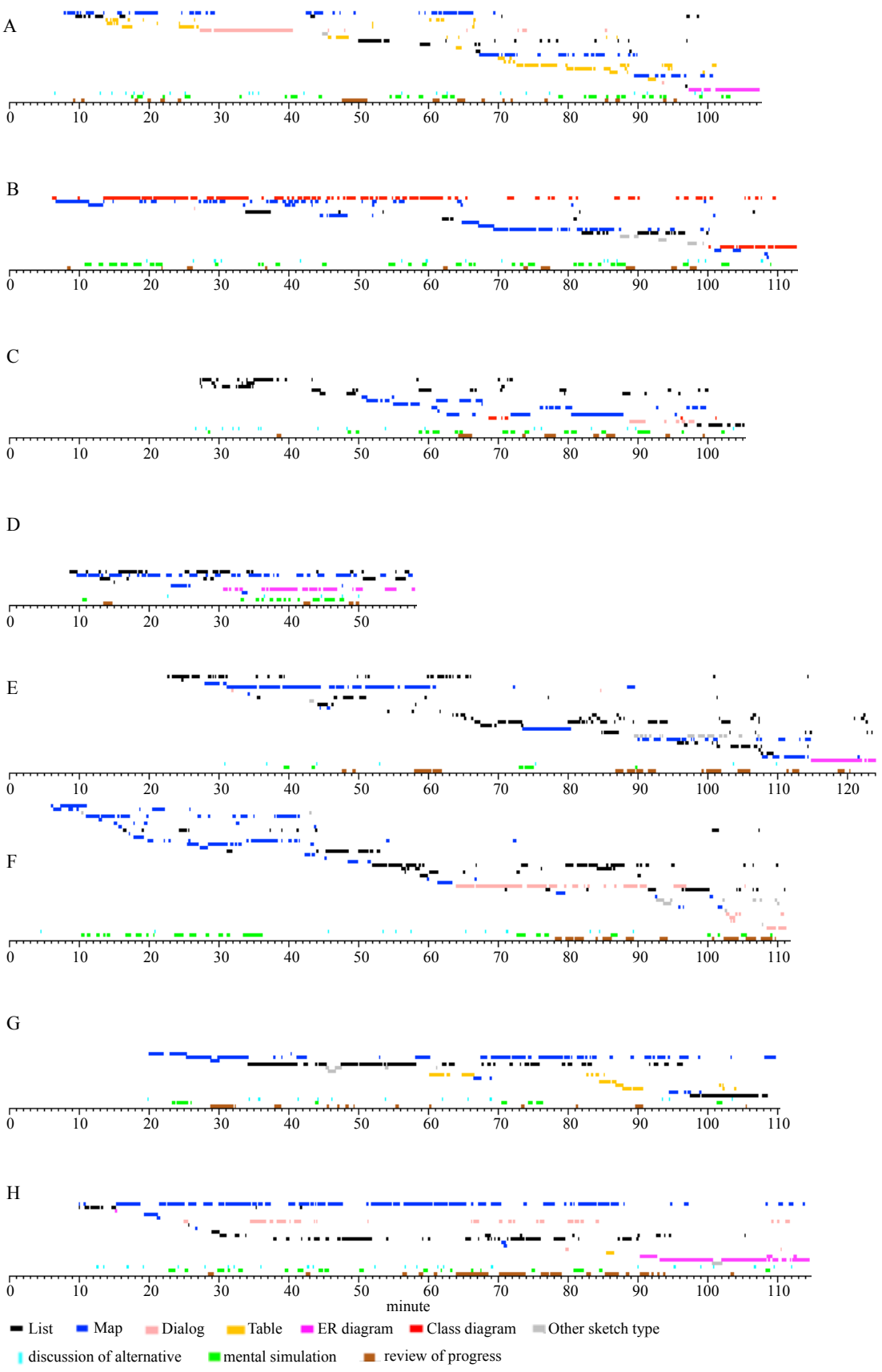


Figure 9. Sketch focus during the design sessions, depicted for each 10 second interval (focus periods shorter than 10 seconds are rounded up to 10 seconds). The time shown begins when pairs were first given the design prompt and ends with the end of the design task. Each horizontal row corresponds to a single sketch or reasoning activity (bottom three rows). Sketches are listed in the order in which they were created.

tem-driven pairs (B, C, E) relied heavily on problem domain sketches to support reasoning about components within their system design sketches.

All system-driven pairs frequently moved between a map that they marked up with circles to represent cars to the equivalent entity in a system sketch. Often, the designers did not write on either sketch, but instead internalized the insights of their discussion. In other cases, designers did edit sketches. For example, several pairs (B, C, G) mentally simulated the movement of cars through intersections using a map. They rapidly switched back and forth between the map and the class diagram to design the data model, drawing relationships between different classes to capture necessary control and data flow. As designers very rarely drew separate sketches to represent alternatives or separate points in time (see Sections 5.3.3 and 5.3.1, respectively), designers very rarely shifted between sketches representing alternatives (1 ± 1%) or points in time (1 ± 3%).

In 25% (± 6%) of the total focus periods, designers quickly referred to a sketch for less than 3 seconds without editing it as a momentary reference. We categorized each momentary reference as a *quick glance*, *point*, or *split focus*. Quick glances (43 ± 14% of momentary references) allowed designers to gather information or seek confirmation. Pointing (37 ± 14% of momentary references) helped guide the attention of the other designer in the pair to explain, review progress, or mentally walk through the design. Split focus (20 ± 10% of momentary references) allowed designers to reason about how a design might work, using the knowledge gained from one sketch to help identify omissions, mistakes, and inconsistencies. For example, the designer on the left in Figure 13 simultaneously pointed at a data model and a map to understand how a car object is passed between components as it travels through an intersection. Mental simulations were often mentally intensive, requiring extensive attention. Split focus enabled designers to externalize the sketches relevant to the discussion, reducing the cognitive overhead of recalling where sketches were located and allowing more cognitive resources to be used on the reasoning activity at hand. This is consistent with distributed cognition and the concept of computation offloading [44].

5.2.1 *Transitions between levels of abstraction*

Shifting to a sketch at a different level of abstraction enabled designers to focus on a particular aspect of the design by omitting non-relevant details. For example, six of the eight pairs shifted at least once from a map of intersecting roads to a sketch of a single road, which helped them focus on the mechanics of individual cars while ignoring details of the rest of the map. Designers did not shift between sketches at different levels of abstraction until their design had become sufficiently complex. Designers began shifting between sketches at different levels of abstraction, on average, 33 (± 17) minutes into the session. After this point, abstraction shifts were frequent, occurring once every 49 (± 25) seconds. Overall, 26% (± 13%) of each pairs' transitions shifted between abstraction levels.

Figure 13. An example of a split focus between two sketches.

The designers frequently shifted to lists in order to discuss components at a high level of abstraction; lists were the target sketch type in 39% (± 15%) of abstraction transitions. These lists were often lists of system components or requirements. All designers created a list of software classes, allowing them to defer discussion of details until later in the session and carry on conversations at a high level of abstraction. As the design session progressed, some pairs (B, D, F) fleshed out classes with more detailed designs, often expressed in ER diagrams. But the more abstract lists continued to play an important role, helping designers to keep track of where they were in the design process by checking off, erasing, and pointing at the lists.

Maps were a frequent target of abstraction transitions as well (35 ± 19%), helping to support less abstract discussions. Designers shifted to map sketches from software and lists sketch types in order to better understand the problem domain. For example, designers moved between sketches of an entire map to a single intersection, allowing them to explore traffic configurations at an intersection without the clutter of a larger map. In turn, focusing at a lower level of abstraction led to a large variety of visual syntactic elements, such as elements for single lanes, many lanes, lights, traffic configurations, and car queues.

5.3 **Reasoning activities**

Sketches played an important role in supporting at least three distinct types of reasoning activities: *mental simulation*, *review of progress*, and *suggesting design alternatives* (Table 6). Pairs averaged 17.9 (± 8.4) mental simulations, 11.8 (± 4.6) reviews of progress, and 14.4 (± 7.4) discussions of alternatives per session (Table 7). Pairs varied greatly in the frequency of these types of reasoning activities: mental simulation (5 to 30), review of progress (5 to 17), and discussion of alternatives (5 to 27). Sketches played an important role in supporting reasoning activities: 81% (± 13%) of reasoning activities used sketches, with 53% (± 8%) of reasoning activities gesturing over existing sketches without editing, 22% (± 11%) editing an

Reasoning Activity	Example
Mental simulation	*"So the cop looks at the intersections [and] says, okay it's a green light on these roads. This road has some number of cars in it. If the light's green one tick, [it] means we pop, and we do that queue and second picture. So from T0 to T1, that's how we've gone through two."*
Review of progress	*"Okay. Maybe we should start talking about the view. We've got kind of a rough sketch of the base class of the model and kind of a rough idea of the controller, which is a combination of this cop with the visitor..."*
Suggestion of design alternative	*"Or you could say, well the queue—let's go back to the model for a second—so instead of just a straight queue you could have like a… Well, we have for interior road[s], we have a maximum number of slots, so let's say this is the head."*

Table 6. An example of each reasoning activity.

existing sketch, and 6% (± 5%) creating a new sketch. In the following sections, we examine each reasoning activity in turn.

5.3.1 *Mental simulations*

Mental simulations allowed designers to walk through the behavior of their design. As in other reasoning activities, mental simulation was often supported by sketches (83 ± 17%). Mental simulation sometimes involved edits (30 ± 30%) or a creating a new sketch specifically to support the mental simulation (9 ± 11%). When designing user interfaces, designers sometimes mimicked using their design by talking and gesturing over sketches, using the experience to demonstrate a nonfunctional requirement that the system should satisfy. For example, one designer (A) simulated a user creating roads by dragging the mouse across the map and stated that in her proposed interaction "you can create something very quickly without fussing with it". Mental simulation helped designers not only to imagine using their design, but to put themselves in the mindset of the end-user.

When designing with system and problem sketches, mental simulation enabled designers to "expose the guts" of their design in order for it to be scrutinized. For example, one pair (D) declared that they were "putting [their] foot into the dirty details" before initiating a mental simulation. All system-driven pairs used mental simulations to explain how data and control was passed between software classes or how the state of the model was changed over time. Mental simulation was typically not strictly sequential from start to finish: designers instead backed up in their explanations, paused to explain pieces, and abandoned simulations when they became distracted.

Surprisingly, designers almost never supported mental simulation by creating separate sketches representing different conditions or points in time. In the single instance in which this did occur, pair B reasoned about the system by drawing two adjacent intersection sketches labeled *T* and *T + 1*. They then simulated the movement of cars through an intersection by shifting between the before and after diagram.

Overall, mental simulations served several purposes:

Propose new design ideas. As designers simulated and walked through the behavior of aspects of the design with a temporal component, such as the behavior of end-users or code, they sometimes generated design ideas. For instance, while walking through the end-user experience, designers (A, G, H) created new interface components 'on the fly' to continue the simulation. As another example, while walking through control or data flow, some designers (B, D, F) created the needed class diagram elements in a similar manner.

Clarify and refine. Mental simulations helped to bring fragmented assumptions about the system together into a cohesive story. For example, after one designer (G) became lost in the details of their design, the other designer clarified the design by mentally simulating its steps.

Reflect and evaluate. Mental simulations brought out details of the design, allowing them to be evaluated. For example, after stepping through data flow between software components, one pair (B) discovered a potential problem in their design: "well that may end up with spaghetti code... so what about a traverser pattern". In another case, a designer in pair F changed a design decision and retreated to a previously-rejected design idea. After performing a mental simulation, he stated: "I didn't before, but now I think [showing individual cars] is important". When a designer walked through a simulation, the partner would sometimes jump in and ask: "What about X happening?" Mental simulations often spurred partners to propose situations and corner cases that could lead to problems in the design.

Reveal implications. Mental simulations helped to reveal implications of the design that were not immediately obvious. For example, one pair (D) had determined that cars must go through an intersection. But after one designer simulated this behavior with an explanation of how this occurs, the other designer asked: "But how does the intersection know where to send it?", prompting a new discussion.

5.3.2 *Review of progress*

Reviewing progress was a central part of designers' activities at the whiteboard, allowing designers to take stock of where they were and evaluate the design as it existed. On average, designers stopped to review progress every 9.3 (± 2.5) minutes for 49 (± 11) seconds, and all 8 pairs reviewed their progress. When reviewing progress, design-

	A	B	C	D	E	F	G	H	Avg
Men. Sim.	30	20	19	14	5	24	8	23	17.9
Rev. Prog.	17	7	8	5	15	12	13	17	11.8
Disc. Alt.	22	12	17	5	7	12	13	27	14.4

Table 7. Number of reasoning activities (by pair).

ers often consulted whiteboard sketches they had drawn (88 ± 13%). These consultations only occasionally led to edits (24 ± 19% of instances reviewing progress). Reviews of progress almost always involved reviewing requirement sketches (86 ± 13%). Designers reviewed lists of requirements, which were then cross-referenced with other sketches to determine if the current design satisfied the requirements as the designers understood them.

Pairs varied in how they tracked requirements, which in turn influenced how they reviewed their progress. Some pairs (D, F, G) did not list requirements. Pairs D and F were system-driven pairs that visited mainly system and problem domain sketches. Pair D performed their reviews of progress to check their work over software components. Pair F, in contrast, performed their reviews of progress by role-playing the designer and implementer roles; one designer told the other to: "pretend I'm the developer, describe the system to me". Pair G, a user-interface-driven pair, used sketches primarily to summarize their progress.

Other pairs (A, C, and H – all user-interface-driven pairs) maintained one or two lists of high-level requirements throughout the design session (e.g., Figure 5a). Designers varied in terms of when the requirements were authored: two pairs (A, H) created their entire list of requirements in a short period of time, later using the list to check that requirements were satisfied but making very few edits; the other pair (C) made many edits and additions to their requirements throughout the design session.

Reviewing progress served several functions:

Verify coverage of requirements. Designers reviewed requirements one at a time, ensuring that each was satisfied by the design by cross-referencing sketches. This sometimes led pairs to return to deferred issues, spawning new discussions to address them. Designers used check marks and other symbols to check off completed requirements.

Identify assumptions. Designers reviewed their sketches to check that the design was logically sound, helping to uncover assumptions and portions of the design that were incomplete. For example, some pairs realized that cars could travel between multiple intersections, spawning a discussion to expand the design for these cases.

Plan what to address next. After reviewing a set of sketches, designers sometimes discussed the sketches they needed next, pointing at existing sketches to establish what they had missed. For example, pair B looked over what they had created, checked it against the requirements, remarked that they had 30 minutes, and agreed that they should redraw everything to communicate the design to the developers who would implement it.

5.3.3 *Discussion of alternatives*

Designers often discussed alternatives; overall, pairs discussed an alternative once every 8.7 (± 4.4) minutes. Discussing alternatives included both situations where alternatives were considered seriously, and those where 'straw man' alternatives were used to clarify an existing design idea without any intention of following them through. When unable to choose definitively between alternatives, designers often made a provisional decision, resolving to re-evaluate the decision later, as: "it's more expensive to get stuck on an idea; [we] need to move on!" (A). For example, 6 of the 8 pairs considered alternative designs for left turn signals. Several chose one approach early on or deferred the decision, revisiting the decision later with in-depth discussion when reviewing progress or mentally simulating the situation.

When considering alternatives, designers often (74 ± 17%) made use of whiteboard sketches, either by pointing, editing, or creating new sketches. Discussing alternatives was most often supported by simply pointing or glancing at existing sketches. In 76% (± 15%) of cases, no trace of discussed alternatives remained on the whiteboard. Only 18% (± 17%) of instances of discussing alternatives involved editing an existing sketch and only 6% (± 5%) involved creating new sketches. Even when creating new sketches, designers rarely made comparisons between them by switching between alternatives in competing sketches. Instead, they discussed the alternatives. Designers almost never (2 sketches) discussed competing alternatives by explicitly representing each alternative in a separate sketch.

While sketches played a crucial role in helping designers to work through the ramifications of their choices, sketches often did not accurately reflect the current alternative under discussion. Nonetheless, designers discussed them as if they represented the most up-to-date state of the design. For example, one pair (C) discussed user interactions for creating roads, sketching alternatives in which roads were represented by either connected square blocks or by straight lines with notches. After deciding on the latter, they subsequently gestured at the first sketch, speaking as if it had lines with notches representing roads. In some cases, designers even pointed and gestured at imaginary sketches. For example, one designer (H) pointed at areas in a blank square and explained a design as if there were sketched content.

6 THREATS TO VALIDITY

There are three major threats to the validity of our results: the selection of participants, the choice of the design task, and the context in which the design task occurred.

First, our selection of participants may threaten the generalizabiliy of the results if our eight pairs are not representative of the general population of software designers. It is possible that the results might differ for designers with different backgrounds or specializations or who are familiar with other design approaches such as model-driven design or agile design. However, we mitigated this issue by recruiting participants with extensive experience in design from a variety of backgrounds. We recruited 16 designers from 7 different companies, each with its own

culture and design methodologies. Participants had a median of 17.5 years of experience as professional designers. Of course, as the designers were experienced, the results might not generalize to less experienced designers.

The second threat to validity is the choice of design task. Designers approach tasks based on their existing experience, including their knowledge in the domain. Individuals with extensive experience in a domain may already know which design alternatives are optimal or should be avoided, or use their intuition based on their experience to design more efficiently. In order to place designers on an 'even playing field', we chose a design task in a domain that most participants were likely to have encountered in their lives but were unlikely to have worked on professionally. This may have influenced how the designers worked. However, a previous study which used the same task [31] compared the overall design process to that of Olson et al. [37], who in turn compared the process to that used in practice, and found a great amount of similarity in practices.

The scope of the design task is also a factor. An overly-specific task might not allow designers sufficient creativity, artificially constraining the ways in which designers might naturally engage in a design process. Conversely, an overly-generic task might lead designers to approaches and solutions so divergent as to be incomparable. To mitigate this issue, we asked for a specific set of deliverables – a verbal summary of the design and an architecture to provide to a team of developers – but allowed the designers complete freedom through the design session in how they arrived at those deliverables. At the same time, we recognize that our task was still relatively small, compared to real-world design projects.

The third threat to validity is the context in which the design task was performed, including the time available for the design session and the environment in which the design work occurred. Design took place amongst pairs of designers, with equal responsibilities and knowledge of the task. In practice, design may involve groups larger than pairs and may sometimes involve designers with unequal knowledge, leading to participant behavior within sessions that is less balanced. Participants were instructed to write exclusively on the whiteboard rather than use other media such as paper; while designers in practice may design primarily using the whiteboard, this may be supplemented by use of mediums such as paper to record and make note of the design. Participants were limited to a single two-hour time span and had no access to other people or resources. In practice, design typically occurs over longer periods of time and involves many different stakeholders, with whom the designers may work and from whom they may obtain feedback during the design process. Design in practice may also occur during a wide variety of software lifecycle phases, often forcing designers to consider the existing design and the constraints that it imposes on the design task at hand. In contrast, our participants approached a new design task.

Despite these potential issues, the designers generally found their experiences during the study to be representative of their experiences in real world design. During the interview at the conclusion of the design session, some designers stated that the design discussions within the session matched those that they have in their own meetings. "This sort of discussion is very much what the engineers do … within the team working with project management and UI design cross-functionally." However, a few designers also reported that they adjusted the way they approached the design problem due to the limited amount of time. For example, one designer stated: "it was, I think, predictable and expected that we spend a lot of time on early discussion and progressively get quicker at making decisions and moving along and putting together the data model at the end."

7 Discussion

Our results reveal that sketches play a crucial role in supporting design conversations. In turn, variations in these conversations, both from designers' approach to the design problem and the evolution of their conversations themselves, influence the types of sketches designers draw and the visual syntactic elements within these sketches. We found that design conversations span small groups of sketches, with each sketch playing a distinct role in supporting the design conversation.

In the remainder of this section, we first list our main findings and then discuss these findings and their implications for tool design. Where our findings corroborate other reported studies, we provide a citation.

7.1 Main findings

7.1.1 *Types of sketches*

What types of sketches do designers create?

Designers create lists, tables, GUIs, ER diagrams, class diagrams, code, drawings, and domain specific sketches [8, 55]. Designers prefer general-purpose sketches that support the discussion of many scenarios over sketches specialized for an individual scenario. While designers sometimes use ER and class diagrams to design the system, domain sketches are crucial to system design [40].

Do the sketches created vary with respect to the approach to design taken?

Designers may approach a design problem from the perspective of the user interface, the system, or requirements, leading them to work with different types of sketches. Nevertheless, all designers made use of a common core of sketch types, as their design process moved beyond their initial focus to other aspects of their design.

How syntactically complex are sketches?

Overall, the average number of distinct visual syntactic elements in sketches ranged from 4.0 for lists to 11.8 for class diagrams. Designers very rarely reordered or rearranged lists, suggesting that order was either unimportant or was clear without exploration or external rep-

resentation.

How do sketches evolve?
Sketches increase in syntactic complexity as design conversations progress and designers introduce new syntactic elements to record aspects of the design [39]. Designers sometimes borrowed syntactic elements from other notations [14, 40]. As lists accumulated additional syntactic elements, they sometimes evolved into class and ER diagrams [14, 39].

7.1.2 *Focus and transitions*

How do designers shift their attention between sketches?
During design dialogues and periods of review, designers often transfer their focus repeatedly among a small group of sketches. These groups of sketches typically provide alternative views on some aspect of the design, for example sketches of different types or domains, sometimes across levels of abstraction (cf. [40]). Focus transitions most often to a sketch with which a designer has worked recently. They are rarely between sketches representing distinct alternatives or points in time.

How long do designers focus on individual sketches?
Designers move rapidly between sketches. Designers rarely (12 ± 5% of total focus periods) focused for more than 30 seconds on a single sketch, and often spent less than 10 seconds on a sketch (66 ± 9%).

How do designers make reference to sketches?
We observed that designers often referred to adjacent sketches in short bursts, even when they do not edit or add to them. Designers use momentary references to gather information, guide the attention of other designers by pointing, and combine information from sketches by dividing their focus.

7.1.3 *Reasoning activities*

How do designers use sketches to understand and advance the state of the design at hand?
Sketches play an important role in supporting reasoning activities: 81% of reasoning activities used sketches. Sketches most often support reasoning activities simply by providing information and something to gesture at. Only 22% of reasoning activities resulted in an edit to a sketch, and only 6% resulted in the creation of a new sketch.

Which types of reasoning activities do the sketches support?
Sketches support mental simulations, helping designers to propose new ideas, clarify and refine, and reveal implciations. Sketches support designers in reviewing their progress, helping designers to check the coverage of requirements, identify assumptions, and plan what to address next. Finally, sketches support the discussion of alternatives, most often through the use of existing sketches rather than through editing (18%) or creating sketches (5%).

Are the outcomes of design discussions always recorded in the sketches supporting these discussions?
Sketches did not represent the entirety of the design or even, for the parts described, its current state. Designers decided on solutions without writing them down and reused sketches from rejected alternatives to support reasoning about different design alternatives.

7.2 Sketches support design conversations

Designers made constant use of sketches throughout their design process, using the whiteboard to discuss design alternatives, tracing paths across the whiteboard to mentally simulate the behavior of the systems they designed, and examining sketches to review their progress. Yet, while sketches were central to designers' work, they did not represent the entirety of the design or even, for the parts described, the current state of the design. Instead, the design existed only in the minds of the designers and in the conversations in which they engaged. Designers decided on alternatives without writing them down, referred to sketches as if they were up to date, and gestured in space at imaginary sketches. As the design evolved through designers' conversations, sketches helped support these conversations, even when the evolution of the sketches did not keep pace with the evolution of the design.

Designers often worked with sketches that were general-purpose, enabling them to quickly simulate a wide range of scenarios by pointing to and annotating a general-purpose sketch, rather than drawing each scenario in a separate sketch. Much as a schematic of a car design might support a discussion of a maintenance issue, or a timeline might support a discussion of scheduling, general-purpose sketches at a higher level of abstraction supported the designers in diving into the discussion of many alternatives and many issues. In contrast to other normative and descriptive accounts of design at the whiteboard which emphasize the use of separate sketches of alternatives to compare, weigh tradeoffs, and synthesize [7, 58], designers almost never explored alternatives by creating separate sketches for each alternative. Rather, they pointed at existing sketches, even pretending that they matched the design that existed only in their minds. One potential explanation comes from the context of our study: designers performed early exploration of a design. This kind of design task may allow designers to be more creative and less constrained.

Designers' preference for general-purpose sketches also shaped the types of sketches designers used. Rather than sketching individual scenarios separately using sequence diagrams, designers used class and ER diagrams which afforded the flexibility to point at and rapidly flesh out multiple scenarios within a single sketch. Even when simulating the behavior of a design over time, designers evidently found it easier to point at and reason about el-

ements within a class diagram than to repeatedly draw sequence diagrams. Similarly, designers working with user interfaces built general-purpose mockups, simulating scenarios through pointing and annotating rather than sketching each state in a storyboard. This preference may reflect the demands of early, informal design in which many designs are considered in quick succession; designers did, for example, report that they would typically create sequence diagrams later in the design process. And designers did choose to externalize more in situations in which the design conversation became complex. For example, in designing a class as part of a larger class diagram, one pair (B) provided more detail by making a list.

Implications. These findings have important implications for the design of both fully interactive whiteboard systems and capture technologies such as electronic pens for physical whiteboards. First and foremost, such tools must deal with the discrepancy between design as it exists in designers' minds and as designers record it on a whiteboard: informal sketches provide support for discussing a design, rather than a record of the design. This has fundemental implications for tools intended to capture an accurate record of a design. Rather than simply record a static view of a sketch, finding ways to record the design process itself, including the reasoning activities enabled by sketching, is crucially important, as the design does not exist in the sketches but in the conversations and interactions they enable. For example, to capture a sketch used in a mental simulation, it may be more helpful to record the strokes designers draw, moment by moment, and designers' corresponding discussion, allowing designers to later play back a video capturing the entire walkthrough of an aspect of a system that took place in the mental simulation.

Second, tools automatically capturing sketches for long-term usage – such as in design documents, wikis, and issue trackers – must take into account that a single sketch may be used to discuss a range of topics. Given sufficient context, it may be possible for the original designer to identify the rationale of a decision from a sketch. But, as sketches are used for multiple purposes, it may be insufficient to simply record a sketch in one place. Thus, it again may be more effective for such tools to consider recording not simply sketches but the reasoning activities that occur with sketches, including the scenarios and alternatives annotated on top of sketches. And even out-of-date sketches may serve an important role in capturing the discussion of newer scenarios.

7.3 The approach to design influences the sketches designers draw

While all designers were provided with the same materials and were given the same prompt, three distinct approaches to design emerged, which in turn influenced the sketches designers drew. Some designers chose to focus on the behavior of the system, leading them to create system sketches. Others focused on the user interface, creating sketches describing user interactions. Others still designed from requirements, creating list sketches to work toward their designs. Yet all designers made use of a common core of sketch types, as their design process moved beyond their initial focus to other aspects of their design.

Previous work has identified the types of sketches produced during design activities [22, 47, 56]. Our findings extend these results, identifying the frequency of individual visual syntactic elements within several types of sketches.

Designers only infrequently made use of traditional notations for software design; of the 155 sketches designers created, 9 of them used a traditional notation, either a class or ER diagram. Instead, much of the design process took place in lists, used domain-specific sketch types to model the problem domain, or worked with user interface mockups. Even when designing the system, designers often did so while working with domain sketches, using them to work through scenarios and generate issues that the design should address. Design sketches are simply not limited to traditional notations such as UML, and informal representations are used throughout design exploration.

Designers used sketches to capture decisions as they were made, managing the level of detail in their sketches to be consistent with their current stage of the design process. This finding is consistent with existing work, such as Shipman's finding that forcing developers into a notation can degrade task performance by introducing overhead [47]. For example, designers sometimes enumerated the components in a system with lists, allowing decisions on component relationships to be deferred. Or, when working with large map sketches, the designers did not concern themselves with the movement of individual cars.

Implications. For sketching tools that seek to provide a fluid experience by avoiding unneeded formalism and supporting a minimal set of syntactic elements, our results begin to identify what those elements might be, identifying the most frequently used types of VSEs within sketches. For example, rather than necessarily provide the full expressiveness of the UML class diagram formalism, our results suggest that an early stage tool for supporting informal design might initially let designers sketch more simply with a subset of these elements. Moreover, our results also illustrate the importance of sketch types beyond traditional system notations, including lists, tables, and representations of the domain. These, too, should be equally supported by electronic design tools. And, as design often takes place across multiple sketches, it is crucially important for tools to support the juxtaposition of sketches of multiple types.

These results also suggest that it is crucially important for sketching tools to consider, in detail, exactly what information notations ask designers to provide and how this matches the information designers are ready to express at various points in the design process. Our results also suggest that tools should provide a gradual way of

conforming to a notation, as designers borrow from multiple sketch types, use impromptu notations, and use formal notations informally. Designers evolve sketches
Our results reveal that designers continue to add new types of visual syntactic elements as they work with a sketch. As the focus of designers' conversations evolved through the design process, so the sketches that designers used to support these conversations evolved as well. For example, designers often began sketches as a list, typically containing only first-order elements and a title. As design conversations progressed and designers wished to express more details about the design, these lists often evolved, adding additional syntactic elements such as second-order elements and titles. Designers rarely reordered or rearranged lists and rarely attempted to encode information in the order of lists. Confirming existing findings [45], the designers sometimes reappropriated sketches, for example adding boxes and arrows as they evolved lists into class diagrams; in some cases designers redrew these sketches. Beginning these sketches as lists enabled the designers to defer discussion of aspects of the design such as relationships between elements until later in the design session.

Depending on the situation, designers may choose either to redraw a sketch or to develop it in place. Redrawing forces a commitment to a decision, as it replaces a provisional hastily drawn sketch with a more complete representation and removes any annotations that may be present. Redrawing creates space for more detail, enabling decisions to be made about aspects of the design for which there may previously have been no space (e.g., fields in a class diagram). In contrast, evolving sketches supports design thinking 'in-the-moment', as designers can spontaneously record decisions in their sketches.

Our results also highlight the support for provisional and informal design as afforded by the nature of the medium of whiteboards [39]. A previous study of informal software design using poster boards and markers found that sketches did not evolve in place, as designers instead redrew sketches [14, 41]. Our results suggest that redrawing, rather than evolving, might make it more challenging for designers to record decisions in the moment and may force designers to commit to decisions earlier. Of course, other differences between media might also have an effect on the design process. Writing on large sheets of paper that can be torn off and posted to a wall affords more space than a whiteboard, allowing more alternatives to be sketched and old sketches to be retained rather than erased.

Implications. Understanding the differences between the roles of redrawing, reappropriating, and evolving sketches – and understanding the effect of the drawing medium on these different activities – is particularly important for interactive sketching tools, as the facilitative characteristics of a particular medium should be preserved in an electronic tool. It is thus important for future work to better understand the effect of the medium on the design process.

The results reaffirm that a key aspect of the whiteboard experience is its fluidity in enabling sketches to evolve, and sketching tools that wish to maintain this fluidity must provide support for evolving sketches. The fluidity of notations has profound implications for sketch tool (and metatool, e.g., [20]) developers. Premature commitment to a notation should be avoided, with where and when sketches get interpreted and turned into more precise notations through reconition determined by the users. Sketching tools should allow designers to begin with simple, reduced notations (used informally) and then add additional elements as needed. For example, a UML tool might allow designers to first list names of classes and then support the development of the list of class names into a class diagram.

Sketching tools can also enable designers to evolve sketches more easily by, for example, helping them to rapidly 'make space', enabling ideas to be rapidly developed in the moment without forcing them to be redrawn. Yet this might introduce other problems, as early sketches may no longer be available for reference. Thus, tool support for recording and replaying the design history may become more important.

Designers often annotated their sketches with VSEs that had no inherent semantic meaning but instead served to guide the attention of a sketcher's audience during an explanation. This finding has two implications for tools. First, tools that attempt to interpret and assign meaning to VSEs within sketches should be able to support VSEs that do not have any fixed semantics. Second, given that these VSEs arise when a designer is explaining a part of their design, there is an opportunity to capture and preserve information about the explanation, such as which sketches designers step through, or to infer importance of elements within a sketch based on how frequently they are annotated.

7.4 Designers work with small groups of sketches

Our results reveal that design conversations span groups of sketches. Designers rarely stayed focused on a sketch for a long period of time but instead rapidly shifted their attention between sketches, frequently spending less than 10 seconds at a single sketch. While each sketch contained specific information and supported specific types of reasoning, design conversations often cut across this structure, drawing together information from multiple sources or using the information in one sketch to support the design activities in another. As a result, pairs often rapidly moved back and forth between small groups of sketches: over 60% of focus transitions were to return to the previously viewed sketch. A pair's attention almost always spanned a fraction of the total whiteboard sketches, including an average of 2 sketches every minute and 3.9 sketches every 5 minutes. Designers worked with groups of sketches in a variety of ways, each serving a unique function: designers worked with sketches for periods of time, gathered information through quick glances, point-

ed to draw attention during conversations or to guide review or simulation, and split their focus in order to reason about the design using information from related sketches.

An interesting question for future work is to better understand how these groups of sketches are structured. How stable are groups, and how much do they evolve as the design conversations evolve? Our results suggest that some sketches were very related, as in several sketches describing related aspects of a GUI. Other relationships may depend more strongly on the design activity at hand, as in a list that describes the behavior of a particular element in an ER diagram, or sketches that are used together only when using a requirements list to review progress.

Implications. These results highlight the importance of designing support for working with groups of sketches within interactive sketching systems. The medium of interactive whiteboards introduces both challenges and opportunities in this regard. On one hand, interactive whiteboards may be physically smaller than large physical whiteboards and may force designers to draw larger due to the difficulty of capturing fine movements [30]. This reduces the number of sketches a designer can draw on a single whiteboard. On the other hand, interactive whiteboards can allow designers to interact with many virtual whiteboards, for example alternating among different virtual whiteboards, or juxtaposing reduced views of related virtual whiteboards. As designers constantly interact with groups of sketches, our results suggest this is crucially important to support.

8 Conclusion

Sketching is a central and fundamental activity in the process of software design (just as it is in other design disciplines). Through the systematic study presented in this paper, we examined the role of sketching within the design process in detail, examining the types of sketches designers create, how designers focus on sketches and transfer their focus between sketches, and how sketches support reasoning within design activities.

Our findings are rich, some confirming what is already reported in the literature (e.g., designers create a variety of types of sketches; sketching helps in discussing alternatives), others adding new knowledge (e.g., designers generally work with multiple sketches at the same time; designers opt to discuss alternatives over existing sketches instead of representing these alternatives explicitly on the whiteboard). By the same token, some of our findings align relatively well with how we think about current software design tools (e.g., designers need tools that allow them to evolve their sketches; designers reason over their sketches in order to understand and advance the design at hand), others conflict seriously with current thinking and seem to demand the creation of new types of tools (e.g., designers move very rapidly among sketches; sketches rarely follow notational convention strictly; designers sometimes reappropriate sketches and transform them into a different sketch type). Together, our findings paint a more complete and complex picture of the relationship between the software design process and sketches made at the whiteboard than has been portrayed previously. The insights underpinning this picture have important implications for the design of interactive whiteboards that seek to reconceive this relationship, suggesting attributes of physical whiteboards that should be preserved and those that need to be re-envisioned.

Our future work will focus on two major research directions. First, we would like to perform *longitudinal* studies of whiteboard use, as we suspect that there are behaviors that play out across software design meetings that also need to be understood. Second, we will continue the development of our Calico [29] software design sketching tool to incorporate many of the lessons learned here and to evaluate their impact empirically.

Acknowledgments

We thank all of the participants in our study and the participants of the Studying Professional Software Design Workshop. This work was supported in part by Microsoft and by the National Science Foundation under grants CCF-1118052 and IIS-1111446.

Nicolas Mangano received degrees in psychology and information and computer science from University of California, Irvine in 2007 and a Ph.D. in information and computer science from the University of California, Irvine in 2013. His research focused on understanding and supporting informal software design at the whiteboard, including in globablly distributed setting and with electronic whiteboards. His dissertation on Calico: An early-phase software design tool won the 2014 ACM SISGSOFT Outstanding Doctoral Dissertation Award. He serves as co-founder of Molimur, a company whose mission is to create creative collaboration tools.

Thomas D. LaToza received degrees in psychology and computer science at the University of Illinois, Urbana-Champagin in 2004 and a Ph.D. in software engineering at Carnegie Mellon University in 2012. He is currently a postdoctoral research associate at the University of California, Irvine. His research focuses on human aspects of software development, including supporting information needs in software development, software design at the whiteboard, and crowdsourcing software engineering. He has served on several program committees and is currently an organizer of the Workshop on Crowdsourcing in Software Engineering.

Marian Petre is a Professor of Computing at The Open University in the UK. Her background includes a B.A. in Psycholinguistics from Swarthmore College and a Ph.D. in Computer Science from University College London. She received a Royal Society Wolfson Research Merit Award in recognition of her research on expert software design behaviour and reasoning. She also held an EPSRC Advanced Research Fellowship, again researching expert reasoning and representation in software design. Her research earned the Design Studies Award (2004) and the ACM SIGSOFT Distinguished Paper Award (at ICSE 2013) for 'UML in practice'. Her research is reflected in over 100 international refereed publications, plus co-authored books including '*Software Designers in Action*'. She is a founding member of the Psychology of Programming Interest Group.

André van der Hoek received joint B.S. and M.S. degrees in business-oriented computer science from the Erasmus University Rotterdam, The Netherlands, and a Ph.D. degree in computer science from the University of Colorado at Boulder. He serves as chair of the Department of Informatics at the University of California, Irvine. He heads the Software Design and Collaboration Laboratory, which focuses on understanding and advancing the roles of design, collaboration, and education in software development. He has served on numerous international program committees, is a member of the editorial board of *ACM Transactions on Software Engineering and Methodology*, was program chair of the 2010 ACM SIGSOFT International Symposium on the Foundations of Software Engineering, and is program co-chair of the 2014 International Conference on Software Engineering. He was recognized in 2013 as an ACM Distinguished Scientist, and in 2009 he was a recipient of the Premier Award for Excellence in Engineering Education Courseware. He is a member of the IEEE.

References

[1] A. Baker and A. v. d. Hoek, "The Design Process as Rotating Subject Pairs," presented at the Studying Professional Software Design Workshop, 2010.

[2] A. Baker and A. van der Hoek, "Ideas, subjects, and cycles as lenses for understanding the software design process," *Design Studies,* vol. 31, pp. 590-613, 2010.

[3] L. J. Ball and B. T. Christensen, "Analogical reasoning and mental simulation in design: two strategies linked to uncertainty resolution," *Design Studies,* vol. 30, pp. 169-186, 2009.

[4] L. J. Ball, B. Onarheim, and B. T. Christensen, "Design requirements, epistemic uncertainty and solution development strategies in software design," *Design Studies,* vol. 31, pp. 567-589, 2010.

[5] J. Bertin, *Semiology of graphics: diagrams, networks, maps*: University of Wisconsin press, 1983.

[6] G. Booch, "Draw Me a Picture," *IEEE Softw.,* vol. 28, pp. 6-7, 2011.

[7] W. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*: Morgan Kaufmann, 2007.

[8] 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*, pp. 557-566, 2007.

[9] H. Christiaans and R. A. Almendra, "Accessing decision-making in software design," *Design Studies,* vol. 31, pp. 641-662, 2010.

[10] R. Chung, P. Mirica, and B. Plimmer, "InkKit: a generic design tool for the tablet PC," In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural* (CHINZ '05), pp. 29-30, 2005.

[11] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM,* vol. 31, pp. 1268-1287, 1988.

[12] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool Support for Cooperative Object-Oriented Design: Gesture Based Modelling on an Electronic Whiteboard," *Proceedings of the SIGCHI conference on Human factors in computing systems,* pp. 518-525, 2000.

[13] U. Dekel, "Supporting distributed software design meetings: what can we learn from co-located meetings?," In *Proceedings of the 2005 workshop on Human and social factors of software engineering* (HSSE '05), pp. 1-7, 2005.

[14] U. Dekel and J. D. Herbsleb, "Notation and representation in collaborative object-oriented design: an observational study," *SIGPLAN Not.,* vol. 42, pp. 261-280, 2007.

[15] E. Ferguson, *Engineering and the Mind's Eye*. Ma. and London: Cambridge, 1992.

[16] F. Guimbretiere, M. Stone, and T. Winograd, "Fluid interaction with high-resolution wall-size displays," In *Proceedings of the 14th annual ACM symposium on User interface software and technology* (UIST '01), pp. 21-30, 2001.

[17] V. Goel, *Sketches of Thought*. Cambridge, Massachusetts: The MIT Press, 1995.

[18] G. Goldschmidt, "The dialectics of sketching," *Creativity Research Journal,* vol. 4, pp. 123-143, 1991.

[19] P. Grisham, H. Iida, and D. Perry, "Improving Design Intent Research for Software Maintenance," in *ATGSE 07: Accountability and Traceability in Global Software Engineering*, 2009.

[20] J. Grundy and J. Hosking, "Supporting Generic Sketching-Based Input of Diagrams in a Domain-Specific Visual Language Meta-Tool," in *Proceedings of the 29th International Conference on Software Engineering* (ICSE '07), pp. 282-291, 2007.

[21] R. Guindon, "Designing the design process: Exploiting opportunistic thoughts," *Human–Computer Interaction,* vol. 5, pp. 305-344, 1990.

[22] D. Hendry, "Sketching with Conceptual Metaphors to Explain Computational Processes," in *Visual Languages and Human-Centric Computing*, Washington, DC, pp. 95-102, 2006.

[23] C. D. Hundhausen, "Using end-user visualization environments to mediate conversations: a 'Communicative Dimensions' framework," *Journal of Visusal Languages and Computing,* vol. 16, pp. 153-185, 2005.

[24] G. Johnson, M. D. Gross, J. Hong, and E. Y.-L. Do, "Computational Support for Sketching in Design: A Review," *Foundations and Trends in Human–Computer Interaction,* vol. 2, pp. 1-93, 2008.

[25] W. Ju, B. Lee, and S. Klemmer, "Range: exploring implicit interaction through electronic whiteboard design," In *Proceedings of the 2008 ACM conference on Computer supported cooperative work* (CSCW '08), pp. 17-26, 2008.

[26] S. Klemmer, M. Newman, R. Farrell, M. Bilezikjian, and J. Landay, "The designers' outpost: a tangible interface for collaborative web site," In *Proceedings of the 14th annual ACM symposium on User interface software and technology* (UIST '01), pp. 1-10, 2001.

[27] J. Landay, "SILK: sketching interfaces like krazy," In *Conference Companion on Human Factors in Computing Systems* (CHI '96), pp. 399, 1996.

[28] J. Larkin and H. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words**," *Cognitive science,* vol. 11, pp. 65-100, 1987.

[29] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," In *Proceedings of the 28th international conference on Software engineering* (ICSE '06), pp. 492-501, 2006.

[30] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "Supporting Informal Design with Interactive Whiteboards," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, 10 pages, 2014.

[31] J. McDonnell, "Accommodating disagreement: A study of effective design collaboration," *Design Studies,* vol. 33, pp. 44-63, 2012.

[32] N. Mangano and A. van der Hoek, "The design and evaluation of a tool to support software designers at the whiteboard," *Automated Software Engineering,* vol. 19, pp. 381-421, 2012.

[33] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *Software Engineering, IEEE Transactions on,* vol. 35, pp. 756-779, 2009.

[34] E. Mynatt, T. Igarashi, W. Edwards, and A. LaMarca, "Flatland: New dimensions in office whiteboards," In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (CHI '99), pp. 346-353, 1999.

[35] E. D. Mynatt, "The writing on the wall," in *Proceedings of the 7th IFIP Conference on Human-Computer Interaction*, 1999.

[36] K. Nakakoji and Y. Yamamoto, "Conjectures on how designers interact with representations in the early stages of software design," in *Software Designers in Action: A Human-centric Look at Design Work*, ed: Chapman and Hall/CRC, pp. 381 - 399, 2014.

[37] A. Newell and H. Simon, *Human problem solving* vol. 104: Prentice-Hall Englewood Cliffs, NJ, 1972.

[38] M. Newman, J. Lin, J. Hong, and J. Landay, "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice," *Human-Computer Interaction,* vol. 18, pp. 259-324, 2003.

[39] G. Olson, J. Olson, M. Carter, and M. Storrosten, "Small group design meetings: An analysis of collaboration," *Human–Computer Interaction,* vol. 7, pp. 347-374, 1992.

[40] H. Ossher, R. K. E. Bellamy, B. E. John, and M. Desmond, "Concern development in software design discussions: implications for flexible modeling," in *Software Designers in Action: A Human-centric Look at Design Work*, ed: Chapman and Hall/CRC, 2014, pp. 295-313.

[41] M. Petre, "Insights from expert software design practice," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 233-242, 2009.

[42] M. Petre, "UML in practice," In *Proceedings of the 2013 International Conference on Software Engineering* (ICSE '13), IEEE Press, pp. 722-731, 2013.

[43] M. Petre, "Reflections on representation: cognitive dimensions analysis of whiteboard design notations," in *Software Designers in Action: A Human-centric Look at Design Work*, ed: Chapman and Hall/CRC, pp. 267-294, 2014.

[44] M. Scaife and Y. Rogers, "External cognition: how do graphical representations work?," *International Journal on Human-Computer Studies,* vol. 45, pp. 185-213, 1996.

[45] D. A. Schön, *The reflective practitioner: How professionals think in action* vol. 5126: Basic Books, 1984.

[46] J. Schumann, T. Strothotte, S. Laser, and A. Raab, "Assessing the effect of non-photorealistic rendered images in CAD," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '96), Vancouver, British Columbia, Canada, pp. 35-51, 1996.

[47] F. Shipman and C. Marshall, "Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems," *Computer Supported Cooperative Work (CSCW),* vol. 8, pp. 333-352, 1999.

[48] H. A. Simon, "The ill structure of ill-structured problems," *Artificial Intelligence,* vol. 4, pp. 181-204, 1973.

[49] H. A. Simon, *The Sciences of the Artificial, 3rd Ed.*: MIT Press, 1996.

[50] A. Strauss and J. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory*: Sage Publications, Incorporated, 2007.

[51] M. Suwa, J. Gero, and T. Purcell, "Unexpected discoveries and S-invention of design requirements: important vehicles for a design process," *Design Studies,* vol. 21, pp. 539-567, 2000.

[52] M. Suwa and B. Tversky, "External Representations Contribute to the Dynamic Construction of Ideas," In *Proceedings of the Second International Conference on Diagrammatic Representation and Inference* (DIAGRAMS '02), pp. 341-343, 2002.

[53] H. Tang, Y. Lee, and J. Gero, "Comparing collaborative co-located and distributed design processes in digital and traditional sketching environments: A protocol study using the function–behaviour–structure coding scheme," *Design Studies,* vol. 32, pp. 1-29, 2011.

[54] B. Tversky, "What do Sketches say about Thinking," 2002.

[55] A. Van Der Hoek and M. Petre, *Software Designers in Action: A Human-centric Look at Design Work*: Chapman and Hall/CRC, 2013.

[56] J. Walny, S. Carpendale, N. Henry Riche, G. Venolia, and P. Fawcett, "Visual thinking in action: Visualizations as used on whiteboards," *Visualization and Computer Graphics, IEEE Transactions on,* vol. 17, pp. 2508-2517, 2011.

[57] Y. Y. Wong, "Rough and ready prototypes: lessons from graphic design," presented at the Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems, Monterey, California, 1992.

[58] K. Yatani, E. Chung, C. Jensen, and K. N. Truong, "Understanding how and why open source contributors use diagrams in the development of Ubuntu," in *Proceedings of the 27th international Conference on Human Factors in Computing Systems*, pp. 995-1004, 2009.

[59] C. Zannier, M. Chiasson, and F. Maurer, "A model of design decision making based on empirical results of interviews with software designers," *Information and Software Technology,* vol. 49, pp. 637-653, 2007.

[60] C. Zannier and F. Maurer, "Comparing Decision Making in Agile and Non-agile Software Organizations," in *Agile Processes in Software Engineering and Extreme Programming*, ed: Springer Berlin / Heidelberg, 2007.