Building an Ecologically valid, Large-scale Diagram to Help Developers Stay Oriented in Their Code

Mauro Cherubini
CRAFT
École Polytechnique Fédérale de Lausanne
Station1, CH-1015 Lausanne, Switzerland
mauro.cherubini@epfl.ch

Gina Venolia, Rob DeLine
Microsoft Research
One Microsoft Way, Redmond, WA 98052
{gina.venolia, rob.deline}@microsoft.com

Abstract

This paper presents the creation, deployment, and evaluation of a large-scale, spatially-stable, paper-based visualization of a software system. The visualization was created for a single team, who were involved systematically in its initial design and subsequent design iterations. The evaluation indicates that the visualization supported the "onboarding" scenario but otherwise failed to realize the research team's expectations. We present several lessons learned, and cautions to future research into large-scale, spatially-stable visualizations of software systems.

1. Introduction

Developers form elaborate mental models of their code [9], which are always incomplete and are often wrong. Being this a well-known issue for the community, some researchers suggest that we should create visualizations of the code to help developers to stay oriented in their problem solving activities.

Like other researchers [2, 11], we can imagine a team-scale visualization that is always "on" and is spatially stable, yet adapts to the evolving source code. We might hope that such a visualization could give each team member a spatial foundation on which to build their mental model, give a visual common ground to the team when engaging in discussions, and provide a starting point for interactive exploration of the code. Given the complexity of professional code, the visualization would have to be presented on a very large, high-resolution display.

Rather than diving straight into the implementation of such a visualization and display device, we wanted to do a low-cost, paper prototype to help us understand the factors that might influence its design. Rather than deciding up front what the visualization should contain, we wanted to work with a development team to create it to their requirements. This paper describes our efforts to do just that. We call the result a "code map" by analogy to the familiar cartographic map.

In this paper we report our effort to use participatory design techniques to create an ecologically-valid [8] code map for a software development team. We created and maintained this code map in the team space for a one-month period, during which we solicited feedback to both refine the design and understand which features were necessary to support the team's activities.

Our hypothesis was that a code map would be used for three scenarios: understanding the new features of the code, re-engineering parts of the code, and transferring relevant information of the code to a new developer (that we called "onboarding").

2. Related Work

Software artifacts and the social practices that produce them are intrinsically intertwined. As noted by de Souza et al. [5], we can find two sorts of dependencies in software development: a technological reliance between software elements and a social one between developers.

Configuration management tools were developed to solve dependencies at technological level but, as noted by de Souza et al. [5], these have the opposite effect at social level, creating a distinction between private and public aspect of the work. The same phenomenon was observed by LaToza et al. [9]: an escalating fracture, rooted in code ownership, which creates black boxes in the code. A developer and his code become deeply intertwined [1, 4, 9, 10].

Recent studies proposed solutions to support collaborative development through increasing the group's awareness, an understanding of the activities of the others [5]. As noted by Storey et al. [14], different proposed solutions mimic existing face-to-face awareness mechanisms. However, software development does not involve the manipulation of physical objects, so it is not usually possible to ascertain a developer's activity just by observing at which part of the code he is looking at.

Another source of awareness in software development is the artifacts themselves. Configuration Management tools are used to map the history of authorship and changes to the system [2].

Many prototypes for software visualization have been proposed with the aim of supporting awareness and collaboration (e.g., SeeSoft, VRCS, Tukan, ADVIZOR, Xia/Creole, Palantír, etc., reviewed in [14], FastDASH [2], the War Room Command Console [11], and Relo [13]). Evolution Matrix [6] also goes in this direction, using program analysis to calculate various metrics based on a set of releases of the software. However, and as noted by Storey [14], there is generally a lack of empirical work on the desirable features that should be provided by such tools.

Most of these systems provide visualizations which are intended to foster greater coordination. Few studies, though, validated this assumption with theories from cognitive science or with empirical data from controlled experiments. And yet after more than a decade of these prototypes, recent

0-7695-2987-9/07 $25.00 © 2007 IEEE
DOI 10.1109/VLHCC.2007.19

field studies show little adoption of tools for automatic visualization of software or for reverse-engineering [3]. We still know little about the practices around collaborative software development and the mechanisms that relate productivity, awareness, and externalized mental imagery [12]. This study is an attempt to gain some insights in this domain.

3. Method

We studied a product team at Microsoft Corporation called Pandora (a pseudonym). We chose this group because the 300-file code base was neither too large or two small, the code base was mostly new code, the team was already using printed visualizations (of UI features and schedules, but not code) placed in the hallways, and the project manager was shared our enthusiasm for the potential benefits of visualizing the code base. The group was composed of 8 software engineers, 5 test engineers, and 5 user interface designers. Team members occupied individual offices of a single floor within an office building. Their documentation was minimal, and most of it was purely textual. Offices were connected by several long hallways. For the period we worked with the group they were in the second sprint of a new product, working on implementing and then testing some new features.

We conducted an initial observation during the first week of July 2006, and then we intensified our observations during the following four weeks. We focused our observations on the core development team, as they were more concerned with the architecture of the system and they frequently interacted with one another.

Predeployment Observations. During the initial week we conducted non-intrusive observations in the team area, to understand the characteristics of the group. We were interested in the current use of visualizations, and particularly the ways these were used to maintain awareness in the group.

Initial Design. To create an initial map design for the Pandora group we invited each developer to draw from memory important elements of the architecture of the Pandora application. We gave each developer a large, blank piece of paper (about 36×60 inches, or ~90×150 cm) and a set of markers of various colors. We asked them to identify and represent the elements that they considered important for coordination with other developers on the team.

We analyzed and merged these initial drawings to produce the first version of the diagram. We printed the diagram using an industrial plotter with rolled paper stock, 60 inches (150 cm) wide. We then presented the initial design to the group and collected their suggestions for improving its appearance and usefulness.

Deployment. We used automated tools to create and update the elements of the drawing that represented classes and methods. We manually positioned these elements and added other graphics and text.

During the daily informal meetings, we collected information on the ways the developers used the map and other forms of visualizations that they created for themselves during the same period. We asked the developers to annotate the map with felt pens and stickers. We were particularly interested in whether and how the developers would personalize and adapt the map to support the specific needs of their work. Each day we took a snapshot of the annotations on the map and of the relevant drawings produced in the team area and in each developer's office.

Each day we updated the map to incorporate the annotations and the changes to the Pandora code base, printed the updated map, and posted it in the team space.

Final Interviews. The last step of our protocol was an interview with each developer to gather the developer's final thoughts about the code map. We structured these interviews following a list of questions targeting the design of the map, asking each developer to recall a particular episode in which the map played an important role for his/her work, and the developer's opinion on the presence of the map in the group space.

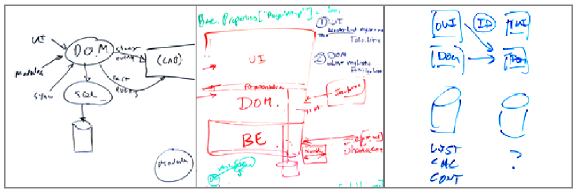

Figure 1. Whiteboard captures reproducing the basic elements of the Pandora architecture

4. Results

Predeployment Observations. During the initial week of observations we could see that developers used transient forms of diagrams, like sketches on whiteboards, scrap paper or notebooks, for exploration activities. The majority of their diagrams were sketched on whiteboards during ad-hoc meetings.

We observed many instances of whiteboard sketches where the developers reproduced the same basic elements of the architecture of the Pandora system (see Figure 1).

Initial Design. In their initial drawings, none of the developers used a formal notation to represent the architecture of the system. While each developer depicted the same basic architectural elements in the same basic configuration, there were notable differences between the drawings.

Across the different depictions, we identified four recurring modules that were always present. These corresponded to the general elements of an application: a data layer (STORE), an interface for the persistence of data; a domain object module layer (DOM), needed for the upper levels to interact with the Store; a user interface layer (UI); and a module named CAB, a framework for dependency injection that was adopted by the core team. We gave the CAB module particular attention as its representation varied greatly between the drawings. We asked the developers specific follow-up questions and learned that the framework was borrowed from another group at Microsoft. Only one developer in Pandora participated in the development of the CAB module; the others tended to treat it as a "black box." Finally, each initial drawing contained elements absent from the others. Developers tended to show more detail in the area that they "owned," which typically occupied a central place in the depiction (see Figure 2).

At the end of this phase we engaged a group discussion to understand the reasons behind these differences and to find better ways to depict these elements on the code map.

Following the findings of the initial observations we decided that the code map should depict the types (classes, interfaces, etc.) and their members (methods, fields, etc.) situated within an architectural block diagram. The types and members represented the microscopic structure of the project, while the block diagram represented the macroscopic structure.

Therefore, our initial design was quite simple. We created an initial architectural diagram by manually synthesizing the developers' initial drawings (Figure 2, gray box) and then rendered it. We examined the directory structure of the source code to find and depict additional architectural structure. There were approximately 200 types represented in the initial code map. We manually positioned the types according to the organization of the project's source code directories. The developers observed that the initial result was not really readable.

Deployment. During the three-week deployment phase, the developers suggested many changes to improve the readability of the map: *Corrections.* Some of the architectural clues that we had extracted from the directory structure of the source code were outdated or simply wrong. *Sub-layers.* There was additional architectural structure that was not represented in either the source code directory or the developers' initial drawings. *Labeling.* The labels that we had found by examining the source code structure were often wrong or outdated. *Categorical hiding.* Developers told us that some categories of types could be omitted from the drawing. In particular exception classes, enumerated types, and iterators were omitted. *Banding.* There were hints in some of the developers' initial drawings that there were architectural concepts that cut across the traditional notion of hierarchical grouping. For example the concept of a "reminder" occurred in the UI, DOM and STORE layers. While the traditional hierarchical architectural layering proceeded horizontally, we rendered these concepts as vertical bands perpendicular to the horizontal layers. *Relationships.* In response to developers' requests we added lines representing the inheritance and reference relationships between types.

We initially posted the code map on the wall of a hallway located near the group's meeting space, but not particularly close to any of the developers' offices.

We observed little interaction in front of the map. Few developers reported using the map for ad-hoc or group meetings. When asked, some explained that the map was "too far" from their offices and having a technical conversation in the hallway was "too exposed." At the end of the first week we tried to overcome this problem moving the position of the map from its initial position to a hallway that was closer to the core of the developers' offices and farther from the major traffic patterns.

Finally, the developers said that the initial map had too much detail for the needs of informal meetings. To address these last two issues, we produced several copies of a reduced-version of the map, that we called the MiniMap. The MiniMap contained the same boxes in the same layout as the large code map, but we replaced the detailed class interfaces with just the class name. Consequently we could print the MiniMap much smaller (38×22 inches, or 95×55 cm), and place a copy in each developer's office, hoping to make them seem more convenient and less "precious" than the hallway map. In addition we broadened our scope to include the test engineers, so they received copies too. We positioned the MiniMap close to each participant's whiteboard, in hopes of supporting ad-hoc conversations.

The final full-sized code map was at or beyond the limit of ergonomic usability. The 60 inch height was awkward for tall people to see the bottom of, and impossible for short people to see the top of. The smallest font size, used for type members, was about 5 points, which some (but not all) participants considered too small to be readable.

During the deployment period, the developers continued intensively using their office whiteboards.

Final Interviews. We found little evidence of interaction on either size of the code map. The notable exceptions were two new hires in the group that used their MiniMaps extensively to support their "onboarding" process. They carefully studied the maps and annotated them, adding missing details such as the name of the Dynamic Link Libraries containing the classes represented on the maps.

The rest of the core team reported mixed feelings about the map. Participants gave three main reasons for not using the maps. First, the maps contained too few or too many details in relation to the scope of the discussion in which it would be used. Second, the information contained in the maps was not dynamic. For instance, as one tested wanted the diagram to show the call graph among the classes instead of inheritance relationships. Finally, the layout of the elements depicted on the map was static, and couldn't be changed to adapt it to the scope of the conversation.

5. Discussion

We had several aspirations for the code map. We hoped to stimulate and ground discussions by providing a common visual referent. We hoped that by providing the architectural elements that every developer sketches over and over during

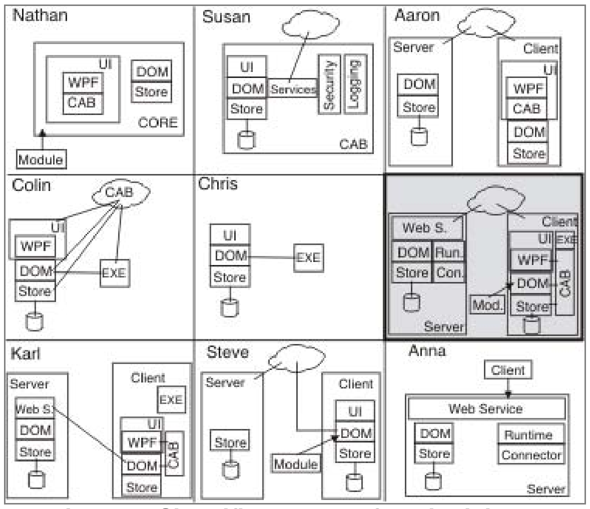

Figure 2. Simplified model of the individual contributions of each developer in the core team to the first assignment. The gray box contains the synthesis that we proposed as the initial design of the code map

informal meetings, a code map could facilitate the developers' interaction on implementation details maintaining the "big picture" in mind. We also hoped that by reifying the architecture and by incorporating the developers' suggested changes to the map design, that the code map would stimulate architectural insights and improvements. We hoped that we could shift some of the cognitive burden of code understanding to the spatial domain.

We do not have conclusive evidence that these aspirations were realized. Developers had very few interactions in front of the map and in almost all the cases they did not annotate the map. Additionally, the level of details of the map did not reflect the dynamism and focus required by most of the meetings: the map simultaneously contained too much and not enough detail.

Plasticity and Adaptability. In the final interviews we asked the participants about their continued, intensive whiteboard use despite the availability of the code map. They explained that the whiteboard was more flexible than the code map or the MiniMap because it was easy to represent exactly the right amount of detail. Additionally, they told us that during ad-hoc meetings they only needed to reconstruct only a few details of the architecture to be able to situate their discussion. One of the developers said: "It will be cool to have a dynamic code map with an interface like [the movie] *Minority Report*."

In our understanding, the map was lacking plasticity and adaptability to different situations. The static arrangement of the elements did not allow changing the perspective on the system to suit the conversation at hand.

Customization. The developers also suggested that could have been important for them to be able to personalize and customize the map according to their necessities and specificities. The suggested, for instance, the possibility to center the map on their methods, or to highlight the methods they owned or they worked on a map that could act, in this case, as a visual reminder.

Location. The position of the MiniMap seemed to have an important role in its use. The developers, in fact, wanted to have the map close to the whiteboard and in the line of sight from their desks. In this way, while working on the code, they could refer to the map and to the discussion that happened on the board to offload some of their cognitive load while working on the code.

6. Conclusions

The validity of our findings is constrained by the length of our chosen observation period, which might have been too short to observe longer-term dynamics. Additionally, the same techniques should have been tested on different group, maybe not following agile work practices, or in different companies with a different culture.

One of the most evident conclusions is that a static, paper display might be not enough for testing different graphical solution and observe their impact on the work practices. Conversely, the resolution and size needed greatly exceeds current display technology (see for instance [7, 15]).

This study should serve as a cautionary tale to researchers who are trying to build large-scale visualizations of code to support team work. The physical and social characteristics of the work environment can be as significant as the design of the visualization in its success or failure.

7. Acknowledgments

We would like to thank Andrew J. Ko, Andrew Begel and Jacob Biehl for their precious advices.

8. References

[1] V. Bellotti and S. Bly. Walking away from the desktop computer: Distributed collaboration and mobility in a product design team. In *Proc. CSCW 1996*.

[2] J. Biehl, M. Czerwinski, G. Smith, and G. Robertson. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. In *Proc. CHI 2007*.

[3] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In *Proc. CHI 2007*.

[4] U. Dekel. Supporting distributed software design meetings: What can we learn from co-located meetings? In *Proc. HSSE 2005*.

[5] C. de Souza, J. Froehlich, and P. Dourish. Seeking the source: Software source code as a social and technical artifact. In *Proc. of GROUP 2005*.

[6] S. Ducasse and M. Lanza. The class blueprint: Visually supporting the understanding of classes. *IEEE ToSE*, 31(1):75–90, January 2005.

[7] K. M. Everitt, S. R. Klemmer, R. Lee, and J. A. Landay. Two worlds apart: Bridging the gap between physical and virtual media for distributed design collaboration. In *Proc. CHI 2003*.

[8] E. M. Huang, E. D. Mynatt, and J. P. Trimble. Displays in the wild: Understanding the dynamics and evolution of a display ecology. In *Proc. Pervasive 2006*.

[9] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *ICSE 2006*.

[10] D. W. McDonald and M. S. Ackerman. Just talk to me: A field study of expertise location. In *Proc. CSCW 1998*.

[11] C. O'Reilly, D. Bustard, and P. Morrow. The war room command console, shared visualizations for inclusive team coordination. In *Proc. Softviz 2005*.

[12] D. E. Perry, N. A. Staudenmayer, and L. G. Votta. People, organizations and process improvement. *IEEE Software*, pages 36–45, July 1994.

[13] V. Sinha, D. Karger, and R. Miller. Relo: Helping users manage context during interactive exploratory visualizations od large codebases. In *OOPSLA'05 Eclipse Technology eXchange (ETX) Workshop*.

[14] M. D. Storey, D. Cubranic, and D. M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proc. SoftVis 2005*.

[15] Ron B. Yeh, Joel Brandt, Scott R. Klemmer, Jonas Boli, Eric Su, Andreas Paepcke. *Interactive Gigapixel Prints: Large Paper Interfaces for Visual Context, Mobility, and Collaboration*. Stanford University Computer Science Department Technical Report. October, 2006.