CodeGraffiti: Using hand-drawn sketches connected to code bases in navigation tasks

Leonhard Lichtschlag
RWTH Aachen University
lichtschlag@cs.rwth-aachen.de

Lukas Spychalski
RWTH Aachen University
spychalski@cs.rwth-aachen.de

Jan Bochers
RWTH Aachen University
borchers@cs.rwth-aachen.de

***Abstract*—Current IDEs excel at text manipulation, but offer little support for sketching and capturing informal visual artifacts that developers create during their work on the code base. Such artifacts promise to help the examination of existing source bases and the orientation therein when linked up to corresponding code fragments. In this paper, we present a design and prototype how to use linked sketches to assist the the developer in orientating in the code base. Our evaluation with 32 users shows that testers adopt the navigation through linked sketches and refer to the spatial documentation significantly more.**

I. Introduction

Understanding the source code is one of the core software engineering activities [7], [9], [16]. To navigate a code base and propose changes to it, a developer has to understand the conceptual model of the program structure and has to find the right places to make changes, relying on the means provided by the IDE. Current IDEs provide rich support for textual organization, e.g., folders or call navigation. But, the domain of visual and spatial support for navigation [2], [5], [6], [12], [17] is employed little. Many of these approaches are automatically generated, high-level abstractions created on the basis of the underlying source code. Their advantage is that they require little effort from the user. But, they cannot be created when a developer starts by visualizing her design prior to coding, and they do not provide insight into her conceptual model.

Sketching is an established technique for ideation, exploration, and communication [15], [19]. Software developers use sketches frequently in different phases of the software development process to depict and convey different views and concepts of the system under development [20]. We approach this by using such rather informal drawings as means for navigation and understanding of code bases by integrating them into the IDE and connecting them to code artifacts (figure 1). This way, sketches fulfill a role orthogonal to generated visualizations. In this paper, we introduce CodeGraffiti, a design of integrating hand-drawn sketches into a software development environment and connecting these sketches to source code. We built a prototype plug-in for the Brackets IDE [1] and present a study of developers exploring a source base, interacting with the connected sketches to solve their tasks.

II. Related Work

Knowledge about source code often gets visualized in transient form during short meetings, e.g., on whiteboards or paper [9]. Sketches are also important for understanding existing code, designing, or refactoring [4]. They retain value after the day of creation [3], but since sketches are rarely recorded and archived afterwards, the knowledge has to be constantly rediscovered [9]. In all studies above, the use of hand-drawn sketches outweighed both tool-based visualizations and visualizations created with automated tools, often because sketching is unconstrained by formal notations, e.g., UML [4], [12]. Developers establish their own individual workflows in dealing with sketches, mostly in paper notebooks or on tablet devices [20]. Developers might redraw a sketch multiple times in order to get a cleaner version of the original sketch that will be recognizable in the future by others [3].

Several designs have been put forward to integrate sketches into the IDE: ReBoard [3] automatically captures and archives whiteboard images. Calico [11] enhances design processes on electronic whiteboards and tablet devices by introducing a grid of multiple virtual whiteboards. Codepad [13] explores interactions with sketches on touch interfaces that are connected to the IDE. Sketches have also been used as a user interface description language and in teaching [14]. Previously, we proposed sketching on the code as a means of expression for the navigator in pair programming tasks [10].

Approaches to provide a higher abstraction to views on a code base include line oriented visualizations, e.g., SeeSoft [6], vocabulary oriented [12] visualizations like Thematic Maps [8], or activity map based approaches like Code Bubbles [2] and CodeCanvas [5]. Corresponding user studies suggest that these tools can help in understanding code, but may also lead to an additional layer of confusion.

III. Design

The main strength of sketches is their informality and the ability cover a wide range of contexts with common place drawing tools. This same quality, however, makes it both tricky to parse sketches algorithmically and hard to integrate sketches into a formal environment like an IDE without loosing the very qualities that make them so powerful in the first place. Building on designs from the related work [5], [13], we developed the **mission control view** (figure 1b). The mission control view is a semi-transparent, fullscreen overlay that the user invokes and dismisses by a menu or a keyboard shortcut. This view is a canvas of infinite size that can be panned and zoomed. It provides an overview of the whole project and contains sketches and connections to the code base. These connections allow the user to navigate to the source code without using the project tree and its folder structure. In our prototype, only one mission control view exists per project. The user interface elements of this view are integrated into the view itself, since

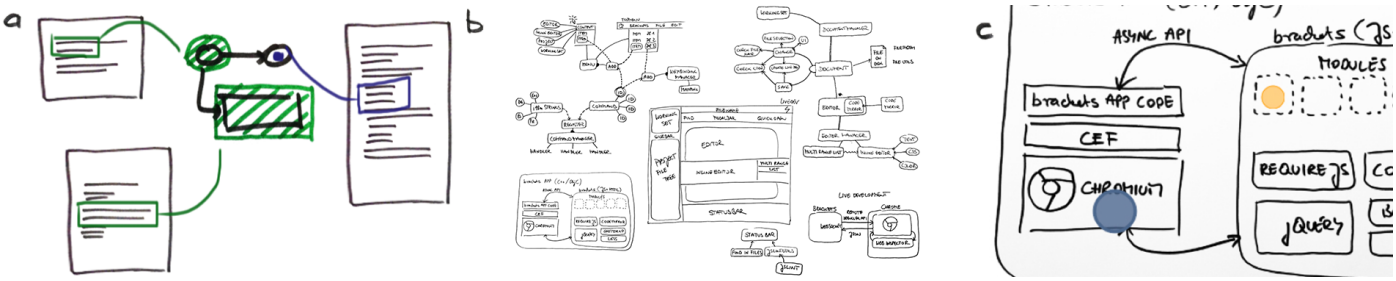

Fig. 1. a) Concept: the sketch refers to many code fragments in multiple files and reveals a structure of the code base. b) The sketch used in the study in the mission control view. c) Detail with two connection dots visible. The blue one is highlighted because it refers to the lines with the keyboard focus.

the mission control view overlaps all other areas of the editor. It is useful for sketches that explain a structure of the code base by bringing together parts of a concept and putting them into a big picture. These sketches may be composed of multiple elements referring to multiple code locations, e.g., a workflow of an algorithm that spans multiple methods.

Our design represents **connections** as color coded one-to-one relations between an (x,y)-position in a sketch and a range of code lines or a file. This does not constrain the semantics of the connections: positions in a sketch are independent of style, formalism, or the IDE's ability to parse the sketch file. Ranges in code can indicate many levels: individual lines, methods, uses of a variable, etc. This way, the user has more freedom to express meaning though connections. The prototype plug-in implements them as **connection dots** (figure 1c), i.e., buttons embedded into the sketches in the mission control view. These connection dots refer to either a file or to a range of lines in a file, e.g., a method. When the user clicks on them, the mission control view is dismissed and the IDE navigates to the corresponding file and code location. There, the code lines are colored like the connection dot in the overlay. If the mission control view is invoked with the keyboard focus in theses lines, the mission control highlights the corresponding connection dot. A user thus can navigate in both directions from and to a sketch: she may look at the sketches in the mission control view and decide to see where parts that the sketch refers to are implemented, or she may look at a method and see that it is referred to in the main sketch. Bringing the mission control to the front may allow for insights into higher level abstractions and help her form a model of the code base. CodeGraffiti monitors edits in the files and, as lines of code move, the connections are automatically updated.

Our CodeGraffiti prototype is build as a plug-in for Adobe Brackets [1]. In the current version, the focus of the design is on exploring navigation through sketches, the user experience of creating sketches remains underdeveloped. The CodeGraffiti plug-in is available for download at alongside all materials used in the user study (hci.rwth-aachen.de/codegraffiti).

IV. Study

To test how the design works for developers, we conducted a between groups user study in which we examined one navigation task across two conditions. We provided a code base with pre-existing sketches for our testers: The source base used in the study is the source code of the Brackets code editor itself (JavaScript). The sketches were created without any knowledge of our tool or the task of this study by an active Brackets developer who works at Adobe. The connections were created afterwards by the authors. As a result, the sketches did not indicate the task solution by presence alone.

The task was to locate and identify multiple locations in the code base, to point out the lines of code in which a change should be made, and to verbally outline these modifications. This task was divided into three subtasks T1–T3, which needed to be accomplished in order (adding a menu item, adding the corresponding command, linking both in the controller). We counted the tasks as successfully completed if the presented solution would result in a working implementation.

In the control condition (C1), users worked with a standard installation of the Brackets IDE with the sketches of the organization of the source base presented on a DIN A3 piece of paper. In the connection condition (C2), users worked with the Brackets IDE and the mission control view, which showed the same materials linked with the code. This way, both conditions used the same editor, code base and sketches, only the sketches were presented in different ways. Our five hypotheses were:
H1 More programmers solve the task correctly in C2.
H2 Programmers solve the task faster in C2.
H3 Programmers look at sketches more often in C2.
H4 Programmers look at sketches longer in C2.
H5 Programmers (subjectively) find that the connection between sketches and source code is an additional tool supporting their software comprehension process by helping them to understand the conceptual model behind the code.

We recruited a total of 32 participants, 5 female, aged 23 to 36 (average age 28). Four participants were familiar with the source code of Adobe Brackets. All participants were asked to fill a pre-session questionnaire in order to assess their prior knowledge. We counterbalanced the groups with regard to JavaScript and source code proficiency. Each participant was given a short introduction to the code editor and its user interface and we explained the CodeGraffiti plug-in and the shortcut to toggle the mission control view.

We asked the testers to familiarize themselves with the tools and condition by working on a pre-task before the actual evaluation. We limited the working time to 20 minutes for this pre-task and 25 minutes for the main task. Participants were asked to think-aloud during the study. Sessions were videotaped and annotated with respect to H1 to H4 by one of the authors. In order to gather qualitative data with reference to H5, we conducted a semi-structured interview about the potential application of the CodeGraffiti plug-in in actual work

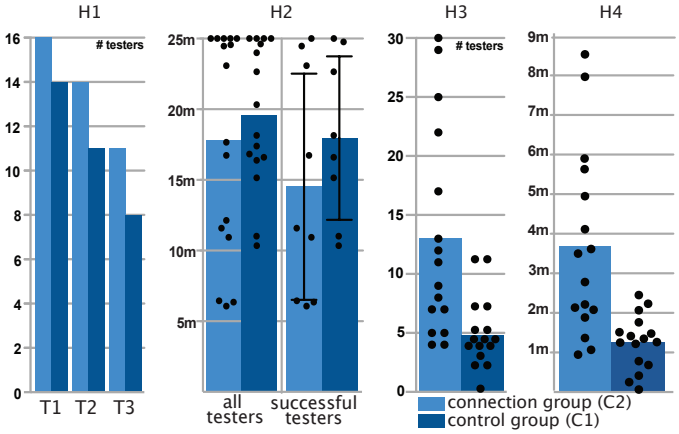

Fig. 2. Quantitative results of the user study for hypothesis H1 to H4.

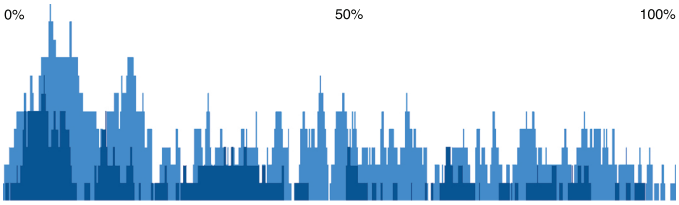

Fig. 3. Histogram of glances towards the sketches, normalized over task completion time. While most participants of both groups looked at the sketches after reading the task description (peaks at around 10%), participants of the C2 continued the engage the mission control view.

projects that the participants are or were recently involved in.

A. Quantitative Results

We reject **H1**, no subtask showed a significant difference (Fisher's, $p_{T1} = 0.48; p_{T2} = 0.39; p_{T3} = 0.47$). We reject **H2**, users did not perform faster in C2 (Fisher's, $p = 0.32$). We accept **H3**, participants in C2 looked at sketches significantly more often (t-test, $p = 0.003$) than in C1. We accept **H4**, participants in C2 looked at sketches significantly longer (t-test, $p = 0.001$) than in C1 (cf. figure 2).

Figure 3 shows the difference in behavior and interaction with the sketches with the task completion times normalized for each of the 32 participants. Consistent with above observations, the histogram graph depicts that the mission control view sketches were consulted more frequently in C2 and testers engaged with them over the whole duration of the task.

B. Qualitative Results

For either group, we observed very clear patterns of behavior. The participants of C1 read the task description and then looked at the sketches provided on paper, they studied each sketch to find potential hints on where to start the task. In the think-aloud comments, most participants stated that they could not find anything helpful and moved the paper aside. Working with the editor, participants used standard navigation operations (tabbed browsing, project tree, scrolling, search, cf. [7], [18]). If participants reached an impasse, they would take another look at the sketches to check for missed clues. At the end of a task, some participants would take another look at the sketch to check if they might have overseen something: *"I will take another look at the sketches, since they have been provided, ... there should be something on them."* All in all, the sketches were used as a reference, but mainly as a last resort. Most participants looked at the sketches since they were provided, not because they felt the need to. They used standard navigation methods to understand the code and solve the task.

Participants of C2 read the task description and then opened the initially closed mission control view. Similarly to the control group, participants in the connection group took the initial glance to get an overview of all sketches. The behavior of this group then deviated from C1: participants constantly switched between the mission control view and the source code in order to navigate the code base. The operations that were used by the C1 members, however, were in part substituted by the navigational facilities of the mission control view. Elements of the mission control view turned out to be suitable even if the names of the elements did not coincide with the filename or the method names they were connected to. However, as soon as the participants felt that the sketches and the connections provided by the mission control view would not help them, they fell back into old habits for a short amount of time and, e.g., started to search within files as well as the whole project or navigated via the file tree, only to come back to the mission control view and use its functionality again to continue with the task. Opposite to C1, testers 'defaulted' to sketch based navigation and used other means as a last resort. A common observation was that when participants of C1 asked: *"What was I looking for again?"*, they immediately opened the mission control view to find the highlighted connection dot in order to see where they were with regard to the sketches, whereas in the same situation most participants of C1 turned to the task description and not to the sketches provided on paper. It is particularly noteworthy that most participants partially or entirely explained the way they understood the tasks and how they worked together by mentally walking through the steps in the sketch: This recapitulation of the progress was made with the opened mission control view, pointing at the sketches and following the sketched lines as well as clicking onto the connection dots to get to the corresponding code segments to prove to themselves, that they had considered every part of the task. We did not observe such behavior in the control group.

C. Interview

At the time of the interview, 17 participants were working on a solo project, 17 were working on team projects. We initiated the interview session by asking testers how they would imagine to use the functionality of being able to connect sketches with source code with regard to their projects: Participants liked the idea of *navigation support* through their own projects via the mission control view using the connection dots. Some instantly imagined their project affiliated sketches and visualizations and were excited to connect them to the source code. Participants imagined the mission control view to provide an adequate *code base overview* of the project and the software architecture, e.g., to see which other team members had to be involved in the task or which other parts of the project had to be considered. Some participants had the idea that the mission control view could be used as a manager's view meaning that a project leader could *capture the development progress* of the project, i.e., new elements that had been added to the view or changes that had been made.

Above all, participants imagined this functionality to be very helpful for *on-boarding* new team members. They reported that it is hard for a new team member to catch up with all the knowledge about the project and the decisions that have been made during the design process and the implementation phase. Sketches created during on-boarding meetings could be an enormous support to get to know the project: *"The sketches were like a road map to me. I think using such a map is easier than searching because you don't need to know exactly what you are looking for. The sketches can complete the missing parts or even tell you what to look for."*. In conclusion, we accept H5: there was a consensus among the participants of C2 that the connections between sketches and source code had helped them formulate a conceptual model.

Participants also identified and confirmed challenges employing connected sketches within their projects: Despite the fact that most participants created sketches or visualizations as a regular part of their work, they still mentioned that this is *time consuming*. Sketches created during team meetings were seen as valuable byproducts without this drawback.Some participants considered the quality of their own sketches and were concerned about the *readability of sketches*. Very few participants reported to re-sketch their own sketches if out-to-date, the foremost mentioned problem was the *currentness* of sketches and connections. All testers agreed that maintaining sketches and connections can be realistic for a rather small team of developers or a medium-sized project, but *larger teams* would have problems to maintain the sketches and their quality.

V. Summary

This paper presents a way to connect source code to sketches that depict anything from the low-level details of the source code to the high-level concepts about the source code. We showed how a code base can be navigated through connected visual sketches and how it helps developers to comprehend the context of source code, to orient within the context, and to support mental walkthroughs. In a between groups user study, participants used the mission control view for navigation instead of traditional means, by quickly invoking it, selecting the target navigation, and dismissing it. Testers looked significantly longer and significantly more frequent at sketches on average, with no adverse effects on total time needed and success on the task. Furthermore, testers made more use of the sketches which they used to formulate a conceptual model and finally solve their task. The connection of a sketch in the IDE promotes sketches to an immediately available source of information, whereas the printed version remains an additional, but not directly utilizable source of information. Hence, CodeGraffiti's design offers a way to promote sketch use and sketch retention. Given that many developers already create these kinds of sketches (on paper, whiteboard, or tablets), this presents an opportunity: not throwing them away and connecting them to the code base as an viable alternative to other software visualizations.

VI. Limitations and Future Work

Our study provided less information on two other areas of developer tasks: team communication and creation of of new code artifacts. For the first, our testers especially expressed agreement for our design to connect any visualization to source code with little restrictions on formality. We look forward to further investigation of communicating about code bases through sketches. The biggest problem is that—given that developers already have sketching practices—our approach introduces the burden of connecting sketches manually as well as keeping the sketches up to date and readable despite the lack of sketching conventions. The focus of our next design efforts will be that formulating the connections to the codebase becomes a lightweight operation that fluidly embeds into the current workflows. Finally, revision management and the ability to browse prior versions of connected sketches is interesting future work.

References

[1] Adobe Brackets. *http://brackets.io/*

[2] Bragdon, A., Zeleznik, R., Reiss, S.P., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., and LaViola Jr, J.J. "Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance". In *Proc. CHI 2010*, pages 2503–2512.

[3] Branham, S., Golovchinsky, G., Carter, S., and Biehl, J.T. "Let's go from the whiteboard: supporting transitions in work through whiteboard capture and reuse". In *Proc. CHI 2010*, pages 75–84.

[4] Cherubini, M., Venolia, G., DeLine, R., and Ko, A. J. "Let's Go to the Whiteboard: How and Why Software Developers Use Drawings". In *Proc. CHI 2007*, pages 557–566.

[5] DeLine, R. and Rowan., K. "Code canvas: zooming towards better development environments". In *Proc. Software Engineering*, 2010.

[6] Eick, S.G., Steffen, J.L., and Sumner, E.E. "Seesoft—A Tool for Visualizing Line Oriented Software Statistics". *IEEE Trans. Software Eng.*, vol. 18, no. 11, pages 957–968, 1992.

[7] Ko, A. J., Myers, B.A., Coblenz, M.J., and Aung, H.H. "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks". *IEEE TSE*, vol. 32, no. 12, pages 971–987, 2006.

[8] Kuhn, A., Loretan, P., and Nierstrasz, O. "Consistent Layout for Thematic Software Maps". In *Proc. WCRE 2008*, pages 209–218.

[9] LaToza, D., Venolia, G., and Deline, R. "Maintaining Mental Models: A Study of Developer Work Habits". In *Proc. ISCE 2006*, pages 492–501.

[10] Lichtschlag, L. and Borchers, J. CodeGraffiti: "Communication by Sketching for Pair Programming". In *Ext. Abstr. UIST 2010*.

[11] Mangano, N., Baker, A., Dempsey, M., Navarro, E., and van der Hoek, A. "Software design sketching with calico.". In *Proc. IEEE Automated Software Engineering 2010*, pages 23-32.

[12] Kurtz, C. "Code Gestalt: a software visualization tool for human beings". In *Ext. Abstr. CHI 2011*, pages 929934.

[13] Parnin, C., Görg, C., and Rugaber, S. "CodePad: Interactive Spaces for Maintaining Concentration in Programming Environments". In *Proc. SOFTVIS 2010*, pages 15–24.

[14] Plimmer, B. and Freeman, I. "A toolkit approach to sketched diagram recognition". In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers*, pages 205–213, 2007.

[15] Schön, D.A. "The Reflective Practitioner: How Professionals Think in Action". Basic Books, 1983.

[16] Singer, J., Lethbridge, T., Vinson, N., and Anquetil, N. "An examination of software engineering work practices". In *Proc. CASCON*, pages 174–188, 2010.

[17] Sinha, V., Karger, D., and Miller, R. "Relo: Helping Users Manage Context during Interactive Exploratory Visualization of Large Codebases". In *Proc. VL/HCC 2006*, pp. 187–194.

[18] Starke, J., Chris, L., and Sillito, J. "Searching and skimming: An exploratory study". In *Proc. IEEE Software Maintenance*, 2009.

[19] Tversky, B. and Suwa, M. "Thinking with sketches". In *Tools for Innovation*, vol. 1, no. 9, pages 75–85, 2009.

[20] Walny, J., Haber, J., Dork, M., Sillito, J., and Carpendale, S. "Follow that sketch: Lifecycles of diagrams and sketches in software development.". In *Proc. IEEE Workshop on VISSOFT 2011*, pages 1–8.