Code Thumbnails: Using Spatial Memory to Navigate Source Code

Robert DeLine, Mary Czerwinski, Brian Meyers, Gina Venolia, Steven Drucker, and George Robertson
Microsoft Research, Microsoft Corporation, Redmond, USA
rdeline, marycz, brianme, ginav, sdrucker, ggr @ microsoft.com

Abstract

Modern development environments provide many features for navigating source code, yet recent studies show the developers still spend a tremendous amount of time just navigating. Since existing navigation features rely heavily on memorizing symbol names, we present a new design, called Code Thumbnails, intended to allow a developer to navigate source code by forming a spatial memory of it. To aid intra-file navigation, we add a thumbnail image of the file to the scrollbar, which makes any part of the file one click away. To aid inter-file navigation, we provide a desktop of file thumbnail images, which make any part of any file one click away. We did a formative evaluation of the design with eleven experienced developers and present the results.

1. Introduction

Recent studies have shown that experienced developers doing maintenance tasks on unfamiliar source code spend a large fraction of their time simply navigating around the code. In a modern development environment, such as Microsoft Visual Studio or Eclipse, a developer typically uses many features to navigate: opening, switching between, and scrolling within tabbed documents; clicking on items in hierarchical overviews (class view, project file view); clicking on source code entities (go to definition); and issuing textual queries or structural queries (find definition, find callers, find all references) and jumping to the results. In a study of experienced student developers modifying a 500-line Java program, Ko, Aung and Myers found that participants spent an average of 35% of their task time navigating [12]. In a similar study, in which experienced professional developers modified a 3000-line C# program, DeLine, Khella, Czerwinski and Robertson found that the inefficiencies of code navigation played a large role in participants' poor task completion rates [5]. In both studies, experienced programmers had difficulty navigating around programs of very modest size; the problem is presumably worse in larger programs.

One reason why code navigation is so inefficient is that most of the provided UI mechanisms are based on memorizing symbol names. To open a file requires knowing its name; to click on a method in a class overview requires knowing the name of the method, its containing class and the class's containing namespace; to find an object using search requires knowing the name of the sought object or a nearby object. The number of symbols in even a modest program can overwhelm a developer's working memory, causing confusion and false navigation steps. A second problem is that source code's visual uniformity makes code difficult to recognize at a glance, leading to disorientation. (DeLine, Khella, Czerwinski and Robertson report an instance where a developer viewed a method, briefly navigated away and returned without realizing it was the same method as before.) Hence, a developer spends considerable cognitive resources both on remembering symbols in order to navigate and on distinguishing one part of the code form another.

To lower this cognitive burden, we introduce new overview features to a development environment to better support the use of spatial memory to navigate source code. We call these new features Code Thumbnails, shown in Figures 1 and 2. The rest of this paper will describe Code Thumbnails (CT) and its preliminary evaluation.

2. Code Thumbnails

Our design introduces two user interface features to Microsoft Visual Studio: the Code Thumbnail Scrollbar (CT Scrollbar, or CTS) for navigating within a file; and the Code Thumbnail Desktop (CT Desktop, or CTD) for navigating between files. The CT Scrollbar, shown in Figure 1, supplements the document's vertical scrollbar with a thumbnail image of the entire document. The document text is shrunk to fit the height of the scrollbar. For short files, the thumbnail font size is capped at a maximum of 2.5 points, keeping the text just below the threshold of readability; when this is the case (not shown in Figure 1), the unused portion of the display is filled with gray. Our intention is that the developer can use the text shape for visual landmarks (a perceptual activity), without reading the text (a cognitive activity). The currently visible portion of the document is reflected both in the scrollbar's "thumb" (as usual) and with a box drawn around the corresponding text in the thumbnail. Since Visual Studio provides a folding editor for code (that is, the text is parsed into a tree with collapsi-

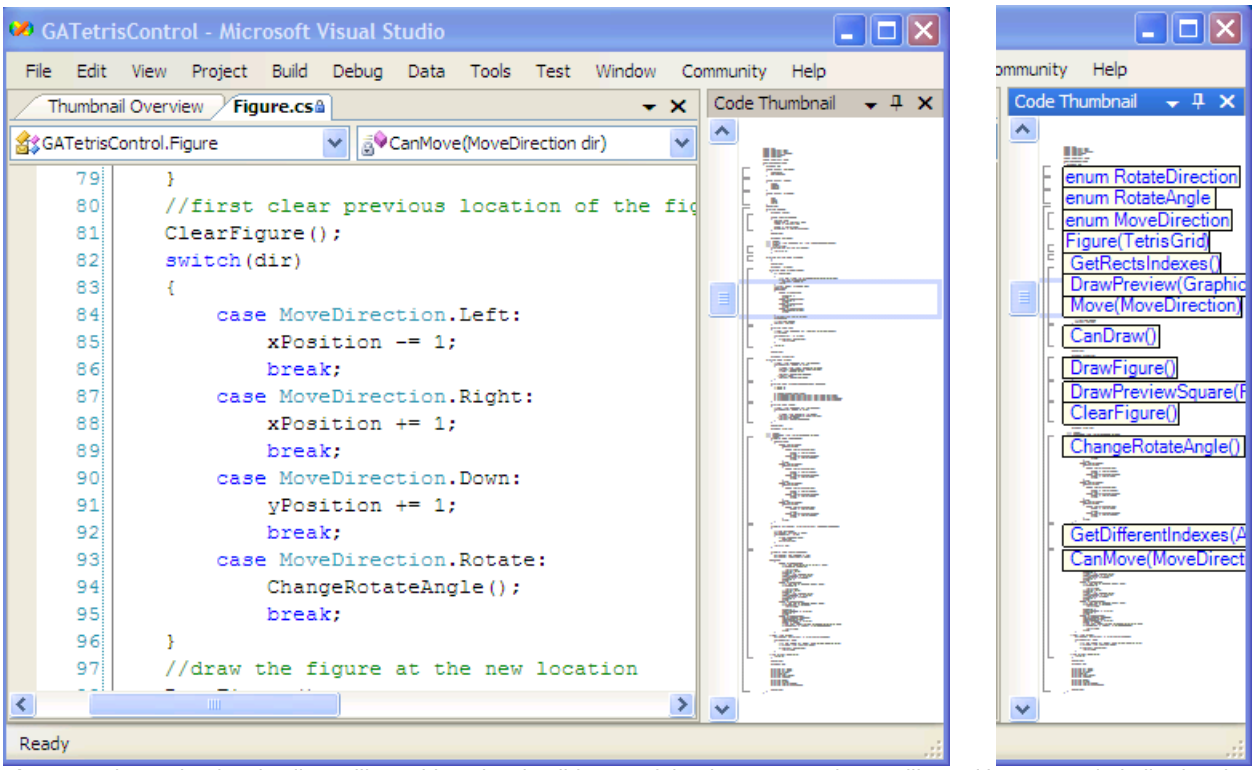

Figure 1. The Code Thumbnail Scrollbar adds a thumbnail image of the document to the scrollbar, with a rectangle indicating the current view (left). On mouse-over, it the names of potential navigation targets are revealed (right).

ble nodes), we reflect this tree in the code thumbnail with brackets representing the second- and third-level nodes, which are typically types and their members. The brackets provide another form of visual landmark.

To navigate using the CT Scrollbar, a developer can either use the scrollbar at left in the usual way, or she can click on a location in the thumbnail to jump to the corresponding place in the code. Whenever the mouse cursor is inside the thumbnail area, labels appear showing the names of likely navigation targets, specifically the names of second-level items with no children (e.g., enums) and third-level items (fields and methods) as shown on the right side of Figure 1. In the current design, these pop-up labels occlude the code shape, which is an area for improvement.

The CT Desktop, shown in Figure 2, shows a thumbnail image of every source file in the project, arranged on a desktop surface. Each thumbnail has a label at the top, which shows the file name and serves as a handle for moving the thumbnail. A developer can arrange the thumbnails on the desktop as she sees fit. The code thumbnails are drawn exactly like those in the CT Scrollbar, except that the currently visible portion is drawn with a filled rectangle to make it more apparent. We use the same font size for all thumbnails on the desktop, which means that each thumbnail's height is proportional to its file's length. The document whose editor is active is highlighted with a thicker border than the others. Documents that are currently closed are shown with a grey background, grey title and no scroll area. As with the CT Scrollbar, moving the cursor over a thumbnail pops up target labels, and clicking on a thumbnail activates the document's editor and scrolls to the chosen part of the document. Clicking a thumbnail's title area activates the document's editor without scrolling the document. Double-clicking a grayed thumbnail opens the document and activates its editor.

When the programmer uses any of the standard search tools, the search results are highlighted in both the CT Scrollbar and CT Desktop. This makes it easy to see all search results at a glance.

Both the CT Scrollbar and Desktop are intended to allow the developer to form spatial memory of the code. The CT Scrollbar provides a stable, one-dimensional space per document, with visual landmarks to help the user distinguish different parts at a glance (namely, the code shape, the brackets and the target labels). The CT Desktop provides a stable, two-dimensional space of all the documents, again with visual landmarks (namely, the thumbnail landmarks, plus their placement).

Our UI design choices were driven by our study goals. Specifically, we were interested in whether developers could form spatial memory of the code and how that would affect their navigation choices. We

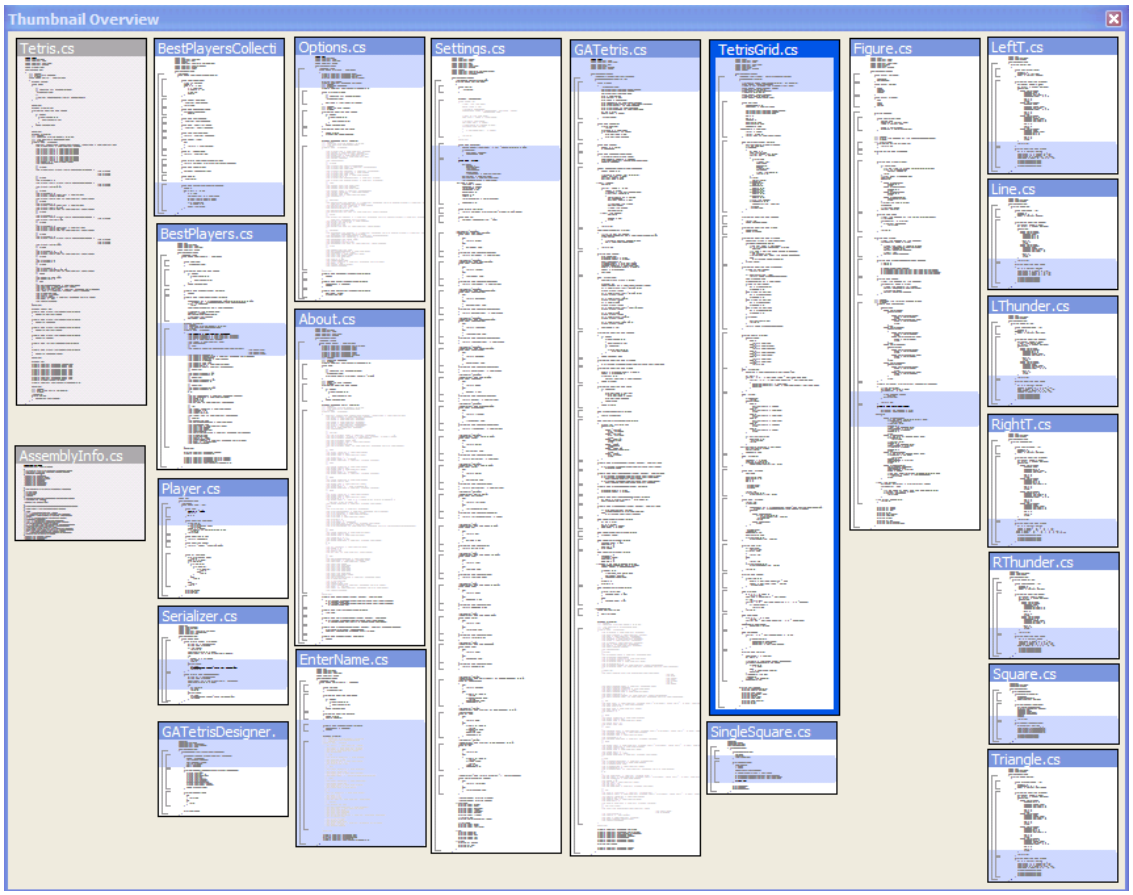

Figure 2. The Code Thumbnail Desktop window shows thumbnail images of all source files, which the user arranges on a desktop surface.

chose thumbnail images of the code to define the space and provide visual landmarks because this depiction requires little learning from the user. In future studies, we could weigh the spatial-memory benefits of Code Thumbnails against existing techniques or other, potentially unfamiliar representations, like UML diagrams, Voronoi Treemaps [1], or Software Terrain Maps [6]. Our intention here is to do an initial evaluation of the potential of the idea of Code Thumbnails.

3. Spatial Memory and Visual Code Cues

There is a large body of literature on the use of spatial cognition while navigating graphical user interfaces (see Ehret's dissertation [8] for a recent example) and wayfinding [3][4][18], both for real and electronic spaces. Some of these studies have culminated in a set of guidelines for designers of virtual worlds [16]. For instance, leveraging knowledge from the architectural domain [13][15], Darken and Silbert [4] have shown that adding real world landmarks, such as borders, paths, boundaries and directional cues, can greatly benefit navigation performance in virtual reality. In particular, they found that stationary or predictably moving cues are optimal, and that multiple sensory modalities can be combined to assist searching through an electronic space. They also have shown that if the space is not divided using a simple, organizing principle, that users will impose their own, conceptual organization upon the space.

However, in previous research, Jones and Dumais [11] suggested that, for document retrieval, adding spatial information to the document has little value beyond giving the document a semantic label. In their study, participants were able to accurately and efficiently retrieve stored documents in the real world with as impoverished a semantic label as a two-letter cue! Storing the document in a spatial position did improve performance over baseline control conditions, however.

Our goal is to build on research as to how landmarks and spatial layouts generalize to the retrieval of information in large, semi-structured, electronic worlds in new domains like software development. In addition, it is our intention here to leverage what we know about spatial memory and perceptual cues in an effort to help

programmers stay oriented in their code, in addition to helping them navigate more effectively.

4. Related Work

SeeSoft [9] was a family of novel software visualization techniques designed to show various textual properties of large software systems. One SeeSoft view showed all the code in all of the files of a software system, shrunken so that they were all visible on the same display. Lines of text were replaced with colored lines (or pixels) that represented various properties of the line of source code. For example, color coding could show where recent changes were made in the code. While this view has some similarity to the CT Desktop of Figure 2, the main objective of CT Desktop is to provide a context for spatial memory for code navigation.

Eclipse displays markers in the editor scrollbar to localize information about the code [7]. Each warning or error is marked by a line in the scrollbar at the location it appears in the source file. These markers enable programmers to navigate directly to regions of concern within the program without reading symbol names.

Similar to both the Eclipse scrollbar and SeeSoft, Aspect Browser presents the source files of a system as rectangles proportional to the file size, tiled from left to right, with color-coded lines showing the locations of search results. [10][17] Although Aspect Browser was designed to visualize search results and not to aid navigation, Shonle, Neddenriep and Griswold note that a developer can use spatial memory to track long tasks by remembering how far she has progressed in a top-down, left-to-right sweep of the results.

The Data Mountain [16] was a novel user interface for document management designed specifically to take advantage of human spatial memory (i.e., the ability to remember where you put something). Users freely arranged document thumbnails on an inclined plane textured with passive landmarks. A user study of the Data Mountain [2] demonstrated that it was an effective alternative for current web Favorites mechanisms, and that it leveraged spatial memory. The Data Mountain allowed users to informally arrange their space in a very personal way. This informality appeared to play an important role in forming spatial memory and was enabled by having the ability to view the whole space, by the spatial relationships between the thumbnails, and by the manual control of those relationships in space. Hence, the CT Desktop retains these aspects of Data Mountain's design.

The Data Mountain user study also suggested that spatial memory did in fact play a role in virtual environments. Subjects were reported to say things such as, "it's right here" and "I know it's back there", and then move directly to the location of the view. Also, when thumbnails were "turned off", users were not slower at retrieving their web pages even after 6 months of elapsed time without interacting with the system. This remarkable result emphasizes the power of spatial memory. Storage times, retrieval times, and retrieval failures were all reduced because of this aspect of spatial memory's influence.

5. Formative Evaluation

Eleven intermediate to experienced developers (as determined by an internal, well-validated screening question) were recruited for a formative evaluation of Code Thumbnails. All participants were recruited from the greater Puget Sound area, although one participant from our own company was included to replace a cancellation. Participants had an average age of 34, had been programming on average for 15 years and had used Visual Studio for an average of nine years. They worked on development teams that had nine colleagues, on average. All participants were male, which was an artifact of the gender-skewed roster of study volunteers, not by design.

Each session started with an overview description of the code base used for the study, which was a C# implementation of the Tetris game. All participants had played Tetris before and remembered the game flow. Following this discussion, each participant was provided a brief overview of Code Thumbnails. We gave participants five minutes to explore Code Thumbnails to see how the software worked, and we encouraged participants to perform global searches to see how Code Thumbnails showed search hits across the project files in the desktop window.

Once participants had successfully used and understood Code Thumbnails, they started the programming tasks. We were presented with three tasks, listed in increasing order of difficulty. The tasks were:

1. The game contains a dialog box for setting the color of each of the seven types of falling game pieces. This color defaults to gray for each type of game piece. Participants were asked to change the code so that each type of game piece defaulted to a different color.
2. In the existing implementation, a game piece always fell at a constant rate of one square per second. Participants were asked change the game so that game pieces fell faster as the player scored more points.
3. The third task was to change the game so that hitting the space key during game play caused the current figure to fall immediately as far down as it can.

Participants programmed and performed the study tasks at their own pace, with the goal of finishing all three

tasks within 75 minutes. They worked independently, though we ran two participants at a time for logistical reasons. Nine participants were able to complete all three tasks within the allotted time, one got very close to completion, and one had not completed the second task when the session concluded.

Once participants had finished the last coding task (or 75 minutes into the session), they performed a series of targeted search tasks, designed to determine how and how well they would utilize existing search and navigation features in Visual Studio with Code Thumbnails. In these tasks participants were asked to find files by name, methods by name, and methods by brief descriptions of their functions. Task times were automatically collected for these trials for a total of 18 search trials (five trials of each type, plus three practice trials which were not included in the analysis). Participants were instructed that they could find these targets using any means available, including local and global searching, scrolling through the code, or using Code Thumbnails. This was important for us to gauge whether or not participants would use Code Thumbnails to navigate if given the choice to use anything. All participant activity in Visual Studio was to be logged to a data file for later analysis, but unfortunately an error only allowed us to collect this data for five of the participants.

Next, participants performed a series of spatial memory trials divided into four sets: searching for files, both without and with visual landmarks, and searching for methods, both without and with visual landmarks. Each set consisted of five trials for a total of 20 trials. For each of the first five trials, the spatial memory quiz prompted the participant with a file name and showed him a blank border the same size as the CT Desktop. The participant's task was to click as close as possible on the location within that border to where that file had been located on the CT Desktop during the programming tasks. If a click was anywhere within the target location, the quiz program announced success and prompted with the next file name. If the click was outside the target location, the quiz program displayed a black "X" at the click location as a reminder of where previous clicks had been attempted. When the participant clicked on the correct location for the file, the next trial would begin. The second set of trials was like the first, except that the quiz program now revealed all the files' thumbnails (without file names) in the CTD window.

The third and fourth sets of spatial memory trials mimicked the CT Scrollbar and focused on finding methods by name. For the third set, the quiz program drew a long, blank rectangle centered in the window, representing the CT Scrollbar. The quiz program prompted the participant with five method names, providing the same correctness feedback as the previous trials. A click was considered "on" a method if a corresponding click in the CT Scrollbar would have caused any part of the method text to appear in the editor window. The fourth set of spatial memory trials was like the third except that the quiz program now revealed the code thumbnail (without any text labels) within the scrollbar rectangle.

For all the spatial memory trials, the quiz program recorded the time and position of each click. From these, we derived the dependent measures of the total search time, the total number of clicks, and the distance of the first click to the target.

As their final task in the session, participants filled out a questionnaire about their experiences using Code Thumbnails and added any comments or suggestions they might have.

All tasks were carried out on identical Compaq EVO 510 desktop computers with an Intel P4 2.8 GHz processor, 2 GB of RAM and 40 GB hard drive, running Windows XP Pro SP2. The computers had Dual NEC 18" flat panel monitors, side-by-side, each running at their native 1280x1024 resolution. We used Visual Studio 2005 as the software development platform.

We ran a formative evaluation because we did not have clear hypotheses about how Code Thumbnails would be used to leverage spatial memory. To examine this question, we explored several manipulations that we believed would be useful in future benchmark studies of Code Thumbnails and related techniques. For example, in our targeted search task we indicated some target methods by name and some by functional descriptions. We thought having a concrete name would make the task easier, but also wanted to see what techniques participants used to search when they were only given a general description. Likewise, for the spatial memory quiz, half the trials were presented to the participant without any landmarks at all– a true test of their spatial memory. We expected these trials to be harder, but wondered whether participants would be more successful for frequently-accessed areas of code.

6. Results

6.1. Usage during Programming Tasks

As mentioned, we logged five of the participants' activities as they performed the coding tasks. We were interested to see whether and how often the Code Thumbnails features were utilized for navigation and selection, relative to other navigation features. It was immediately clear that all our participants frequently used the Code

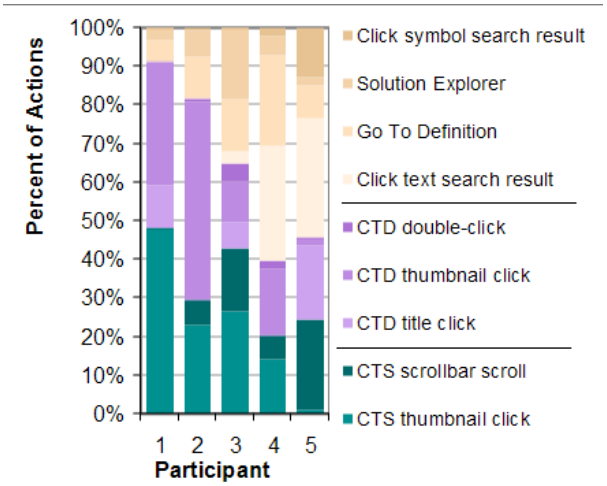

Figure 3. Percentage of navigation actions where participants used standard features (top four actions), CT Desktop (middle three), and CT Scrollbar (bottom two), for each of the five participants.

Thumbnails features for navigation, searching and selection. For a majority of the participants, Code Thumbnail activities represented by far the most frequent ways to navigate and search. For the few participants for which this was not the case, Find and Replace and Find Results were the most frequent navigation features, followed closely by both Go To Definition and Code Thumbnails. Figure 3 summarizes the five logged participants' navigation actions during the programming tasks. Both the CTS and CTD interactions are broken down into specific actions. Usage of CT varied by participant between 40% and 91% of all logged navigation activities.

Experimental observations of participants' use of the tool verify the log data. Participants appeared to quickly understand how to navigate with Code Thumbnails early in their programming tasks and continued to use them often, augmenting their use of more-familiar navigation methods. Given the time pressure of the programming tasks, the rapid adoption of Code Thumbnails was surprising and encouraging.

6.2. Targeted Search Tasks

As we expected, searching for files was significantly faster than searching for methods (9.3 v. 20.3 seconds on average), which was in turn significantly faster than searching for method descriptions (49.0 seconds on average). More interestingly, since we logged five of the participants' actions in the IDE, we were able to analyze what features they used during the targeted search trials. Because of the time pressure of the search trials, we expected users to favor tried and true search techniques. Nonetheless, the CT Desktop was used in 64% of the search trials. The next most popular way of finding targets was text search, which was used 16% of the time. The CT Scrollbar and the Solution Explorer were used 11% and 8% respectively. (The percentages do not add up to 100% as participants sometimes used multiple navigation features techniques in a trial). In short, even in a "race", participants frequently used Code Thumbnails, even when given the choice of familiar navigation features.

6.3. Spatial Memory Task Times

We used a Repeated Measures Analysis of Variance (RM-ANOVA) to analyze the time to click on the correct spatial location for files and methods and the number of misses before finding the target. Our analysis was 2×2×5, namely landmarks absent vs. present, file target type vs. method target type, and five search trials. There were significant main effects for target type, $F(1,10)=24.5$, $p<0.001$ and landmarks, $F(1, 10)=14.08$, $p=0.003$ and a significant interaction between target type and landmark availability, $F(1,10)=5.7$, $p=0.04$. No other interactions were significant.

In the spatial memory quiz, searching for files was significantly slower than searching for methods (13.7 vs. 2.5 sec, on average). Also, file searches were significantly slower when the thumbnail landmarks were not available (files: 18.9 sec without landmarks vs. 5.7 sec with landmarks, on average), as compared to method searches (2.4 vs. 0.15 seconds, respectively, on average, no significant difference). The fact that the landmarks are providing speed benefits for file searching in the CT Desktop means that the participants were beginning to build up a cognitive map of the spatial layout of their thumbnails in the overview window– a nice finding for using the tool for such a relatively short period of time. Spatial cognition research has consistently shown that mental maps of spaces build up piecemeal, anchored around landmarks [4, 11, 13, 15, 16]. We conjecture that visual landmarks were not as useful for method searches because method targets are much closer in the CTS relative to file targets in the CTD and could be found within a few clicks, even with no visual landmarks.

6.4. Spatial Memory– First Click Distance

For the spatial memory tasks we calculated the distance from the participant's first click to the target. For file targets, this distance is the pixel distance between the click and the target rectangle; for methods, this distance is the number of lines of code between the line clicked and the lines occupied by the method's code. Therefore, we separately analyzed these two target types. For the files, we used a 2×5 RM-ANOVA analysis (landmark absent vs. present, over five trial repetitions). We did

not detect a main effect for landmarks. The main effect of trial repetitions was significant, $F(4,36)=4.8$, $p=0.003$. Also, the interaction between whether landmarks were absent or present and repetitions was significant, $F(4,36)=10.6$, $p<0.001$. This can mostly be explained in terms of varying familiarity with the targets. The first two blind targets were files that the participants frequently accessed during the programming tasks; the next three blind targets were infrequently accessed files. For the first two blind search trials, the average first click pixel distance was 368 pixels; for next three blind trials, the pixel distance was 511 pixels, showing users were much closer with familiar targets than unfamiliar. This is also suggestive of the formation of spatial memory with Code Thumbnails Desktop.

For the method data, we ran a similar 2×5 RM-ANOVA analysis. Here a significant main effect for landmark presence, $F(1,9)=23.5$, $p<0.001$ was a observed, and nothing else was significant. When landmarks were absent, the average number of lines from the target method was 54.9; when present, the average dropped down to 3.6 lines away from the method, on average. This is further evidence that participants were beginning to form a cognitive map for the layout of the code within files.

6.5. Satisfaction Questionnaire

The results from the satisfaction questionnaire came out surprisingly well for a first iteration of the Code Thumbnails design. Table 1 shows the average results across all participants. Highlights included high marks for overall ease of use, learnability, overall satisfaction and preference for Code Thumbnails over the existing Visual Studio user interface.

6.6. Participant Feedback

The participants gave us many good ideas for improving the design of Code Thumbnails. Most mentioned that they wanted the method names' hover text to be wider, so that the whole name could be seen. Some thought it would be nice to be able to drag and drop the methods (i.e., edit the file) within the thumbnails. A few wanted more features in the CTD thumbnails, like highlighting code that refers to the current definition (callers, callees, field uses). Participants appreciated the search hits in the global overview, although suggested that making the search results readable in the thumbnails would useful. One participant mentioned how useful Code Thumbnails were for navigating when you did not know where you needed to go, but suggested better support for keyboard shortcuts to improve navigation speed. Most participants saw the need for multiple monitors to devote enough screen space to the CT Desktop and wanted us to think about ways to quickly bring up the Desktop and then easily send it away again. A few mentioned that it was maybe not necessary to see all of the files in a project, just those frequently accessed (what Ko, Aung and Myers call a working set). Some participants recognized that the study's code base is small and were concerned about scaling to larger projects.

7. Discussion

For a first iteration design, users found Code Thumbnails easy to learn and obviously helpful for navigation, even under time pressure. Participants used the Code Thumbnails frequently to navigate and to search for and find methods and files, even when they were told they could use any method for these tasks that they preferred. Usability issues were observed and valuable feedback

Table 1. User satisfaction questions and average responses (with standard deviations in parentheses), sorted by decreasing average response. For most questions (except as noted) a high-number response was more favorable to Code Thumbnails.

Satisfaction Question	*Avg. Response (St. Dev.)*
Learnability: The version of Visual Studio you used today was easy to learn. (Disagree=1, Agree=5)	4.6 (0.5)
Ease of Use: The version of Visual Studio you used today was easy to use. (Disagree=1, Agree=5)	4.5 (0.5)
Preference: How much would you prefer this version of VS over existing techniques for developing software? (Not at all=1, Very much=5)	4.3 (1.0)
Satisfaction: How satisfied were you with the version of VS you used for accomplishing the tasks? (Low=1, High=5)	4.1 (1.1)
Global navigation: The code thumbnails (if you used them) were useful for rapid, global navigation through the code. (Disagree=1, Agree=5)	4.0 (1.0)
Utility: How useful were the code thumbnails (if you used them) overall? (Not at all useful=1, Very useful=5).	3.8 (0.9)
Divided attention: How much did you have to divide your attention between the code thumbnails (if you used them) and the code displayed? (High=1, Low=5)	3.5 (0.8)
Local navigation: The code thumbnails (if you used them) were useful for minute, local movement through the code. (Disagree=1, Agree=5)	3.3 (1.5)
Frustration level: How discouraged, irritated, stressed or annoyed did you feel while completing the programming tasks? (Low=1, High=5)	1.7 (0.8) (*Note: lower is better*)

for future designs was obtained. We note that this study focuses on professional developers navigating unfamiliar code. How these navigation patterns differ when the code is familiar and how code familiarity affects the utility of Code Thumbnails remain open questions. We hope to address these questions in a future longitudinal field study.

For the spatial memory test, several different kinds of data were explored with the hope that they would reveal something about Code Thumbnails supporting better spatial memory. For files, participants could leverage the file outlines as landmarks which improved search speed in the overview window when available. We think that this implies that our participants were beginning to build up a cognitive map for the layout of files in the overview window in just over an hour of usage. This hypothesis needs to be followed up with a more careful analysis and benchmark against a situation in which code thumbnails are not available.

For methods, having the outline of the code visible significantly improved participants" ability to click near the target. We suspect the participants were learning something about the distinctive shape of the code within files in order for this effect to be as strong as it was, and further tests of this are certainly needed.

8. Conclusions

With our formative evaluation we were able to show that users were quickly able to learn to navigate using thumbnail images of the code. They enjoyed this style of navigation, as measured both by their navigation choices during programming and search tasks and by their subjective ratings. We have some initial evidence that they were forming a spatial memory of the code, although the evidence could be strengthened by a longer study with more participants. This leaves two important questions unanswered: Is a developer more efficient (spending less time) while navigating via spatial memory rather than traditional means? And, does navigating via spatial memory free up cognitive resources that can be applied to programming tasks– that is, does navigating via spatial memory increase overall programming productivity? We intend to address these two questions with a follow-on study.

9. References

[1] Balzer, M., Deussen, O., and Lewerentz, C. (2005), "Voronoi Treemaps for the visualization of software metrics," In *Proc. ACM Symposium on Software Visualization* 2005.

[2] Czerwinski, M., van Dantzich, M., Robertson, G.G. & Hoffman, H. (1999). The contribution of thumbnail image, mouse-over text and spatial location memory to web page retrieval in 3D. In *Proc. of Interact '99*, IOS press, pp. 163-170.

[3] Darken, R. & Sibert, J. L. (1993). A Toolset for Navigation in Virtual Environments. In *Proc. of UIST'93*, ACM.

[4] Darken, R. & Sibert, J. L. (1996). Navigating large virtual spaces. *International Journal of Human-Computer Interaction*, 8, 49-72.

[5] DeLine, R., Khella, A., Czerwinski, M., and Roberson, G. (2005), "Towards understanding programs through wear-based filtering," In *Proc. Symp. on Software Visualization 2005*.

[6] DeLine, R. "Staying oriented with Software Terrain Maps (2005)," In *Proc. of the Workshop on Visual Languages and Computation* 2005.

[7] Eclipse home page, http://www.eclipse.org.

[8] Ehret, B.D. (2002). Learning where to look: Location learning in graphical user interfaces.

[9] Eick, S.G., Steffen, J.L., Sumner, E.E.Jr. (1992), "Seesoft-A Tool for Visualizing Line Oriented Software Statistics", *IEEE Trans. on Software Engineering, 18:11*.

[10] Griswold, W., Yuan, J., and Kato, Y. "Exploiting the map metaphor in a tool for software evolution," *Proc. International Conference on Software Engineering*, 2001.

[11] Jones, W. & Dumais, S. (1986). The spatial metaphor for user interfaces: Experimental tests of reference by location versus name. *ACM Transactions of Office Information Systems, 4*, pp. 42-63.

[12] Ko, A.J., Aung, H.H., Myers, B.A. (2005), "Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks", *Proc. International Conference on Software Engineering 2005*.

[13] Lynch, K. (1960). *The Image of the City*. Cambridge, Massachusetts: The MIT Press.

[14] Mander, R., Salomon, G. & Wong, Y.Y. (1992). A „Pile" Metaphor for Supporting Casual Organization of Information. *Proc. of CHI'92*, 627-634.

[15] Passini, R. (1984). *Wayfinding in Architecture.* New York: Van Nostrand Reinhold.

[16] Robertson, G. , Czerwinski, M., Larson, K., Robbins, D., Thiel, D. & van Dantzich, M. (1998), Data Mountain: Using Spatial Memory for Document Management, In *Proc. of UIST '98*, pp. 153-162, ACM.

[17] Shonle, M., Neddenriep, J. and Griswold, W., "AspectBrowser for Eclipse: A case study in plug-in retargeting," *Proc. of the OOPSLA workshop on eclipse technology eXchange*, 2004.

[18] Thorndyke, P. W. & Hayes-Roth, B. (1982). Differences in Spatial Knowledge Acquired from Maps and Navigation, *Cognitive Psychology, 14*, pp. 560-589.

[19] Vinson, N.G. (1999). Design guidelines for landmarks to support navigation in virtual environments, *Proc. of CHI'99*, p.278-285.