CodeGraffiti: Communication by Sketching for Pair Programming

Leonhard Lichtschlag *Jan Borchers*
RWTH Aachen University
52056 Aachen, Germany
{lichtschlag, borchers}@cs.rwth-aachen.de

ABSTRACT
In *pair programming,* two software developers work on their code together in front of a single workstation, one typing, the other commenting. This frequently involves pointing to code on the screen, annotating it verbally, or sketching on paper or a nearby whiteboard, little of which is captured in the source code for later reference. *CodeGraffiti* lets pair programmers simultaneously write their code, and annotate it with ephemeral and persistent sketches on screen using touch or pen input. We integrated CodeGraffiti into the *Xcode* software development environment, to study how these techniques may improve the pair programming workflow.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Pair programming, code annotation, pen input

INTRODUCTION
In *pair programming*, two developers share one workstation: while the *driver* edits the code, the *navigator* observes his work, with roles swapped frequently. The navigator cannot perform any editing, but has to communicate her thoughts to the driver who then edits the code, making the team more aware of design decisions, pitfalls, and mistakes. This extreme collaboration technique has been shown to significantly lower development times and coding error rates [2].

However, this communication happens not just verbally or through written code. By pointing, gesturing, sketching diagrams and scribbling annotations, programmers explain graphical properties to each other, test ideas, or visualize complex software structures. Current IDEs rely on keyboard and mouse input to create syntactic content such as code, and offer little support for graphical content. Thus, programmers have to express documentation and other semantic content either *textually*, in *external programs*, or *physical artifacts*: Automated tools like Doxygen (see doxygen.org) that parse textual comments in the code are widespread but unsuitable for capturing quick annotations or graphical content. Graphics created in separate pen-based applications offer greater expressiveness, but require mode switches, and do not link sketches to code. Physical artifacts such as whiteboards or paper offer quick creation of highly expressive, informal notes, but outside the IDE. Such notes are also easily thrown away or wiped from the board, being captured digitally only if they seem valuable enough.

UIST'10, October 3–6, 2010, New York, New York, USA.
ACM 978-1-4503-0271-5/10/10.

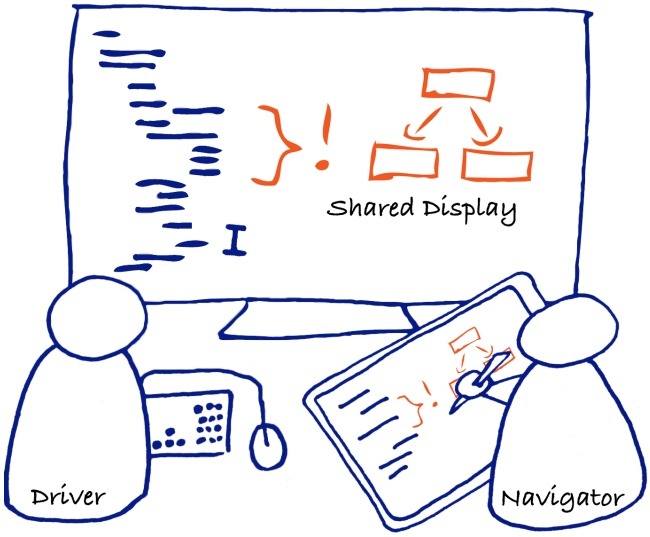

Figure 1: Pair programmers using CodeGraffiti. The *driver* controls syntactical code input (left), the *navigator* controls the sketching cursor (right). Both share the same view on the code, but with independent cursors. The code annotations point out some semantical errors and improvements.

Other professions such as manuscript editors often use editing marks to communicate revisions and opinions [3, 6], which has also been proposed for teachers when reviewing code [4, 5]. *CodeGraffiti* extends graphical input for general semantics [1] in the IDE. It places sketches, documentation, explanations on structure and design decisions directly in line with the code. This lets pair programmers communicate graphically right in the IDE. Our hypotheses are that this lets programming teams communicate graphical ideas more effectively, that they retain more sketches for documentation than via physical artifacts, and that sketches are integrated better into code structures.

INTERACTION DESIGN
Our design rationale is to give the navigator a way to support team conversation by sketching directly on the shared screen (Fig. 1). Sketching (Fig. 2) and annotating should be as easy as picking up a pen to draw or write on paper or a white-

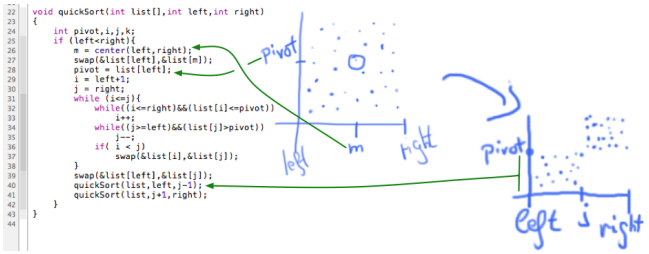

Figure 2: *QuickSort* explained via a sketch. The sketch directly references the declared variables.

board. The driver is still the only person entering syntactical code (Fig. 1). The navigator enters semantic graphics that do not compile, making it easy to quickly jot down a sketch, explain a design idea or point out a typo to the driver. Similar to copy editor marks, *annotations* provide information to a word or line using underlining, exclamation marks, or circling. *Sketches* (Fig. 2) are broader in scope and may refer to multiple points, e.g., to highlight a software design pattern. Hand-drawn sketches and annotations look appropriately informal, and can be adapted easily as design decisions change.

A common focus is considered essential to maintain the higher quality of pair-programmed code. Therefore, despite separate text and sketching cursors, we decided not to allow the navigator to scroll to a different part of the code. By sharing the same view, neither member can wander off and work on separate tasks or even files.

The IDE links graphics within code bodies *persistently* to one or more anchors in the text. It also links anchors visually to the graphics so that sketch and code can provide explanations for each other. These graphics show up for every programmer who opens the code base with CodeGraffiti installed, and can become a permanent part of the software documentation over time.

We also allow ink over interface widgets, reference documentation, and other screen elements not related to source code for illustration and gesturing purposes. Missing this link to source code, however, they are treated as *ephemeral*, fading out after five seconds.

PROTOTYPE

We implemented *CodeGraffiti* in two parts: a server running as a plug-in in the IDE, and clients for various input modalities. The plug-in was written for Xcode[1], the free, native development IDE for Apple devices. It accepts input from three clients:

- finger painting gestures on iPad and iPhone (Fig. 3a),
- pen input on a graphics tablet (Fig. 3b),
- pen input on a tablet display (Fig. 3c).

The server stores sketches in a companion file to each source file, and keeps a table linking them to anchors in the text. This way, users without the plug-in do not see any artifacts, and CodeGraffiti integrates well into revision control systems. As lines are inserted, anchors move to another file, or referenced variables are renamed, the IDE updates sketch

[1] www.apple.com/macosx/developers/

Figure 3: CodeGraffiti clients on a) Apple iPad/iPhone, b) Wacom Intuos tablet, c) Wacom Cintiq tablet display.

positions accordingly. When the text anchor is removed the sketch becomes uncoupled, and the programmer can reattach it to a different position.

An interesting challenge is that the driver might scroll the view while the navigator is sketching. If both share only one physical display (Fig. 3b), this is unlikely. If the navigator has a mirrored display (Fig. 3a,c), we wait until she finishes drawing and then snap it to the driver's new view.

SUMMARY and FUTURE WORK

We presented a rationale and prototype for simple and integrated sketching in pair programming. While designed with pair programming in mind, a single programmer or code reviewer can also benefit from the sketching facilities provided.

Our next steps are to deploy CodeGraffiti to both single and pair programmers, and explore how sketching input is adopted, and how it fits into the workflows of single programmers, pair programmers, and code reviewers. We also hope to learn how the three input modalities compare for different tasks. Finally, preliminary tests already indicated that the second cursor might be distracting to the driver. To alleviate this problem we plan to fade out the pen cursor if it moves near the main cursor.

ACKNOWLEDGMENTS

This work was funded by the German B-IT Foundation. We would also like to thank Tom Igoe for fruitful discussions and Thorsten Karrer for his support.

REFERENCES

1. Borchers, J. HyperSource: A Hypermedia Program Development And Documentation System. *Proc. Third International World-Wide Web Conference*, 1995.

2. Bryant, S., P. Romero and B. du Boulay. Pair Programming and the Mysterious Role of the Navigator. *Int. J. of Human–Computer Studies*, **66**(7):519–529, 2008.

3. Liao, C., F. Guimbretière and K. Hinckley. PapierCraft: a Command System for Interactive Paper. *Proc. UIST 2005*, 241–244.

4. Plimmer, B. and P. Mason. A Pen-based Paperless Environment for Annotating and Marking Student Assignments. *Proc. AUIC 2006*, 37–44.

5. Plimmer, B., J. Grundy, J. Hosking and R. Priest. Inking in the IDE: Experiences with Pen-based Design and Annotation. *Proc. VLHCC 2006*, 111–115.

6. Schilit, B. N., G. Golovchinsky and M. N. Price. Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations. *Proc. CHI 1998*, 249–256.