Calico:
A Prototype Sketching Tool for Modeling in Early Design

Nicolas Mangano, Alex Baker, and André van der Hoek
University of California, Irvine
Department of Informatics
Irvine, CA 92697-3440 U.S.A.
+1 949-824-6326
{nmangano,abaker,andre}@ics.uci.edu

ABSTRACT
Design is an inherently creative process, particularly so during the early stages of design when a solution is just beginning to form. At this time, the more formal models and modeling languages to which we are so accustomed in software engineering serve a limited if non-existent role. But this does not mean that modeling is not relevant during early design; on the contrary, it is critical. It is just that a different form of modeling and overall design process takes place. In this paper, we present early results from our foray into exploring how designers can be supported in the early, highly creative stages of software design. We particularly build upon the existing body of work in creativity and general design, and apply key lessons found there to the construction of Calico, a prototype sketching tool for modeling in early design.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *computer-aided software engineering*

General Terms
Design, Human Factors

Keywords
Creativity, modeling, sketching, early design, software

1. INTRODUCTION
Models and modeling play a role in every design discipline, and so it is in software engineering: for any development project of reasonable scale, we must model what we are about to build. But, as has been reported in many a discipline outside of software engineering, the final model that describes what is about to be built is not the only type of model used in design. Three types of models are usually distinguished: (1) exploratory models that help the designer think and work out the solution in their mind (these are called "thinking sketches" in [1].), (2) models that communicate the final solution that the designer has developed ("called prescriptive sketches"), and (3) models that facilitate rapid, impromptu exchanges between collaborating designers (called "talking sketches").

It can be argued that the software engineering modeling languages that have been developed to date largely aim to support prescriptive sketching, that is, capturing designs that are well thought out and sufficiently stable to go through the effort to document them in detail. It would be difficult for a designer to justify the effort of, say, fully modeling a rough idea in Statecharts [2] or even UML [3], if a next refinement is already popping up in their heads – as typically the case in early, creative design [4]. The flow [5]of idea generation would be disturbed, the creative process stifled, and an opportunity to do a broad search for a solution quickly limited to making one alternative work as best as one can.

Yet designers in the early stages of problem and solution framing do use models, extensively [6]. These models are of a different nature than those which we are used to support with our modeling tools to date. They generally are much more exploratory, incomplete, informal, and of mixed abstraction levels. It is no surprise, then, to find designers involved in early software design to use a whiteboard or just pen and paper to explore their thoughts – both these media are extremely fluid and place no restrictions on their use in terms of the kinds of diagrams, lists, comments, sketches, and other markings generated by the designer.

This paper presents early results from our research into the question as to whether it may be possible to provide tool support for the early design process that provides a similar kind of fluidity to the whiteboard or pen and paper, but adds the kinds of functionality through which the tool enriches and makes more effective the design experience. Specifically, we present a prototype of Calico, a new software design sketching tool that we have explicitly designed for use on an electronic whiteboard (or tablet PC). Calico distinguishes itself in supporting lightweight informal sketching, upon which it provides facilities for rapid exploration and organization of different design ideas – as geared towards software design.

2. CHARACTERIZING EARLY DESIGN
Early design, and the role of thinking and talking sketches in early design, can best be understood with a look at the general literature in both creativity and design, as it spans across a vari-

ICSE'08, May 10–18, 2008, Leipzig, Germany.

ety of disciplines. Whereas software engineering certainly has recognized the importance of design and has built a fair amount of support for the design phase, most of the support has focused on capturing what amount to relatively complete if not final designs in a variety of design languages, analyzing such designs with a set of algorithms that verify the presence or absence of certain desired or undesired properties, and supporting a user in documenting a design in powerful tools. The topic of early design, or how software designers engage in the creative exercise of sketching an initial approach to a particular software design problem, however, has received little attention.

In other fields, however, understanding early design has emerged as a key topic and, more recently, the study of design in general as it spans across disciplines has received a lot of attention. Through experiments, observational studies, theory development, and other approaches, slowly but surely an understanding is arising of what some call "designerly ways of knowing" [7], that is, the unique aspects of design as it differs from scientific endeavor or artistic expression. Cross sums up the collective evidence gained thus far by characterizing the creative act of design as: (1) rhetorical, (2) exploratory, (3) emergent, (4) opportunistic, (5) abductive, (6) ambiguous, and (7) risky [7].

In and of itself, this characterization is useful in understanding the *realm* of design, but it does not yet state what designers *do* when they actually design. A broad variety of sources, however, is beginning to provide a coherent picture of what happens when designers design. Here, we briefly summarize the main observations stemming from this body of literature, particularly as the observations pertain to the early phases of design, when the design solution(s) is (are) still very much in flux.

Designers frequently shift focus

Jones was among the first to observe these shifts and labeled them as three distinct phases in the overall design "process": convergence, transformation, and divergence [8]. When working on an idea, a designer will typically rapidly generate a number of alternative high-level approaches, some of which are easily dismissed, others of which are slowly but surely refined and combined until eventually one design emerges that exhibits characteristics from a number of the alternatives that were originally created. Good designers will know how to leverage this process, will know when to diverge more when their current solutions do not seem quite right, and generally will even perform much of this process in their head rather than explicitly worked out through external models.

Goel echoed these ideas several decades later in his theories about the importance of sketching [4]. 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 a lateral transition, a designer switches from one idea to another, attempting different strategies to solving the same problem. Recent studies suggest that experienced designers have a roughly equal balance between lateral and vertical transitions in the early design phases, although more vertical transitions begin to happen as the models mature [4].

The importance of vertical transitions is not to be underestimated. While top-down and bottom-up methods of design have had their share of proponents, it has been observed time and again that the expert designer fluidly moves back and forth over multiple levels of abstraction, sorting out details that may have impact on certain high level choices, as readily as they lay out the overarching structure of the solution [1, 7].

Designers use quick, low detail models for exploration

Studies conducted of designers in action show that they prefer to use tools that allow liberal expression over tools that are rigid in the kinds of models they allow [9], unless they have largely finished the process and are ready to create a prescriptive model for the builders to use or unless they truly must analyze certain properties of their envisioned solution. But early on, during exploratory design or when talking through potential solutions with some colleague, the models that they employ are rapidly sketched and at best have some detail, but they certainly are not complete.

At the heart of this tendency is the reason why the designers create these models in the first place, which is to help them personally understand an idea by visualizing what is appearing "in their mind's eye" [1]. By visualizing the ideas through an explicit and external model, it becomes possible to closely examine them and let the model "talk back" at the design (Schön's concept of "conversation with the situation" [10]). Because thoughts and ideas in the mind are so fleeting, external representations that can be visible and examined over an extended period of time, that can be parked and revisited at a later time in the exact same condition, and that can be shared with others for joint exploration, are a critical part of design.

Therefore, the tools that designers use must enable them to work quickly enough to keep pace with their internal thought process [1, 6], otherwise the tools would obstruct the flow of ideas. For this very reason, too, detail is often omitted. As long as the sketch is sufficiently complete that it will remind the mind's eye of missing detail, it suffices for the design process. Not surprisingly, expert designers need very little information for this process to happen.

Designers use models that are ambiguous

Whether consciously or subconsciously so, designers' early models tend to be ambiguous, which has been shown to have beneficial effects on the quality of the eventual design that is delivered [7]. While a general lack of detail in the sketches that are used contributes, it is a conscious strategy of many expert designers to leave as much as possible of their early designs ambiguous, purposely not resolving ambiguities early on to leave room for (often radical) changes later on. Locking in too many choices early, even if those are deemed non-consequential with respect to the overall strategy, tends to lock in this overarching strategy more so than intended, partially because of hidden dependencies among parts of the solution that emerge later and partially because of a subconscious effect in the mind of the designer who has more and more vested in the solution at hand. Overall, then, the presence of ambiguity early on helps in broadening the spectrum of solutions that are considered and tends to deliver a design of higher quality [4].

In addition to broadening the search space, ambiguity supports a reinterpretation of the problem, which is helpful to reframe the problem differently. Reframing a problem subconsciously cues relevant knowledge on the issue at hand, causing the designer to connect current ideas to past experiences and otherwise amassed knowledge that may not have been evident at first [11]. It has

often been observed that design is as much about framing a solution to a problem as it is about framing the problem; successful designers realize that a key to the design process is the careful examination of the problem space and adjusting both the problem statement and its solution under development hand-in-hand [7]. Particularly because most design problems tend to address wicked [12] or ill-formed problems [13], carefully co-evolving one's understanding of both is critical to a successful design process.

Designers use a broad variety of languages in their models

We already highlighted that most models are quickly produced, low in detail, and often ambiguous. A critical element in the creation of such models is that designers rely on a broad range of languages in which they express their ideas. Furthermore, these languages may not be formally defined. Many are ad-hoc and created impromptu and nearly seamlessly by the designer as they express their ideas [9]. As a result, it is not uncommon to see design sketches that involve some high-level architecture diagrams that are not in some ADL but rather pictorial descriptions of entities, that at the same time include some low-level pseudo code, that include a few lists and short notes that explain progress and important decisions, and that may even include some primitive GUI drawings.

Early Software Design

While many of the studies upon which we have drawn took place outside of software, slowly but surely evidence is emerging that the lessons learned also apply in software and that the same kinds of behaviors emerge when software designers are involved in the early design of a new system. A study on observing how people use models in design brainstorming, conducted at OOPSLA, revealed that designers may begin with formal methods, but often will deviate greatly and create their own notation to reflect their ideas [9]. A recent paper focusing on whiteboards use by developers focused on the question of how and why programmers used the whiteboard [14]. It reported such findings as "The adherence to modeling languages, such as UML, or standards of any sort was very low." and "Our results indicate that in most cases, an informal notation was used to support face-to-face communication and that current tools were not capable of supporting this need because they did not help developers externalize their mental models of code."

In some of our own (as of yet unpublished) work, we are studying photographs of whiteboards taken during and after design exercises. It is remarkable how much these photographs indicate the similarities with other fields: the models are typically incomplete, have little detail, do not use standard language and in fact generally use language that emerge spontaneously, and show evidence of shifting both vertically as well as horizontally.

In sum, we believe there is a need to support software designers in the early phases of design, when their initial ideas are formed and they rapidly cycle through many ideas in search of a set that leads them to a suitable solution. In the rest of this paper, we introduce Calico, a prototype of a sketch-based software design tool that we are designing explicitly with early software design needs in mind.

3. CALICO

We have begun addressing these lessons and considerations in a new software design tool that we are currently in the process of developing, called Calico. Calico is a prototype, with the present version operational, some personal experience gained in its use, and additional features we have already identified. We designed Calico in an iterative manner by developing a prototype, trying it out ourselves, verifying whether or not certain desired properties were present, and refining and rearchitecting Calico in subsequent iterations to address shortcomings and extend its functionality. The present version is the third incarnation, and a first in which we are satisfied with the basic sketching features as a basis upon which to build further in our research. Here, we discuss the features that are currently implemented, relate them to the observations of the previous sections, and use an illustrative example to highlight the features.

The example concerns the hypothetical design of a simple Pitfall game. The objective of the game of Pitfall is to navigate a character across a map without falling victim to a trap. The game progresses over several levels, with each level presenting hurdles and an overall puzzle of how to navigate the map of incremental difficulty. In this session, the designer is laying out the basic elements of the world and the various kinds of pitfalls that will form the obstacles that a player has to circumvent. Simultaneously, the designer is laying the basic groundwork for what may be an architecture for how the game will actually be internally structured.

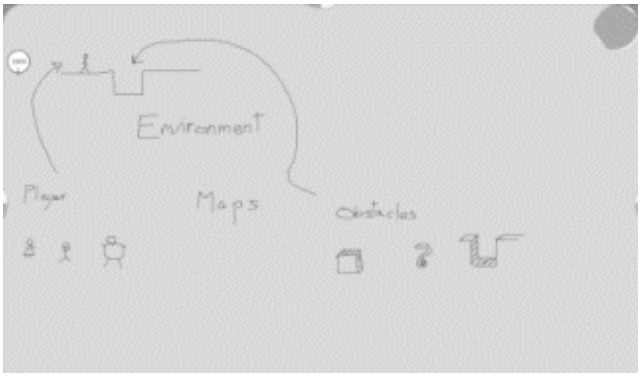

Figure 1. Basic Sketching.

Figure 1 shows the basic operation of Calico: designers sketch on a large open space on electronic whiteboard (or tablet PC). In this space, they can sketch what they want, whether it involves figures, drawings, lists, annotations, and so on. Everything they do is freehand, as this is the "fastest" mode and does not impede the early design process. As such, the basic mode of Calico supports the designer in exactly the same way they can operate on the whiteboard. Of course, Calico being a computer tool, already one advance makes a critical difference: the ability to undo and redo operations, which the traditional whiteboard does not support.

We note the grey circle on the top right of the canvas, which is a mini view of the entire sketch, and the small white circle on the left, which enables a designer to zoom in and out. Together with the ability to pan, this effectively presents the designer with an infinite canvas on which they can work (see also below).

In the figure, it can be seen that the designer started with a really basic sketch of a walking game character, which has to deal with a literal pitfall ahead of them. The designer has labeled this the

"Environment". The player character is elaborated on with three quick sketches of figures, indicating that the game probably will allow players to choose a character, or that the character may be changing its appearance depending on game play. Similarly, the designer has drawn three different pitfalls, and also mentioned the presence of maps. All this is done simply by drawing and writing as one would draw and write on a regular whiteboard.

But clearly, not being able to do anything with the scribbles and diagrams that the designer creates is limiting. The key advance underneath Calico is the ability to create scraps from any part of the canvas. These scraps are irregular in shape, indicated simply by the designer holding right click and circumscribing an area. This "lifts" a diagram or group of sketches from the background, giving the raised region an informal importance.

Figure 2 shows the same sketch as in Figure 1, but this time with scraps that group elements important for the designer. It is clear that this is in some sense a small but critical addition to the sketching experience that immediately changes the nature of how one examines and interacts with the design. As a result, one interprets the diagram much more easily, particularly the groupings of, and relationships among, the elements.

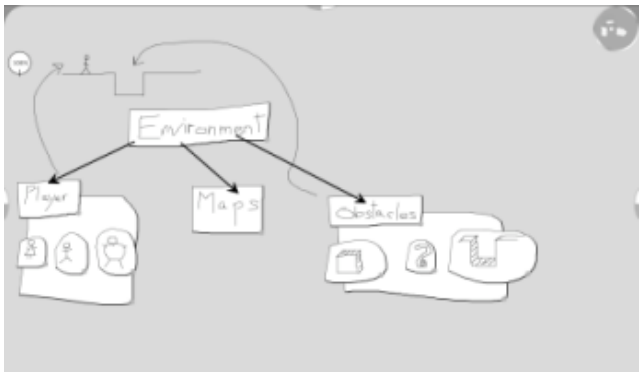

Figure 2. Pitfall Design with Scraps.

But beyond this visual cue, scraps offer a number of other affordances. First, a designer can use any shape for a scrap, providing the designer with a mechanism to assign informal meaning to differently shaped scraps. This gives rise to a key point we made earlier: designers often use temporary, informal languages that simply arise of conventions they use in drawing. Being able to have differently shaped scraps enables this.

Scraps also are at the heart of how Calico enables rapid exploration of a design space. Scraps can be moved around on the canvas, easily duplicated or erased, grouped, and related with arrows. In Figure 2, for instance, each obstacle is a separate scrap and they are all grouped on top of an obstacles scrap (the label of which is also a scrap). The environment is linked to player, maps, and obstacles through arrows, and the player scrap has a number of other scraps on top of it.

Scraps follow the metaphor of an actual scrap of paper. A designer can continue marking them, drag them around, as well as group them. The grouping mechanism follows a strict physical metaphor. The bottom most scrap will drag every scrap on top of it when it is moved, and the top most scrap moves freely without dragging the scraps below it. This is consistent with how a stack of papers would roughly behave in real life. This metaphor thus provides a familiar interaction mechanism and thereby a clearly understood behavior. Note that new scraps are created at the bottom most layer, so a group of scraps can be quickly grouped by creating a scrap that encapsulates all desired scraps.

Scraps, once created, do not "freeze" their content – designers can continue drawing on the scraps just as they would draw on the background. Moreover, the content of a scrap can be dropped back onto the background, or even onto another scrap that lies behind it. This enables designers to flexibly combine the contents of multiple scraps. Translucency of the scraps when moving them helps in aligning the contents.

Drawing a line from one scrap to another creates an arrow. The arrow is tied to both scraps, so it will move along with its corresponding scraps. These arrows can be thickened and thinned to signify stronger and weaker relationships between scraps, giving the designer the option to weigh arrows and provide visual clues as to which ones may be more important, or which scraps might be more connected, or which arrows represent the greatest data flow, and so on. A designer could choose to create scraps in the shape of arrows as their own notation, but Calico's arrows are important since they provide support for a very common activity in a more semantic way without sacrificing the usability of other basic Calico functions.

A final piece of the Calico functionality is its grid. We described Calico's "infinite" and zoomable canvas previously, but an infinite canvas can require a serious amount of zooming, searching, and other navigation that hinders the design exercise. In an effort to lighten the mental load, Calico actually provides the designer with a grid of canvases. Shown in Figure 3, this grid view provides an overview of all the canvases and allows the designer to navigate to any cell on this grid with the click of a button. It is, therefore, possible for them to explore different ideas or different aspects of the problem on different canvases.

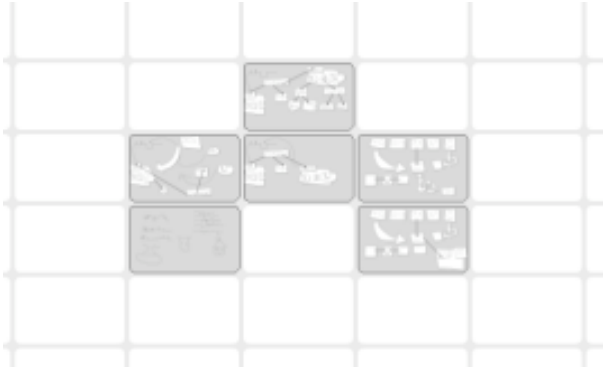

Figure 3. A cropped screenshot of Calico's grid view.

By virtue of there being a grid of canvases, the grid provides an organizing capability that is more difficult to achieve in an infinite canvas. Designers have more of a sense of "place" where they are in the overall space and where they "have left" some of their ideas to later return to. Moreover, they can move individual canvases around the grid to organize and reorganize their ideas at a conceptually high level.

A key functionality complementary to the grid is the ability for a designer to, with the click of a button, copy an entire canvas to a grid cell below, above, to the left, or to the right of the current

canvas. This can be done without switching to the grid view, as the primary drawing interface of Calico has small "handles" at each side of the canvas with which the designer can move to an adjacent grid cell or copy the contents of the current canvas to an adjacent grid cell. The idea here is to support the designer in freely exploring alternatives and fluidly keeping a history of the alternatives that they explored. In our own design sessions with Calico, we would, before each major change, copy and switch to a new canvas where we made the change, sometimes returning to the previous canvas and exploring a different alternative from there. The ideas become increasing advanced and refined as they snake out from the original iteration, creating an often distinct path at the grid level view where we can retroactively see the historical buildup of our design ideas. At times we would explicitly use the grid to organize our ideas, but we primarily operated within the canvas but still relying on the grid as a structured memory.

We now return to some of the characteristics of early design that we introduced in Section 2 (shifting focus, quick and low-detail models, ambiguity, and broad variety of languages) and compare these general needs with the functionality provided by Calico.

We first observe that everything in Calico is done via sketching and via gestures that the user sketches. That means that they stay in their primary mode of interaction, without the need to cognitively switch to menus or to reach across the canvas for a certain feature. Everything is "ready-at-hand" at the tip of the pen. This means that the flow of design is uninterrupted, or at least minimally interrupted, supporting the stream-of-consciousness kinds of experiences that can take place in early design.

Second, we observe that the features of Calico thus far particularly help with the rapid shifting of focus – particularly through the grid and use of scraps – and the creation of quick, low-detail models – particularly through being a sketch interface that does not place any requirements on what and how the designer draws. This, combined with the fact that the grid captures a history of design and supports the easy reconciliation of ideas, encourages divergent thought and exploration of alternatives.

With respect to ambiguity, there is little explicit support that one can provide, though we note that the basic choice of supporting sketching helps. It is more the absence of features (i.e., requiring only particular shapes, requiring every box to have a clear label, requiring arrows to have precise semantics) that is the key determinant in enabling ambiguity to enter the models. The developers can work freely without being prompted to make any design decisions.

The same holds true for the use of multiple languages: sketching is the key determinant, and lets designers both use symbols that are standard accepted as well as use emergent languages of their own. Nonetheless, we believe there is more support that can be provided here than what we have incorporated into Calico thus far. It would be interesting, for instance, to explore how such use of an emerging language can be supported, perhaps through automatic construction of palettes of symbols that continuously reappear in a design exercise, or via auto-completion of pairs of symbols that appear.

4. RELATED WORK

In addition to the background materials we discussed in Section 2, there are several items of related work in which others have explored the use of sketching. Within the software engineering community, the closest related work is that of Grundy and Hosking, who have implemented a generic sketch-based front-end to the drawing of arbitrary diagrams [2]. Their tool allows designers to flexibly sketch, and the sketched elements are (eagerly or lazily) mapped onto known kinds of elements. A sketch interpreter, that is suitable trained with practice sketches, makes this mapping. Grundy and Hosking agree with us on the critical need of maintaining the sketched nature of the drawn elements, and allow a designer to easily move back and forth between sketch and interpreted diagram. Their focus, however, remains with the interpretation of the sketch into eventual diagrams, whereas our focus is more flexibly supporting the designer in the exploration of ideas with such features as scraps, grouping scraps, the nature of the gestures, and the grid.

In the field of computer-supported cooperate work (CSCW), the potential of sketching has been recognized for quite a bit longer than in software engineering. When interactive whiteboards first emerged, the CSCW community rapidly explored their potential. A number of tools resulted (e.g., [15], SILK [16], PostBrainstorm [17], InkKit [18]). While some of this work is similar in nature to that of Grundy and Hosking in focusing on translation of sketch to diagram (e.g., Web page design in DENIM, GUI design in SILK, and UML design in Knight and SUMLOW [19, 20]), a number of different projects also explored the creative process afforded by electronic whiteboards. A number of features similar to those provided by Calico have been reported, for instance the use of post-it notes in PostBrainStorm [17], which is similar to our use of scraps (though it requires notes to be created first before they can be written on), the ability to "scrape" drawings from a background into a scrap [21], or the desire to maintain informal drawings [18]. These tools all provide steps in the right direction for creative design exploration, though the particular set of features provided by Calico is unique and its focus on software as the primary domain of interest enables it to handle certain parts of the drawing process differently.

5. CONCLUSION

The creative exploration of a design problem in the early phases of design can involve as much modeling as when a solution has been worked out and is being documented in full. The kind of modeling performed, however, differs and brings with it some unique requirements for supporting tools. In this paper, we have introduced Calico, a new prototype sketching tool for modeling in early software design, through which we are exploring how early design can benefit from the availability of tool support.

The design of Calico is strongly influenced by literature describing general design and creativity as it takes places in other design disciplines. We strongly believe software design can benefit from the lessons found in this literature, and will continue in our exploits of the similarities in needs for inspiration.

Our immediate technical goal is to explore how we can provide more software specific support. To date, we have focused our attention mostly on developing a "natural" sketch environment in which ideas can be flexibly explored. But we do believe it is possible to augment this environment with support for software specifically, some in terms of modeling facilities and some in terms of the kinds of feedback the environment provides to the designer. Additionally, we are currently planning for trial use of Calico, first in a different research group at our university but shortly thereafter in an industrial setting – both of which experi-

ences we hope will lead us to refinements and further explorations in how Calico can support the early design process.

6. ACKNOWLEDGEMENTS

This effort is partially funded by the National Science Foundation under grant numbers DUE-0536203 and IIS-0534775.

7. REFERENCES

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

[2] Grundy, J. and Hosking, J. 2007. Supporting Generic Sketching-Based Input of Diagrams in a Domain-Specific Visual Language Meta-Tool. In Proceedings of the 29th International Conference on Software Engineering (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 282-291.

[3] Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 2003.

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

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

[6] Pye, D. The Nature and Aesthetics of Design. Cambium Press, 1995.

[7] Cross, N. Designerly Ways of Knowing. Springer, 2006.

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

[9] Dekel, U. and Herbsleb, J. D. 2007. Notation and Representation in Collaborative Object-Oriented Design: An Observational Study. In Proceedings of the 22nd Annual ACM Special Interest Group on Programming Languages Conference on Object Oriented Programming Systems and Applications (Montreal, Quebec, Canada, October 21 - 25, 2007). ACM, New York, NY, 261-280.

[10] Schön, D.A. The Reflective Practitioner. Basic Books, 1982.

[11] Purcell, A.T. and Gero, J.S. Drawings and the Design Process: A Review of Protocol Studies in Design and Other Disciplines and Related Research in Cognitive Psychology. Design Studies (1998), 19 (4). 389-430.

[12] Rittel, H.W.J. and Webber, M.M. Dilemmas in a general theory of planning. Policy Sciences (1973), 4 (2). 155-169.

[13] Simon, H.A. and Fernandes, R. A Study of How Individuals Solve Complex and Ill-Structured Problems. Policy Sciences (1999), 32 (3). 225-245.

[14] Cherubini, M., Venolia, G., DeLine, R., and Ko, A. J. 2007. Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In Proceedings of the Special Interest Group on Computer Human Interaction Conference on Human Factors in Computing Systems (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM, New York, NY, 557-566.

[15] Newman, M., Lin, J., Hong, J. and Landay, J. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. Human-Computer Interaction (2003), 18 (3). 259-324.

[16] Landay, J. A. and Myers, B. A. 2001. Sketching Interfaces: Toward More Human Interface Design. Computer 34, 3 (March 2001), 56-64.

[17] Guimbretière, F. Fluid Interaction for High Resolution Wall-Size Displays Department of Computer Science, Stanford University, 2002, 157.

[18] Plimmer, B. and Apperley, M. 2004. Interacting with Sketched Interface Designs: An Evaluation Study. In Special Interest Group on Computer Human Interaction 2004 Extended Abstracts on Human Factors in Computing Systems (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 1337-1340.

[19] Damm, C. H., Hansen, K. M., and Thomsen, M. 2000. Tool Support for Cooperative Object-Oriented Design: Gesture Based Modelling on an Electronic Whiteboard. In Proceedings of the Special Interest Group on Computer Human Interaction Conference on Human Factors in Computing Systems (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 518-525.

[20] Chen, Q., Grundy, J., and Hosking, J. 2003. An E-Whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams. In Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments (October 28 - 31, 2003). HCC. IEEE Computer Society, Washington, DC, 219-226.

[21] Kramer, A. 1994. Translucent Patches—Dissolving Windows. In Proceedings of the 7th Annual ACM Symposium on User interface Software and Technology (Marina del Rey, California, United States, November 02 - 04, 1994). UIST '94. ACM, New York, NY, 121-130.