UNIVERSITY OF CALIFORNIA,
IRVINE

Calico: An early-phase deisgn tool to support the design behaviors of software designers at the whiteboard

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Nicolas Francisco Mangano

Dissertation Committee:
Andre' van der Hoek, Chair
Professor David Redmile
Professor Gary Olson

2013

DEDICATION

(Optional dedication page)
To ...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank...

(You must acknowledge grants and other funding assistance.

You may also acknowledge the contributions of professors and friends.

You also need to acknowledge any publishers of your previous work who have given you permission to incorporate that work into your dissertation. See Section 3.2 of the UCI Thesis and Dissertation Manual.)

CURRICULUM VITAE

Nicolas Francisco Mangano

EDUCATION

Doctor of Philosophy in Computer Science **2012**
University name *City, State*

Bachelor of Science in Computational Sciences **2007**
Another university name *City, State*

RESEARCH EXPERIENCE

Graduate Research Assistant **2007–2012**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2009–2010**
University name *City, State*

REFEREED JOURNAL PUBLICATIONS

Ground-breaking article **2012**
Journal name

REFEREED CONFERENCE PUBLICATIONS

Awesome paper **Jun 2011**
Conference name

Another awesome paper **Aug 2012**
Conference name

SOFTWARE

Magical tool `http://your.url.here/`
C++ algorithm that solves TSP in polynomial time.

ABSTRACT OF THE DISSERTATION

Calico: An early-phase deisgn tool to support the design behaviors of software designers at the whiteboard

By

Nicolas Francisco Mangano

Doctor of Philosophy in Computer Science

University of California, Irvine, 2013

Andre' van der Hoek, Chair

The abstract of your contribution goes here.

Chapter 1

Introduction

Often when developers are faced with a design challenge, they will turn to the whiteboard. This is typical during the conceptual stages of software design, when no code is in existence yet, but may also happen when a significant code base has already been developed, for instance, to plan new functionality or discuss optimizing a key component. Design sessions at the whiteboard may even arise spontaneously, such as when a developer has to refactor some code or discuss how to best integrate a new feature.

Compared to the polished, precise, and typically detailed models software designers produce using modern software design tools, the content they create at the whiteboard consist of rough sketches and imprecise approximations of the design they have in mind, which are continuously modified and refined as part of the design activity [3]. The sketches on the whiteboard wall shown in Figure 1.1 illustrate this point. The sketches are the result of many design sessions at a startup software company. While the sketch clearly is unintelligible to those who were not present during the design sessions, they serve a crucial role for those who were: they were the vehicle for working through a complex design problem and making key decisions that defined the software to be developed.

Figure 1.1: Whiteboard wall at a startup company.

Developers turn to the whiteboard for the flexibility and fluidity that it offers in the design experience [12]. On a whiteboard, developers can freely sketch, branch off to another part of the design problem, return to a previous part, erase some portion of their work, redraw it, and so on, all without the typical restrictions one might find in a traditional software design environment. They can focus on designing without worrying about particular notations, how to navigate within a tool, etc.

While a preferred medium for design, a significant disadvantage of the whiteboard is that it is a passive medium: it has no facilities that purposefully support the design process. Particularly, whatever is drawn or written remains static and cannot be manipulated, other than drawing over it or erasing it. This is a less than desirable situation, because it is known that software designers often wish to manipulate a design at hand in more advanced ways than merely adding or erasing content [24]. The whiteboard, thus, limits what they are able to do.

The research community has acknowledged this problem and has contributed many approaches that rely on an electronic whiteboard to provide more advanced support (e.g., [10, 47, 36, 22, 14]). This support can be divided into two broad categories: (1) approaches

that focus on sketch recognition, and (2) approaches that focus on management of sketched content. Approaches in the first category, sketch recognition, attempt to interpret the strokes made by the user to turn them into formal objects. Early work offered a predefined visual vocabulary for converting sketches into formal objects, such as UML diagrams [10] or user interface mockups [47], with tools that provided feedback to the designer based on the rules of the formal notation they support. Later work made visual vocabularies expandable by users [36] and made using the tools more flexible by delaying interpretation until it was desired by the user [22], sometimes even while retaining a sketchy appearance [14].

Approaches in the second category, sketch management, help organize the potentially many and varied sketched artifacts that may be produced during meetings (Figure 1.1 is an illustration of this point). Early approaches provided access to a large number of whiteboards through a filmstrip [84], hyperlinks [85], or hierarchical perspectives [60]. Later work automated particular aspects of managing sketches by automatically grouping clusters of sketches in close spatial proximity [58], shrinking sketches when moved to the periphery [34], or using metaphors such as Post-It Notes to organize and relate sketches [43].

In examining these and other existing sketching tools, it is useful to consider their respective underlying motivations. In so doing, I observe that every sketch tool was designed to support a particular way of working at the whiteboard. For instance, Knight supports designers in refining initial rough sketches into more formal representations [22]. As another example, Flatland supports designers in creating many different diagrams by automatically clustering sketches and adding specialized behaviors to those clusters [58].

In this paper, I define these ways of working as design behaviors. More precisely, I define a design behavior as a recurrent, recognizable set of actions serving a single purpose within a design meeting. In this dissertation, I scope these actions to pertain to the whiteboard, that is, focus on creating and modifying its content, navigating the content created, and collaborating in all of these. That is, two designers calling up a colleague is not a design

behavior in which I am interested. One designer refining a diagram drawn by another is.

Quite a few design behaviors have been identified in the literature, despite the fact that the study of software designers "in action" is still in its infancy. For example, in addition to refinement of sketches and supporting multiple different types of sketches, studies of designers at OOPSLAs DesignFest found that software designers improvise their own notations and evolve their diagrams across many canvases [24]. As another example, in-the-field observations at software companies found that software designers deliberately switch among formalisms and use provisionality to engage in a dialog with incomplete ideas [72].

Broadly speaking, design behaviors that software designers perform can be be described as falling into three high level categories . The first is how they create and modify the kinds of things they darw, such as the different types of sketches they create and their improvisation of notations. The second is how they navigate those sketches, such as shifting focus between sketches of different types. The third is how they collaborate over their sketches, such as when designers switch from working together on a sketch synchronously to working asynchronously.

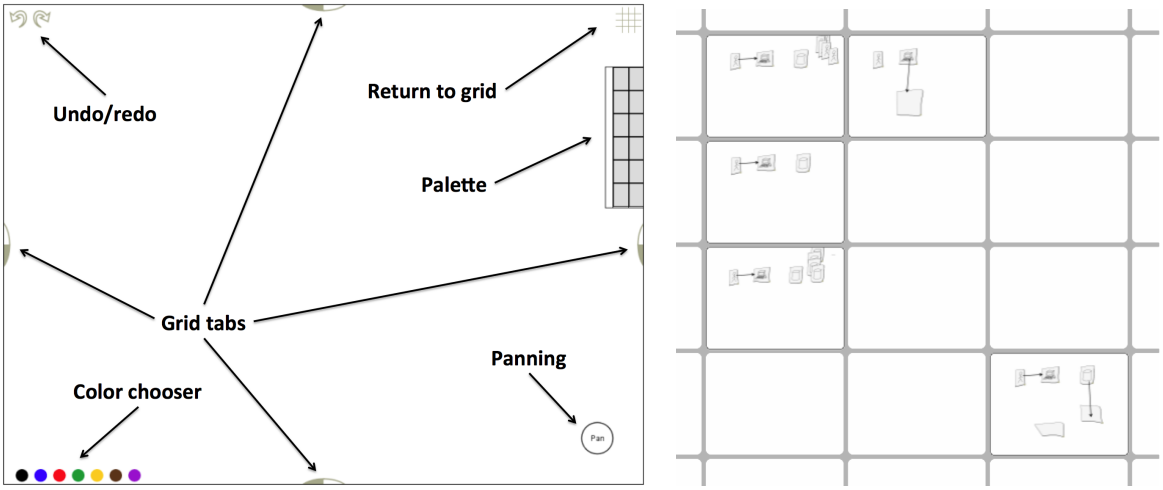
The key insight motivating our research is that software designers do not "operate in" or apply just one behavior for an entire design meeting. Rather, designers interleave design behaviors over the course of a design meeting, switching among them as they see fit to navigate a design problem and its potential solutions. For instance, a designer may first sketch two diagrams and juxtapose them side-by-side to evolve them in parallel, then record patterns of execution in one of the diagrams using an impromptu notation, and thereafter shift to a different aspect of the design problem altogether. Throughout, the designer fluidly exhibits a range of these different design behaviors, typically without an explicit trigger. Designers shift opportunistically, addressing the part of the design they see as most important at the moment.

During a meeting, it is a natural choice for designers to limit themselves to a single tool

to support them. That tool is typically a whiteboard or paper [72], though in some cases it may be a computerized tool like the ones I have described above (e.g., SUMLOW [10], Knight [20], Flatland [58]. In the latter case, the choice of tool determines the behavior or small set of behaviors that are now supported, as designers will seldom move between tools during meetings, because of the high cost associated with switching, both in terms of the cognitive burden on the user to switch contexts and in terms of the effort required to import or manually copy the contents. The cost is simply too great and designers, thus, are stuck with support for at best a few of their behaviors as embedded in the tool they happen to be using.

What is desired is a tool that supports a broad range of behaviors and allows developers to use the features naturally when they need them. Creating such a tool, however, is a non-trivial exercise. Simply picking up functionality from one tool and dropping it in another, and doing this repeatedly to support a multitude of behaviors, leads to tools that are highly disjoint. It is unclear, for instance, what it would mean for a tool to have available both multiple canvases in a filmstrip and functionality that automatically makes room on the current board? As another example, a tool that automatically recognize sketches and also support emergent notations is equally different to envision as being conceptually clear to its users. Existing solutions do not necessarily stack their functionality gracefully, and the approach taken by one tool may collide with the support provided by another.

This dissertation explicitly addresses the interleaving behaviors that designers exhibit during software design at the whiteboard by taking a step back, examining a collection of behaviors, and contributing a new tool that is designed from the ground up to support this collection of behaviors with a small set of conceptually coherent functionalities. In order to put this into practice, I first built a basic sketching interface so that, as a baseline, designers may sketch and perform the same activities as they normally would on a whiteboard, but in a digital medium. Building on this foundation, I incrementally introduced features that support one



(a) Canvas perspective

(b) Grid perspective

Figure 1.2: Calico version one

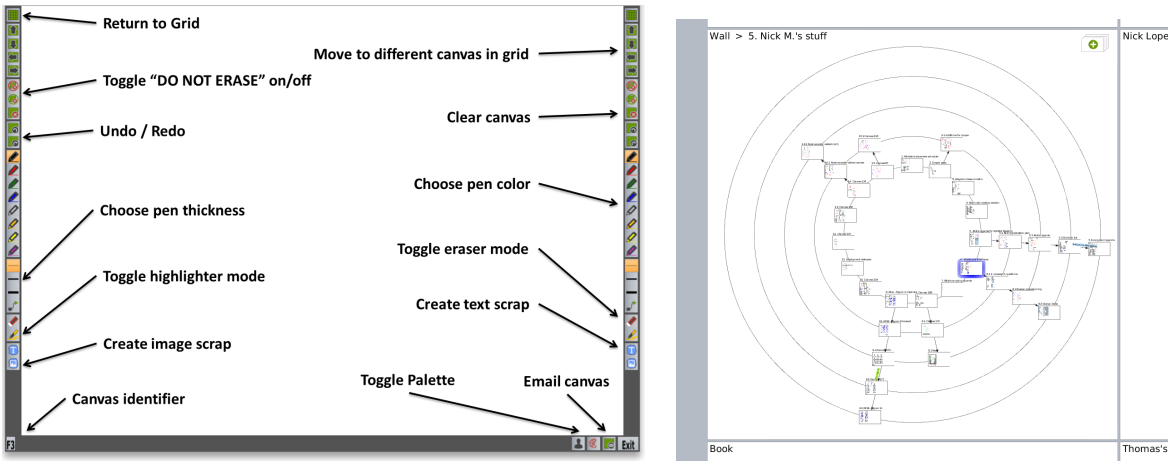
or more design behaviors, yet minimally obstruct the performance of other behaviors.

This approach was realized by the construction of a software tool called Calico. In this dissertation I describe two versions of the tool, the first of which is an initial exploration in supporting a subset of the design behaviors with a sketch-based tool, and the second of which significantly iterates on the first to support the complete set of design behaviors I set out to support.

The first version of Calico supported an initial subset of the design behaviors with three main features. Figure 1.2 depicts the interface for this version of Calico. The first feature, scraps, supported the kinds of sketches that designers created by providing a mechanism to create representations for box-and-arrow type diagrams and for manipulating sketches. The second feature, the grid, supported designers in navigating between sketches by providing a grid-layout of all canvases. Depicted in Figure 1.2(b), the users can partition their work over the canvases in the grid, and move the resulting canvases around to organize their work. The third feature, the palette, allowed users to save scraps (depicted on the right of Figure 1.2(a)), and reuse them by dragging them back onto the canvas. Chapter 3 explains these features in greater detail.

Calico version one was evaluated in a controlled laboratory study in which I compared the design activity of computer science graduate students engaged in a challenging software design problem using Calico against those using a regular whiteboard. I presented all participants with a prompt to design an educational traffic simulator and asked them to either design it using either Calico or the whiteboard. I found that the same design behaviors performed at the whiteboard were also performed in Calico, but in a slightly different manner. The participants in the Calico groups did use Calico's advanced feature set of perform the design behaviors, and reported that they found the features useful. However, the participants rarely used the palette. The design conversation of the groups were also analyzed using protocol analysis, which was done by breaking down the sessions into segmented phrases belonging to different categories of design. The breakdown of the categories demonstrated a very high correlation with reports from past research of how software design was performed at actual software companies, providing evidence that the design as it happened in our study matched that of design "in the wild". Lastly, the groups reported that the limitation of the interactive whiteboard I used to only accept a single person drawing at a time limited their ability to perform parallel work.

Following our experiences with Calico version one, I rebuilt the tool from the ground up in Calico version two. The features in the second version of Calico significantly iterate on those in the first version to support the whole set of fourteen design behaviors I identified as crucial to designers at the whiteboard. This version is depicted in Figure 1.3. The scraps feature was significantly revised based on user feedback to better support the designers in the kinds of sketches designers create. The grid feature was replaced with intentional interfaces, depicted in Figure 1.3(b), in order to better support designers in navigating between projects and the canvases capturing each project. In our experience with the grid, I saw that users typically dedicated entire rows or columns to a single topic. That behavior inspired the intentional interface view, in which canvases are layed out in a circle, and a set of canvases focusing on a particular topic can extend outwards radially (Figure 1.3(b)). Relationships between



(a) Canvas perspective (b) Intentional interfaces perspective

Figure 1.3: Calico version two

associated canvases are explicitly captured as well using light-weight tagging. Lastly, the software architecture of Calico itself was rebuilt using a client-server architecture in order to allow designers on different machines to collaborate on the same sketch. Chapter 5 explains these features, as well as others, in more detail.

Calico version two was evaluated by deploying it in the field to two software companies over a period of several weeks. [More to come after field deployment!]

1.1 Thesis structure

This thesis is organized into nine chapters. The remaining chapters are structured as follows:

Chapter 2 - Motivation. This chapter introduces the definition of design behaviors in detail and presents the grand set of design behaviors that I aim to support. The research question of this thesis is also introduced here.

Chapter 3 - Calico Version One. I describe the approach and features of the first version of Calico. This chapter introduces the first concepts of scraps, the grid, as well

as the palette. The approach is evaluated in a comparative study between Calico and the regular whiteboard, both with respect to the design behaviors performed as well as the design conversations that took place.

Chapter 4 - Notations. In order to ground the design behaviors introduced in Chapter 2 and observed in the evalution of Calico version one, I perform a quantitative and qualitative analysis of the use of notations by professional software designers designing at the whiteboard. The professional software designers engaged in the same task as the computer science graduate students in Chapter 3.

Chapter 5 - Calico Version Two. The second and final version of Calico is introduced in this chapter. The new features are introduced, including the revised interaction for scraps, the intentional interfaces features that replaces the grid, the distributed nature of the architecture, as well as other features targeted at supporting the full set of design behaviors.

Chapter 6 - Evaluation. Here I present our evaluation of the final version of Calico "in the wild" at two software companies. [Note: more to come after field trial!].

Chapter 7 - Discussion. In this chapter, I take a step back and put my work in a broader context. I discuss the implications of my findings, reconnect my work to the broader design literature, and provide some findings and observations that are outside the evaluative framework of the design behaviors, but are worthwhile to highlight nonetheless.

Chapter 8 - Related Work. I review the existing literature of tools that support software design on an electronic whiteboard, covering both tools that support sketch recognition and those that support the management of sketches.

Chapter 9 - Conclusions. This chapter summarizes the contributions of my dissertation, and also provides some concluding remarks.

Chapter 10 - Future Work. This final chapter suggests possible future avenues of research

for this dissertation, including the concept of compositional notations, which builds on scraps to provide more targeted support for software design notations.

Chapter 2

Motivation

2.1 Design Behaviors

Before introducing the design behaviors that I aim to address in our research, it is useful to return to the definition of a design behavior and explore it in greater detail. *A design behavior is a recurrent, recognizable set of actions serving a single purpose within a design meeting.* I make four important observations about this definition:

- ***recurrent*** – A design behavior can be observed to happen consistently across many design meetings and across many designers. A particular sketch seen once or twice in a meeting does not qualify as a design behavior. It must be a repeated and general.
- ***recognizable*** – A design behavior stands out as a coherent set of actions within an overall design meeting. The actions clearly belong together and can be distinguished from other groups of actions.
- ***set of actions*** – A design behavior necessarily unfolds over time with multiple actions that a designer undertakes. A single stroke or gesture does not qualify.

- ***serving a single purpose*** – A design behavior has a purpose in the overall exploration of a design problem and its potential solutions. The set of actions contribute to furthering is exploration.

With this definition in hand, I examined both the software design literature and the broader design literature for design behaviors. What I found was that the general design literature is more mature than the software design literature, in that it has identified quite a few design behaviors that span across different design disciplines. For instance, designers across building architecture, engineering, and product design use constraints to guide their design thinking [16]. As another example, designers in multiple fields use visual similarity (i.e., how they foresee the final product) in their sketches to help them imagine their final product [26]. The software design literature is only now beginning to catch up to the topic of design behaviors, with the emergence of studies that are beginning to look at software designers "in action" (e.g., [3, 12, 24, 72]. These studies, thus far, confirm the behaviors seen in other design disciplines, but at the same time do not confirm all of them yet, simply because the number of studies remains small. In the below, I include a subset of the design behaviors found in the general design literature, quite a few of which have already been confirmed in the software design literature (i.e., Behaviors 1, 2, 3, 4, 6, 7, 8, 9, 10, and 13) and some of which I simply believe will be confirmed in future (i.e., Behaviors 5, 11, 12, and 14). We feel confident in making this assumption because of my own informal observations of software designers in action, particularly in studying the videos of the SPSD 2010 workshop [3]. While I have not performed full studies of those videos expressly for the purpose of corroborating the general design literature, informally I have seen instances of all of the behaviors I discuss in the below.

I separate the fourteen behaviors I intend to support in three categories, each of which I detail below.

2.1.1 Kinds of sketches software designers produce

The first category deals with what the designers draw. The sketches that designers make at the whiteboard are typically not the goal in and of themselves and, as such, the types of sketches that designers make will vary depending on their current design activity. Sometimes, the sketches are used to help the designer in their thinking, by externalizing their ideas and thoughts onto the whiteboard [49]. Other times, the whiteboard is simply a medium to explain a thought, idea, or design in progress. The developer is not using it for problem solving, but instead to communicate information to a listener or collaborator [28]. Because of these different purposes, what designers draw is dependent on what they work on at what point during a design meeting. This leads to the following design behaviors.

1. *They draw different kinds of diagrams.* In order to explore a design problem, software designers sketch many different types of diagrams, often within the same canvas [5,17]. They may sketch, for instance, entities and relationships, interface mockups, scenarios, architectures, and other kinds of diagrams [12]. The freedom to sketch multiple kinds of diagrams in the same space is fundamental to supporting the exploration of the design space, as it enables designers to explore an issue from different angles, at different levels of abstraction, or even in different ways altogether. The sketches in Figure 2.1, which is taken from a design session in which two software designers are designing an intersection in a traffic simulator, present such an example. On the right, they used the sketch of a traffic intersection to help them in working with the data model on the left. In turn, their thoughts about the data model may lead to revisions to the sketch of the intersection on the right. This behavior of working with multiple representations also aligns well with the observed phenomenon that tools which restrict designers to use one notation hinder the design exploration and lead to fewer alternatives being considered [82].

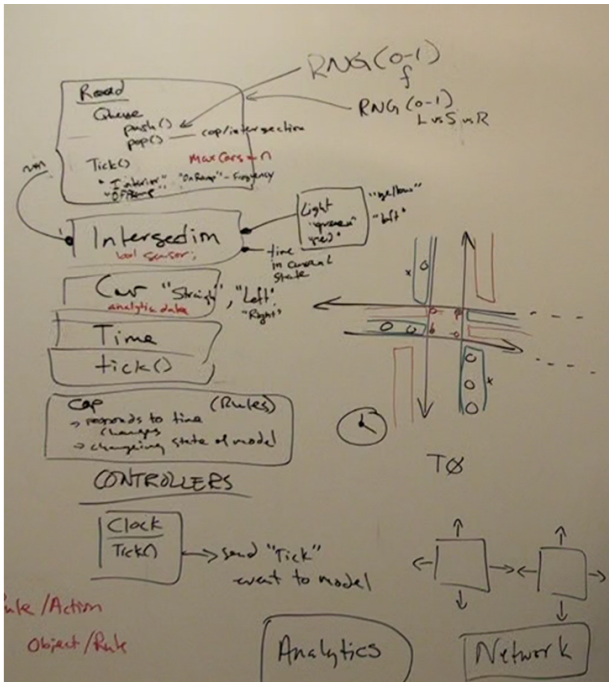


Figure 2.1: Diagram from a design session that includes UML diagrams, a map, and annotations.

2. *They produce sketches that draw what they need, and no more.* Of the many sketches that software designers create, few are drawn in full detail. Software designers typically can get what they need from a quick and incomplete sketch, e.g., a barebones user interface or boxes-and-arrows sketch [92]. The benefit of a low-detail sketch is that it can be created quickly, and modified easily, giving the designer more rapid feedback [12, 72]. Further, providing too much structure too soon can create unconscious barriers to change, resulting in a less exploratory and a less broad search for candidate solutions [93]. This behavior further breaks down into two parts:

   (a) *They only draw what they need with respect to the design at hand.* Low-detail sketches tend to incorporate relevant information and omit the irrelevant [88], including only as much detail as necessary to advance the designers thinking. When talking, they will only draw what they need to reinforce what they want to communicate [72]. When thinking, they will only draw the details relevant to the immediate issue to help them reason [24]. For example, the elements in the

left of Figure 2.1 differ in the amount of detail they contain, where some contain data attributes and others contain only a name.

(b) *They use only those notational conventions that suit drawing what they need.* Sketches only include as much notational convention as the designer needs in a given situation [72]. For instance, if a sketch can express an idea using only boxes-and-arrows, then no more will be drawn, but if a sketch must represent a hierarchical relation, then a richer array of arrows will be present, typically following the convention of an existing formal notation.

3. *Over time, they refine and evolve their sketches.* The level of detail that designers want in their sketches varies over time. Early on, they may need very little, but later they may need much more as they expand on their ideas [67]. Figure 2.2 presents an example of such a refinement, which details the evolution of the diagrams from Figure 2.1 from a relatively simple list, to a complex box-and-arrow diagram. As designers discuss and work through an idea, they will evolve their sketches with additional details to capture their decisions. In general, as part of this refinement process, the sketches will contain more visual precision, and the designer will rework the design to fix any inconsistencies [22]. This behavior, too, breaks down into two parts:

(a) *They detail their sketches with increasing notational convention.* As a designers understanding of the design space matures, so does the representation that they use. As I already discussed, while the designer is aware of the full expressive powers of formal notations, they only borrow from those notations what they need at the time. However, as the designer progresses and the decisions firm up and additional, less-critical decisions are made, they tend to use more and more of the formal notational convention to represent their commitment to the chosen direction [66]. The formal elements in Figure 2.2 illustrate the decisions captured by the designers, where some arrows contain information about cardinality and

some elements are contained in a bounding box to reflect their status as an entity in the model. Note that this move towards a more formal notation is not a strictly uniform activity, as different parts of the design may exist at different levels of maturity [72].

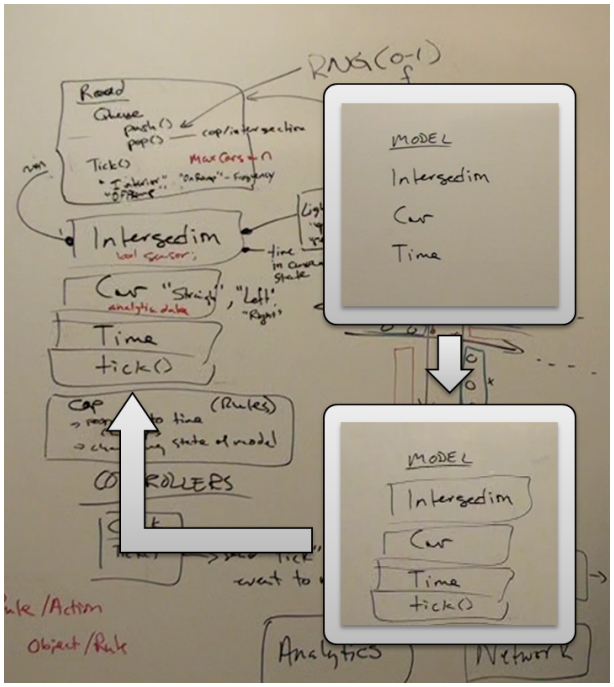


Figure 2.2: Diagram from a design session that was refined from a list into a UML diagram.

(b) *They appropriate a sketch in one notational convention into another notational convention.* Refinement of sketches does not always mean refinement toward and in a single notational convention. Sometimes, designers appropriate one kind of diagram into another [24]. For example, the designers initially created the data model in Figure 2.2 as a list, then later added boxes, then boxes with arrows, and finally evolve into a UML class diagram. Separately, at roughly the same time in the exploration, the same boxes and arrows might refine into a user interface diagram. The designer in all likelihood did not foresee this, but in working out their design in place, they re-appropriated the sketch to suit their needs [52].

4. *They use impromptu notations.* Designers do not exclusively work with the notational

convention they know (e.g., UML, ER, etc.), but also, at times, will improvise in the moment. The deviations that they make from standard notations, such as annotating UML diagrams with custom symbols, are deliberate additions that break convention to capture insights before an idea is forgotten. Beyond such annotations and minor deviations, developers also will sometimes adapt wholly new notations on the fly. These often relate to the problem domain that they are explaining, since few domain-specific notations exist, but shorthand is still needed to support the design process [24].

2.1.2 How they use the sketches to navigate through a design problem

The second category pertains to how designers use sketches to navigate the design space. While designers may create many different sketches over the course of a design meeting that vary in detail, notation, and what they represent, there is typically a thread of thought that relates the sketches and the ideas represented in them to one another.

5. *They move from one perspective to another.* Software designers create many sketches through which they shift their focus between perspectives. The designer in Figure 2.3, for example, is working on the user interface component for a map, and may navigate to the two different views of the intersection to his left, or explore the architecture of its data model to the far left. The designer uses each new perspective to better understand how the parts of a design fit into the whole, asking questions such as: "[what] if we look at it like this, from this angle, it fits together like this" [72]. Each perspective presents a new way of looking at the same design, and what may be subtle in one perspective, may be more pronounced and easier to understand in another.

6. *They move from one alternative to another.* In a sufficiently complex software design task, a software designer will generate sketches of competing solutions before commit-

ting to a particular choice [98]. Expressing alternatives as sketches rather than simply mentioning them aloud allows designers to manage their focus and more effectively explore alternative solutions [56]. Once created, designers can compare alternatives and weigh their trade-offs [8]. They may shift their attention back and forth between alternatives and adopt ideas proposed in one into another, or synthesize the ideas of several alternatives into an entirely new alternative [41].

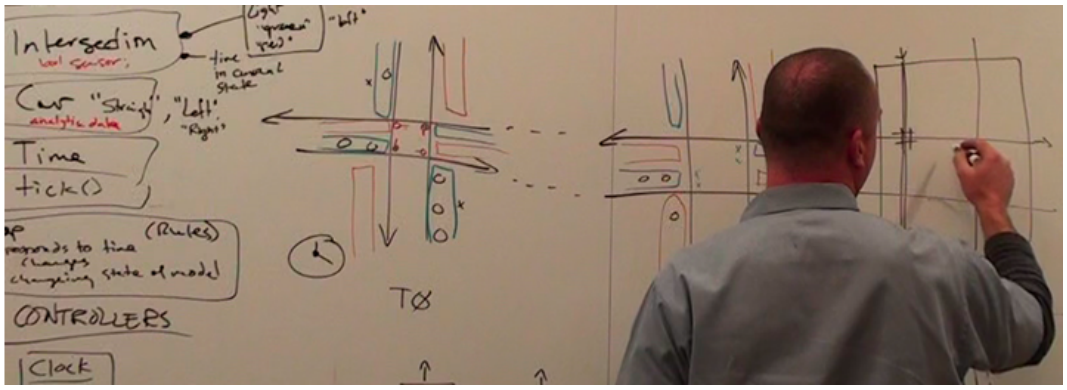


Figure 2.3: The designer navigates between different types of diagrams and abstractions.

7. *They move from one level of abstraction to another.* Software designers move between different levels of abstraction, either by "diving into" parts of their design to explore them in more detail or by shifting "back up" to the higher-level representation. For example, the designer in Figure 2.3 have navigated to the sketch of a map, which presents a zoomed-out view of the two intersections to his left. This shift in abstraction happens often in software design, as many of its notations are hierarchical in nature. A software architect may shift their focus from working out how software components interact with each other to choosing a component and working out how it functions, perhaps by drawing its internal architecture and diving in even further. This behavior typically leads to a multitude of sketches that together consider different abstractions simultaneously [72]. Many scenarios requiring shifts of abstraction have been documented, including the design of user interfaces [18], web pages [90], and so on.

8. *They perform mental simulations.* Software designers use mental walkthroughs to gain insight into the consequences of their design [97]. They may need to understand how information flows among components, or inspect their design by mimicking how an end user would interact with it. The software designers 'interrogate' their design by testing it against hypothetical inputs and scenarios, often marking over their existing sketch while simulating. Through these mental exercises, the designers can bring to light their implicit assumptions and expose flaws in the design [72].

9. *They juxtapose sketches.* In order to compare and contrast ideas, software designers will often juxtapose sketches across perspectives, alternatives, and abstractions [72]. A class diagram may be examined in parallel with a sequence diagram to aid the designer in determining how a message is passed between components. Alternatively, the designers in Figure 2.4 point to both a data model and a map to understand how a car object is passed between components as it travels through an intersection. The juxtaposed diagrams help the designer in reasoning how the design might work, using the knowledge gained from one diagram to help identify the omissions or mistakes in another, as well as any inconsistencies between them [72].

10. *They review their progress.* Not all time spent during design is dedicated towards producing new content or verifying whether the design does what the designer intends it to do. At some point, the designers must take stock of what they have done. They momentarily take a step back, away from the design, and consider the progress that they have made and what they have yet to do [52]. They may return to the problem statement or list of requirements and mark off everything they have done to address it, they may generate a new list of issues that they further need to address, or simply just talk amongst themselves, to assess where they are.

11. *They retreat to previous ideas.* Periodically, designers may reach a stopping point in the exploration of their current set of sketches, such as when they become stuck or simply

Figure 2.4: Designers juxtaposing two sketches by pointing.

have exhausted an alternative. They then may choose to return to a previous state of the design (and its sketches) to start anew [97]. For example, an abandoned proposal for a time-based architecture may become a more lucrative option if an event-based architecture proves too costly in system memory usage. In returning to past ideas, the designer may bring new insights and a matured understanding from the exploration they just exhausted, which they can use to explore the past ideas further.

2.1.3 How they collaborate on them

Software design is a highly collaborative activity, especially at the whiteboard where sketching and design exploration is almost always performed in collaboration with others. The behaviors in this section result from the collaborative aspects of working toward a single vision of the design that is shared by all parties.

12. *They switch between synchronous and asynchronous work.* While much of the work that

takes place at the whiteboard is typically synchronous, with all participants focusing on a single aspect of the design they are discussing, it is known that designers occasionally break away to explore an idea independently while the others continue with the main discussion [23]. This typically occurs when a sudden inspiration strikes, or when a designer wishes to develop a counterexample or alternative to what is being discussed now.

13. *They explain their sketches to each other.* After any independent work takes place, and even in cases where one designer drawing on behalf of the group and "has the floor", the designers must synchronize their mental models of the state of the design [24]. This behavior relates to Behavior 9, but represents its collaborative version. They will need to verbalize their mental simulations to explain the consequences of a particular choice, clarify the meaning of a sketch, or even simply explain their assumptions or inspiration. While explaining, they may sketch on top of the existing diagram, using their marks to guide attention or add detail, or simply gesture over the diagram if they do not want to edit the sketches.

14. *They bring their work together.* Sometimes, as a result of asynchronous work, the designers need to integrate their ideas from separate sketches into a unified design. This may involve bringing parts of a sketch over, creating a new sketch that integrates both, or sometimes working on a third alternative that combines the best aspects of each but requires a different underlying approach to make that work [24].

2.2 Research Question

Now that I have introduced this set of design behaviors, I return to what it would mean to build support for it. I first recognize that there is a spectrum of ways in which I could support the design behaviors. On one end of the spectrum lies the whiteboard itself, a

minimally intrusive, informal medium that supports only drawing and erasing of content. The naturally occurring behaviors are permitted, but not necessarily supported, which is the basic problem that this research is trying to address. On the other end of the spectrum is the formal design tool, a highly structured, heavy-weight environment with hosts of explicit features. Examples of such tools are Rational Rose [37] and ArgoUML [78]. Theoretically, they support a number of the behaviors, such as, for instance, Behavior 5 by providing multiple views on the same model, or Behavior 6 by maintaining many projects. They, however, do not support all, and the ones that they do are not nearly as fluidly supported as necessary.

Between the informal whiteboard and the formal tool lies a range of possible approaches that provide different blends of the strengths of each. This leads to our research question:

> *What minimally invasive, coherent set of features can be designed that is sufficient to effectively support these behaviors?*

It is useful to examine the phrasing of this research question:

- ***minimally invasive*** – *I seek to build support that does not completely abandon the whiteboard experience. People go to the whiteboard for its fluidity and flexibility. Any solution that I design must preserve the feel of the traditional whiteboard as much as possible.*
- ***coherent set of features*** – *I wish to arrive at a set of features that build on each other using a unified set of design principles and metaphors. Building an isolated feature for every behavior would not satisfy our research goal.*
- ***sufficient to support all of these behaviors*** – *Each behavior should in some way be supported by the overall set of features, and support for any particular behavior should not come at the cost of another.*

Having put forward our research challenge, I must analyze the resulting effect of the support for the design behaviors as well. This is a difficult question to answer, particularly since the literature to date has documented that these design behaviors take place, but has not yet fully articulated their effects on the design process and design product. For instance, switching among perspectives (Behavior 5) seems to have a positive effect on the eventual design [2], and, in certain cases, it has been documented that the consideration of multiple alternatives (Behavior 6) also seems to lead to a better design [8]. However, even these studies are hard pressed to provide absolute answers. The first study does not examine the optimal length of time a perspective should be explored: is thirty seconds too short, thirty minutes too long, or what is the general distribution? Similarly, the second study does not talk about the number of alternatives: is three sufficient, should twenty be explored, or does it depend on the design problem in some way? These are questions that I cannot objectively answer at this time.

For this reason, our evaluation must be exploratory. An obvious evaluation would examine factors such as the frequency of design behaviors, time spent working in each behavior, and how our tool impacts the interleaving of the design behaviors. A more comprehensive evaluation would attempt to correlate the appearance of these design behaviors with the quality of the process and the quality of the design by, for instance, relying on outside experts to rate the designs that are produced and correlating this quality with the frequency, interleaving, etc. of the design behaviors. While this would possibly lead to a conclusive statement with respect to the impact of our tool, it would be very challenging to perform this evaluation given that a typical design exhibits complex interleavings of many design behaviors, making achieving any statistical relevance difficult.

Our evaluation will take a qualitative approach. By observing Calico in practice and by interviewing users of the tool, I will analyze the usage of Calico within the scope of the design behaviors and analyze the impact of the features on those design behaviors. Additionally,

while the observations will likely pertain to the existing set of fourteen behaviors, I of course do not rule out that new behaviors may emerge, as enabled by the features of the software, that might be worthy of study.

Chapter 3

Calico Version One

3.1 Introduction

While many software design tools exist and are in use daily, when faced with a given design problem, more often than not developers will turn to the whiteboard, informally sketching and writing to work through potential solutions [13, 19, 51, 62, 73]. This is typical during the conceptual stages of software design, when no code is in existence yet, but frequently also takes place during maintenance design [31], when a code base has been developed but must be modified to, for instance, incorporate new functionality or optimize a key part. Design issues also may arise spontaneously, such as when a developer works on a task, becomes stuck, and involves one or more colleagues in an impromptu design session at the whiteboard to work through the issue [13].

One reason developers turn to the whiteboard is the flexibility and fluidity it offers in the design experience. Existing software design tools, with their requirement to precisely use prescribed notations, focus on correctness and completeness and can be said to primarily support documenting a design after it is thought out. However, software design, as any

kind of design, is a highly creative endeavor [73, 7]. On a whiteboard, developers can freely sketch, branch off to another part of the design problem, return to a previous part, erase some portion of their work, redraw it, and so on all without being forced into using a specific notation or being required to provide any more detail than they want.

At the same time, the whiteboard does not help the exercise either. Particularly, whatever is drawn or written remains static and cannot be manipulated other than drawing over it or erasing it. This is a less than desirable situation, as it is known that software designers often wish to manipulate a design at hand in more advanced ways than merely adding or erasing content [25].

In this paper, we present Calico, a novel software design tool for the electronic whiteboard that supports software designers in the process of sketching. Calico targets four natural behaviors exhibited by software designers while they sketch. Particularly, they: (1) regularly shift focus, (2) typically draw low-detail models, (3) move from abstract to more refined representations, and (4) use a variety of notations. Calico offers the following set of features that is designed to maintain the same flexible and fluid nature of the whiteboard while offering specific support for these behaviors. The features are: (1) a grid to manage multiple canvases in the workspace, (2) scraps that enable advanced manipulation of sketched content, (3) a palette that works in harmony with the new interaction offered by scraps, and (4) a gesture-based interaction scheme to tie all features to stylus-based input.

To evaluate Calico, we conducted a laboratory experiment in which pairs of participants designed an educational traffic simulator using either Calico or a regular whiteboard. For each of the sessions in the laboratory experiment, we reviewed video recordings, analyzed detailed logs of use created by Calico, and interviewed the participants after each session. We then performed a three-step evaluation. We first investigated if participants moved beyond using Calico as a basic whiteboard by observing the frequency of use for each feature. We then performed a qualitative analysis of the videos and reviewed the videos with respect to the

design behaviors we identified. Lastly, we compared the structure of the design conversations as they occured within each pair of participants to identify any potential differences between those pairs that used Calico and those that used the regular whiteboard.

Results are promising, and show that the participants did move beyond Calico as a basic whiteboard by using its more advanced functionalities to carry out their normal software design behaviors. The first analysis, of the frequency of use for each feature, shows that both the grid and scraps were used repeatedly and often by participants, though the palette received much less use in comparison. The second, qualitative, analysis reveals many occurrences of the four software design behaviors, including several creative uses of Calico's features to accomplish these behaviors. Finally, the comparison of the structure of design conversations between the Calico pairs and the whiteboard pairs shows that the design activities across the two conditions are remarkably similar, providing evidence that Calico does not inadvertently influence the normal design conversation in undesirable ways. Overall, then, Calico represents an advance over typical design at the whiteboard, providing designers with advanced capabilities while at the same time preserving the fluid and flexible nature of their work.

The remainder of this paper is organized as follows. In Section 2, we review the role of sketching in software design. Section 3 reviews the four behaviors of software designs that we support. Section 4 presents Calico and how it addresses the stated goals. Section 5 presents our methods for evaluating Calico. Our results are presented in Section 6. Section 7 includes a discussion of our results and Section 8 reviews the threats to validity within out study. Section 9 presents related work. Lastly, Section 10 concludes our study and presents future work.

3.2 Sketching

To position our work, we introduce relevant background material regarding the role of sketching. We first discuss sketching in design in general, then examine sketching in software design in particular, and conclude with a discussion of the specific behaviors of software designers that we want to support.

3.2.1 Sketching in Design

Sketching plays an important role in the design process, regardless of discipline. Designers use sketching as an extension of their own thinking process [73, 50]. Sketching not only acts as an external memory [59] by offloading ideas from the mind onto paper, but there is also additional value in the act of doing so [80]. Sketching is unique in that it affords great fluidly and flexibility in expression [17], enabling designers to focus on the ideas and discussion rather than the means by which the discussion is being recorded. The benefits of utilizing sketching in the design activity are numerous, and here we list those that stand out most.

First, sketching provides a mechanism to quickly externalize thought, both for the individual and a group, so that abstract ideas can be viewed, analyzed, and discussed [71]. A person may want to scribble an idea before it escapes their mind, or may create a partial diagram in reference to an idea while explaining a concept to another individual. Ferguson [27] calls these thinking and talking sketches, respectively, because of their ability to support the thinking process and the discussion of an idea, respectively. Second, the rapid form of externalization allows the designer to engage in a tight cycle of drawing, understanding, and reinterpretation. Schön described this process as having a "reflective conversation" with the material [79]. Often this leads to "unexpected discoveries", as Gero and Suwa call them [87],

stemming from attending to visual details that were not intended when they were drawn. Goldschmidt distinguished between "seeing that", the act of summarizing what is seen, and "seeing as", the act of reinterpreting the sketching or making analogies [30]. Buxton noted that experts are better at distinguishing between "seeing that" and "seeing as" [9], and will get more information out of a sketch than novices [30].

Third, and last, sketching allows designers to reason with the abstract and intangible by using symbols to create representations. Larkin and Simon argue that designers create symbolic representations of abstract concepts to engage in diagrammatic reasoning. Once an abstraction is reduced to symbols using external imagery, designers can use spatial metaphors, such as grouping or scale, to reflect on qualities of the drawn representations and develop new insights [48]. Sketching also allows the designer to generate new notations on the fly if no suitable notation exists. Indeed, designers not only use different symbols for different activities, but also for different phases within that same activity [29].

3.2.2 Sketching in Software Design

Software designers sketch too, and we are not the first to observe this. Dekel and Herbsleb's studies of participants in OOPSLA's DesignFest, for instance, resulted in a set of observations on how software design teams use and manipulate sketches [25]. Petre's work added observations on use of sketches to support design thinking and mental imagery in actual professional practice [73].

The advantages of sketching reviewed in Section 2.1 are beginning to be echoed in these and other studies. For example, Cherubini et al. articulate goals such as sharing thoughts, grounding ideas, manipulating concepts, and brainstorming as reasons why software designers sketch [13]. In both co-located and distributed teams, diagrams are indispensible for explaining core design concepts to new team members [96]. Although future research is

necessary to extend these studies, they begin to show that sketching plays a crucial a role in software design, just as it does in other design disciplines.

The tools that software designers use have an impact on the quality of exploration of a design problem. For instance, Zannier et. al. examined factors involved in design decisions that software teams make, finding that tools that encourage conversation and do not force structure encourage a broader consideration of alternatives [99]. These kinds of tools allow software designs to generate several solutions concurrently and then choose the solution or mixture of solutions that leads to the optimal result. Petre provides an indepth look at a variety of aspects of software design sketches [73]. One aspect of particular interest is how software designers use mental imagery and how sketching serves as a natural extension. Petre observes that software designers use mental images because they allow flexible selection of focus, have implied provisionality, and enable juxtaposition of elements. That is, sketching allows software designers to shift their focus fluidly among provisional ideas, which allows concepts to be juxtaposed early and often. Moreover, she found that the degree of formalism of notations varies with the completeness of the idea, i.e., provisional ideas will lack many details of their notations and mature ideas will have more detail.

3.2.3 Behaviors of Software Designers While Sketching

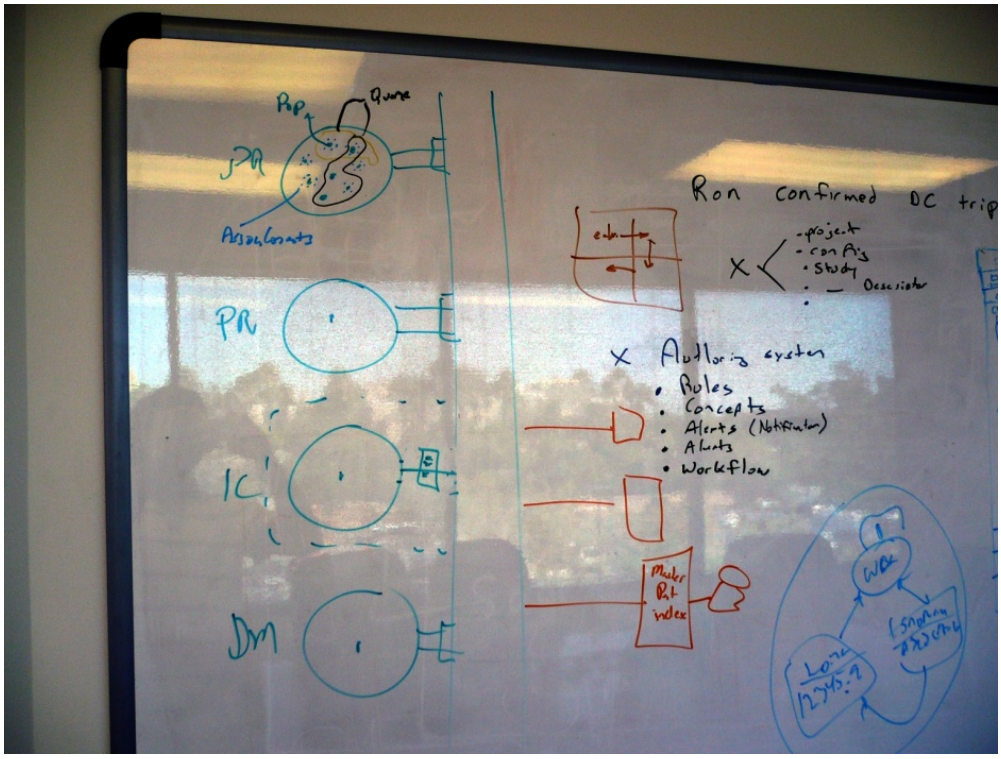
Also emerging from the literature studying software designers "in action" is a set of behaviors that software designers exhibit while they work at the whiteboard. These behaviors are ubiquitous, and give rise to exactly the question Calico attempts to answer: can these behaviors be supported such that the design experience at the whiteboard is enhanced? In the below, we discuss the four behaviors towards which Calico is designed.

Shifts in Focus

When working on a design problem, software designers switch context continuously. For instance, in Figure 3.1(a), the designers first drew the different kinds of databases for patients that they will need as circles on the left, then jumped to brainstorming a list of elements relevant to the database on the right, and then began drafting a high-level perspective of how those databases would interact with the rest of the system on the bottom. Such switching between contexts is an essential part of design, as designers must consider alternatives, explore different perspectives, and develop detail where and when necessary to create a satisfactory solution [73, 55, 99].

Jones observed that three identifiable phases occur in creative design: divergence, transformation, and convergence [40]. In the divergence phase, the objective of the activity is to generate a high quantity of alternatives without evaluating any particular one. The transformation phase includes the refinement of those alternatives by splitting them into subproblems, identifying constraints, and evaluating those subparts. The third phase, convergence, involves distilling those alternatives until a single choice is selected from the many, or emerges from combining aspects from those choices. These phases are clearly not linear; shifts in focus are pertinent in all of these phases.

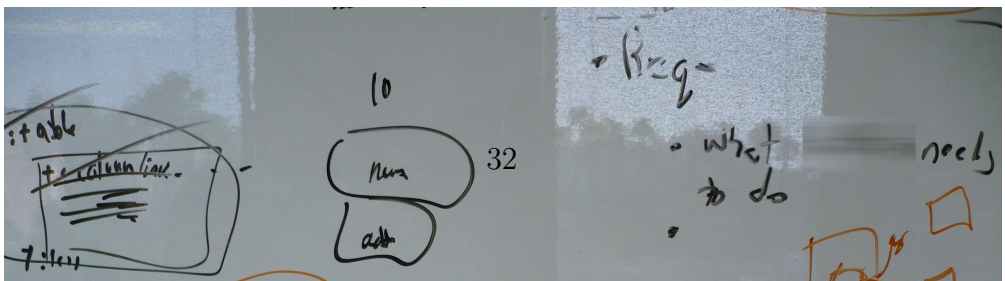
Goel echoed these ideas several decades later in his theories about the importance of sketching [29]. Goel closely examined the transitions that take place when designers work with sketches, and he observed that two, quite different, transitions frequently take place, namely vertical transitions and lateral transitions. In vertical transitions, designers shift their focus from a high level of abstraction to a lower level with more detail. In lateral transitions, designers switch from one idea to another, attempting different strategies to solving the same problem.


PR
IC
DM
Ron confirmed DC trip
Authoring system
Roles
Concepts
Alerts (Notification)
Alerts
Workflow

(a)


Lab
Hub
Site

(b)


10
32
Req
what
needs
to do

Use of Low-Detail Models

Software designers make use of low-detail models during exploration at the whiteboard. Figure 3.1(b) highlights this observation. Most boxes remain unlabeled, various forms of shorthand are used, and the sketches remain abstract in nature. Various studies of software designers confirm the prevalence of low-detail models, particularly early in design sessions, noting that they allow ideas to be expressed quickly and modified easily [13, 73]. Often, too, quick sketches of prototypes can find the same usability problems as high fidelity prototypes [91]. Additionally, people tend to "incorporate relevant information and omit the irrelevant" [89] in their sketches, using only as much detail as necessary to advance their thinking.

The sketchy look, as well as the flexible nature of sketching, allows designers to not commit to ideas too early. If ideas are too structured too soon, or even just appear too formal when they are put into drawing form, they tend to become unconscious barriers to change [94]. A design that is too high fidelity too soon causes a designer to be less likely to reconsider the ideas underlying it, resulting in a less exploratory and a less broad search for candidate solutions [94].

Also, whether consciously or subconsciously so, the low-detail sketches of software designers' early models tend to leave room for alternative interpretations, which has been shown to have beneficial effects on the quality of the eventual design that is delivered [29, 95]. While a general lack of detail in the sketches that are used contributes to this effect simply by the nature of sketching, it is a conscious strategy of many expert designers to create interpretation-rich representations, purposely not committing unnecessary detail early on so that they can use the same drawing to imagine several different perspectives.

Use of a Mix of Notations

Software designers make use of a mix of notations. In Figure 3.1(c), on the left-hand size, a module within the system (Client Plugin) is represented using a box-and-arrow representation, while a custom notation is used to represent database interactions in the bottom-right of the same image (the diagram with the "L's" inside circles that connect to "slots" on the right).

This mixture of notations is commonplace in design, and forcing the designer to use just one notation at a time can be harmful to the design activity [29, 96]. A study by Goel showed that groups using tools with structured diagrammatic elements to perform a design activity was less successful in exploring alternatives than groups using a tool that did not have structured elements [29]. In another study by Shipman and Marshall, participants circumvented formalisms early in their design process, stating that they did not want to prematurely commit to ideas. Further, the study found that the cognitive overhead required for participants to adapt their thinking to a given notation hindered their creativity [81].

Additionally, software designers have been observed to create models that are specialized to support their understanding in a specific context [73], and the notations for such models may emerge on the fly to adapt to the needs of a new context [25]. The internal data structure of L's and slots in Figure 3.1(c), for instance, uses an impromptu notation that does not resemble any known approach (e.g., UML or Entity Relations).

Refinement of Representations Over Time

Crosscutting the other behaviors discussed thus far is the tendency for software designers to move from generic representations to more refined ones. Early sketches are typically low in detail and visually imprecise, as in Figure 3.1(b). These early sketches are then iterated

upon and refined over time, with each refinement recording new design decisions and each iteration reflecting a revised understanding of the design. Later iterations, then, tend to be more organized to reflect the relationships between elements of the system and tend to be drawn with more aesthetic detail to reflect a firmer commitment to the design decisions made.

In addition to the visual quality, the formalism of the notations, i.e., the amount of visual syntax that a computer could hypothetically recognize, that software designers use also increases as they refine their diagrams. During the early stages, while the design problem is still ill defined and the constraints have not yet been clearly defined, software designers may liberally omit parts of notations to delay decisions. However, as the constraints become increasingly defined, so does the formality and detail of the representations [68].

Lastly, the evolution of these notations is not linear, but instead involves designers moving back and forth between notations opportunistically. The developer may create a symbol to serve a particular purpose, abandon the use of that symbol in favor of another one, and then return to using that original symbol again later [25]. The choice in representation during the early phases can be volatile, but designers typically settle into a uniform notation eventually.

3.3 Calico

As already alluded to, Calico was created to address the behaviors described in Section 3. To do so, we introduce a number of new features, specifically: the grid, scraps, and the palette. In this section, we present these novel features, describe them in detail through various examples, and carefully relate them back to the design behaviors of Section 3. As a guide, Table 3.1 shows a mapping between each of the four behaviors of Section 3 and the main features of Calico. Before we discuss the features, we reiterate that Calico is meant to

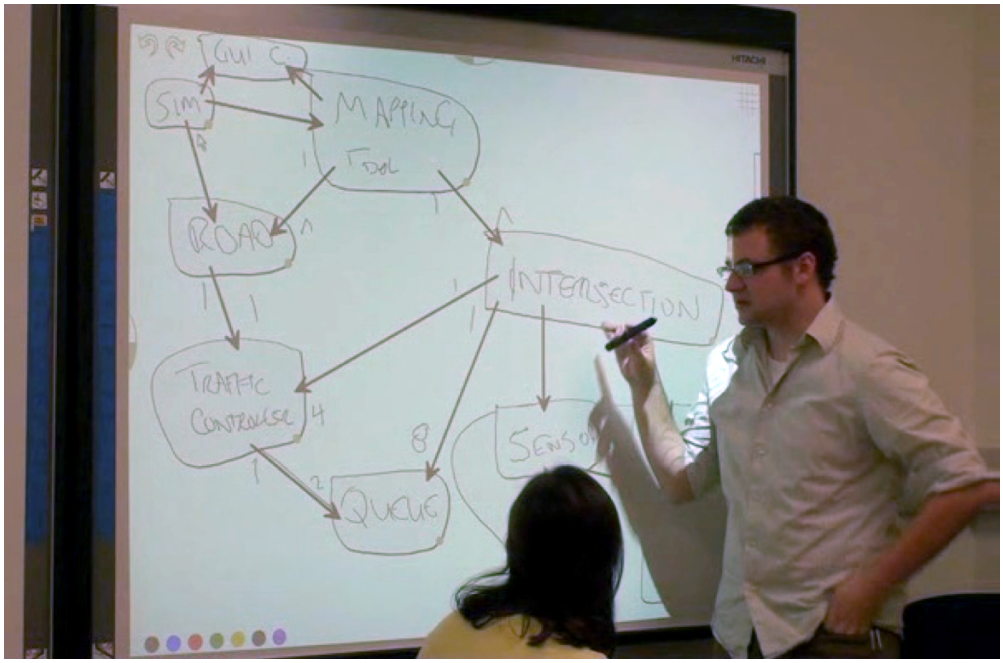


Figure 3.2: Physical setup of Calico

be used on an electronic whiteboard (or Tablet PC, in the case of sole person use), and is optimized to take its input from a digital stylus, not a mouse. The designer stands in front of the electronic whiteboard and uses the digital stylus to draw, write, and control Calico (see Figure 3.2). The nature of this interaction shaped the design of Calico, particularly the mechanisms with which it enables designers to manipulate a design.

Note that all of the examples we use in this section draw from the evaluation design sessions we describe in Section 5.

3.3.1 Basic Features

Figure 3.3 presents Calico as it first appears when a developer starts it. Users can immediately draw or write, without needing to enter any mode or selecting a widget to create new content. Just as on a standard whiteboard, they make any marks they wish, anywhere, in any shape. The drawing canvas has just a few visible widgets to maintain the appearance

Table 3.1: Calico features as they address the behaviors discussed in Section 3

Behavior	Feature	Effect
Shifts in focus	Grid	Sketching effort can be partitioned across multiple canvases Tabs permit quick shifting between, and copying of, canvases
Use of low-detail models	Scraps	User-drawn shape can be preserved in order to enable informal models, Strokes are grouped implicitly, which makes them moveable, stackable, and relatable
Use of a mix of notations	Scraps	Emerging notations are captured by scraps and can then be easily copied for reuse
	Palette	Scraps are stored in a central location and can be reused anywhere
Refinement of representations over time	Grid	Abstract sketches can be partitioned and expanded upon across several canvases, which can be meaningfully organized on the grid
	Scraps	Sketches can be transformed from plain sketches to first order objects by making them into scraps, and then related to one another with arrows

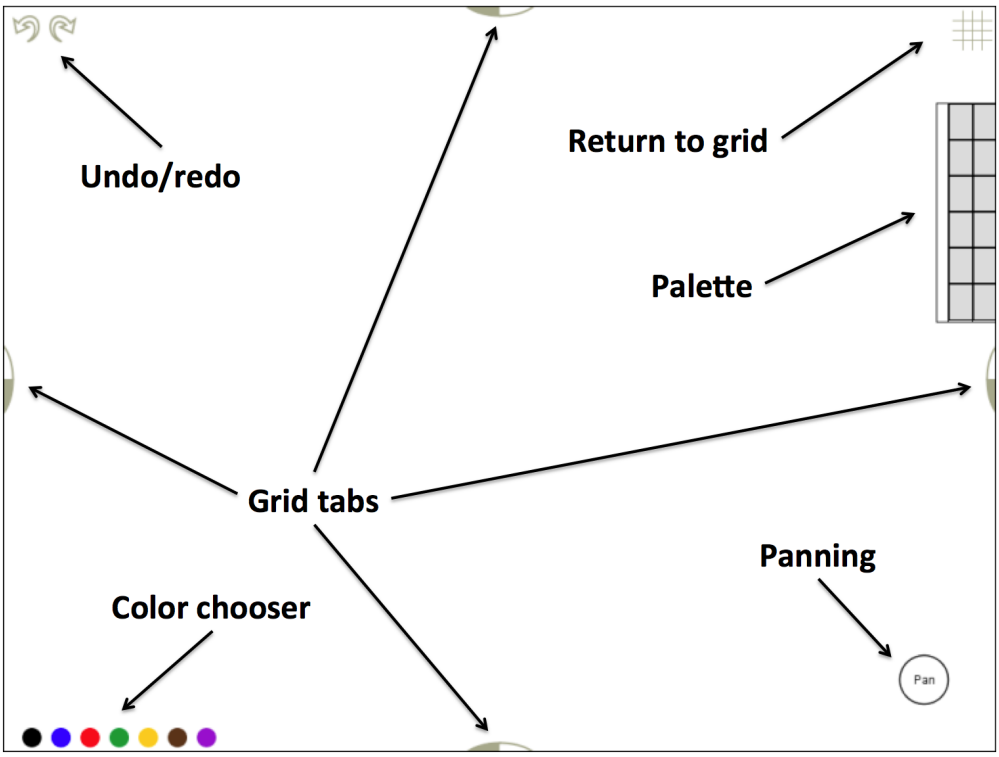


Figure 3.3: Calico appearance at the start of a design session

of a standard whiteboard. Seven colored buttons at the bottom left allow a user to change colors. The colors were carefully chosen to be in direct correspondence to the standard pen colors available on a regular whiteboard.

In addition to changing colors, Calico provides a small handful of features familiar to computer users. Users can pan a canvas, save and load designs, export sketches as images for use in other programs, as well as undo and redo actions. The undo and redo actions are globally stored, so if a previous action was performed on another canvas, the user's view will shift to that canvas and the action will be undone. Also, while Calico does not recognize or formalize shapes drawn by the user, Calico does compensate for the typically low polling rate of touch-based hardware by appoximating the curve of the user's strokes using Bézier curves. Calico does not support text entry, relying on users to simply write on the board in order to avoid unwanted breaks in concentration due to the need to switch to an extraneous keyboard. While Calico will run on any touch-based device with Windows, Linux, or Mac

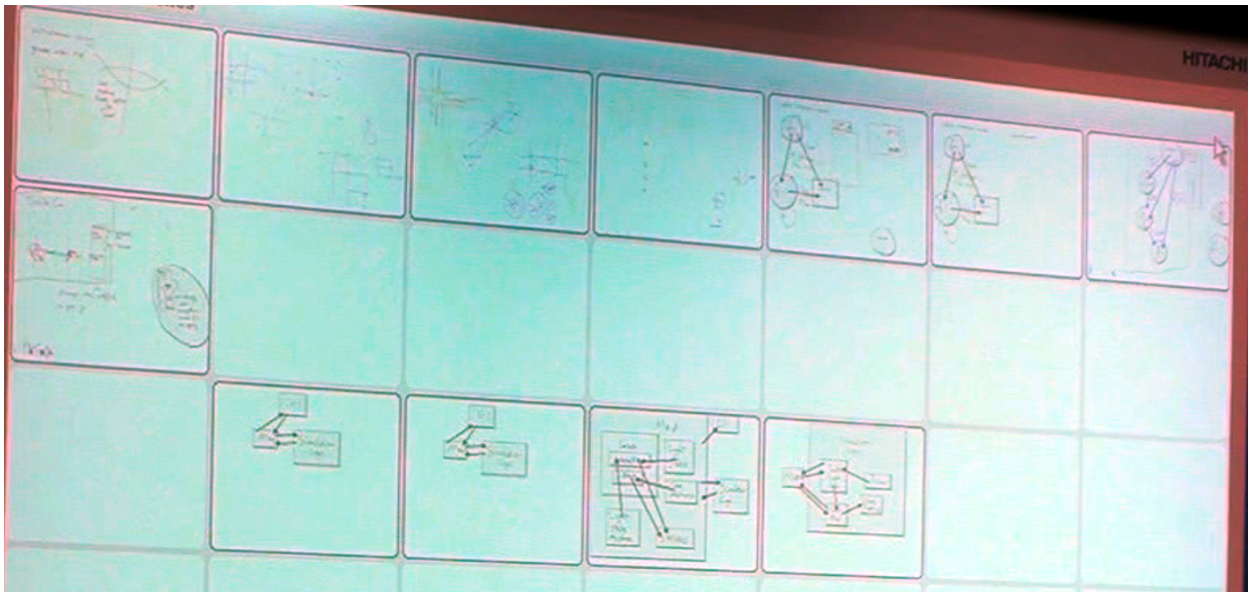


Figure 3.4: Grid in use

OS (see Section 3.5), all of our experimentation was performed on a board with a projector resolution of 1024x768.

Calico's first novel feature is the grid, which is our answer to the designer's need to not only easily shift focus, but to also be able to keep track of their design at hand. Shown in Figure 3.4, the grid provides the designer with a bird's eye perspective of the design process as it has unfolded thus far by showing multiple canvases at once. By tapping on a canvas on the grid, the developer enters that particular canvas, where they can create new sketches or modify any sketches already in existence. By tapping the grid icon (see top right of Figure 3.3), the designer returns to the grid.

3.3.2 Grid

The grid naturally leads to a partitioning of design effort, while at the same time supporting straightforward movement between the emerging parts of this overall effort. For instance, a designer can make a to-do list in one canvas, use other canvases to work out each item on the

to-do list in isolation, and return to the to-do list periodically to check and mark progress. As another example, the designer could choose to examine a design problem from various perspectives or at varying levels of detail in different canvases. The grid becomes the record keeper of this exploration, allowing rapid shifting from one perspective or level of detail to another.

Previous tools have used the filmstrip metaphor to support multiple canvases, with a scrollbar at the bottom, typically in reverse order of manipulation [83]. With frequent updating of the content of different canvases, this leads to a volatile representation in terms of the order of the canvases in the filmstrip, making it difficult to navigate and easily shift focus. The grid advances on the filmstrip by transitioning from a time-based metaphor to a spatial metaphor. The grid keeps all of the canvases in a constant location, which means that a user can not only easily locate individual sketches based on where they exist in the grid relative to each other, but also navigate to adjacent canvases without first having to switch to the grid. They can move left, right, up, or down one canvas at a time by using the tabs that are located in the middle of each edge of the canvas (see Figure 3.3). Tapping the white part of the tab enacts the move to the adjacent canvas in that direction.

Tabs also support a different form of shifting focus: exploration of alternatives. On a standard whiteboard, it is difficult to fork ideas. One has to manually replicate a sketch elsewhere before modifying it. This is clearly undesirable and either leads to less exploration or direct manipulation of the sketch in question, the latter destroying the original from which the departure is being made and making it difficult to return to a previous state. By tapping the grey half of a tab, a canvas' content is copied in its entirety to the corresponding adjacent canvas, which in turn is placed into focus. With a single "click", thus, the designer is provided with a fresh copy of an entire canvas, which they can then further explore, refine, or modify. Using this technique repeatedly, a trail of historical revisions is built that documents how an idea evolved, providing a safety net to always return to previous versions.

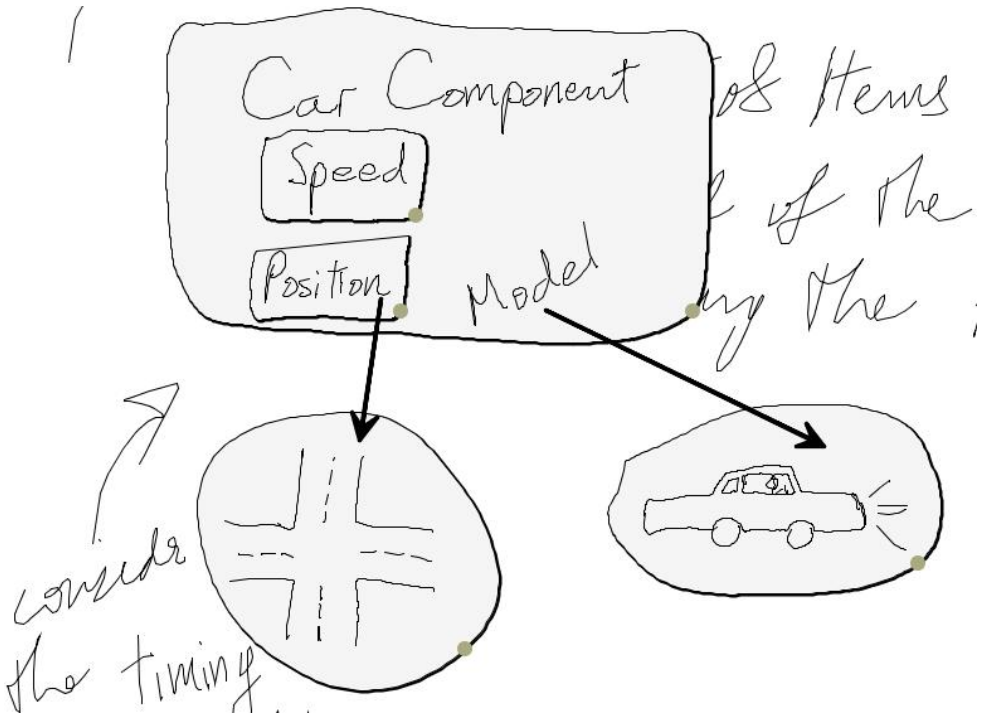


Figure 3.5: Scraps

Shifts in focus are further aided by the fact that canvases are moveable on the grid, allowing a user to rearrange the various canvases according to whichever concern they wish to address. That is, they may move "old" ideas aside, cluster canvases by phases of design, group topics, or even juxtapose different perspectives.

3.3.3 Scraps

Scraps address three of the behaviors discussed in Section 3: (1) the use of low-detail models, (2) the use of a mix of notations, and (3) the tendency to move from generic representations to more refined ones.

Scraps leverage a key action that users of a whiteboard naturally perform: circumscribing some area of sketched or written content [37]. They typically do this to indicate some object of sorts, or to simply mark an area as important. In Calico, the act of circumscription

using the secondary button on the digital stylus has visually the same result: the area that is circumscribed is highlighted with a thin border and grey background. In addition, however, the area becomes an object in Calico that can be further manipulated and has several preassigned behaviors. In particular, scraps are implicit groups that are movable, stackable, and relatable. We review these properties one by one.

Implicit groups. Scraps build upon the approach taken in Translucent Patches [45], which allows users to explicitly declare an area as a group. Anything that is either entirely circumscribed in the first place or otherwise written or drawn in this area afterwards is automatically part of the group. Consider the literal sketch of a car visible in the bottom right of Figure 3.5. It was first drawn on the canvas, then circumscribed by the stylus to become a scrap. The scrap is now a persistent object with a grey background. Any further additions to the car automatically become part of the scrap.

Movable. Scraps are movable. Right-clicking and dragging using the digital stylus will move a scrap and its contents to a different location on the board. This seemingly innocuous action in reality represents a significant improvement over the standard whiteboard: content drawn can be rapidly reorganized. It particularly is important that such reorganization takes place in the language of the user: elements that they have deemed of sufficient importance to promote to being a scrap are the elements that are moved.

Stackable. Moving a scrap to a position where some part, or all of it, overlaps another scrap attaches it to the scrap behind it, allowing users to quickly create a stack of scraps (thereby creating hierarchically composed groups), as one would a pile of papers. For instance, the scraps labeled "Speed" and "Position" in Figure 3.5 are part of the scrap labeled "Car Component." If "Car Component" is moved, "Speed" and "Position" are moved as well. Dragging a scrap off of another scrap un-groups it. Moving the scrap labeled "Position" from its current location on the canvas to where it overlaps the scrap containing the intersection will ungroup it from "Car Component" and group it with the intersection scrap in one fluid

motion. Note that dragging a scrap implicitly moves it to the top of the order of scraps; scraps do not slide under other scraps.

Relatable. By dragging the digital stylus from one scrap to another, an arrow is created between two scraps. The arrow is persistent and anchored to the places where it originated and ended. When scraps are moved, the arrows move accordingly and keep the two scraps related. In Figure 3.5, the scrap "Position" relates to the intersection scrap, and the scrap "Car Component" relates to the car scrap.

Complementing these basic scrap behaviors are several more specialized behaviors accessible via a small radial menu on the scrap itself. First, a scrap's content can be dropped onto a scrap behind it or, if there is no scrap behind it, back onto the canvas. This allows designers to combine content from multiple scraps. Second, scraps can be given a more regular box shape, and, third, scraps can be made transparent. This latter functionality supports divergence at the micro level of individual scraps (as opposed to the macro level of entire canvases through the grey tabs). By drawing on a translucent scrap that overlays another scrap, developers can explore modifications to this other scrap's content without overwriting its content. If they are satisfied with the result, they could choose to combine the two scraps by dropping the content of the translucent scrap. Alternatives can be compared in this manner as well via multiple translucent scraps capturing different deviations.

Scraps are also fundamental to addressing the behavior of using a mix of notations. By virtue of having physical presence, scraps provide a natural basis for serving as a representation for more structured, though still informally drawn, figures, such as user interface sketches or class diagrams. Since scraps are amorphous, and take on the shape of a designer's stroke, shaping them in visually identifiable forms (e.g., boxes, circles, buttons) allows the designer to informally convey a certain meaning. Particularly when combined with arrows, this supports sketching of numerous types of box-and-arrow-like diagrams.

3.3.4 Palette

The palette is the third feature that we incorporated in response to the behaviors discussed in Section 3. It specifically addresses the issue of varying notations.

Palettes are not new, but their typical incarnation in drawing programs is to include a prepopulated set of figures that are not configurable beyond changing the entire set. In design, however, it is not uncommon that a spontaneous notational convention emerges that does not necessarily adhere to any pre-existing or fixed set of figures. Calico's palette, thus, starts empty, and is filled with content by the designer, enabling the reuse of sketched elements. This allows a temporary vocabulary to be created and leveraged within a design session (as exemplified by the impromptu notations in Figure 3.12, as further described in Section 6).

The palette leverages scraps for this purpose. Designers can store a scrap simply by dragging it into the palette on the side of Calico's canvas. The palette has a number of cells, each of which may hold one or more scraps. By dragging from a palette cell onto the canvas, any scraps inside that cell are copied to the canvas at the position of the stylus. Once populated, the user can rapidly create variations of a design simply by dragging key scraps from the palette. Note that the palette serves as a global clipboard; scraps can be stored and reused from any canvas.

3.3.5 Implementation Notes

Calico's implementation consists of approximately 37,000 lines of code written in Java 6.0. Calico was developed using the Eclipse development environment, and was tested on Hitachi FX-Duo Starboard interactive whiteboards. It is portable across several operating systems (i.e., Windows XP/Vista, Linux, and Mac OSX), though all of our laboratory evaluations

described below were performed using a Starboard connected to a Windows laptop that ran Calico.

Architecture

Figure 3.6 provides a summary of the architecture of Calico, which is based on the model-view-controller pattern. The model is responsible for keeping track of all of the sketched content, including strokes, scraps, arrows, and any relationships that exist among them. Strokes and scraps are stored as sequences of raw coordinates; arrows are stored as a start point and end point. Two relationships are supported. First, when scraps fully contain other content, whether strokes, other scraps, or arrows, a containment relation is kept so that, when scraps are moved, copied, or deleted, the contained elements are also moved, copied, or deleted. Second, an anchoring relation is kept if an arrow's start point or end point is within a scrap, so that when the scrap is moved, the corresponding point also moves. All content is managed by an ObjectHandler, which is responsible to provide not just convenient access, but also accessory methods for creating and restoring from a serialied back-up.

Lastly, the grid of canvases exists virtually as a set of two-dimensional coordinates within the components in the model. Each stroke, scrap, or arrow has attached an (x, y) identifier to which canvas it belongs.

The Controller is responsible for interpreting the actions of the user on the canvas and translating those, if needed, to Calico actions. The Gesture Controller is the primary point of access, taking as input the mouse events that are generated when the user interacts with the Hitachi Starboard (we use mouse events only and specifically chose not to take advantage of any special Hitachi Starboard features in order to avoid limiting ourselves to just this electronic whiteboard). Based on the location of the stroke and any pre-existing

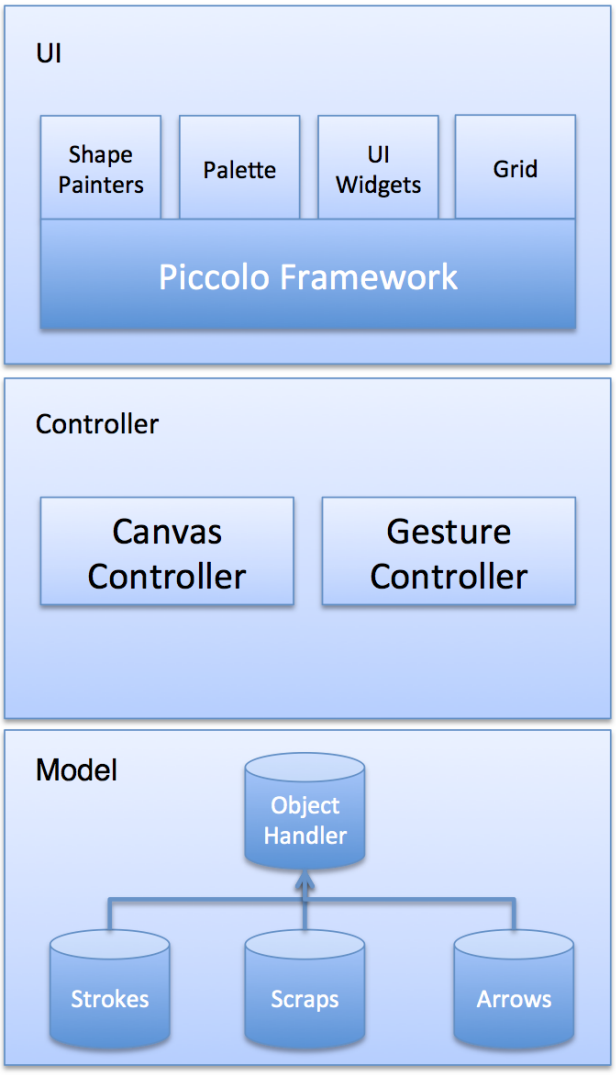


Figure 3.6: Calico architecture

content that may already exist on its path, the Gesture Controller distinguishes strokes that draw on the canvas from strokes that indicate some action to be performed (see also Section 3.3.5). Strokes that draw on the canvas are passed on directly to the Canvas Controller for creation in the model. Strokes that represent actions are passed on to the Canvas Controller as "action objects" using a command pattern, in order to facilitate future extensions with other gestures and actions.

The User Interface is built using Piccolo [5], a zoomable interface that optimizes screen refreshing. Piccolo was chosen because of its available source code and relatively lightweight

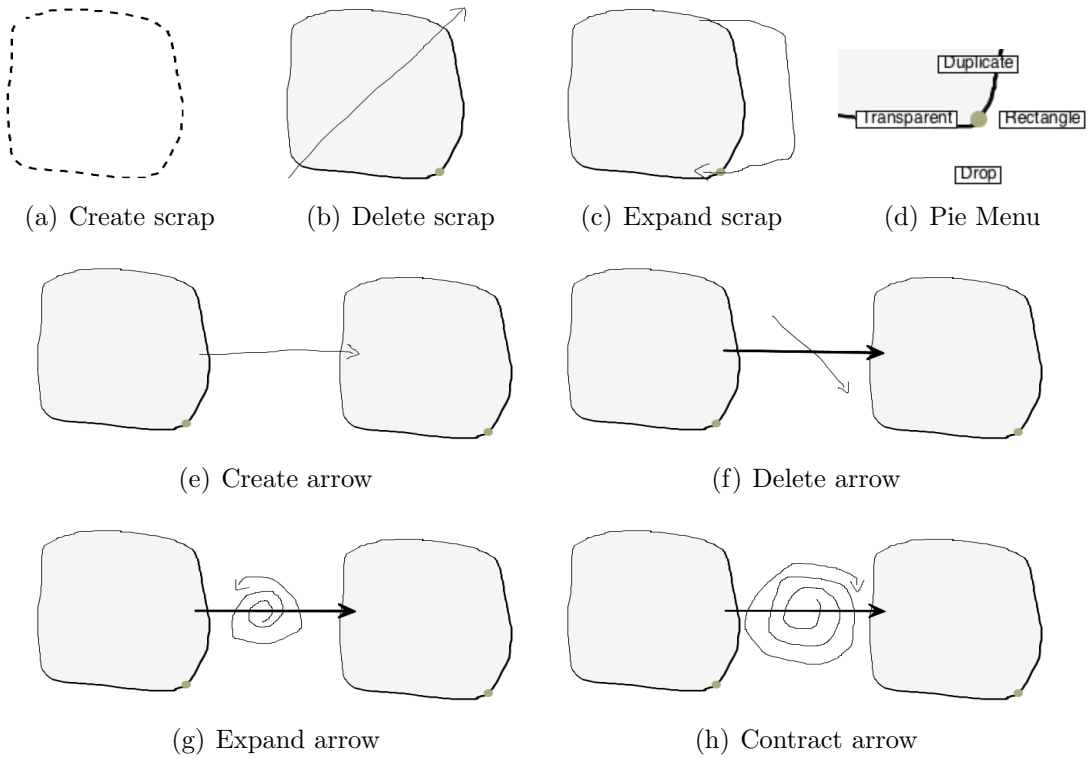
footprint. The Shape Painters, UI Widgets, Palette, and Grid all extend the Piccolo Framework and comprise the interface with which the user interacts. The Shape Painters mirror the data model using nodes from the Piccolo Framework to draw content on the canvas; any changes to the data model are then automatically propagated. The UI Widgets take care of Calico functionality such as the tabs, panning, and undo and redo. The Palette also exists as a separate UI widget, albeit with special functionality to hold scraps that can be reused. Upon initialization, Calico instanciates a fixed number of canvases and stores them in the grid, which can be referenced using the (x,y) identifier.

Gesture-Based Input

Permeating across the four design behaviors discussed in Section 3 is the need for gesture-based input. Many of the creative, exploratory activities that Calico supports rely on the fluidity and quickness of sketching. Interactions supplemental to the primary sketching activity should be equally quick. The design flow must be maintained, shapes should be created the way designers want them, and the effort of making changes should be low. Essentially, "viscosity" [73], the cost of making changes, has to be low or designers will be discouraged from exploring opportunities in design.

In order to maintain the natural feel of sketching in the interaction, we have chosen to use a mixed mode approach: many features are ready-at-hand through simple gestures, but some more esoteric functionality that would otherwise require integration of complicated gestures is available in a pie menu off of scraps (visible in Figure 3.7(d)).

Interaction is driven by a small set of context-sensitive gestures, as shown in Figure 3.7. Each stroke of the stylus performs a unique action depending on where it travels. For example, dragging the pen while holding the secondary stylus button on the canvas creates a scrap, but performing the same action on a scrap will move the scrap. With ordinary strokes, dragging



(a) Create scrap (b) Delete scrap (c) Expand scrap (d) Pie Menu

(e) Create arrow (f) Delete arrow

(g) Expand arrow (h) Contract arrow

Figure 3.7: Gestures in Calico for scraps and arrows

the stylus along the canvas draws directly onto the canvas, but slashing it across a scrap, as shown in Figure 3.7(b), will delete it. The gestures take into consideration where the stylus begins, which object(s) it intersects along its path, and where the stylus ends. This is sufficient to build a straightforward interaction scheme that allows the user to create scraps, manipulate them in various ways, draw arrows from one scrap to another, and navigate the entire Calico interface with ease. Each action has a priority, such that stylus strokes spanning multiple interactions behave predictably. For instance, when a user draws a line from one scrap to another that crosses an existing arrow, a new arrow is created and the existing arrow is not deleted, despite the fact that the stroke struck through it (a gesture that normally deletes that arrow).

3.4 Experimental Design

To evaluate Calico, we conducted an exploratory, comparative study in which we asked pairs of participants to perform a software design task using either Calico or a regular whiteboard. Our goals were threefold: first, we wanted to ensure that, when Calico is used for an actual design activity, people moved beyond basic whiteboard sketching and used Calico's advanced features. Second, given that Calico was designed to support specific design behaviors, we wished to observe whether these behaviors actually did occurred. Lastly, we wished to assess the structure of the design conversation in the pairs that used Calico, and compare it with that of the pairs that used the whiteboard.

3.4.1 Recruitment and Participants

Sixteen pairs of participants (32 participants in total) were given one hour and fifty minutes to complete a given design task. Eight of the pairs used Calico to perform the design task, and eight of the pairs used the regular whiteboard. We recruited participants at our university using email advertisements and snowball sampling. In a pre-experiment survey, we asked participants to declare their area of expertise, and their industrial experience. All participants were computer science graduate students with some degree of experience designing software systems. The average amount of industrial experience was three years, and ranged from none to seven years of experience. To balance pairs, we paired participants with the similar industrial experience together to avoid having the more experienced designer take over the session. We then gave each pair a rating based on the average experience of its members, and distributed these pairs equally among the whiteboard and Calico conditions so that each condition had an equal amount of experienced and inexperienced pairs.

3.4.2 Procedure

Each session lasted between two and two a half hours. When participants arrived, they were given an informed-consent form that included details about the nature of the study. Those in the Calico session were given a 30-minute tutorial, while those in the whiteboard activity were allowed to begin immediately, upon which they were given the two page prompt (see below). Participants were asked to use the whiteboard and to not write on the prompt or any other paper so that cameras could have a clear view of anything they wrote. Pairs were given one hour and fifty minutes of design time, after which they briefly recapped their design in their own words (5-10 minutes). After pairs finished their design activity, they were given five minutes to collect their thoughts and then to summarize their design in a ten-minute explanation. Afterward, we interviewed them for ten minutes about their opinions and experience concerning the activity, and their past experience with similar technology. At the end of the session, each participant was given $100 as an incentive and a $250 prize was awarded to the pair with the best design. As a result, all pairs took the exercise seriously and were fully engaged throughout the design exercise.

Each session was recorded on video camera for subsequent analysis. Additionally, Calico produced detailed logs that captured each individual user action (e.g., scribble drawn; scrap created, moved, or deleted; switch canvas in the grid).

3.4.3 Task

Each pair received the same design prompt, asking for the design of an educational traffic signal simulator to be used by students in a civil engineering course (the same prompt was used in the Studying Professional Software Design workshop; it is included in its entirety in the introduction to the Design Studies journal special issue dedicated to this workshop [74]). The prompt, provided a series of open-ended goals and requirements, asking the pairs to

design a system that allowed engineering students to: (1) create a visual map of the roads, (2) describe the behavior of the lights at each intersection, (3) simulate traffic flow, and (4) change parameters of the simulation, such as traffic density. The prompt also instructed pairs to produce a design that they could present "to a pair of software developers who will be tasked with implementing it."

3.4.4 Measures

Our goal was to: (1) measure how often Calico's advanced features were used, (2) examine how those features were used with respect to the design behaviors we outlined earlier, and (3) assess the structure of the design conversation in both the whiteboard and Calico sessions. We also asked a standard set of exit interview questions concerning the group's satisfaction with the tool and the design process that they followed.

Use of Features. In order to objectively verify that participants move beyond just sketching like they would on a regular whiteboard and actually use the advanced features of Calico, we measured the amount of usage that each feature received. To perform this, we reviewed the videos and made a note of each time a particular feature was used.

Design Behaviors. After we had recorded all instances of the features being used, we performed a qualitative analysis of how the features were used. We were interested in seeing how well those features supported the behaviors outlined in Section 3. When considering the shifting of focus, we were particularly attentive to the reason that participants moved out of one canvas and into another within the grid, and we classified the general activities that occurred in each one as well. For low-detail models, we noted the general diagrams that participants sketched, and paid particular attention to what models participants created, and if and how they used scraps to do so. For the third design behavior, mix of notations, we watched out for situations where participants mixed several notations together in

a single canvas, and highlighted occurrences of impromptu notations that were unique to a particular pair and did not appear in others. Lastly, for the design behavior of moving from abstract to concrete, we compared photos of diagrams at various stages, and noted how pairs restructured their diagrams and added additional detail.

Structure of Design Conversations. After qualitatively assessing the videos for evidence of the design behaviors, we assessed the impact that Calico had on the structure of the design conversation by coding audio transcripts of the design sessions for what kinds of activity took place at each moment. The categories and guidelines for the coding scheme were adopted, with minor changes, from a previous study on design meetings [63]. As the authors of that study explain, the categories were derived from the Design Rationale literature [54] as well as studies of group activity [77], and reflect key aspects of the design activity. With respect to the activities from design rationale, statements were separated into *Issues* at hand, the *Alternatives* or solutions raised, and the *Criteria* used to evaluate an idea. Within the same coding scheme, statements could also be organized into organizational activities, i.e., conversations the group had to organize itself (specifically called *Meeting Management*, *Summary*, *Walkthrough*, and *Goal*), to *Clarify* their ideas, or to engage in *Digressions*. Two additional categories not part of the original coding scheme, *Technology Management* and *Technology Confusion*, were inspired by a previous study based on the same coding method that analyzed a tool's effect on the design process [65]. *Technology Management* refers to the times when the participants' focus was devoted to the tool itself rather than the activity at hand, and *Technology Confusion* refers to time lost due to system failure. Lastly, any category that did not fit into any of the above was categorized as *Other*.

Due to limited time and resources, only a subset of the sessions, specifically six Calico and six whiteboard sessions, were coded. While this is not enough to claim statistical significance, it is enough to gain a sense if there is, in fact, an apparent difference in process between the two conditions.

In order to verify the validity of the coding, we performed an interrater reliability test, and also consulted with researchers who previously applied this coding in their own past studies. After initial training, two individuals independently coded a session, and then compared their coded transcripts to perform an interrater reliability test. We obtained a Cohen's k value of 82% at this level of granularity. We then compared the total time that each coder had for each category, and found a correlation of .998. These measures are well within the accepted tolerance for behavioral analysis.

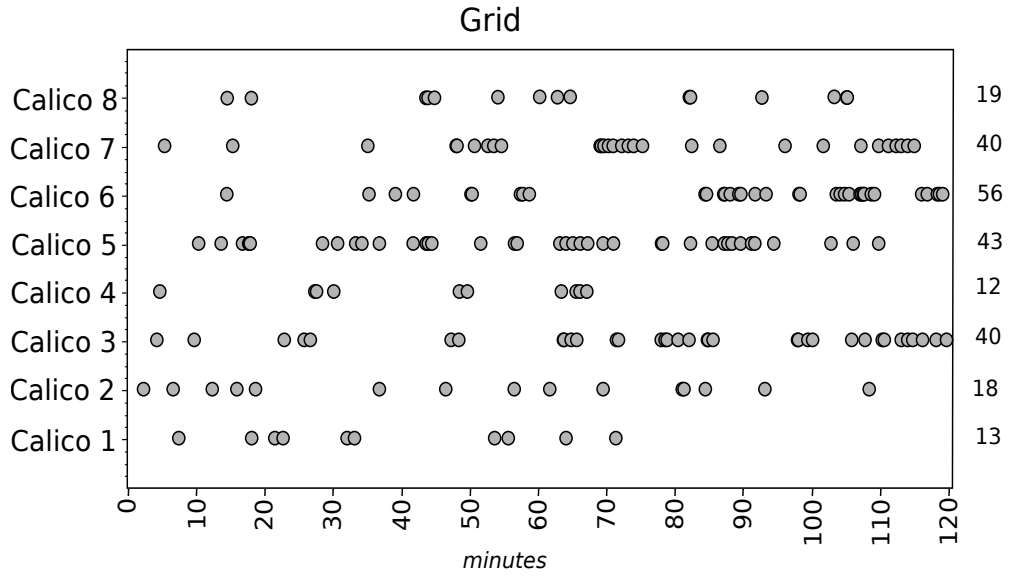
Satisfaction and perceptions of participants. At the end of each session, we interviewed participants in order to learn how Calico affected their approach to the design task, and what the positive or negative aspects of Calico were for the users. We did this by first asking participants to reflect on their designs and the process they used to get there. We then asked them explain how they used each of the advanced features, their overall satisfaction with each feature, and whether they had any suggestions for improvement.

3.5 Results

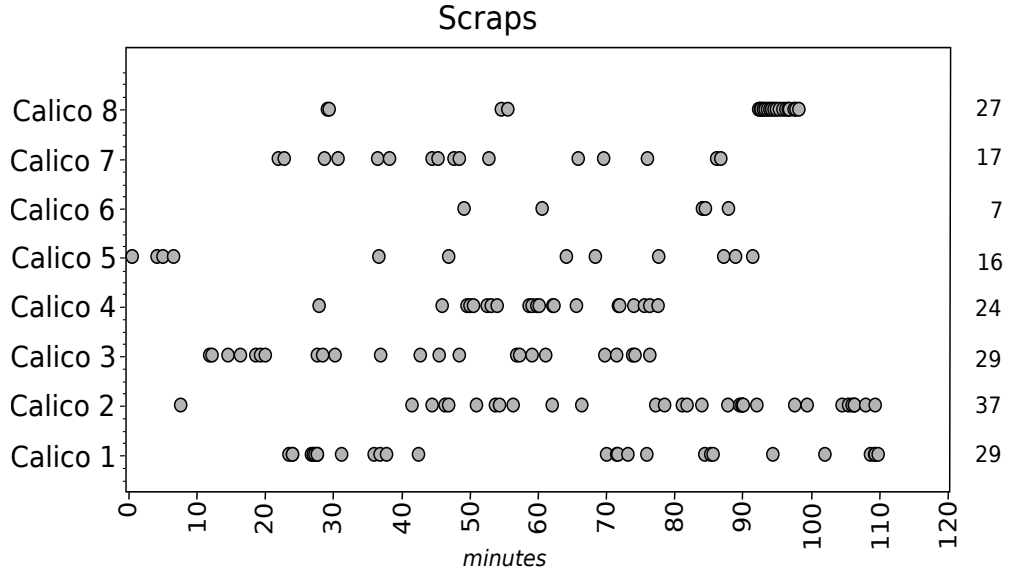
The discussion of our results is organized by the major categories of analysis described in the previous section: use of features, design behaviors, design conversations, and satisfaction. In this section, we only present the results that we observed; we do not make attempts to interpret them. In Section 7, we bring the results together and draw our conclusions about the value of Calico.

3.5.1 Feature Use

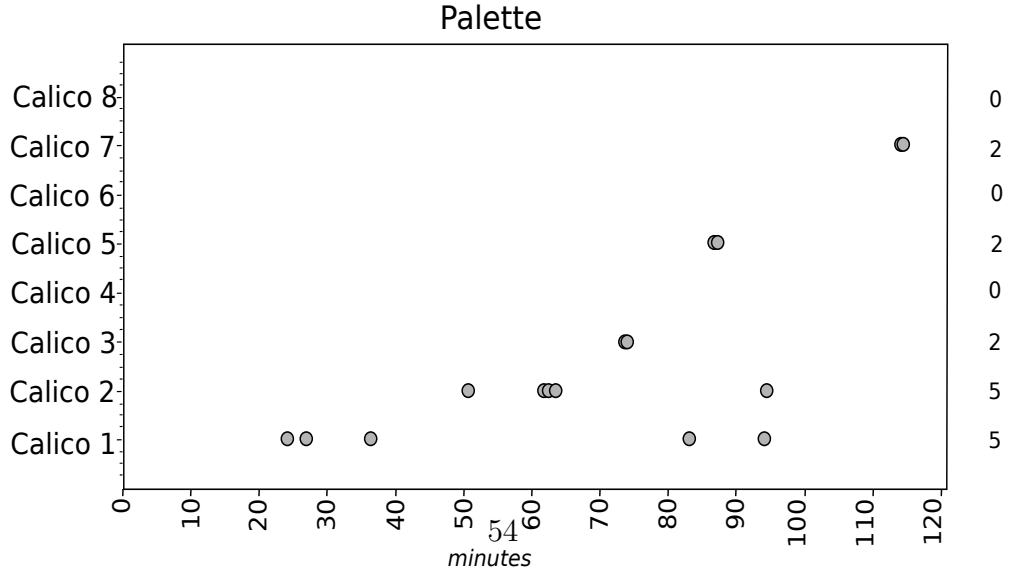
We first focused on whether participants would move beyond just basic sketching to using the advanced features of Calico. This point is important, given that there was no incentive


Grid
Calico 8
Calico 7
Calico 6
Calico 5
Calico 4
Calico 3
Calico 2
Calico 1
19
40
56
43
12
40
18
13
0
10
20
30
40
50
60
70
80
90
100
110
120
minutes

(a)


Scraps
Calico 8
Calico 7
Calico 6
Calico 5
Calico 4
Calico 3
Calico 2
Calico 1
27
17
7
16
24
29
37
29
0
10
20
30
40
50
60
70
80
90
100
110
120
minutes

(b)


Palette
Calico 8
Calico 7
Calico 6
Calico 5
Calico 4
Calico 3
Calico 2
Calico 1
0
2
0
2
0
2
5
5
0
10
20
30
40
50
60
70
80
90
100
110
120
minutes

(c)

for participants to use any of the advanced features other than the features being useful to their task at hand; they could have instead chosen to sketch as they normally would on a standard whiteboard without the help of scraps or the grid.

Figure 3.8 shows the result, marking each time the grid (a), scraps (b), or palette (c) was used. From the graph, we note that the grid was unanimously used by all pairs, with heavy usage by most pairs and moderate by some. The pairs predominantly switched to the grid view first for navigating to different canvases, though as time went on, several incorporated the use of the tabs to navigate to adjacent canvases (see Section 4).

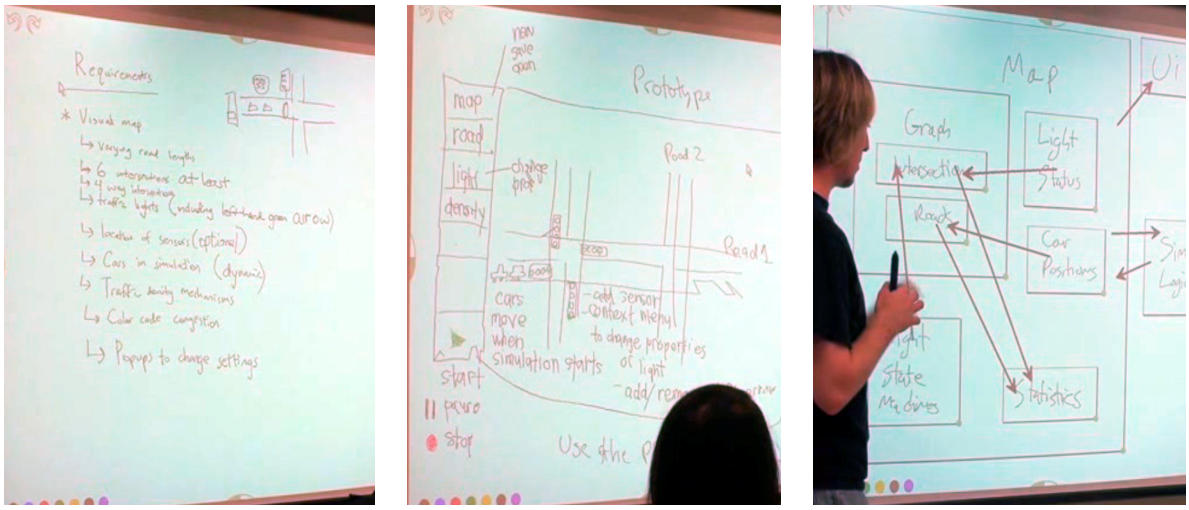
There was a wide distribution in the frequency of scrap use, with some pairs strongly relying on scraps, others exhibiting more moderate use, and a few barely using them. The palette was the least used feature, with just two pairs using it some and three other pairs using it twice each.

3.5.2 Design Behaviors

Given our goal of supporting the specific design behaviors identified in Section 3, this section focuses on occurrences of these behaviors in the various pairs, as well as how the advanced functionality of Calico was used in the presence of these behaviors.

Shifting focus

As the heavy use of the grid would suggest, Calico assisted designers readily in shifting focus. All sessions resulted in grids similar to the one shown in Figure 3.4, with the designers moving back and forth among canvases frequently. Pairs switched to a new canvas to either: (1) address additional detail generated by the current sketch, or (2) generate a new alternative. In the majority of cases it did not matter where on the grid a pair went to continue their



(a) List of requirements in bullet point form

(b) UI mockup of the traffic simulator interface

(c) Architecture in the form of boxes and arrows

Figure 3.9: Participants tended to focus their efforts towards a particular theme in each canvas

design activities, as a consequence of which we observed quite a few linear chains of canvases on the grid in use. Several interesting cases did emerge, however. For example, in one session, each of the pair members "owned" a grid row of design content in that they were the primary designer for all of the sketches in their row. As another example, one pair used the spatial metaphor extensively, with members talking about moving in a given direction (i.e., left, right, up, down) to navigate to certain sketches. While not all pairs operated as explicitly in terms of direction, the grid's spatial orientation did provide a consistent layout that enabled the pairs to shift focus by navigating to different canvases that they knew existed in certain locations.

A number of pairs exhibited bursts of back-and-forth switching between canvases to compare content, as illustrated in Figure 3.8(a) by the overlapping sequences of dots. This happened for two reasons. First, participants moved back-and-forth when they needed to mentally juxtapose two concepts, such as when they were working to improve their design for the user interface in conjunction with the underlying model. Second, they would take stock to verify that the different sketches were consistent with one another to make sure they had

not inadvertently introduced problems when they furthered some aspect of their design.

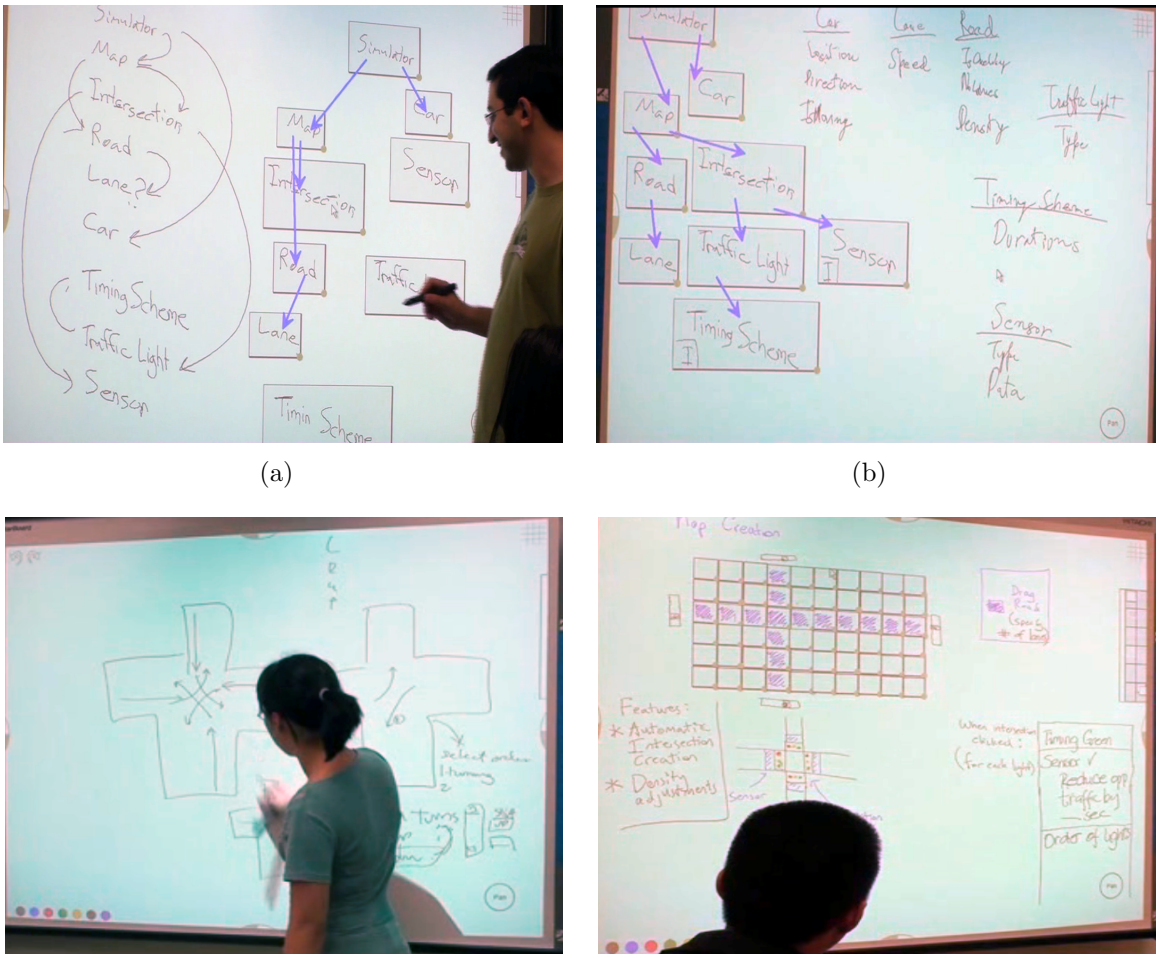
The contents of the canvases tended to break down in three categories: (1) requirements, (2) code structures, and (3) UI mockups, with examples of each depicted in Figure 3.9. Every pair used the first canvases to lay out their requirements in a list, as in Figure 3.9(a), after which they would move to another canvas to embark on the design proper, producing diagrams similar to Figures 3.9(b) and 3.9(c). Pairs would frequently return to the requirements for inspection and assessment of their progress.

The ability to copy canvases proved useful not only to generate alternatives (one can see several variant sketches in Figure 3.4), but also when it was desirable to divide the contents of a canvas between two canvases so to be able to work on each part separately. Copy followed by subsequent deletion of the respective halves of the copied content achieved this desired result. One of the pairs used the copy feature to create backups of individual canvases, and moved these backup copies out of the way by dragging the canvases to the outer edges of the grid. Several other pairs moved canvases in the grid around to more clearly organize their design.

Overall, we observed that participants use the grid as a tool to divide design content across canvases, where a shift in attention was commonly accompanied by a change of canvas, and long periods of attention were marked by extended periods of continued activity in a single canvas (visible as gaps between points in Figure 3.8). Additionally, we observed the spatial layout to be helpful to designers, both in providing clear references to design content and in organizing the overall design effort.

Low-detail models

We observed the use of low-detail models in every single design session. The pairs that used Calico created the same type of low-detail models that we saw on the whiteboard, and in



(a) (b)

(c) (d)

Figure 3.10: Examples of models in varying amounts of detail

most cases they used scraps to create these models. Of the eight Calico pairs, two simply sketched directly on the canvas as they would have on a traditional whiteboard, and did not further manipulate their sketches. In the cases where pairs did use scraps, they benefited from the extra functionality, such as moving scraps to reorganize a design, copying them in order to create variations, and arrows to define relatioships.

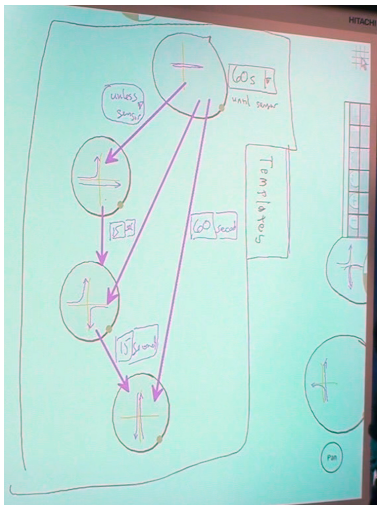
The images in Figure 3.10(a) and 3.10(b) illustrate an example of how many of the pairs used scraps to create box-and-arrow representations of code structures. The pairs that did not use scraps, such as the pair that produced Figure 3.10(c), drew boxes and arrows, but did not heavily restructure their diagrams. The pairs that did use scraps engaged in more organization of their box-and-arrow diagrams through the moving and copying of scraps. Additionally, they typically refined their scrap-based models into UML-like models, showing how scraps can be used as the basis for such refinement from low detail to more detail. In the case of Figure 3.10(b), the designers annotated some of their scraps with "I" to indicate that it is an interface, and on the right-hand side they created lists of parameters for each object.

Many of the other diagrams produced in Calico shared similar qualities with the low-detail diagrams, with Figures 3.10(c) and 3.10(d) providing two more examples. These diagrams were quick sketches that participants initially used to help them understand a particular situation. The designers in Figure 3.10(c) did not use scraps heavily and so they experimented with traffic configurations by simply drawing and erasing different scenarios. The designers in Figure 3.10(d) used scraps to accomplish a similar task. In this case, they used scraps of different shades to represent traffic configurations, and experimented with different configurations by moving and copying the scraps. Additionally, they used scraps to pull pieces of the design from surrounding canvases in order to understand how a particular traffic configuration would work within the context of other parts of the system.

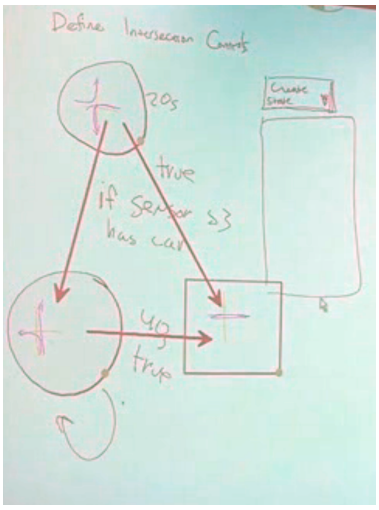
Mix of notations

As expected, numerous notations were used by the participants as part of their design process. These include class diagrams, user interface components, and diagrams specific to the domain of traffic simulation. These different types of diagrams were often created in response to one another. In the whiteboard sessions, this led to heterogenous content in different notations spread out over the whiteboard. In the Calico sessions, the pairs broke up their designs across different canvases, as exemplified in Figure 3.9, and as a result most canvases used just a single notation. Pairs sometimes did mix different notations in a single canvas, such as in the Figure 3.10(d), where one of the pairs uses representations of a traffic intersection in the top of the image with pieces of code representations in the bottom-right of the image, on the same canvas. This pair, as well as other pairs that similarly mixed representations, used the different representations to juxtapose different views of the design.

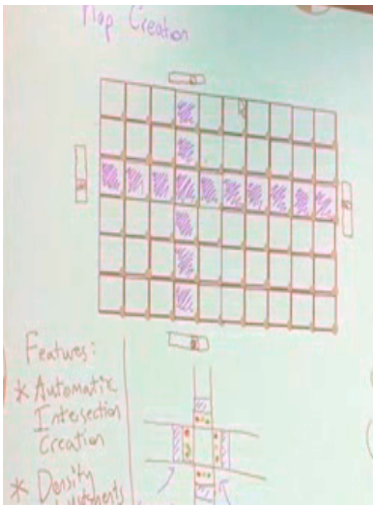
Pairs had no ready notation to represent traffic structions, and so they created their own on the fly. For instance, the image in Figure 3.11(a) shows how one pair used scraps to model a state diagram that is part of their vision for how civil engineering students will specify the timing of traffic lights in their simulation. Note how the state diagram exists on one of two tabs meant to be part of the user interface. Next, in Figure 3.11(b), the same pair applied a similar notation to define the logic that is executed once a car arrives at an intersection. Here the pair reused the same diagrammatic elements by copying scraps. In Figure 3.11(c), another pair created dozens of small scraps and placed them in a grid configuration to simulate the interface for traffic flow. They filled in select canvases with the pen to simulate a particular route, and experimented with different routes by erasing and filling in other scraps. In all three examples within Figure 3.11, pairs were able to create copies of their scraps and reuse them to attempt alternate combinations.



(a) State diagram for defining light combinations at an intersection



(b) State machine that is executed when a car arrives at an intersection



(c) Simulation of traffic flow on the map and the controls to regulate it

Figure 3.11: Several impromptu notations emerged in the design sessions

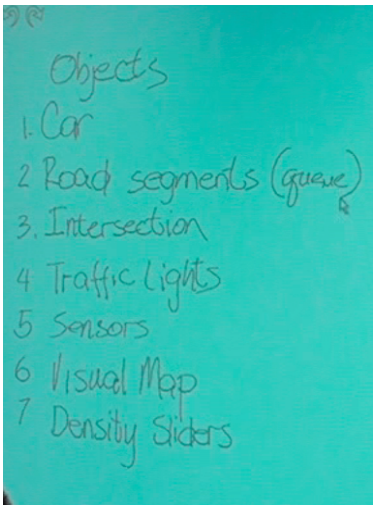
Refinement of representations

Both the grid and scraps were used in the refinement of representations over time. At the most general level, the grid partitioned the design space so that participants could separate high level and low level representations. Many pairs used spatial orientation to navigate from higher level representations to lower level representations, where higher level representations would exist at the typically left-most canvases, and the details of components in adjacent canvases to the right or below the originating canvas. Next, when members made the transition to more concrete representations, they would transform sketches into scraps, and create copies on other canvases where they would expand them in more detail and create relationships between them.

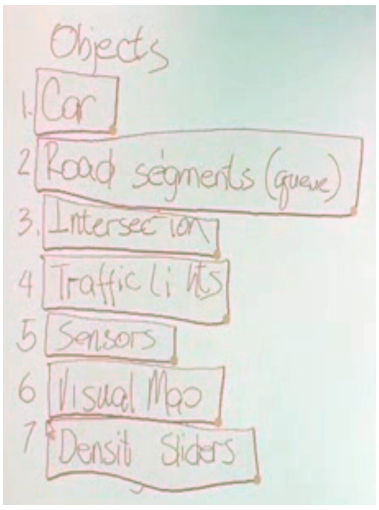
There were two patterns of this behavior that occurred frequently within pairs. Figure 3.12 illustrates a representative example of the first pattern, which occurred in over half of the Calico pairs. Most pairs, but not all, began by creating several lists of design requirements, goals, and what they viewed as major aspects of the system. The example in Figure 3.12(a)

contains several high-level components such as Maps, Intersections, Cars, and so on. After brainstorming lists such as these, pairs would commonly convert them into scraps (Figure 3.12(b)) and copy them into another canvas, where they would expand on these in more detail, as in Figure 3.12(c). However, creating lists such as these was not common in all pairs, two pairs chose to jump right to diagramming. In the exit interviews, these two pairs both reported that they were first concerned with understanding the requirements and the world that they were modeling. They both began by creating concrete representations of what they knew was true within intersections, and then developed a high level understanding by drawing on top of and discussing these models with their partner. To accomplish this, they created many diagrams of intersections, and from these jumped to writing down assumptions in lists contained in other canvases. When they encountered a new concept that they sensed was complex, such as how to handle the timing within intersections, they would loudly say, "let's leave this for later", and leave it as a generic annotation.

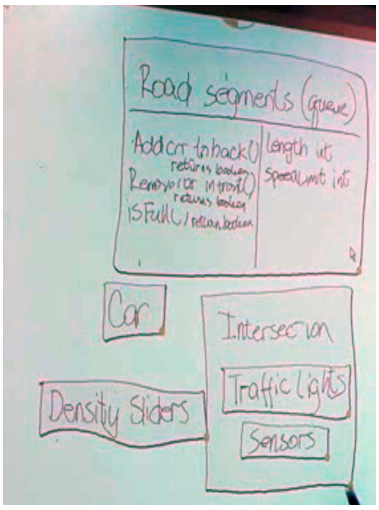
A second pattern that occurred more frequently was the breaking down of the design across multiple canvases. A representative example is depicted in Figure 3.13, in which the participants from that pair partitioned their code structure across many spaces. They created a high level perspective of the architecture in one canvas using scraps (Figure 3.13(a)) and then copied these scraps to adjacent canvases, where a particular scrap would be expanded to include more detail. In Figure 3.13, the participants divided the architecture of their program into UI, Map, and Simulation Logic. They copied the contents of this canvas to an adjacent canvas, where they subsequently fleshed out the details of Simulation Logic, but left Map and UI with no additional detail. They then used a third canvas to work out the details of Map, while leaving Simulation Logic and UI as is. Using this divide and conquer strategy, the participants were able to effectively partition their design. This behavior of spreading diagrams across several canvases is similar to what Dekel and Herbsleb [25] observed in their studies, where diagrams in one drawing space would depend on references located in other spaces.



(a) List of high level objects

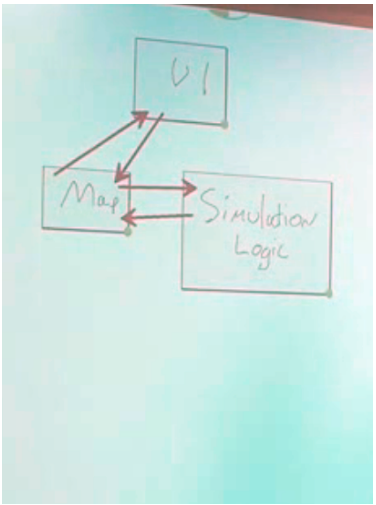


(b) List of high level objects converted to scraps

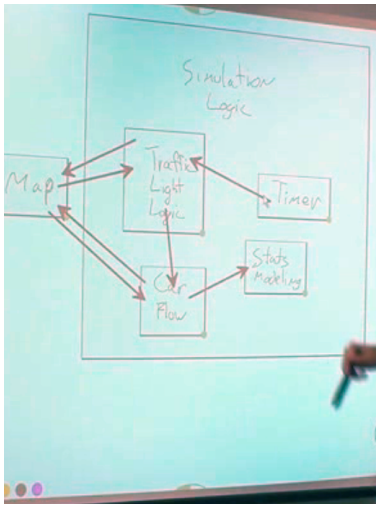


(c) High level objects refactored and expanded

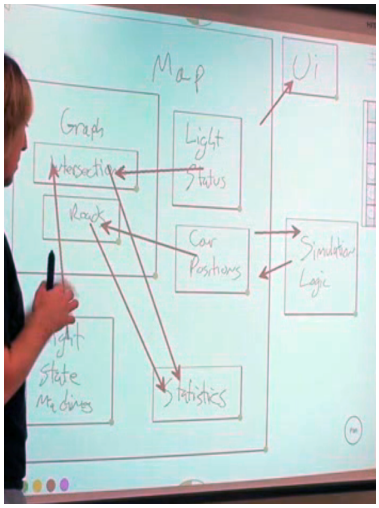
Figure 3.12: Lists were converted into representations of software components using scraps



(a) High level perspective of system architecture



(b) Definition of Simulation Logic



(c) Definition of Map

Figure 3.13: Representations of software components were broken up across several canvases

Table 3.2: Summary of design conversation categories.

Category	Whiteboard Pairs	Sum	Calico Pairs	Sum
Issue	6.37		7.34	

3.5.3 Structure of Design Conversations

We now examine the structure of the design conversations within the Calico pairs, and see how it compares with the pairs that performed the activity on the whiteboard.

How time was spent

We first determined the structure of the design process by measuring the total time spent in each design category during conversation. We coded the transcripts of the spoken parts of the meetings, summarizing the total time spent in each category, as shown in Table 3.2.

Categories related to design rationale (i.e., *issues*, *alternatives*, and *criteria*) took up the majority of the design sessions. In the Calico pairs, these took up 67.5% of the meeting time, and in the whiteboard pairs they took up 62.7% of the meeting. Both Calico and the whiteboard had a similar breakdown within the design rationale categories. Within the Calico and whiteboard pairs, discussing *alternatives* occupied 41.8% and 37.0% of the time, respectively. Both pairs spent roughly the same amount of time discussing *issues*, at 7.3% and 6.4% for Calico and whiteboard pairs, respectively, and also the same amount of time evaluating solutions in the *criteria* category, with 16.5% and 15.6% for Calico and whiteboard, respectively. However, the average length of stay in each category, i.e. the average uninterrupted time for a particular category, was higher for the Calico pairs than the whiteboard pairs. The average length of stay for *alternative* in Calico pairs was 13.0 seconds compared to 8.7 seconds for whiteboards. The average length of stay for *criteria* in Calico pairs, 11.0 seconds, was also longer than that for the whiteboard pairs, 7.5 seconds. However,

the average length of time per *issue* proposed was similar, with Calico pairs averaging 8.1 seconds and whiteboard pairs average 7.1 seconds.

Roughly double the amount of time was devoted to management related activities in Calico pairs compared to whiteboard pairs, however this time difference was due to discussion of the technology. Management related categories occupied 22.8% of the meeting in Calico, while they only occupied 12.4% of the meeting in the whiteboard pairs. Of that time in Calico, 8.8% was spent discussing the technology (combined sum of *Management of Technology* and *Technology Confusion* categories). Both pairs spent relatively the same amount of time explicitly discussing meeting management, 7.3% and 6.4% for Calico and whiteboard pairs, respectively. The other categories were roughly similar as well.

We also recorded the amount of time spent not talking, which we recorded as *pause*. We found that the whiteboard pairs were quiet for more than double the period of Calico pairs, with whiteboard pairs silent for 24.0% compared to 9.7% for Calico pairs. Upon closer inspection, we saw that the majority of whiteboard pairs used this period of silence to work independently of each other. Calico pairs were not given this opportunity since the system only permits one user to write at a time. While we do not measure the impact that this had here, during the exit interview participants reported frustration over their inability to work independently. As a result, they reported that they had to force their attention on what the other person was writing, and abandon ideas they were thinking of independently because they could not write them down.

We then recorded time spent clarifying answers across all categories, though we found that little time was dedicated to clarification within both sessions. The combined percentage of clarification across all categories for the Calico pairs was 2.8%, while it was 4.3% for the whiteboard pairs.

Lastly, we examined pairwise levels of similarity across all pairs, shown in Table 3.3. We

Table 3.3: Correlations between sessions for total time spent (dark cells pertain only to Calico sessions, white cells pertain only to whiteboard sessions, and lightly shaded cells are the intersection of both)

	C.1	C.2	C.3	C.4	C.5	C.6	W.1	W.2	W.3	W.4	W.5
C.1											
C.2	0.90										
C.3	0.98	0.94									
C.4	0.97	0.96	0.98								
C.5	0.94	0.98	0.97	0.98							
C.6	0.95	0.95	0.98	0.98	0.97						
W.1	0.94	0.96	0.97	0.96	0.99	0.97					
W.2	0.41	0.20	0.44	0.33	0.27	0.43	0.29				
W.3	0.92	0.97	0.94	0.95	0.98	0.94	0.98	0.20			
W.4	0.92	0.98	0.94	0.98	0.99	0.95	0.98	0.16	0.99		
W.5	0.61	0.55	0.73	0.67	0.61	0.74	0.61	0.92	0.54	0.53	
W.6	0.91	0.91	0.95	0.93	0.93	0.97	0.93	0.57	0.91	0.90	0.83

compared the average time spent within each category across Calico and the whiteboard conditions, and found a correlation of .92, indicating a strong relationship. All Calico pairs had a strong correlation with one another, ranging from .90 to .98. Within the whiteboard pairs, there was a much larger variability, with correlations ranging from .16 to .98, with a median of .58. A major way in which the meetings varied, particularly between Whiteboard Session 2 and Session 5, was in the amount of time that was categorized as *pause*. Removing the pause category from the pairs raises the correlations significantly to the range of .93 to .99 (not visible in table). Overall there appears to be much similarly across all sessions, with pairs within the Calico condition having the tightest range when considering all of the categories.

Transitions Between Activities

In addition to total time, we also examined all incoming and outgoing transitions that pairs made between categories in order to understand the flow of design conversations. We first computed the total number of times the participants transitioned from one category to

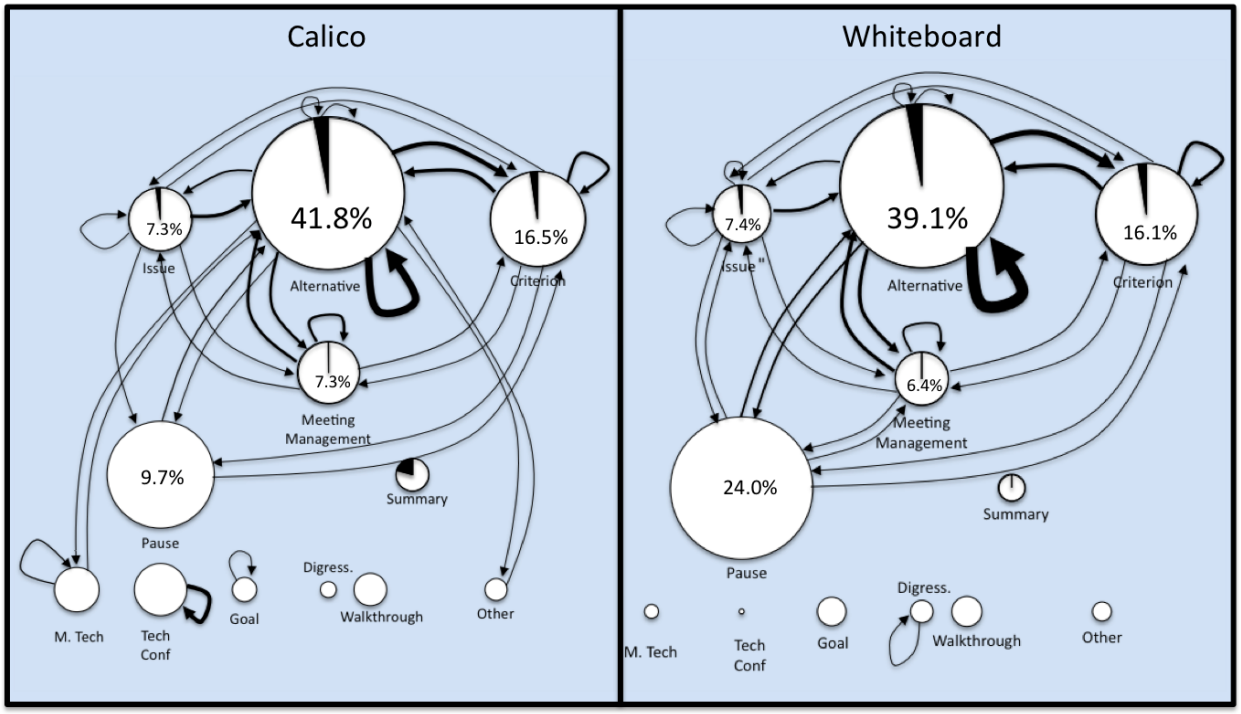


Figure 3.14: How time was spent within and between the design activities

another, which is shown in Figure 3.14 through arrows, the thickness of which representing the relative frequency that a particular transition occurred. The most common transitions that occurred belonged to the design rationale categories: *issues*, *alternatives*, and their *criteria*. In Calico they were involved in 76.8% of all transitions, and 77.7% of all transitions in the whiteboard pairs.

We then computed the correlation matrix for the Calico and whiteboard pairs, and found striking similarities between the transitions of all 22 categories. In order to get the correlation matrix, we computed the average percentage that a given transition happens between one category to the next, and established the correlation matrix shown in Table 3.4. The average transition that happens in the Calico pairs was strongly correlated with that of the whiteboard pairs at .92. As with the previous table of correlations, the Calico pairs, on average, are more highly correlated with one another than the whiteboard pairs and have a tighter range, The Calico pairs range from .63 to .93, compared to whiteboard pairs, which

Table 3.4: Correlations between sessions for transitions (dark cells pertain only to calico sessions, white cells pertain only to whiteboard sessions, and lightly shaded cells are the intersection of both)

	C.1	C.2	C.3	C.4	C.5	C.6	W.1	W.2	W.3	W.4	W.5
C.1											
C.2	0.82										
C.3	0.86	0.88									
C.4	0.70	0.75	0.76								
C.5	0.90	0.88	0.89	0.63							
C.6	0.90	0.86	0.95	0.76	0.93						
W.1	0.83	0.83	0.83	0.45	0.94	0.84					
W.2	0.45	0.45	0.66	0.45	0.46	0.64	0.44				
W.3	0.85	0.84	0.80	0.46	0.93	0.83	0.98	0.41			
W.4	0.88	0.88	0.82	0.59	0.90	0.87	0.91	0.47	0.93		
W.5	0.58	0.66	0.81	0.49	0.65	0.76	0.62	0.88	0.57	0.62	
W.6	0.88	0.88	0.80	0.54	0.91	0.86	0.94	0.42	0.95	0.97	0.60

range from .41 to .98. As with the previous table of total time spent, whiteboard pairs 2 and 5 also have low correlations with the other whiteboard pairs, and a high correlation with each other, suggesting a distinctive difference in work style. Also, unlike the previous table, Calico pair 4 correlates slightly less, on average, than other Calico pairs, and much less with whiteboard pairs, suggesting they used a slightly different process as well.

Additionally, we looked for patterns of transitions. In order to discover patterns, we use the same method described in Olson et al. [64]. In this method, we observe for transitions that occur more often, or less often, than they are statistically expected to. In order to determine the expected value, we first calculate the number of times that a particular category occurs, and divide that number by the total number of instances of all categories, which gives us the expected percentage of occurrence. We then compare this expected percentage with the actual percentage for a particular transition. If no structure exists in the design session, then the probability of moving from one activity to the next will be the same as the expected percentage. However, if a pattern exists, then we will observe a deviation between those two numbers.

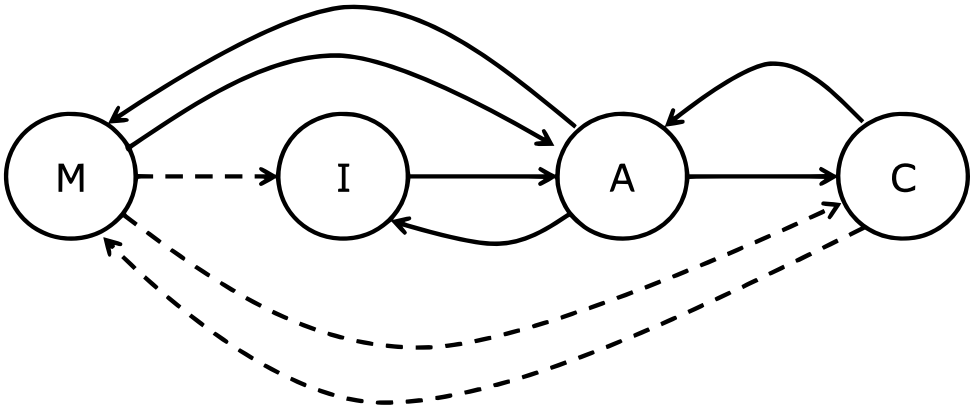


Figure 3.15: Emergent pattern of transitions across both the Calico and whiteboard pairs

Using the above method, we observed sequences of up to four transitions, and a pattern began to emerge that was present in both the Calico and whiteboard pairs. The pattern that emerged is shown in Figure 3.15. The solid lines in Figure 3.15 represent strong trends, while the dashed lines represent transitions that are weaker, but still occur well above probabilistic levels. M stands for any management category, I for *issue*, A for *alternative*, and C for *criteria*, in accordance with the categories in Figure 3.14. There was a strong tendency to make frequent transitions between alternative-criteria, or between management-alternative. Additionally pairs commonly engaged in serial evaluation, where one participant would challenge the evaluation of the other, leading to new criteria with which to judge the alternative. Within the pure design categories, sequences of alternative-criteria transitions were typically punctuated by either the discussion of an issue, or a category from management. Olson et al. called these combinations IAC or MAC, based on their category names. These specific combinations are examples of "design episodes", which were found by them to be common in design meetings. We found these design episodes to be present in both the Calico and whiteboard pairs. The episodes of IAC, which has an expected value of .04%, appeared to happen more commonly in Calico pairs, occurring an average of 3.25% of all transitions, compared to 2.33% of all transitions for whiteboard. Episodes of MAC, also with an expected value of .04%, appeared to happen more commonly in whiteboard pairs at 2.64% of all transitions, versus 1.61% of all Calico transitions.

3.5.4 Satisfaction and Perceptions of Participants

At the end of each session we verbally interviewed the participants and asked them to reflect on their own design process and satisfaction with the tool. Despite some technical difficulties, six out of eight pairs reported that they greatly enjoyed using Calico. The grid was unanimously praised as a tool to organize the design space. One participant praised that "it was the first time [they] felt like they were using a whiteboard [in a digital medium]", and expressed that other pen-based software tools that paginated their drawing space, such as Microsoft OneNote, made navigating designs frustrating. Five of the eight pairs explicitly praised scraps as tools for quickly copying content and moving it across canvases, however many expressed frustration that they were not able to rotate or resize them. Two pairs reported mixed feelings about the overall tool, stating that Calico merely felt like a digital whiteboard and did not compensate for the bulky hardware. All pairs reported dissatisfaction with the hardware in general, stating that it made their writing very sloppy, and that single user input was a "deal-breaker." When asked if they would use Calico in their own design sessions, seven pairs expressed a great deal of interest, but only if it allowed more than one user to draw at a time. All participants reported frustration over this issue, with one participant stating that it caused them to "spin their wheels" while they waited for their partner to finish writing.

Seven out of eight pairs reported that Calico did not alter or interfere with how they would normally approach a design problem, and that the ability to easily copy canvases encouraged them to explore more solutions than they normally would have. Also, pairs reported that the presence of scraps changed the way they approached creating lists, stating that they kept in mind the ability to easily copy list items and transform them into software architecture components. One pair of the eight felt that the benefits of Calico did not outweigh the hardware drawbacks, saying that they could have achieved the same design using a stack of papers. In general, nearly all participants were content with the final design that they had

created, with most participants calling their final design "a good start" given the limited amount of time they had to create their design.

3.6 Discussion

Building upon the results from Section 6, we now turn our intention to interpreting what they mean with respect to Calico use in support of software design at the whiteboard.

3.6.1 Calico Features

Bringing the results described in Sections 6 together, it is clear that Calico does provide support for the behaviors that we identified in Section 3 and that designers are able to use the features in support of these behaviors when they so desire. However, it is also clear that features were not always used when they could have supported a certain behavior and that not every pair utilized every feature.

We interpret the differences in use between the different pairs as stemming from two factors. First, some pairs had early success in using scraps, and discovered ways in which they made their life easier. They subsequently stayed with scraps more persistently. Other pairs simply drew on the canvas and ventured rarely into attempting to use scraps. With some early troublesome interactions (e.g., an accidental deletion by a slash-through, using the wrong stylus button to create a scrap), they became discouraged and used traditional methods instead.

Second, the benefits of using scraps are not always apparent until later. One has to anticipate needing to alter a drawing by creating it using scraps in the first place, otherwise one ends up drawing it twice: once as background sketches and once to create reusable elements from

parts of those sketches. This second step represents additional work, and may be skipped in favor of simply redrawing a sketch, especially if it is small or not-too-involved.

In both cases, we believe more training and the build-up of experience over time has the potential to overcome the issue. Simply sketching on a whiteboard is so ingrained, it takes time to internalize and adjust to a new form of interaction with sketched content.

Post-experiment conversations with the participants confirmed some of these observations. While we generally received very positive feedback, several issues were identified. First was that the nature of scraps was not as intuitive to learn as we had hoped. The participants did not think this was an outright failure, but did note the need for more training and more examples of scrap use. The participants also noted that tabs did not inform the user as to whether a neighboring canvas is occupied. Spatial navigation directly from canvas to canvas without first moving to the grid, therefore, is hampered. A simple mark to indicate a neighboring canvas has content, together with a popup on hover, may overcome this problem.

The participants also wished for extra functionality on scraps. Some of the requested functionality is generic in nature and straightforward to add, such as rotation, resize, and different types of arrows. Some, however, is more specific and pertains to the fact that designers mentally assign meaning to scraps and expect functionality commensurate to that meaning. Such expectations arose later in the sessions, when the design had been largely worked out and the designers now wished to provide additional detail to complete the design. For instance, they asked for functionality to refine scraps into UML classes with a name, variables, and functions, or to turn scraps into lists for easy reorganization and requirement tracking. We believe such refinement can be integrated without disturbing scraps' present sketchy, informal nature, which is so crucial in the early stages of design. Particularly, we envision typing of scraps, with the assignment of type (e.g., UML class, UI element, architectural component, ER element) signaling additional functionality becoming available.

3.6.2 Design Behaviors

With respect to how pairs approached the design task, the Calico pairs exhibited some very similar behaviors, while the whiteboard pairs noticeably differed at times. For example, not all whiteboard pairs used transient diagrams, some maintained all of their diagrams for the entire session and refined these over a period of time. In contrast, other whiteboard pairs continuously erased what they drew and often worked from memory. Some pairs chose to work together the entire time, while other pairs worked on their design siliently in parallel at extreme ends of the board. One of the pairs that did this eventually did review each others' design and reconciled their different approaches, while another pair largely operated independently the entire session with few design decisions being contested or brought together. The Calico pairs, however, seemed to be implicitly guided into certain behaviors by the tool, and these behaviors were consistent across the majority of Calico pairs. The abundance of space led pairs to save and evolve their diagrams over time to a much greater extent. Scraps made it possible to reuse, as well as refine, diagrams, which nearly all pairs did. Additionally, the grid caused the pairs to create clear divisions between different aspects of their design. Overall, it appears that Calico's features led pairs towards the specific behaviors of design that we planned for, while the whiteboard pairs, who did not have these features, varied much more greatly in their approaches. Whether or not these behaviors lead to a positive impact on the overall design is yet to be determined, but, regardless, our observations point to the conclusion that these behaviors are supported by Calico.

3.6.3 Interference

Another important result of the study is that Calico did not interfere with the design activity. If anything, it led to more issues, alternatives, or criterias that were discussed. In the exit

interviews, all designers agreed that Calico did not much effect the nature of their design conversations, and the results from the analysis of the design structure tend to support that. They dedicated relatively the same amount of time of each session discussing design activities at 62.6% and 67.5% for whiteboard and Calico conditions, respectively. Further, the use of Calico had little effect on the amount of time spent managing the activity, at 6.4% and 7.3% for the whiteboard and Calico pairs, respectively. The pairs intuitively understood how to manage their design using the grid, and so they did not need to exert additional effort in managing it. This behavior falls in line with previous research that supports the notion that designers make heavy use of spatial properties to orient themselves and organize their design [62, 6]. In the normal whiteboard sessions, it was not uncommon for participants to look back and forth between two diagrams on opposite ends of the board, and sometimes place a hand on one diagram while they looked at another. Participants used Calico in much the same way by rapidly moving back and forth between two canvases when considering two diagrams. In one exit interview, one pair requested the ability to view two canvases simultaneously in a juxtaposed view to allow a similar activity.

Calico did, however, seem to cause some changes. The Calico tool seemed to force a greater degree of homogeneity between the pairs. The pairs that used Calico had a higher amount of conformity between them with respect to the total time spent in each category than the whiteboard pairs. The correlations between pairs for transitions between categories demonstrated the same tendency. This discrepancy in design conversations between the Calico and whiteboard pairs seems to, by and large, be related to the amount of time spent working independently. Two whiteboard pairs, 2 and 5, spent a significant portion of the session working quietly in parallel, which is reflected in their low correlation with other pairs for time spent in Table 3.3. Interestingly, the transition frequency correlation matrix in Table 3.4 for these two pairs shows the same pattern of discrepancy. This suggests two things: (1) the pairs that chose to spend a significant amount of time working silently in parallel had a different design conversation structure than those whiteboard pairs that did not spend

much time working silently, and, conversely, (2) the pairs that did not spend significant time working independently had a sequential process that correlated relatively highly with the Calico pairs. This second point is interesting because it shows that, while many of the Calico pairs considered themselves handicapped by the inability to work independently, the structure of their design process was similar to the highly collaborative whiteboard pairs.

3.6.4 Focus

Finally, after comparing the videos with the coded transcripts, it seemed that the pairs that used Calico had a greater amount of focus on their partner's ideas and a slightly better shared understanding of recorded ideas than the whiteboard pairs. While both the Calico and whiteboard pairs spent a relatively similar amount of total time per session discussing *issues*, *alternatives*, and *criteria*, the Calico pairs had a larger average of *consecutive time per alternative and criteria* discussed. The average length of continuous time that an *alternative* was 50% longer in Calico pairs than whiteboard pairs (13 seconds versus 8.7 seconds) and the average length of continuous time a criteria was expressed was also nearly 50% longer in Calico pairs than whiteboard pairs (11 seconds versus 7.5 seconds). While this was, in part, due to the single input interface of the electronic whiteboard, other factors in Calico forced the focus of participants well. Within Calico, the pairs would partition their design into different canvases in the grid, and so the drawing spaces in Calico tended to have a tighter focus than the whiteboard, which shows all of the content at once. As a result, the participants in the Calico pairs were not distracted by drawings within other parts of the design, whereas the whiteboard pairs could gaze at any part of their design at any time. When a participant in a Calico pair did refer back to other parts of the design, they had to switch to an entirely different canvas via the grid, effectively forcing their partner's attention to move with them to that part of the design. While Calico pairs were not able to move between topics as quickly as whiteboard pairs, it led to more elaborate explanations

and longer individual responses when presenting a *criteria*. Within the whiteboard pairs, it would sometimes be the case that while one partner would be explaining their solution, the other would ignore them while they considered another part of the design. The split attention was reflected in the longer time spent clarifying ideas for the whiteboard pairs, which was 53% longer than Calico pairs (4.3% of the total session for whiteboard pairs, on average, versus 2.8% of the total session for Calico pairs, on average).

3.6.5 In Sum

Overall, our detailed analysis of how Calico was used and influenced the design behaviors and conversations, as well as the positive responses to Calico from the participants, show that Calico has promise in enhancing design at the whiteboard. More elaborate training is likely needed for users to properly take advantage of scraps, but we still saw that Calico's features are helpful, while avoiding impeding the design process.

Calico's strength lies in its support for the informal process of design, during which the emphasis is placed more on unstructured exploration and less on a precise analysis of the design. Some pairs benefited more than others from Calico's features, but overall, Calico assisted them in using an effective design process in which their natural behaviors were supported with explicit features. Most importantly, they were able to use the grid to partition their design effort, scraps to create diagrams with easily manipulable elements, and, occasionally, the palette to reuse impromptu design languages. Most used the grid only for working with content across multiple spaces, but some benefited from the palette as well, such as when they reused scraps on multiple canvases. Calico's strengths also include supporting the individuals in focusing on each other during the design session, as shown by our conversation analysis.

The tradeoff to Calico's support for informal design is that the resulting designs remain

sketches, and cannot be analyzed or used in ways that more formal design tools support. This kind of tradeoff is a recognized problem with informal tools, and overcoming it is becoming an active area of research [69]. Another observed weakness is that Calico's interface is so different, that it takes users significant time to become familiar. Many of its benefits require the user to think ahead, otherwise they may not be using Calico optimally. Scraps in particular have a delayed pay-off, which led some pairs to not use them since they were not needed in the moment. Finally, Calico's single user input limits how a group of people may use the tool. While it improves focus, it can effectively become a bottleneck to expressing ideas if the participants want to temporarily work in parallel, a behavior which we saw put to both good and bad use in the whiteboard pairs.

3.7 Theats to Validity

Several threats to validity exist that we must keep in mind when considering our results. First, with respect to construct validity, the participants were limited to a single two-hour time span and had no access to others. Real design typically happen over longer periods of time and involve many different stakeholders. Despite this difference, during these longer periods of time, design at the whiteboard certainly happens [13]. While our work will not support all of a design, it usefully support that slice of design that happens then and there.

Second, we must consider threats to internal validity, i.e., the factors that affect our ability to claim cause-effect relationships from the results. The single user input limitation and therefore the prevention of parallel work, is the primary concern. Exit interviews confirm this as participants in the Calico pairs felt handicapped by this limitation, and given the opportunity they would have preferred parallel work. On the other hand, results from our analysis also show that Calico pairs spent a greater amount of uninterrupted time explaining alternatives and criteria to each other, building greater cohesion. Moreover, Calico sessions

still exhibits significant similarities with the non-Calico sessions in the ways the designers worked. Third, the conditions of the experiment may pose a risk to its external validity, i.e., its ability to generalize to what people do. In order to use Calico to its maximum potential, it would require a longer period of time than what was available in the experiment in order for people to become familiar with its features and power. Another factor was that the participants in the study were all graduate students, and so there is a chance that they may not represent the behaviors of experienced professional software designers. Novice designers have a greater tendency to become fixated on a design decision prematurely [4], and novices tend to mismanage their time by focusing on a single issue for an extended period whereas professional designers know when to move on [1]. Given that the comparative nature of our experiment focused on Calico versus non-Calico design, and not on expertise, we did not further examine this factor.

Despite the fact that the experiment has its differences with real-world design, it is interesting to note that there is a relationship with how real designers work. The study conducted by Olson et al. [65] demonstrated that a design session conducted in a controlled laboratory experiment is representative of a design session as it happens in the field with professional designers. Since our study uses the same measurement as those from [65] and also has a high correlation with their results ($r = .85$), by transitive logic, we can say that the study performed here is representative of a design session as it occurs in the field.

3.8 Related Work

Existing sketch tools can be classified into two categories: (1) those that interpret sketches with the purpose of turning them into formal objects, and (2) those that support sketching activities in general. The survey on sketch-based systems by Johnson et. al. [39] provides a more extensive look of all tools created to date than what we necessarily can provide in the

blow.

3.8.1 Tools that Interpret

The tools within this category interpreting a digital freeform sketch and then generate a formal representation. While we purposefully chose not to interpret sketches in Calico, these tools remain relevant as background work. Many tools that interpret sketches focus on generating prototypes for user interfaces. One of the earliest such tools is SILK [46], which allowed users to generate a usable GUI from a sketch, and then generate the corresponding Java code to implement that GUI. Another tool, DENIM [61], allows users to create low fidelity mockups of websites and then run simulations of that website. InkKit and Freeform, two other systems that created usable GUI's from sketches [15, 75], allowed interpreted content to visually retain its sketch appearance, based on the insight that visually beautifying content too early can be harmful to the designer [81].

Another group of interpretive sketch systems includes those that recognize UML elements. The earliest such tool was Knight [21], which allowed designers to create partial UML diagrams by selecting a portion or all of their sketch to be interpreted as UML elements. A later tool, SUMLOW [11] allowed interpreted sketches to retain their sketchy appearance (citing InkKit as inspiration), and allowed the mixing of different UML notations. Hosking and Grundy later adopted this approach in Marama-Sketch [32], which designers can use to sketch diagrams in a broad variety of notations. Numerous other UML-oriented sketch tools have been created, generally following similar strategies (for a survey, see [39]).

Other tools allow mixed elements to coexist in the same area as well, though interpretation is left to explicit manual choice by the user rather than automated recognition. In Kramer's system, the user can create sets of irregular shapes called "patches" [45]. These can be moved around, be made translucent, and, most importantly, the patch can be assigned an

interpretation where it can become a list, outline, table, or other element. Kramer's work later inspired the overall functionality of the sketch framework SATIN [38], which was later used to create the tools DENIM, SketchySPICE, and Designer's Outpost. Patches is probably closest to Calico of all these tools, with the primary difference being that user-defined shapes in Calico, scraps, stay away from interpretation, and these scraps make pieces of the design reusable across the grid canvases by using both the palette and by copying.

3.8.2 Systems that Support Sketching Activities

The second type of sketch system is aimed at supporting the general design activity. These systems help organize sketched artifacts during meetings, make them retrievable, and help manage and maintain the plethora of sketched design content that gets generated during design sessions.

The very first of these systems were created for the Liveboard system at PARC. Colab was the first system to work with this hardware, and acted as a meeting tool [83]. It had a simple, uncluttered interface with a mode selector and a visible set of saved sessions in the fashion of a filmstrip. The next system developed for the Liveboard was Tivoli [70], which used the filmstrip metaphor to manage sketches in a side panel. Tivoli also organized sketches into chunked items for easy moving, and had additional functionality for creating empty space. After Tivoli, Dolphin [86] introduced a client-server architecture for group collaboration and allowed users to create links between canvases. A significant difference between these systems and Calico lies in the grid, which provides a constant spatial orientation of sketches as compared to the paginated style, which leads to older sketches moving around.

Many other tools provide methods for managing the presence of multiple sketches within a fixed space. Post-BrainStorm [33] uses the border areas as areas where sketches auto-shrink, thereby minimizing space use by elements that are temporarily set aside. Flatland

[57] implemented a variant of this last idea by automatically grouping sketches into clusters, and shrinking and moving them out of the way when more space is needed. Range [42] extended this approach by using a person's physical proximity to the board as a trigger to create space. Compared to these systems, Calico's grid organizes the sketches into separate, but spatially fixed, spaces and allows the designers to easily move back and forth without explicitly worrying about space management.

Two final systems provide functionality somewhat different from Calico. Designer's Outpost [44] bridged the gap between physical Post-It notes and digital content, transitioning content of Post-It notes into digital artifacts that then could be organized in various ways. Finally, TEAM STORM [35] addressed the issue of collaboration by providing mechanisms for integrating tablet PC based input on a large display in front of a group of designers.

3.9 Conclusions

In this paper, we have presented a sketch-based software design tool that provides explicit support for the creative and exploratory behaviors of software designers when they work at the whiteboard. Using Calico, software designers can fluidly create, manipulate, and explore a esign problem and its possible solutions.

Our contributions include:

1. A unique software design sketching environment that leverages a carefully-tuned combination of scraps, a grid, a palette, and gesture-based input to enhance the experience of whiteboard software design;

2. A multi-pronged, detailed evaluation that demonstrates that Calico is of value, with participants in the experiment using Calico's advanced features and exhibiting the

kinds of design behaviors toward which Calico was designed; and

3. A rigorous analysis of the design conversations that demonstrates that Calico not only does not interfere with the design activity, but also may lead to more shared focus between participants and longer discussions of various aspects of the designs.

Overall, then, we believe Calico has beneficial impact on the design process as it unfolds at the whiteboard.

We view our work to date as a starting point, and have three directions of future work. First, we want to streamline the current functionality of Calico to address some of the observed problems in using Calico. This includes not only basic operations, such as allowing rotation and resizing of scraps, but also addressing the single user issue. We have already made progress in this direction with a revised interaction interface that we designed based on the feedback that we received [53]. Second, we wish to pursue the ability to refine scraps to support notation-specific features. This would further enable designers to move from very rough initial sketches to more polished diagrams (in the spirit of [76], although in a more gradual way that does not rely on automatic recognition). With converstion, additional, notation-specific features would become available, such as automatic layout, cardinality annotations, different kinds of arrows, and greater visual fidelity. Finally, we wish to automatically bring outside artifacts, particularly source code, but perhaps also lists of requirements or even simply domain-specific images, into Calico as scraps. Many software design discussions happen in the context of an existing project, and we envision being able to populate a canvas with sketchy representations of relevant artifacts that can then be manipulated in all of the ways that Calico offers.

3.10 Background

Lorem ipsum

Chapter 4

Notations

4.1 Background

Lorem ipsum

Chapter 5

Calico Version Two

5.1 Background

Lorem ipsum

Chapter 6

Evaluation

6.1 Background

Chapter 7

Discussion

7.1 Background

Lorem ipsum

Chapter 8

Related Work

8.1 Background

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do

Chapter 9

Conclusions

9.1 Background

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do

Chapter 10

Future Work

10.1 Background

Lorem ipsum

Bibliography

[1] A. Baker. *Bringing Software to Design: Theoretical and Empirical Studies of Software's Role as a Design Discipline*. PhD thesis, 2010.

[2] A. Baker and A. van der Hoek. Ideas, subjects, and cycles as lenses for understanding the software design process. *Design Studies*, 31(6):590–613, 2010.

[3] A. Baker, A. van der Hoek, H. Ossher, and M. Petre. Guest editors' introduction: Studying professional software design. *Software, IEEE*, 29(1):28–33, 2012.

[4] L. Ball, B. Onarheim, and B. Christensen. Design requirements, epistemic uncertainty and solution development strategies in software design. *Design Studies*, 31(6):567 – 589, 2010.

[5] B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Trans. Softw. Eng.*, 30:535–546, August 2004.

[6] F. P. Brooks. *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley Professional, 1st edition, 2010.

[7] F. P. Brooks, Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20:10–19, April 1987.

[8] B. Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design: Getting the Design Right and the Right Design*. Morgan Kaufmann, 2010.

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

[10] Q. Chen, J. Grundy, and J. Hosking. Sumlow: early design-stage sketching of uml diagrams on an e-whiteboard. *Software: Practice and Experience*, 38(9):961–994, 2008.

[11] Q. Chen, J. C. Grundy, and J. G. Hosking. Sumlow: Early design-stage sketching of uml diagrams on an e-whiteboard. *Software - Practice and Experience*, 38:961–994, July 2008.

[12] 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*, pages 557–566. ACM, 2007.

[13] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: how and why software developers use drawings. pages 557–566, 2007.

[14] 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*, pages 29–30. ACM, 2005.

[15] R. Chung, P. Mirica, and B. Plimmer. Inkkit: a generic design tool for the tablet pc. pages 29–30. ACM, 2005.

[16] N. Cross. *Designerly ways of knowing*. Birkhauser Boston, 2007.

[17] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, New York, 1991.

[18] P. P. Da Silva. User interface declarative models and development environments: A survey. In *Interactive Systems Design, Specification, and Verification*, pages 207–226. Springer, 2001.

[19] C. Damm, K. Hansen, M. Thomsen, and M. Tyrsted. Supporting several levels of restriction in the uml. pages 396–409. Springer-Verlag, 2000.

[20] C. H. Damm, K. M. Hansen, and M. Thomsen. Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 518–525. ACM, 2000.

[21] C. H. Damm, K. M. Hansen, and M. Thomsen. Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard. pages 518–525, 2000.

[22] C. H. Damm, K. M. Hansen, M. Thomsen, and M. Tyrsted. Supporting several levels of restriction in the uml. In *UML 2000The Unified Modeling Language*, pages 396–409. Springer, 2000.

[23] U. Dekel. Supporting distributed software design meetings: what can we learn from co-located meetings? In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM, 2005.

[24] U. Dekel and J. D. Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. *ACM SIGPLAN Notices*, 42(10):261–280, 2007.

[25] U. Dekel and J. D. Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. *SIGPLAN Not.*, 42(10):261–280, 2007. 1297047.

[26] E. Y.-L. Do. The right tool at the right time–investigation of freehand drawing as an interface to knowledge based design tools. 1998.

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

[28] E. S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, 1992.

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

[30] G. Goldschmidt. The dialectics of sketching. *Creativity Research Journal*, 4(2):123–143, 1991.

[31] P. Grisham, H. Iida, and D. Perry. Improving design intent research for software maintenance. 2009.

[32] J. Grundy and J. Hosking. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. *Proceedings of the 29th International Conference on Software Engineering*, pages 282–291, 2007. 1248861.

[33] F. Guimbretière. *Fluid Interaction for High Resolution Wall-Size Displays*. PhD thesis, 2002.

[34] F. Guimbretière, 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*, pages 21–30. ACM, 2001.

[35] J. Hailpern, E. Hinterbichler, C. Leppert, D. Cook, and B. Bailey. Team storm: demonstrating an interaction model for working with multiple ideas during creative group work. pages 193–202. ACM, 2007.

[36] T. Hammond and R. Davis. Ladder: A language to describe drawing, display, and editing in sketch recognition. In *ACM SIGGRAPH 2006 Courses*, page 27. ACM, 2006.

[37] D. Hendry. Sketching with conceptual metaphors to explain computational processes. In *Visual Languages and Human-Centric Computing*, pages 95–102, Washington, DC, 2006. IEEE Computer Society.

[38] J. Hong and J. Landay. Satin: a toolkit for informal ink-based applications. pages 63–72. ACM, 2000.

[39] G. Johnson, M. D. Gross, J. Hong, and E. Y.-L. Do. Computational support for sketching in design: A review. *Foundations and Trends in HumanComputer Interaction*, 2(1):1–93, 2008.

[40] J. Jones. *Design Methods*. John Wiley and Sons, Inc, New York, 1970.

[41] J. C. Jones. *Design methods*. John Wiley & Sons, 1992.

[42] W. Ju, B. Lee, and S. Klemmer. Range: exploring implicit interaction through electronic whiteboard design. pages 17–26. ACM, 2008.

[43] S. R. Klemmer, M. W. Newman, R. Farrell, M. Bilezikjian, and J. A. 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*, pages 1–10. ACM, 2001.

[44] S. R. Klemmer, M. W. Newman, R. Farrell, M. Bilezikjian, and J. A. 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, pages 1–10, New York, NY, USA, 2001. ACM.

[45] A. Kramer. Translucent patches—dissolving windows. *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 121–130, 1994. 192474.

[46] J. Landay. Silk: sketching interfaces like krazy. page 399. ACM, 1996.

[47] J. A. Landay and B. A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 43–50. ACM Press/Addison-Wesley Publishing Co., 1995.

[48] J. Larkin and H. Simon. Why a diagram is (sometimes) worth ten thousand words**. *Cognitive science*, 11(1):65–100, 1987.

[49] B. Lawson. *Design in mind*. Butterworth Architecture London, 1994.

[50] B. Lawson. *Design in mind*. Butterworth Architecture, 1994.

[51] N. Mangano, A. Baker, M. Dempsey, E. Navarro, and A. van der Hoek. Software design sketching with calico. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 23–32, New York, NY, USA, 2010. ACM.

[52] N. Mangano and A. van der Hoek. The design and evaluation of a tool to support software designers at the whiteboard. *Automated Software Engineering*, 19(4):381–421, 2012.

[53] N. Mangano and A. van der Hoek. A tool for distributed software design collaboration. In *Proceedings of the ACM 2012 conference on Computer supported cooperative work*, CSCW '12, New York, NY, USA, 2012. ACM.

[54] T. P. Moran. *Design Rationale: Concepts, Techniques, and Use*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1996.

[55] B. Myers, S. Park, Y. Nakano, G. Mueller, and A. Ko. How designers design and program interactive behaviors. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 177–184. IEEE, 2008.

[56] B. Myers, S. Y. Park, Y. Nakano, G. Mueller, and A. Ko. How designers design and program interactive behaviors. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 177–184. IEEE, 2008.

[57] E. Mynatt, T. Igarashi, W. Edwards, and A. LaMarca. Flatland: New dimensions in office whiteboards. page 353. ACM, 1999.

[58] E. D. Mynatt, T. Igarashi, W. K. Edwards, and A. LaMarca. Flatland: new dimensions in office whiteboards. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 346–353. ACM, 1999.

[59] A. Newell, H. Simon, and C.-M. U. P. P. D. O. C. SCIENCE. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ, 1972.

[60] M. W. Newman, J. Lin, J. I. Hong, and J. A. Landay. Denim: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18(3):259–324, 2003.

[61] M. W. Newman, J. Lin, J. I. Hong, and J. A. Landay. Denim: an informal web site design tool inspired by observations of practice. *Hum.-Comput. Interact.*, 18:259–324, September 2003.

[62] J. Nickerson, J. Corter, B. Tversky, D. Zahner, and Y. Rho. The spatial nature of thought: Understanding systems design through diagrams. *ICIS 2008 Proceedings*, page 216, 2008.

[63] G. Olson, J. Olson, M. Carter, and M. Storrosten. Small group design meetings: An analysis of collaboration. *Human-Computer Interaction*, 7(4):347–374, 1992.

[64] G. Olson, J. Olson, M. Storrsten, M. Carter, J. Herbsleb, and H. Rueter. The structure of activity during design meetings. *Design rationale: Concepts, techniques and use*, pages 217–240, 1996.

[65] J. Olson, G. Olson, M. Storrsten, and M. Carter. Groupwork close up: a comparison of the group design process with and without a simple group editor. *ACM Transactions on Information Systems (TOIS)*, 11(4):321–348, 1993.

[66] H. Ossher, R. Bellamy, I. Simmonds, D. Amid, A. Anaby-Tavor, M. Callery, M. Desmond, J. de Vries, A. Fisher, and S. Krasikov. Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. *ACM Sigplan Notices*, 45(10):848–864, 2010.

[67] H. Ossher, B. John, M. Desmond, and R. Bellamy. Are flexible modeling tools applicable to software design discussions. In *NSF Sponsored Workshop on Studying Professional Software Design (SPSD)*, volume 12.

[68] H. Ossher, B. John, M. Desmond, and R. Bellamy. Are flexible modeling tools applicable to software design discussions? 2010.

[69] H. Ossher, A. van der Hoek, M.-A. Storey, J. Grundy, R. Bellamy, and M. Petre. Workshop on flexible modeling tools (flexitools 2011). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1192–1193, New York, NY, USA, 2011. ACM.

[70] E. R. Pedersen, K. McCall, T. P. Moran, and F. G. Halasz. Tivoli: an electronic whiteboard for informal workgroup meetings. In *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, CHI '93, pages 391–398, New York, NY, USA, 1993. ACM.

[71] M. Petre. Team coordination through externalized mental imagery. *International Journal of Human-Computer Studies*, 61(2):205–218, 2004.

[72] 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*, pages 233–242. ACM, 2009.

[73] 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*, ESEC/FSE '09, pages 233–242, New York, NY, USA, 2009. ACM.

[74] M. Petre, A. van der Hoek, and A. Baker. Studying professional software designers 2010: Introduction to the special issue. *Design Studies*, 2010.

[75] B. Plimmer and M. Apperley. Computer-aided sketching to capture preliminary design. *Proceedings of the Third Australasian Conference (2002) on User interfaces*, pages 9–12, 2002.

[76] B. Plimmer and J. Grundy. Beautifying sketching-based design tool content: issues and experiences. In *Proceedings of the Sixth Australasian conference on User interface - Volume 40*, AUIC '05, pages 31–38, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[77] L. Putnam. Procedural messages and small group work climates: A lag sequential analysis. *Communication Yearbook 5*, page 331, 1981.

[78] J. E. Robbins and D. F. Redmiles. Cognitive support, uml adherence, and xmi interchange in argo/uml. *Information and Software Technology*, 42(2):79–89, 2000.

[79] D. Schön. *The reflective practitioner: How professionals think in action.* Basic books, 1983.

[80] M. Schutze, P. Sachse, and A. Rmer. Support value of sketching in the design process. *Research in Engineering Design*, 14(2):89–97, 2003.

[81] 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)*, 8(4):333–352, 1999.

[82] F. M. Shipman III and R. J. McCall. Incremental formalization with the hyper-object substrate. *ACM Transactions on Information Systems (TOIS)*, 17(2):199–227, 1999.

[83] M. Stefik. Wysiwis revised: early experiences with multiuser interfaces. *ACM Transactions in Information Systems*, 5(2):147–167, 1987.

[84] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, 1987.

[85] N. A. Streitz, J. Geissler, J. M. Haake, and J. Hol. Dolphin: integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 345–358, New York, NY, USA, 1994. ACM.

[86] N. A. Streitz, J. Geißler, J. M. Haake, and J. Hol. Dolphin: integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 345–358, New York, NY, USA, 1994. ACM.

[87] M. Suwa, J. Gero, and T. Purcell. Unexpected discoveries and s-invention of design requirements: important vehicles for a design process. *Design Studies*, 21(6):539–567, 2000.

[88] B. Tversky. What do sketches say about thinking. In *2002 AAAI Spring Symposium, Sketch Understanding Workshop, Stanford University, AAAI Technical Report SS-02-08*, pages 148–151, 2002.

[89] B. Tversky. What do sketches say about thinking. 2002.

[90] D. K. Van Duyne, J. A. Landay, and J. I. Hong. *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience.* Addison-Wesley Professional, 2003.

[91] R. A. Virzi, J. L. Sokolov, and D. Karis. Usability problem identification using both low- and high-fidelity prototypes, 1996.

[92] R. A. Virzi, J. L. Sokolov, and D. Karis. Usability problem identification using both low-and high-fidelity prototypes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 236–243. ACM, 1996.

[93] Y. Y. Wong. Rough and ready prototypes: Lessons from graphic design. In *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems*, pages 83–84. ACM, 1992.

[94] Y. Y. Wong. Rough and ready prototypes: lessons from graphic design, 1992.

[95] Y. Yamamoto and K. Nakakoji. Interaction design of tools for fostering creativity in the early stages of information design. *Int. J. Hum.-Comput. Stud.*, 63:513–535, October 2005.

[96] 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*, pages 995–1004. ACM, 2009.

[97] 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*, 49(6):637–653, 2007.

[98] C. Zannier and F. Maurer. Comparing decision making in agile and non-agile software organizations. In *Agile Processes in Software Engineering and Extreme Programming*, pages 1–8. Springer, 2007.

[99] C. Zannier and F. Maurer. Comparing decision making in agile and non-agile software organizations. In *Agile Processes in Software Engineering and Extreme Programming*. Springer Berlin / Heidelberg, 2007.

Appendix A

Appendix Title

Supplementary material goes here. See for instance Figure A.1.

A.1 Lorem Ipsum

dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

"I am glad I was up so late,
for that's the reason I was up so early."
William Shakespeare (1564-1616), British dramatist, poet.
Cloten, in Cymbeline, act 2, sc. 3, l. 33-4.

Figure A.1: A deep quote.