DesignMinders: Preserving and Sharing Informal Software Design Knowledge

Gerald Bortis and André van der Hoek

University of California, Irvine
Department of Informatics
Irvine, CA 92697-3440
{gbortis, andre}@ics.uci.edu

Abstract. Software design is a knowledge-intensive activity. Knowledge is both generated and used when making crucial design decisions, and it is important to capture and share the knowledge in order to take advantage of the valuable insights gained. A particularly challenging setting for knowledge reuse is during informal software design sessions when developers are gathered at the whiteboard working to solve a design problem. It is during this activity that many existing approaches fail to address the needs of the developer to remain "in the moment" while engaged in discussion and sketching. In this paper we introduce DesignMinders, a tool currently being developed to address these issues by augmenting an electronic whiteboard with the ability to capture, refine and explore design knowledge in the form of notecards. This paper describes the problem of knowledge reuse during informal software design, and our decisions and progress with respect to DesignMinders so far.

Keywords: Software design, knowledge reuse, whiteboards, notecards

1. Introduction

Software design is a knowledge-intensive activity that creates a system's software architecture, or principal design decisions [1, 2]. Because decisions are made throughout the development process, design takes place throughout the development life cycle [2]. One aspect of software design that has received little attention is informal software design, or the activity that consists of software developers gathered at the whiteboard to solve a specific design problem. The evidence of such design sessions can be found on the whiteboards of most software development companies, which are typically full of sketches, diagrams, and lists [3-5].

During a design session, developers bring to bear knowledge from past projects or personal experiences that influence the decisions that are made [6]. The knowledge that is involved in such sessions ranges from general knowledge of software development practices, to knowledge about the domain that the system will reside in, and finally knowledge that is specific to the organization or project. This knowledge can be explicit, like facts about a specific development process or the system requirements, or tacit, based on the developer's own experiences from working on other projects [7]. As this knowledge is applied to solving a problem, knowledge is also generated in the form of informal sketches, diagrams and lists that are created on the whiteboard. This output, the decisions underlying it, and any other information that was generated during the design session, provide valuable insights into how the system was designed; insights that can be applied during subsequent design sessions.

Unfortunately, much of this knowledge often "vaporizes" at the end of a design session, except for the occasional photo of the whiteboard and the unreliable memory of the participants [8]. Existing approaches to capturing design knowledge either address this issue during the more formal specification or implementation phases of the development process when the informal software design knowledge has already been lost, or they fail to address the fundamental tensions that exist in capturing, representing, and applying design knowledge. For example, knowledge management systems that allow developers to populate and search a knowledge base containing formal design documents fail to address the need for concise and relevant representations of knowledge during informal software design sessions when the design problem is still being understood. On the other hand, design rationale techniques attempt to capture the decisions that were made during a design session using an argumentation schema that is incompatible with the activities and processes that developers engage in while at the whiteboard. These approaches are difficult to incorporate into the setting of informal software design when developers are engaged both with the whiteboard and with each other in discussion, and are unlikely to interrupt an opportunistic exploration of the design problem to populate the knowledge base with recently acquired knowledge about the design problem, or to formalize their partial solutions into an argumentation schema [9].

In order to be useful during informal software design, an approach must provide a lightweight means of capturing knowledge and making it available when needed, and do so in a way that does not interfere with the developer's design activities.

2. DesignMinders

DesignMinders (portmanteau of *design reminders*) is a software design knowledge reuse tool currently being developed to address these issues. The goal of DesignMinders is to support developers in: capturing design knowledge in an easy way, organizing the collected knowledge as a set of notecards, and making this collection available for exploration and search during design sessions. This approach provides developers with a lightweight knowledge reuse tool that addresses the various needs that arise during design sessions. For example, developers exhibit behaviors like solution conjecture, where one creates a partial solution based on existing knowledge of the problem, which is then used to generate some initial ideas for the form of the design concept, which is then transferred back to understanding the problem [10]. Developers also exhibit opportunism when they deviate from a structured plan or methodical process into the opportunistic pursuit of issues or partial solutions that catch their eye [10]. In order to support the capture and reuse of knowledge in this setting, an approach must address the tensions that exist between initially capturing the knowledge, the required effort on the part of the developer beyond their normal work to refine the knowledge into a reusable representation, and the obtrusiveness of presenting the knowledge to the developer at the appropriate time. DesignMinders is designed to address these tensions, and allows developers to capture, refine, and recall relevant design knowledge “in the moment”, while engaged in informal software design.

DesignMinders will extend the electronic whiteboard and sketching tool Calico [11] that provides developers with distinct advantages over traditional whiteboards, the most important of which is the ability to directly manipulate sketches that are made on the whiteboard by selecting and moving them around. This allows for diagrams and text written on the whiteboard to become elements of design knowledge that can be built upon.

We envision DesignMinders being used alongside Calico by developers during software design sessions while engaged in various informal design activities. During such sessions, developers are involved in generating ideas and converging to a possible solution. Because this is an intensive process, there is a need to remain in this mode without unnecessary interruptions. DesignMinders addresses these needs by providing an interface that allows developers to quickly capture useful design knowledge in an *ad hoc* way within the context of a design session. The ability to explore and search collected design knowledge allows developers to easily recall and incorporate knowledge captured from previous design sessions. For example, a developer might recall a specific diagram, keyword, or date range in which a particularly insightful piece of design knowledge was created. Or, the developer may encounter a design problem that he or she does not have much experience with, and may want to browse a subset of the collection of knowledge for ideas or inspiration about how to approach the problem. By easily recalling and incorporating knowledge from previous design sessions into a current design, or by simply being aware of the commonly encountered issues through reminders, it is possible that the design session will result in a better design. This approach supports the opportunistic methodology employed by designers and presents knowledge in a form that can be used as input for developing a partial solution. It also reduces interruptions since it is driven by the developer’s demand for capturing and retrieving knowledge, and allows for the presented knowledge to be based on the context of the current design. With DesignMinders, the valuable insights that are gained from previous design sessions can be readily brought to bear on design decisions that are made in future sessions.

2.1 Notecards

The most basic element of the DesignMinders interface is the *notecard*. A notecard is an element of design knowledge that consists of a *name*, the elaborating *details*, descriptive *tags*, and an optional *visual representation*, such as a diagram or sketch. Similar to a design pattern, the notecard’s name should be a brief description of the situation in which the knowledge can be applied, like “Multiple Monitors”. Brief, descriptive names allow for notecards to be easily identified when browsing a large collection. The body of the notecard contains the details of the design knowledge that is being captured, like a concise description of the commonly encountered situation and a suggestion for a possible solution. For example, if reminded that the user may be using multiple monitors, the developer can decide to allow certain elements of the user interface to be extended onto a secondary display. Unlike existing approaches to capturing knowledge as patterns, notecards do not require complete problem descriptions and solutions that are more difficult to internalize and contextualize to a specific problem during informal software design. By simply presenting a piece of design knowledge, the designer is more aware of the issues that can occur, and may be more likely to apply the knowledge when making important design decisions. This approach allows developers to capture a vast body of design knowledge in the form of design decisions, problem-solution pairs, or architectural styles, and in such a way that it can be reused during future design sessions.

Like a physical notecard, a DesignMinder notecard also has two sides. The reverse side of the card is where a visual representation such as a sketch or diagram can be added to complement the details on the front side. The developer can flip the notecard by selecting the flip button on the bottom right corner of any notecard currently in focus.

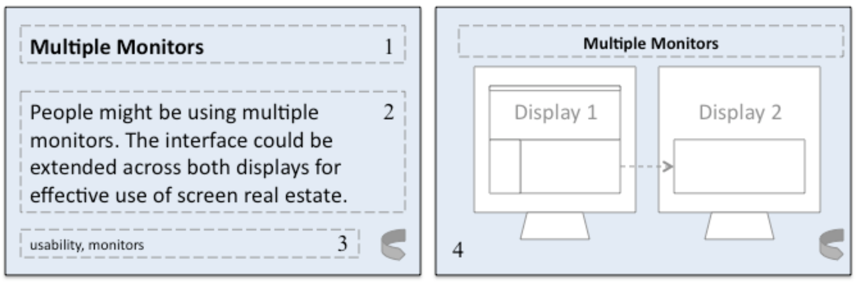
Multiple Monitors
1
People might be using multiple monitors. The interface could be extended across both displays for effective use of screen real estate.
2
usability, monitors
3
Multiple Monitors
Display 1
Display 2
4

Fig. 1 Front and reverse of a DesignMinders notecard showing the name (1), details (2), tags (3), and a diagram (4).

One of the goals of the DesignMinders interface is to simulate the interaction with actual index cards. Physical representations have been employed in creating Object-Oriented designs with CRC cards [12], and in capturing system requirements through user stories in Agile development processes [13], and have distinct advantages since they can be easily manipulated, annotated, and serve as a visual reference when explaining a design. Because knowledge that is applied during informal software design must be concise and easy to internalize and apply to the current problem, notecards serve as an appropriate representation for brief reminders of common design situations. By using an electronic representation, DesignMinders allows for the easy creation of notecards, as well as the computation and indexing of the information that is captured on the notecards to make the subsequent retrieval and presentation of knowledge more relevant to the designer.

Notecards can also be assigned one of several colors to allow developers to quickly and easily assign localized meaning to a notecard, and to visually identify it at a later time. For example, notecards pertaining to usability could be made blue, while those pertaining to security could be made red. Notecards also can be tagged using keywords, allowing developers to quickly annotate the captured knowledge, and to create a rich taxonomy of descriptions that can be used to retrieve them at a later time. [14]. For example, notecards can be tagged with keywords describing the areas of concern (ex. usability), the specific issue (ex. monitors), or contextual factors such as the project name or even the initials of the design session participants.

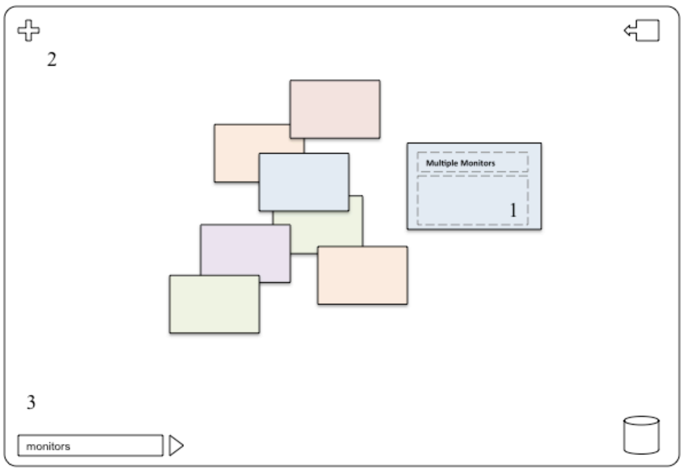

Fig. 2 The DesignMinders noteboard showing a newly created notecard (1), the new notecard button (2), and the tag input field (3).

2.2 Noteboard

Newly created DesignMinder notecards reside on the *noteboard*. As new notecards are created, they form a stack on the noteboard ordered by their creation date. A notecard can be created in two ways. The first is by selecting the new notecard button in the upper left corner of the noteboard, which places the focus on a newly created notecard. Here, the developer can click on any of predefined text areas on the notecard (name, details, and tags), or click the color button to cycle through the available colors for the notecard. Since none of the fields on the notecard are required, a developer can easily create a notecard given the currently available information (just a name, for example), and complete any of the remaining fields at a later time or during a pause in the discussion. This allows for the incremental refinement of the knowledge that is captured through the notecards without undue interruption to the ongoing design activity.

Notecards can also be created through the Calico sketching interface. A developer can select any area of the screen and select the new notecard button. This places the focus on the new blank notecard in the noteboard, only showing the reverse face of the card with the selected visual representation and the name of the notecard. This seamless interaction between Calico and DesignMinders allows for developers to rapidly create new notecards that contain valuable knowledge, such as an architecture diagram or a rough sketch of a user interface component, which can be refined with a name and brief description when appropriate. A notecard can also be removed by dragging it to the trash icon in the lower right corner of the noteboard.

An important feature of the noteboard is the *tag input field*. Located in the lower left corner of the noteboard, the tag input field allows the developer to easily tag a group of notecards either created during a window of time, or explicitly selected by the developer. If the developer selects the *tag record* button next to the tag input field, any notecards that are created will be automatically tagged with the keywords that have been entered in the tag input field, until the developer stops the tag recording. The developer can also lasso a stack of notecards in the noteboard and tag the selected cards with the entered keywords. The applied tags will be reflected on the notecards themselves.

2.3 Exploration and Search

Knowledge reuse goes beyond just capturing knowledge to making it available to the developer at the appropriate time. DesignMinders makes this possible through an exploration and search interface that can be viewed alongside the noteboard. The *exploration view* can be invoked by clicking the explore button in the upper left corner of the noteboard, and consists of a *grid* of notecards, a *search filter*, and a *color chooser*. By default, the grid shows all notecards ordered by most recent creation date. A scroll bar on the right side of the grid allows the developer to easily scroll through the notecards, and selecting a notecard will bring it into focus, making it slightly larger.

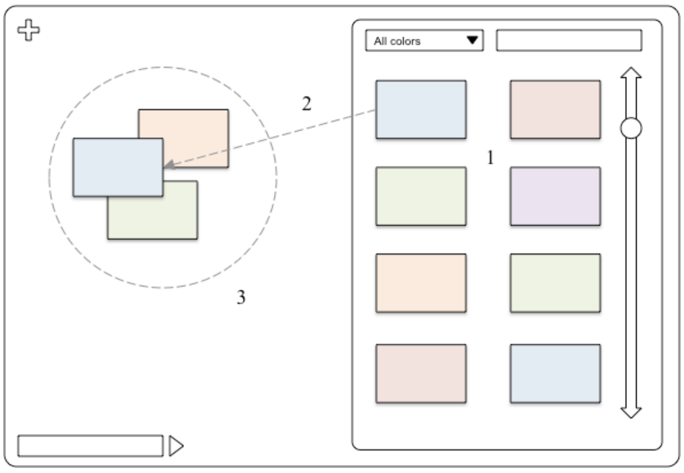

Fig. 3 The exploration view showing the grid with the search filter and color chooser (1), dragging a notecard from the grid into the noteboard (2), and selecting a group of notecards (3).

While this method of browsing is effective for a small number of notecards, it quickly becomes difficult as the number of notecards increases. To address this, the number of notecards in the grid can be reduced using the *search filter*. Keywords entered in the search filter field are used to immediately search the collection of notecards using the name, details, and tag fields. The search filter also allows for freeform input of date ranges to retrieve any notecards that have been created between those two dates. Finally, the developer can filter the notecards by clicking the color chooser and selecting the desired colors.

Developers can make use of the notecards and incorporate them into their design session by simply browsing the collection for relevant notecards. Since there may be several notecards that are relevant, a developer can also drag a notecard from the grid onto the noteboard. When several notecards are dragged over, they create a stack that can be arranged and grouped in arbitrary ways. This allows the developers to create localized collections that are relevant to a particular design session. A notecard can be removed from a stack by simply dragging it off of the noteboard.

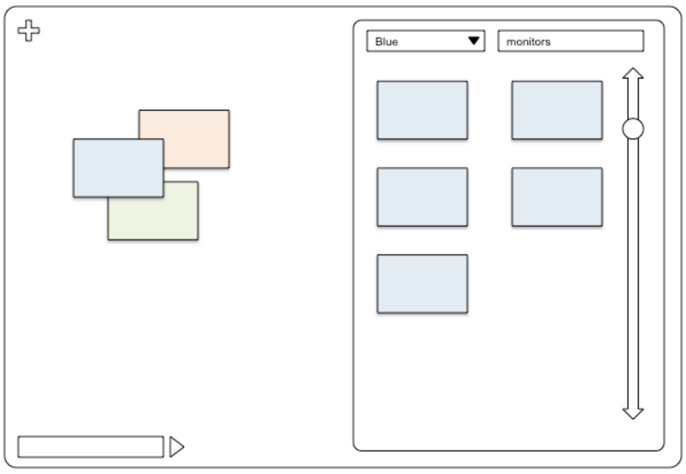

Fig. 4 The grid showing blue notecards tagged with "monitors".

3. Conclusion

A prototype version of DesignMinders is currently being developed. While the basic functionality that we have described provides a novel approach for capturing and presenting knowledge during informal software design sessions, DesignMinders can be further integrated with Calico to present design knowledge by actively scanning the Calico canvas for keywords or diagrams and automatically performing a filtered search for any relevant notecards and presenting them on the noteboard. We envision this as only the first step in developing a platform for knowledge reuse during informal software design that is based on the principle of addressing the needs of the developers in capturing and presenting knowledge during this distinct activity.

References

[1] P. N. Robillard, "The role of knowledge in software development," *Communications of the ACM,* vol. 42, pp. 87-92, 1999.

[2] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*: John Wiley & Sons, Inc., 2010.

[3] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," *Proceedings of the SIGCHI conference on Human factors in computing systems,* 2007.

[4] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool support for cooperative object-oriented design: gesture based modeling on an electronic whiteboard," *Proceedings of the SIGCHI conference on Human factors in computing systems,* 2000.

[5] A. Murray and T. C. Lethbridge, "On generating cognitive patterns of software comprehension," *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research,* 2005.

[6] G. Fischer, "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments," *Automated Software Engineering,* vol. 5, pp. 447-464, 1998.

[7] I. Frank M. Shipman and C. C. Marshall, "Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems," *Computer Supported Cooperative Work,* vol. 8, pp. 333-352, 1999.

[8] R. Farenhorst, "Tailoring knowledge sharing to the architecting process," *SIGSOFT Software Engineering Notes,* vol. 31, p. 3, 2006.

[9] A. Aurum and M. Handzic, *Managing Software Engineering Knowledge*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[10] N. Cross, *Designerly Ways of Knowing*: Birkhäuser Basel, 2007.

[11] N. Mangano, A. Baker, and A. v. d. Hoek, "Calico: a prototype sketching tool for modeling in early design," *Proceedings of the 2008 international workshop on Models in software engineering,* 2008.

[12] K. Beck and W. Cunningham, "A laboratory for teaching object oriented thinking," *OOPSLA '89: Conference proceedings on Object-oriented programming systems, languages and applications,* pp. 1-6, 1989.

[13] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2 ed.: Addison-Wesley, 2004.

[14] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary problem in human-system communication," *Communications of the ACM,* vol. 30, pp. 964-971, 1987.