CodeMaps - Smart Browsing of Code Bases

Henning Weiss

Kongens Lyngby 2013
IMM-M.Sc.-2013-30

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2013-30

Contents

Chapter 1

Introduction

The size and scope of software projects seems to grow constantly. Projects have evolved from small one man efforts with a couple of modules to huge undertakings with hundreds (or even thousands) of contributors and millions of lines of code. When working on a large scale projects, it is often necessary to read and understand large portions of the source code before any meaningful impact can be made on the project.

With the size of codebases growing, the developer's ability to navigate source code must improve as well. The structuring and design of a code base can help significantly, allowing the project to be divided into smaller modules. Documentation delivered along with the source can also aid the developer's understanding. In practice, documentation has been proven to be problematic, as it becomes outdated quickly and developers tend not to update it when making changes to the source code. This usually leaves the source code as only mean of getting an accurate, up-to-date understanding of a software project.

Most modern IDEs[1] ship with multiple tools that use syntactical and structural elements of the source code to enhance navigation and presentation (like code folding, class browsers, etc.). Although they augment the interface with useful information, they still use a "file centric" approach. Code Bubbles is an Eclipse

[1] Integrated development environments

plugin that uses a novel, code centric approach. It presents functions as atomic elements that can be freely arranged on a two-dimensional canvas.

In this thesis I will describe my efforts of creating CodeMap, a Code Bubbles-like code reading tool for the Android tablet platform. The intention behind CodeMap is, as the name implies, to give developers a method of creating a "map" of a code base. Geographical maps contain a lot of information, but by starting at one point and exploring paths from it, a large territory can be gradually explored, without becoming overwhelmed by the amount of information. I will examine whether this method can be implemented as a tool to improve a developer's ability to navigate and understand large code bases. Additionally I will examine how the form factor and input methods of tablets can be used optimally and how they impact the way users interact with the source code.

1.1 Personal Motivation

My motivation for working on this project was the result of trying to extending a plugin of the QT-Creator IDE. That project posed the problem of having to understand functions that contained deeply nested function calls. This lead me to write down notes with pen and paper to keep track of call stacks, as well as noting which module each function is located in and a short description of its purpose. What I needed was similar to the "breadcrumbs"[2] interface, but supporting multiple traces and allowing me to add annotations.

An extensive search for a suited tool only came up with Code Bubbles. Code Bubbles seems to be abandoned by its creators, as they started pursuing other projects. Conceptually it seems like a good approach, but the execution is incomplete, especially with regards to controlling the interface with a mouse. This gave me the idea that a touch interface might result in a more satisfying user experience. Tablets seem to become more ubiquitous and they normally have a touch capable screen. This lead me to conclude that creating a similar user experience as presented in Code Bubbles might be worth examining.

1.2 Structure of report

This report is structured in the following way.

[2]Breadcrumbs is an interface component that tracks a users navigation through multiple documents.

Section 2, entitled “Background”, describes some of the existing solutions for navigating and understanding code bases. In addition, I will briefly discuss the cognition behind reading source code, as I believe it influenced my design decisions.

Section 3 describes the ideal design of CodeMap. The following section will describe the current state of my implementation.

Section 5, entitled “Evaluation”, will describe the design of the usability tests I did.

Section 6, entitled “Discussion and Future work”, will synthesize the knowledge gained through the usability tests and how future work could improve on the current state of CodeMap, as well as other related topics worth studying.

Finally I will conclude the report. The appendix contains a more detailed documentation of the usability tests.

Chapter 2

Background

In this section I will describe some of existing tools that aid developers in navigating code bases.

2.1 Integrated Development Environments

2.1.1 Smalltalk code browser

The smalltalk code browser [III], displayed in figure 2.1, is considered by some a milestone in developer productivity. It has many of the features one would expect from a modern IDE, with its most novel feature being a hierarchical code browser. This browser allows filtering and browsing of the entire smalltalk code base by submodules, classes, functions and variables. It also provides modern refactoring capabilities, like the ability to rename classes, functions and variables. The smalltalk browser also features an integrated debugger that is similar to graphical debuggers shipped with modern IDEs.

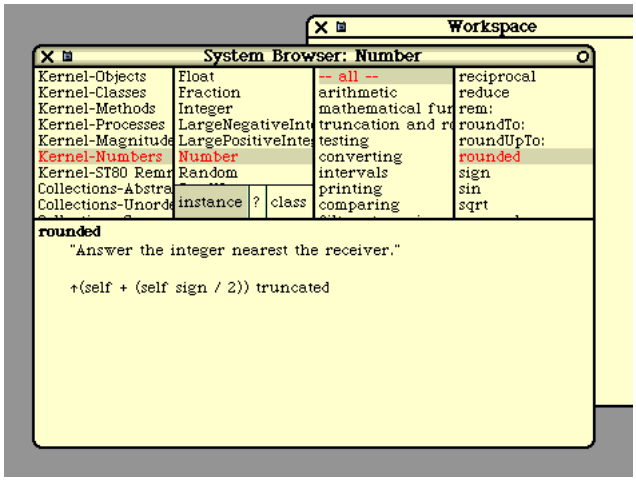

Figure 2.1: Smalltalk code browser

2.1.2 Modern IDEs

Modern IDEs, like Eclipse [Foub], Visual Studio [Micd], NetBeans [Cor] or IntelliJ [Jet], contain many additional features. Declarations of, and references to symbols can be found with a press of a button. Syntactically search, integration with version control, advanced refactoring functionality etc. make these tools very powerful. Here I will discuss features relevant for reading and navigating code found in most modern IDEs.

Code folding allows the developer to hide elements of a file. This is usually a syntax aware feature, making it possible to fold entire constructs like blocks of documentation and entire functions with one click. This allows several functions that aren't adjacent to be on the screen at the same time.

It is also possible to view several files on the screen at once by using the "split screen" feature. This allows developers to display multiple files in the same editor window simultaneously. The orientation of a "split" and the size of the resulting window can usually be adjusted manually. When the developer tries to use this feature to display multiple functions, it can be time consuming, especially when each split's size needs to be adjusted to optimize the amount of elements to be displayed. Multiple, large monitors with a high resolution provide more screen space, allowing additional modules to be displayed.

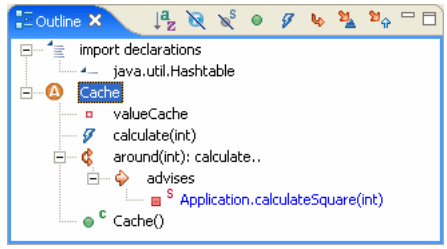

Figure 2.2: Eclipse outline browser

Navigation through source code is aided by the "bread crumbs"[1] feature. It displays the list of functions a developer took to reach a particular piece of code. Only one instance of this "trace" is supported at one time. When a user navigates through a new set of functions, the trace is lost. It is also not possible to annotate this call path. It is also possible to "undo" the navigation to return to starting function.

The component usually called "package explorer" or "outline" gives a structured overview of the source code. It displays a hierarchical view of the contents of the current file, including classes, functions and variables. This allows a user to browse a code base based on the source code's structure. Eclipse's "outline browser" is depicted in figure 2.2.

2.1.3 Code Bubbles

Code Bubbles [Bra], displayed in figure 2.3, is a novel interface for code base browsing. It is a plug-in for Eclipse. The main interface consists of a large, panable, 2-dimensional workspace. On this workspace, so called "bubbles" can be placed. These bubbles can contain the source code of a function, a note, a special icon or several other items of content. Bubbles can be re-arranged through drag-and-drop. By arranging the bubbles in meaningful way and adding annotations it is possible to document source code.

A paper was published by the authors [AB10], in which they describe their findings. 23 people where selected to test Code Bubbles. The users where given a number of tasks to perform. They measured the time used by the participants to solve the tasks and compared them to users solving the same tasks in a "traditional" IDE. The results where positive, and they claim that users where

[1]The name is inspired by the story of Hans and Gretel. In order not to get lost in the woods, they used a trail of breadcrumbs to help find their way back.

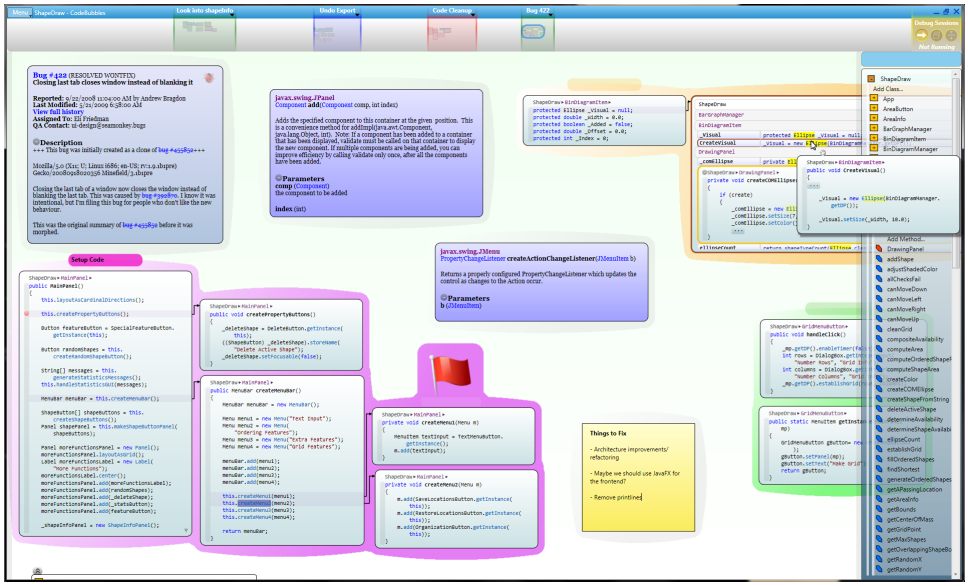

Figure 2.3: Code Bubbles

faster at navigating and reading source code and better at resuming interrupted tasks. It seems like the authors have discontinued development on Code Bubbles and started working Microsoft “Code Space”. The focus of Code Space is collaboration and being used as presenter software with a beamer. Microsoft has developed a plug-in for Microsoft Visual Studio called “Code Canvas” [Mica] that is very similar to Code Bubbles.

2.1.4 Structured Editors

There have been attempts at making editors more context aware. Structured editors let the user navigate the AST (abstract syntax tree) of a program. An advantage to this approach is that all of the non-semantic information can be omitted. Spaces and indentation are thereby not part of the source code, but part of the presentation to the user. This could potentially improve the speed of navigation, as spaces and empty newlines can be skipped over by the editor when the user is navigating through the code.

It also allows the editor to determine what constructs make syntactically sense to insert at a given cursor position. This can be used to make suggestions to

Figure 2.4: TouchDevelop for loop editing

the user, like adding semicolons or closing parenthesizes and braces.

Microsoft's Touch Develop [Micc] is a platform that allows users to write source code on their phone. Context aware lists replace the need for editing a file with the keyboard. Users can select, for example, constructs like "for" loops and the interface will present a dialog to prompt for upper and lower limits of the loop.

2.2 Working Sets

Eclipse Mylin [Foua] is an issue tracker tool for Eclipse. Like other issue trackers, it allows its users to create issue tickets. Tickets usually represent a feature or bug request in a software engineering project. Mylin allows users to associate source files to tickets. The idea is that the user specifies an active ticket before he starts working. This signals that the user is trying to solve the issue(s) associated with that ticket. When marking a ticket as inactive, all files that are currently open will be closed and associated to that ticket. This allows users to rapidly change between sets of files and work on multiple issues conveniently.

The tickets are also synchronized with a web server, making this a feasible way of communicating bug reports.

Code Bubbles has a similar concept with task shelf. This allows the user to save a segment of bubbles and re-open them later. Bubbles are divided into working sets by proximity. When bubbles are close to each other, they are marked in the same color. When creating a task shelf, the user simply selects the relevant working sets to be included. It is also possible to export task shelfs and send them to other users.

2.3 Annotations

Annotations are used to add additional documentation. Comments in source code are often employed to communicate the purpose of a function or a class. This is reasonable, as the comments can be written "inline" inside the file, along with those components. Information about structural connections or working sets is harder to convey, as there is no predefined place for it.

Code Bubbles allows annotation of source code by creating "bubbles" containing Javadoc documentation or notes written by the developer. By allowing the developer to arrange these notes on the same level as the the source code components, they can be used to annotate structural elements, like entire modules.

2.4 Debugging

The Gnu data display debugger (DDD) [GNU] is a visual debugger. The user can re-arrange a graphical representation of variables and data structures on a two-dimensional plane. Pointers to other data structures can be dereferenced and displayed as well. This makes it possible to get an overview of many variables or structures, as well as visualizing algorithms like, for example, operations on a linked list. An example is displayed in figure 2.5.

Code Bubbles features a debug mode. It displays the source code being traced along with watched variables in bubbles. When "stepping into" functions, new bubbles will be created automatically. Debugger Canvas [Micb] is a similar plug-in for Microsoft Visual Studio.

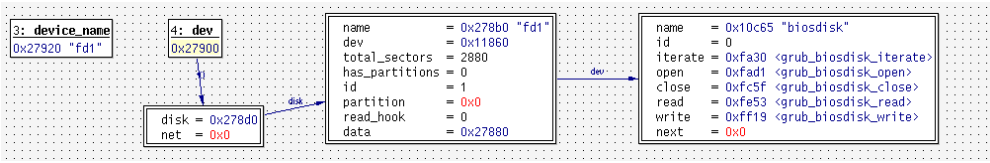

Figure 2.5: Data display of Gnu DDD

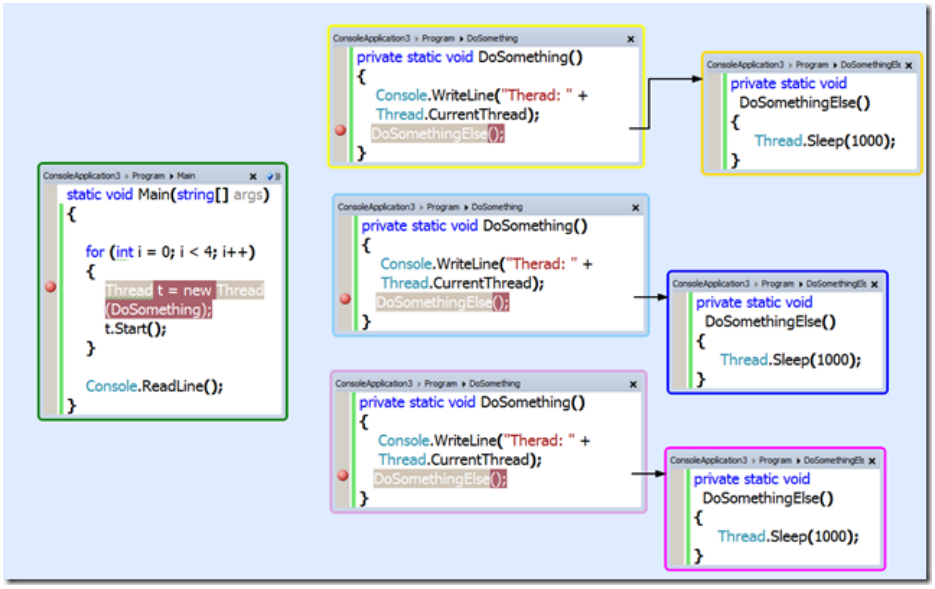

Figure 2.6: Thread debugging with Debug Canvas

Chapter 3

Analysis and Design

In this section I will describe the design for a code reading tool inspired by Code Bubbles.

3.1 Interface

User interfaces (UI) for tablets are different than interfaces for personal computers. The most important differences between the two is that the (slate) tablet does not have a physical keyboard and mouse, but instead relies on the touch sensitive screen to receive its input. Most of the UI of Code Bubbles can be controlled by the mouse. This makes it possible to adapt the original interface with minimal changes.

3.1.1 Workspace

Text editors use horizontal scrolling to allow the user to view files that are larger than the screen. The Code Bubbles workspace allows scrolling in two dimensions, making it possible to navigate amongst multiple open files. Tablets

tend to have smaller displays than workstations. This makes it important to design an UI that makes effective use of the screen. Having a rigid structure that is file oriented might not use the screen space optimally. Instead, the structure of the code should be taken into account to optimize how much information can be presented on the screen.

Code Bubbles allows the user to arrange fragments of code arbitrarily on this workspace. Presenting source code in this way allows the user greater control over the presentation. It allows users to create their own structure and take advantage of spacial cognition to remember where code is located. The user can indicate the relationship of bubbles by the vicinity of them. By making this workspace persistent, users can find a previously read code fragment by remembering "where" it is, instead of having to remember its name. By gradually exploring the code base in this fashion, users can create a "map" of the source code.

3.1.2 Working sets

Code Bubbles has a large virtual workspace in which the user places all open bubbles. It is not clear to me what the advantages of this solution are. Therefor I suggest using multiple workspaces, each containing a working set. By displaying so called "tabs" in the menu bar, users can easily get an overview of open workspaces, as well as changing quickly between them. By separating the workspaces, some performance increase might be gained, as well as simplifying actions to manage working sets.

3.1.3 Fragments

Code Bubbles has small, post-it style notes that users can add at arbitrary positions in the workspace. It is also possible to add small icons to bubbles, allowing the user to call attention to a given area. Bubbles can contain other types of contents, like JavaDoc or Doxygen documentation as well. By giving the user annotations that can manipulated the same way as the source code, users can document structural elements of the source code. Instead of using the term "bubbles", I will refer to these elements as fragments in my application internally. I will use the terms fragment and bubble interchangeably.

3.1.4 Code Browser

It is useful to have a structured overview of a code base. I propose that this should be provided by a code explorer, similar to the one found in Eclipse or Visual Studio. This outline allows the user to select the relevant functions, which are then opened as fragments on the workspace. Function calls inside fragments are displayed as links. Clicking on the links will open a fragment containing the declaration of that function. To give more structure to the explored code, a line is drawn between the fragment and the fragment containing a function called by the first fragment. When a user tries to open a fragment that already exists in the workspace, the code browser should indicate so. By clicking on a function that is already opened, that function should be focused and if multiple fragments of that function are opened the focus should cycle among them. This browser should also have search functionality to enable the user to quickly navigate the source code.

3.1.5 Zooming and Overview

The workspace should be zoomable. By reducing the size of the fragments, more information can be presented on the screen at once. Progressive zoom refers to gradually removing and abstracting information, similar to a digital maps application like Google Maps. When the workspace is zoomed out to a certain degree, text will become too small to be readable. By replacing a fragment's source code with the title of the function, useless information is removed. This will give an overview of the workspace, displaying where functions are located on the workspace and what their relation is.

3.2 Code Analysis

Before the source code can be presented in a meaningful, structured way, it must be parsed and analyzed to provide useful meta information. The structure of programming languages needs to be take into account.

For C (and most other procedural languages), the code browser simply presents the file system, until a file is selected. Then, all declarations contained in that file should be displayed. When the user clicks on a declaration, the source code of the definition should be displayed. In this fragment all the function calls and global variables should be turned into clickable links, allowing navigation

to their declaration in similar fashion. This requires a basic "use-def" and "def-use" analysis of the source code. The "use-def" chains should be able to be queried on a file basis. The "def-use" chains should be able to be queried on a function and variable basis. Function pointers and macros are not covered by this analysis.

Object oriented, statically typed languages (like Java and C++) present additional challenges. Class hierarchies make it more difficult to find the declaration of a function being called. This is a well researched area and several papers have been published on the subject [DFB96] [JD95]. Advanced language features, like C++ templates, are even more difficult to analyze. An existing static analysis framework for C++ is called Pivot [Uni], which aims to provide a complete set of analysis, excluding macro definitions. Eclipse features a complete code analysis framework for Java. Having those frameworks available would make it possible to determine the correct declarations of classes, overloaded functions and templates.

Functional programming languages (like Lisp or SML) are simple to represent in the CodeMap metaphor, as the most central construct of these languages are functions. Emacs Lisp has a global name space and a regular syntax, making it easy to determine the correct definitions. Other languages provide similar difficulties discussed above.

3.3 Version Control

Software tends to undergo rapid development. Version control (VC) makes it easy to stay current with updates to a project. By integrating VC directly into the application, it enables the user to easily transfer source code onto the device and update it regularly.

The synchronizer module should be modular enough to allow many different VC systems. Among the most popular are CVS, SVN, Mercurial and Git. All of them have similar concepts, like having an URL to specify location of source code repository. Actions like updating and committing source code are also similar. This makes it easy to create a single interface to interact with all of these systems.

3.4 Collaboration

CodeMap might be useful in a collaborative effort. Documentation and bug reports could be created by sharing workspaces. Users could simply export an entire workspace into a portable format, like an image file or PDF, and send it to other developers. It might be more useful to share workspaces in a fashion that allows 2-way synchronization, allowing users to collaborate on the same workspaces. It might be also be convenient to associate workspaces with issues in a bug tracker, similar to Eclipse Mylin. A typical usage scenario could involve one user opening relevant code fragments and adding annotations. This workspace could be associated to an issue. A developer who later wants to fix the bug opens the workspace to get an overview of the issue.

Instead of sending these workspaces by email or creating a dedicated server system, CodeMap could use existing solutions. It might be easiest to "piggy-back" on top of a version control system, as many software projects make use of them already. This would also give the benefit of making it easy receive updates and merging conflicting changes. The only requirement to make full use of this is that the state of CodeMap is saved in plain text files.

3.5 Web Interface

It could be useful to have a web interface for CodeMap. By allowing users to access the application with a browser they don't have to install or download source code to their machine. By integrating with Github [Git] or similar sites, it might be possible to augment the existing interface, making it easy for users to try a Code Bubbles like interface.

Chapter 4

Implementation

In this section I will describe the current state of the implementation of CodeMap. All development was done on a Samsung Galaxy Note 10.1 tablet, running Android. The project is written in Java. I have only tested browsing and reading C projects.

4.1 Code Analysis

My implementation uses cscope [Ste] for doing the code analysis. Cscope is a command line tool that can find declarations and references of functions, global variables and macros in a code base. It indexes them into a database to speed up searching. Although it is designed for C, it can also index C++ and Java, and can be expanded to support additional languages.

Cscope's command line UI requires libncurses [FSF]. The ncurses library is currently not available on Android, making it necessary to remove this dependency. This allowed me to compile cscope on Android using the Android NDK (native development kit). The executable is embedded into CodeMap's ".pkg" [1] file. Licensing issues might prevent the distribution of this Pkg on Android Play.

[1] Pkg is the standard format Android uses to distribute applications.

Instead of shipping the executable of cscope embedded in the package, the executable could be downloaded from a remote server the first time the application is started.

I created a Java wrapper for cscope. The wrapper can find declarations, references and list all declarations. All of these actions can be restricted to a single file to speed up search. After invoking cscope with the proper arguments, the wrapper parses the command line output of cscope. Cscope was designed to be used in conjunction with other tools, making the output trivial to parse. This allows the wrapper to map a symbol name to a line in a file.

Cscope only provides the beginning line of symbol entry. To retrieve an entire function from a file, the end position of the function must be known. My solution is to determine the next symbol in the file. This symbol usually indicates the end of the previous declaration, resulting in the function declaration. By removing trailing new lines and comments, the complete declaration of the function can usually be found reliably. The only problems I experienced with this approach are macro definitions in the middle of functions. Fortunately they can be detected easily by their "#" suffix and ignored. An alternative solution to find the end of a function would be to parse the source code. This might not result in a robust solution, as parsing poses additional challenges, like comments.

The wrapper described above is capable of proving the required information to the user interface. All actions, with one exception, are fast enough to not give a noticeable delay in the user interface. Finding the declaration of a function, without knowing which file it is contained, can cause noticeable delays. Cscope generates a indexing database to speed up this search. For the Linux kernel, which is around 15 million lines of code, a 320 megabyte database is created. On the Samsung Galaxy Note 10.1 tablet it takes approximately 6 seconds to find a declaration in that database. For code bases smaller than 500.000 lines of code, the search is almost instantaneous.

4.2 Version Control

Version control (VC) is implemented to give the user a convenient method to transfer source code to their Android device. The VC functionality of CodeMap is designed to be modular, making it easy to add additional protocols. Besides VC, source code commonly distributed as "tarballs" [2]. Features for fetching and unpacking of those archives is already provided by Android. The synchronization infrastructure is flexible enough to support this behavior as well.

[2] Tarballs are archive files that are packaged with the tar utility and compressed.

Currently CodeMap only has a Git synchronizer and allows importing source code from the external storage. The Git synchronizer is implemented using the Java library jGit [Fouc]. It provides a full client implementation of Git. The user can specify a Git URL and CodeMap is able to fetch and index the source code. The synchronizer also allows the refreshing ("pulling") of new changes to the source code.

4.3 Interface

The entire interface has the standard Model-View-Controller architecture and uses the Android widgets as primitives. I will describe the different components of the interface in the order a user would typically be exposed to them.

4.3.1 Project Browser

The first screen the user is presented contains the project list (figure 4.1). Here the user can add, remove, edit and update projects. By long-clicking on an existing item, the user can do project specific actions. By clicking on the "plus" button in the action bar, users can add new projects. The user specifies the name of the project and the URL of a remote git repository (figure 4.2). After clicking "ok", the project is fetched from the server into a local git repository. When a project entry is selected, the "Code Map" view will be shown.

4.3.2 Code Map

Code Map (depicted in figure 4.3) is the central component of the application. It is the canvas on which the function fragments are displayed. It is implemented with an "AbsoluteLayout". This Android layout allows the specification of an absolute position of each widget and can expand to any arbitrary size. A "GestureListener" allows user to interact with the canvas through touch gestures like scrolling, dragging fragments and multi- touch zoom. This provides the user with an illusion of a large workspace.

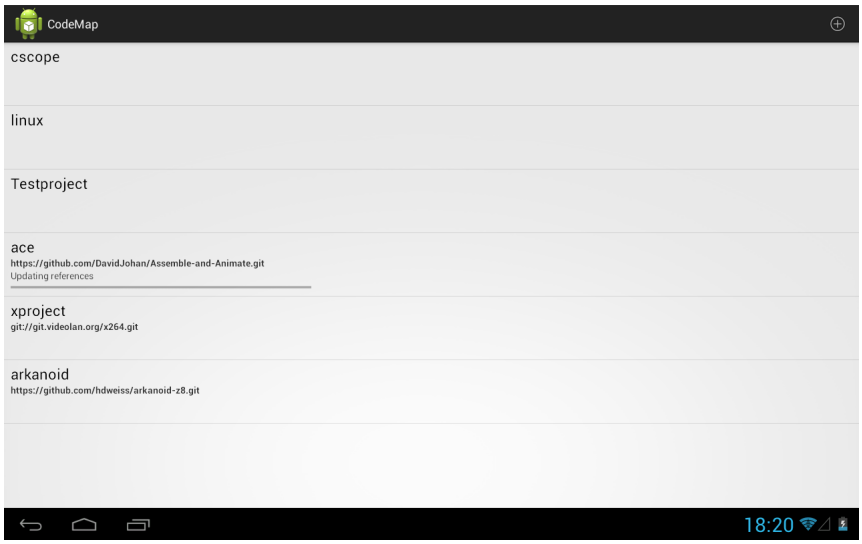

Figure 4.1: Project Browser. Project "ace" is being updated

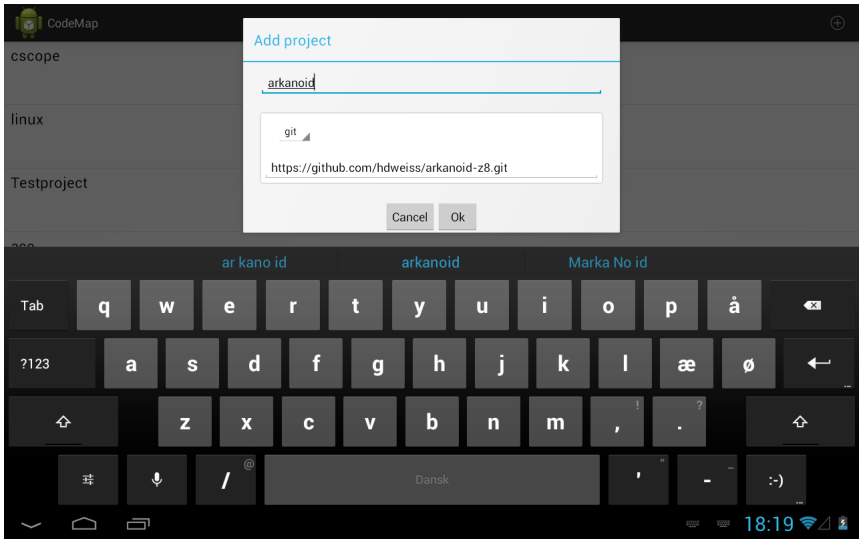

Figure 4.2: Adding project

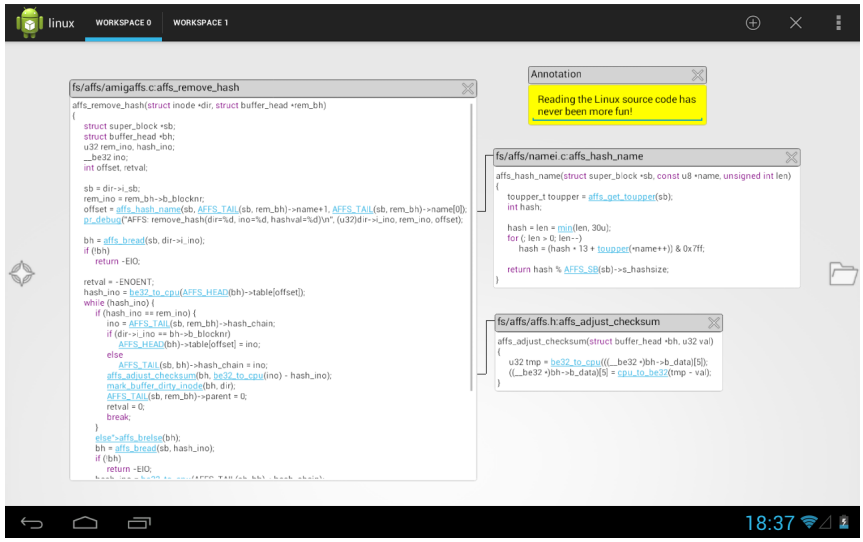

Figure 4.3: Map view

4.3.3 Code Browser

The Code browser can be accessed through the sliding drawer on the right (figure 4.4). It contains a hierarchical, expandable tree browser of the directories and files in the project. When clicking on a source file, all symbols contained in it are displayed. This makes the code browser behave similar to Eclipses "package explorer". When clicking on a symbol, a fragments is created on the code map containing its source code. On the top of the code browser is a search bar, allowing the user to search for symbols contained in the project.

By clicking on a symbol in the code browser that is already opened on the map, the symbol will be focused. When multiple fragments of the same symbol are opened, the focus will cycle through the opened fragments on each click. The number of fragments that are opened is displayed along with each symbols name. By long-clicking on a file, its entire contents is opened in a fragment.

4.3.4 Workspaces

The horizontal bar at the top of the screen is called "Action bar" in Android terminology. This bar contains the name of the current projects, tabs and

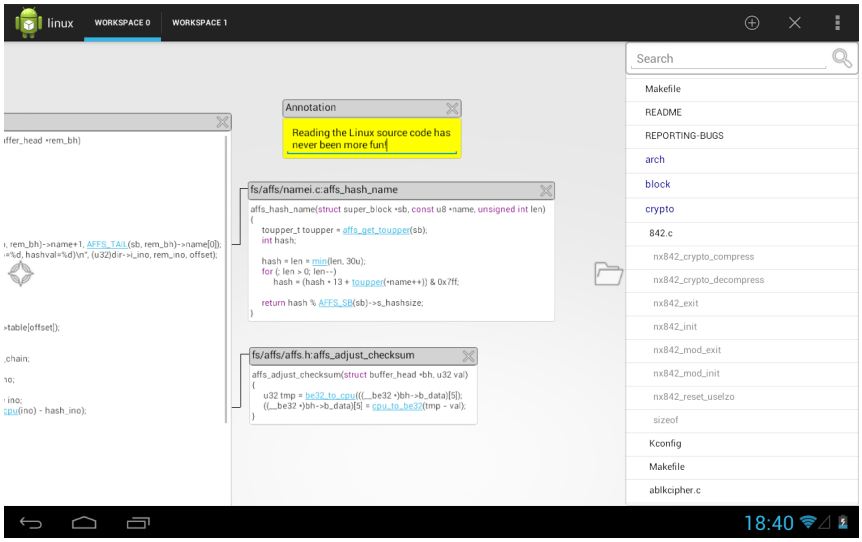

Figure 4.4: Code Browser

buttons. In figure 4.3, two workspaces are open, each represented as a tab. The "plus" and the "cross" button on the right of the bar allow opening and closing of tabs. Each workspace has its own set of opened fragments and its own state. When the user presses the back button, he will be returned to the project browser screen. All state of the opened code maps (opened fragments and their positions, scroll position and zoom) will be saved. The entire state is serialized and written to the applications data directory. Next time the user opens that project, it will be returned to its last state.

To simplify navigation, CodeMap offers a "workspace browser". It is accessed through the sliding drawer on the left side of the screen (figure 4.5). This browser contains, similar to the code browser, a hierarchical, collapsible tree of all workspaces associated with the project and the fragments they contain. Clicking a fragment's name in the browser centers it on the screen.

4.3.5 Fragments

Fragments (or bubbles) are the main component of the canvas. My implementation supports fragments containing source code, annotations or images. Image fragments are currently not enabled in the UI. Each fragment has a "title bar" on the top, containing its name and a button to close it. Fragments can be

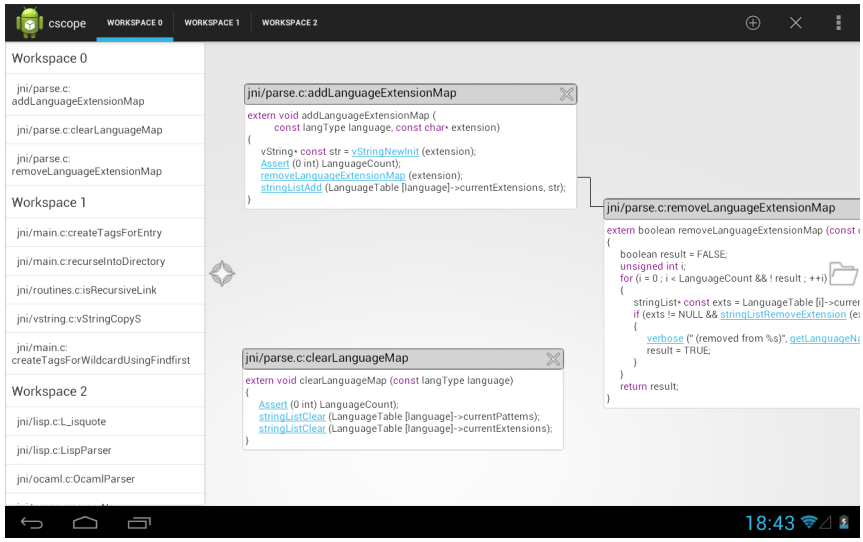

Figure 4.5: Workspace Browser

moved freely around the canvas by dragging their title bar.

The source fragments are essentially HTML displays containing marked up source code. The source code is syntax highlighted and function calls are converted to clickable links. When the links are clicked, CodeMap will try to find the declaration of the clicked function and display it in a new fragment. The newly created fragment is placed to the right of its parent and a line is drawn between the two. The line is supposed to indicate the relationship between the two fragments. When clicking on links, only the function name is used to find its declaration. This can result in cscope returning multiple matches. The user is then prompted with a pop-up menu to select the correct symbol. A more powerful analysis tool would be able to infer the correct symbol without user interaction.

When fragments are added to the canvas they should not overlap. I have implemented a simple algorithm that tries to find collisions of any fragments contained in a workspace. When a fragment is found that overlaps with the newly placed fragment, the inflicting fragment is "pushed away". This means that it will be moved in the direction that, with minimal effort, frees up the space that the newly placed fragment will occupy. This is done recursively, to prevent a "pushed" fragment to overlap with other fragments, until no further moves are necessary. This allows users to drag fragments into each other, causing the po-

tentially overlapping fragment(s) to be "pushed out of the way". This simple collision detection is sufficiently fast not to cause any noticeable delay in practice. If performance issues arise, the workspace could be split into "cells". These cells could be checked for collisions individually, causing the algorithm to skip unnecessary checks. In its current state, the collision detection will sometimes misplace newly added fragments. The way Android calculates the size of widgets makes it sometimes difficult to determine when the collision checks should occur.

4.3.6 Zoom

Android's AbsoluteLayout has no reliable way of zooming the entire layout [3] and all of its widgets. I have attempted multiple solutions to implement zooming, like modifying the layout algorithm of all layouts to change their size individually.

The solution that resulted in the most satisfying behavior was changing the "onDraw()" method of the canvas. This allows me to scale the entire canvas after it has been drawn (displayed in figure 4.6). The problem with this attempt is that Android's widget toolkit receives touch events that map to the un-zoomed canvas. This prevents users from interacting with the fragments the way they expect. A method called "onTouchEvent()" can be overloaded to allow a translation of the touch events to the drawn canvas. I have not been able to get the translation working properly at the time of writing.

[3] Android layouts are containers that can hold widgets and other layouts.

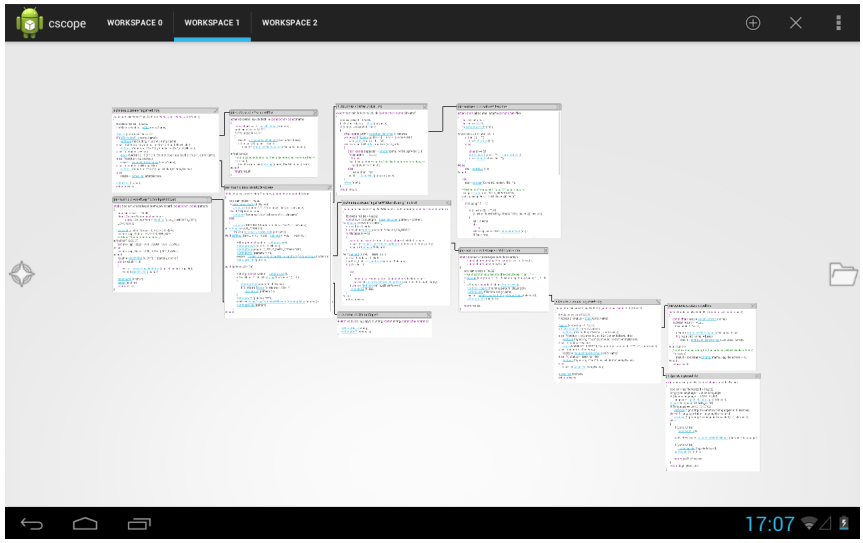

Figure 4.6: Zoomed out workspace

Chapter 5

Evaluation

The core contribution of CodeMap is a code reading interface for tablets. This makes it important to test how users interact with the application. I have chosen usability testing as my main method of evaluation. Usability tests can expose flaws in the UI and give insights into how users would use the application under normal circumstances. In this section I will describe the design of the test scenario, summarize the results I gotten through testing and discuss the the most important suggestions participants made.

5.1 Usability Test setup

Each participant was provided with a Samsung Galaxy Note 10.1. This tablet has a 10.1 inch screen with a 1280x800 resolution, running Android 4.0 (Jelly-bean). After a short introduction to the project and to usability testing, I asked participants to fill out a pre-test questionnaire. This form had some basic questions about the participant and their background with computer programming. Then the users where guided through the test scenario. Afterwards they where asked to fill out a post-test questionnaire. This questionnaire mostly consisted of fields for participants to give feedback.

The test scenario was designed to cover common tasks and most interface com-

ponents in CodeMap. In the paper from Virzi, R.A [Vir92] it is argued that four to five users will provide around 80% of the insights gained through a test scenario. The value of the contributions of more participants seems to be drastically reduced.

Creating a scenario that evaluates user efficiency in a quantitative manner is difficult to design. Test participants have different skill sets and preferences when it comes to reading source code. It would be possible to ask participants to try completing a task with both their preferred IDE and CodeMap. Both tasks could be timed to discover speed advantages. However, the tasks might be difficult to design to be comparable but not giving users an advantage of having solved a similar task. Having reliable numbers might also require a larger set of participants. I have chosen to evaluate CodeMap in a more qualitative manner.

The test scenario I have designed consisted of five tasks for the users to complete. I encouraged users to "think out loud" while completing the tasks. I also tried taking notes on notable things users did or said. Occasionally I asked users questions like "Is this what you expected?" and "Why did you do what you just did?". These questions had to be worded carefully, as they otherwise volunteer information or suggest behaviors.

5.2 Usability Test Results

I have interviewed five people. All of them had some form of computer science background and experience with the C programming language. The tablet was pre- loaded with the source code of cscope and Linux. Participants where given five tasks with regards to the cscope code base (see appendix A). The filled out pre and post questionnaires, along with my notes, can be found in appendix B.

All users where able to complete every tasks in the test scenario. They where also able to find and use all components of the UI. The first participant (Thomas) was given a version of CodeMap that didn't support workspaces. The implementation of this feature was partially a result of his input. Thomas also suggested implementing a "minimap" to improve navigation. This idea was eventually turned into the workspace browser.

Further tests allowed me to discover bugs as well as usability issues with the interface. This enabled me to improve the user experience gradually. In the post-questionnaire I asked users to rate their general impression ("How do you feel about CodeMap overall?") and willingness to use CodeMap ("Considering that the issues are ironed out, would you find CodeMap a useful addition to

your daily coding routine?") on a scale from one to five. The scoring was (in chronological order) 2, 4, 4, 4, 4 for the former and 2, 2, 4, 4, 4 for the former. This might indicate that users opinion was improved somewhat by the improvement of the application.

5.2.1 Code Map

Most users where familiar with the Android platform and touch interfaces. My application tries to follow many of the Android UI guidelines [Gooa], making the canvas of the workspace behave similar to other Android applications (in particular the Google Maps application). It was clear to all users that the presented canvas was "pannable" by touching and dragging it. It was also clear that fragments could be re- arranged freely on the canvas by dragging their title bar. Three out of the five users also discovered the multi-touch zoom functionality. Those users found it helpful, but regretted that it was not fully working (see section 4.3.6).

The biggest issue with regards to navigation on the canvas was that panning only works when the canvas itself is touched. When the contents of a fragment is panned, it will result in interacting with the fragment instead of the canvas. Currently, function fragments are not allowed to expand to be larger than the devices screen. With long functions (or when entire files are displayed) this will result in a fragment that is vertically scrollable. This prevents the fragments from passing the touch events to the canvas, preventing the desired behavior. A way to resolve this issue would be to allow fragments to expand completely, preventing the need for vertical scrolling in fragments. Fixing this issue would improve the canvas navigation, especially when many fragments are on the screen.

5.2.2 Fragments

Links in fragments where recognized by the users. The creation of new fragments containing the declaration of the clicked functions source code was in line with users expectations. Users where given the choice of using their finger or a stylus for interacting with the tablet. The two users that chose their hands as primary input reported a more enjoyable experience compared to the ones using the stylus. However, non-stylus users had occasional problems clicking on links in fragments, caused by the loss of precision. This resulted in the addition of a setting to the application that controls the font size of the source code displayed in fragments. By increasing the font size, the clickable region of links

is increased. The draw-back is that using a larger font size decreases the amount of source code that can be displayed on the screen. A better solution would be to increase the clickable region without changing the font size.

All participants reported a desire to resize the fragments without being able to express their motivation. One participant speculated that he didn't trust the application to resize the fragments to fit the source code into. It was also suggested that "folding" or "minimizing" of the fragments might be beneficial. This would allow the user to temporarily reduce the space taken up by some of the fragments. Almost all users said that line numbering of the source code in the fragments would be helpful.

Adding of annotations by long-clicking on the canvas was only discovered by two users. Only one reported to find this intuitive. This suggests to me that it might be better if annotations could be added through an entry in the menu. The code and workspace browser also allow long-clicking entries for various additional actions. No participant discovered those features, although they where looking for the functionality provided by long-clicking. It seems like implementing a "Tip of the Day" feature is necessary to ensure users eventually discover those features.

5.2.3 Code Browser

Most users found the code browser to be mostly intuitive, stating their familiarity with a similar component from other IDEs. The search interface was used by all participants with ease, although some tweaks to the visual presentation are necessary. One user suggested to embed the search bar into the action bar of the application for easier access. This user used the search function extensively throughout the test scenario.

The popup context menu for handling ambiguous declarations was generally understood by the users, but had some problematic behavior. When the menu first pops up, a fragment is created to be populated. When the menu is closed without selecting an entry, the empty fragment will stay on the canvas, confusing users. Additionally, the menu closes to easily. Several users tried to drag the canvas while the popup was open, resulting in the menu closing without giving the user an opportunity to make a selection.

The code browser might benefit from a button that allows users to collapse and expand all items. Especially with larger projects this might be convenient, as a large directory structure might be explored, resulting in many expanded entries. Furthermore, the browser could be enhanced by allowing different sorting

strategies of entries, like sorting by filenames, file extensions or reversing the order.

5.2.4 Workspace Browser

The workspace browser was not immediately usable by most of the participants. After a short orientation period they had no problems using it and generally agreed that it is a useful addition for navigation. Improving the visual presentation of the workspace browser might reduce this "orientation time". One participant suggested that sorting of the fragments in the order they where created, as well as focused, might be helpful.

Chapter 6

Discussion and Future Work

In this section I will discuss some of results achieved and try to speculate of the impact that they have. I will also suggest enhancement and new features that could be added to CodeMap in the future.

6.1 Usability

Using a touch interface for manipulating the canvas seemed mostly intuitive to users. One major difference between CodeMap and Code Bubbles is, besides the touch interface, is the available screen space. Code Bubbles was designed to be used on workstations with big, high resolution screens. In contrast, CodeMap is designed to be used on tablets that tend to have a smaller screen size and lower resolution. The idea of using the bubbles metaphor is to enable users to customize screen space to allow a large amount of source code to be on the screen at once. Although quantitative data needs to be collected to support this claim, I'm currently under the impression that CodeMap allows a more compact display of source code than traditional split-window environments.

One aspect that I'm eager to explore further is the persistent workspace feature. Two out of five users tried to keep the workspace "uncluttered" by closing explored fragments. The other three users left fragments open and utilized several

workspaces that they referred back to one or more occasions during the usability test. This might indicate that some users will use CodeMap as a persistent documentation of the source code they have read. Workspaces where used by the participants as "working sets". Having the ability to name workspaces and closing them when they are not needed, allows the developer to manage large amount of open fragments. There seem to be no disadvantages to having multiple workspaces instead of one.

It is somewhat difficult to design long running usability tests that are able to measure the potential benefits of having a large, persistent set of workspaces. The most realistic impression could be gained by allowing developers to use CodeMap as part of their daily routine. In section 6.4 I will describe a method of collecting data remotely, allowing the collection of data efficiently on a large scale.

6.2 Improvements

A big improvement to CodeMap would be the replacement of cscope as code analysis tool to a tool. The design of CodeMap makes it simple to do so, as the only part that interacts directly with the cscope wrapper is a single Java class. The code analysis tools of Eclipse would be a good choice, as they support many languages, are written in Java and support static analysis of most language features. This would allow a faster search for definitions in large code bases and provide additional information about the source code. Unfortunately no Android port of Eclipse currently exists. Eclim [Dew] allows running Eclipse in "headless mode", allowing other clients to connect to it and use Eclipse's features. If Eclim offered support for network clients, it would be possible to provide CodeMap with the same amount of information as a complete IDE, as well as compilation and debugging support.

The usability tests have also revealed that users want to be able to display variable declarations. Cscope unfortunately does not deliver this information. If cscope is replaced by a framework that can deliver this information, it must be presented to the user in a meaningful fashion. The simplest approach would be to present each variable in a fragment. If an object oriented language is used, it might also be useful to show all variable declarations of a given object in a single fragment. Perhaps each fragment could have a collapsible list at the top instead, containing all variable declarations used in its function.

6.3 Additional Features

CodeMap is not able to edit source code. It might be useful to be able to transfer the layout and annotations generated in CodeMap to an external IDE. This could take the form of a plugin for Eclipse. The user could import the annotation as comments to the source code. It might also be conceivable that integration with similar tools, like Code Canvas or Code Bubbles, might be possible.

It would also be useful to have a synchronization and collaboration feature similar to the one discussed in section 3.4. Some users also suggested support for debugging. This could be implemented by using Eclim (described above) and CodeMap could provide a UI similar to Microsoft's Debugger Canvas (discussed in section 2.4).

It might also be conceivable to research the implementation of a web interface (see section 3.5), allowing users to try a CodeMap application without the need to install additional software. The biggest issue with this feature might be the lack of support for mobile touch interfaces or the lack of precision input.

6.4 Remote Testing

Besides my direct usability testing, remote testing can be employed. Google analytics [Goob] is a tool that might be helpful for that purpose. It allows logging of user activity, including profile data from of the users phone (screen size, resolution, phone brand, etc.) and recording of actions that users take. This can include where the user clicked and what functions where called as a result of user input. When a crash occurs, it can be logged, together with a log of what the user did to make this crash occur. Other useful metrics, like the amount of fragments and workspaces a user has opened, could be collected as well. This makes it possible to determine what features are most used and, to some degree, whether the user understands the interface.

Having all these logging features would allow a large scale testing of CodeMap by publishing it on Google Play[1]. Although Google Analytics is a versatile tool, it is unable to report the users subjective feelings or collect feedback and suggestions. In addition to the logging, users could be asked to fill out a simple form, similar to the usability questionnaire. Having this information might be

[1] Google Play is the official application repository of Android.

enough to be able to determine the best modifications to the application and the advantages and disadvantages of the CodeMap approach.

Chapter 7

Conclusion

In this thesis I wanted to explore a new method of reading code bases inspired by Code Bubbles. This tool should be able to have a persistent workspace state, allowing the user to resume previous code reading sessions. I also wanted to examine what effects a reduced screen size and a touch interface have on the UI design.

I successfully implemented a tool similar to Code Bubbles for the Android platform. It supports multiple projects. Source code can imported and synchronized inside the application with the Git version control protocol. The tool uses cscope to index code bases and supports projects written in the C programming language. It is possible to explore the file system and list symbols contained in files. Source code can be opened in small widgets (ie. fragments) that can be moved freely on a large 2D canvas. Source code fragments are syntax highlighted and their function calls are turned into clickable links. When one of these links is clicked, a new fragment containing the source code of that function is added to the canvas. Fragments containing annotations can be added as well. Multiple workspaces are supported for each project, allowing the user to create multiple canvases. Each canvas can be opened or closed individually and their complete state is saved. A "workspace browser" is provided to ease navigation between workspaces. I created a usability test scenario to test all components of the UI. Five people participated in the test. All participants where able to access all components of the interface as well as completing all tasks in the scenario.

Changing from cscope to a better code analysis framework would make it possible to increase CodeMap's feature set, especially with regards to displaying variables. Sharing of workspaces with other users or the integration with an IDE was mentioned by participants of the usability test as feature they would like to see. It might also be worth exploring editing of source code on tablets. Perhaps the use of a structured editor, similar to Touch Develop, could be used instead of having to attach a keyboard to the device. CodeMap could also be expanded to allow debugging, similar to Code Bubbles and Debugger Canvas.

I personally think that persistent workspaces for reading source code shows promise. Unfortunately I have not tested its long-term impact on developers ability to navigate and remember large code bases. The prospect of publishing CodeMap on Google Play and collecting data remotely might lead to some useful insights on how developers use CodeMap in practice. I have published the entire source code of CodeMap on Github[1] and posted many of the discovered problems to its associated issue tracker. The project is released under the Gnu Public License (GPL) to encourage other developers to contribute to the project.

[1] http://www.github.com/hdweiss/codemap

Appendix A

Usability Test Tasks

- **Task 1:** Open the project “cscope”.
- **Task 2:** Understand what function “createTagsForWilcardEntry” in file “jni/main.c” does.
- **Task 3:** Open a new workspace/tab.
- **Task 4:** Understand what function “isTagFile” does.
- **Task 5:** Load an arbitrary project from git into CodeMap.

Appendix B

Usability Test Results

B.1 Thomas

B.1.1 Observations

- Found code browser
- Alphabetic ordering of files
- Expansion of symbols of files makes sense
- Declarations are opened by clicking on links
- Panning the workspace make sense
- Function fragments can be in different files, title bar explains where files are
- Sometimes icons are non-responsive
- Found search bar
- Confused over declaration choice popup
- Windows are familiar (drag, close)

- Feels like windows should be resizable
- Lack of "an overview", as the case with viewing a file. CodeMap seems unfamiliar
- Creating projects is intuitive
- Git is the only method?
- I'm closing windows as I don't want to clutter the workspace
- Nothing happens when clicking on files containing no declarations

B.1.2 Suggestions

- Syntax highlighting for variable declaration
- No ability to show variable declarations outside of functions
- Search should be case insensitive
- Line numbering in fragments would be helpful
- Vertically scrollable source code fragments should have scrollbar to indicate scrolling
- Difficult to get an overview with many opened functions, minimap?
- Multiple workspaces
- Function specific annotations. "I want to document a single function"
- Start screen is empty and unwelcoming

B.1.3 Email

```
Hej Henning

Jeg sad og brugte lidt tid på dit program, og har skrevet mine tanker
ned omkring hvad jeg oplevede.  Jeg ved ikke hvor meget af det du kan
bruge, men skrev bare alt ned.

Hilsen Thomas
```

---------Warum?---------

Der bliver vist ude i højre side, hvor mange
bobler der er åben ud fra en given function, men hvis man åbner en hel fil, står der ikke noget om hvor mange gange man har åbnet den.

Når jeg søger på noget, kommer der en liste frem ude i højre side. Alle tiders. men når jeg trykker på en af tingene kommer der nogle gange en liste frem med filer jeg kan vælge fra. men jeg har jo valgt et bestemt udtryk så hvorfor kommer den? Det ser ud til at den også tager alle de andre søge resulter med i listen, hvilket virker som en meget underlig opførsel.

Er det meningen at Annotations skal kunne vokse næsten ubegrænset i bredden? Burden den ikke kun kunne vokse til samme størrelse om de "almindelige" vinduer.

Burde Annotation ikke starte med at åbne tastaturet op, så man kan skrive med det samme i den, i stedet for at man skal klikke på den? fordi det kan godt drille lidt med at ramme det rigtige sted i den.

-----Godt lavet--------

Det var rart at se en progressbar når deres laves en Git pull.
Det samme gælder også at man kan se url'en til projektet.

Det er godt at linjen der kommer ud af vinduet, når man har klikket på en funktion, kommer ud på samme linje som funktionskaldet.

------Features------

Kunne man ikke lave det, så hvis en kommentar til en funktion starter med BUG eller TODO, så får vinduet en anden farve i kanten, eller som du viste fra CodeBubbles, at der kommer et icon i toppen af vinduet.

Hvis man et Long Click på en funktion som er åben, kunne man så ikke lave det så man kunne flytte fokus/hvad man ser, hen til en af de åbne vinduer, eller i hvert fald få en menu med mulighederne om enten at lave en ny eller finde en eksisterende.

Man kunne lave Checkpoints. Altså man havde f.eks. en liste af checkpoints man kunne trække frem fra venstre side og når man trykkede på dem, blev man flyttet hen til sit checkpoint.

```
Hvis man har fået åbnet nogle vinduer, ved at klikke på funktionskald,
og det er blevet ved et stykke tid, og man har fået lavet sig et træ
af funktioner. Så kunne det måske være meget rart med en måde at kunne
lukke hele "Call stack" på en måde, i stedet for at skulle lukke alle
vinduer individuelt.

---------- Bugs --------

Jeg havde ikke nogen Internet forbindelse, og var ude på siden med
projekt listen, lavede et Long click på et project og valgte Update,
hvor efter der kom op med en pop-up "Codemap is not responding" Efter
jeg kom på nettet, så virkede det også fint på testProject, men på
linux projectet crashed det.

Jeg gik ind i linux projectet og åbnede lib/gcd.c og så:
#include
  stå der 3 gange, men der stod ikke noget om hvad den inkluderedet.
Så vidt jeg har kunnet finde ud af så burde der havde stået:
#include <linux/kernel.h>
#include <linux/gcd.h>
#include <linux/export.h>

Jeg åbnede en .html fil og det virker som om den ikke er så glad for <>,
kort sagt så fjernede den store dele af filen.

Jeg var inde i kernel/cpu.c:__cpu_notify, hvor inde jeg fandt linjen
 ret = ifier_call_chain">__raw_notifier_call_chain(.....
men koden fra version 3.7.9 skriver
 ret = __raw_notifier_call_chain(....

 Når jeg søger på noget, får jeg masse svar tilbage, klikker jeg så på
 en og ser en lang liste komme frem, og klikker ved siden af, så er
 der en scrollbar, ved det top-vesntre hjørne af pop'upen, som bliver
 siddende, så hvis jeg åbner en ny list op, så kommer der endnu en
 scrollbar op.

 Jeg var inde i Linux projektet og søgte på 842, hvor efter jeg fik en
 "Error finding entries", men inde i crypto mappen, er der en fil med
 det navn, samt nogle funktioner der også indeholder 842.

 Hvis teksten i header'en bliver for lang, så lægger den sig bag ved
 lukkeknappen, eller ned på næste linje, hvilket bare uheldigvis ikke
 er synlig. Headeren burde vel vokse i højden eller klippe teksten.
```

Hvis man har åbnet en funktion ved at klikke på et funktionskald i et vindue(1), og trækker det åbnet vindue(2), så den enten er til venstre, oppe over, eller nedenunder det andet vindue(1), så lægges stregen oven i vinduerne.

Jeg ved ikke om det er mig som gør noget forkert. men jeg har meget svært ved at flytte en Annotation, uden at den forsvinder.

Hvis man prøver at tilføje et nyt projekt med samme navn som et allerede eksisterende projekt, får man lov til det, det burde man vel ikke kunne.

Pre-Test Questionnaire

Your Name Thomas Pedersen

Your Age 22

Your Major Compotel / IT

Years of experience (Coding) 4

Known Languages

Java, C#, Dart, C, C++, Ada, VHDL
F#

IDE/Editor proficiency Visual Studio, emacs, eclipse

IDE/Editor preference Visual studio

Familiarity with CodeBubbles none

Android proficiency none

Tablet proficiency none

Are you familiar with ~~code base X~~ Cscope? nope

How many usability tests have you participated in? none

What do you expect of CodeMap?

Post-Test Questionnaire

What where your biggest issues?

That it do not know exactly What functions are called.

What parts of the interface are you dissatisfied with?

What parts of the interface do you like?

The menu with the searchbar and list of files. It's simple but efficient.

Do you have any advice for the future development of CodeMap?

I think it needs to understand the code in depth, to make a better experience.

How do you feel about CodeMap overall? (1 = Very unfavorably, 5 = very favorable)

[1 (2) 3 4 5]

Considering that the issues are ironed out, would you find CodeMap a useful addition to your daily coding routine? (1 = not useful, 5 = very useful)

[1 (2) 3 4 5]

Optional comments

B.2 Martin

B.2.1 Observations

- Used stylus
- Found code browser
- Found panning of browser
- Opened fragments
- Links are intuitive, opening a new declaration makes sense
- Participant had tendency to close function fragments again
- Found search bar
- Found new tab button
- Doesn't make use of new tabs/workspaces
- Praised case insensitive search
- Found search interface intuitive
- Keyboard popped up unexpectedly, causing user to become confused
- Realized adding new projects is in project browser
- Found annotations
- Declaration popup for resolving ambiguous declarations made sense

B.2.2 Suggestions

- Expected source browser to be open on startup
- Icons for code and workspace browser are normally associated with sharing capability
- Intuitive that directories are blue
- Search list should be empty when no results are found
- Search should support "wildcard" matching

- Function fragments that have function calls to other fragments should have links between each other, even when they are not opened by clicking on link in function
- Show title of project in action bar
- Hierachy of CodeMap widget elements is not clear - browser should be on top of action bar etc.
- Color scheme for code browser is confusing - gray symbols look like they are disabled
- Workspace browser could promote most recent fragments in lists

Pre-Test Questionnaire

Your Name Martin Wiboe

Your Age 24

Your Major Engineering Management

Years of experience (Coding) 10

Known Languages

C#, Java, Python, VB.Net, HTML, C, 中文

IDE/Editor proficiency Visual Studio, Sublime Text, vim, Eclipse, Xcode

IDE/Editor preference Visual Studio, Sublime Text

Familiarity with CodeBubbles None

Android proficiency User experience, basic development

Tablet proficiency Development for iPad (skilled)

Are you familiar with cscope? No

How many usability tests have you participated in? 3

Post-Test Questionnaire

What where your biggest issues?

Confusion about workspace vs. project, and the document will not appearing "shared" between workspaces. It wasn't easy to move the fragments around.

What parts of the interface are you dissatisfied with?

The left-hand "list of recent functions seems cluttered and difficult to navigate. Icons don't make sense ("sharing" is used as a handle,

What parts of the interface do you like?

Right hand navigation was intuitive (though search should allow partial input). Clear syntax highlighting and visible links.

Do you have any advice for the future development of CodeMap?

Fix control hierarchy. Rename "Workspace".
Recent fragments: Method name should be prominent: Order by latest used.

isTagFile(name) ctrl/mouse

How do you feel about CodeMap overall? (1 = Very unfavorably, 5 = very favorable)

[1 2 3 (4) 5]

Considering that the issues are ironed out, would you find CodeMap a useful addition to your daily coding routine? (1 = not useful, 5 = very useful)

[1 (2) 3 4 5]

Optional comments

I rarely navigate large codebases just for reading the code. CodeMap would be useful to me if integrated into an IDE.

B.3 Walter

B.3.1 Observations

- Uses fingers instead of stylus
- Intuitively used zoom
- Tried to resize fragments - wants entire header declaration on same line
- Occasional trouble clicking on links with fingers
- Tried to view structures and variable declarations
- Praised title bar of fragments
- Leaves code browser open under entire test
- Some problems with overlapping
- Moved fragments by dragging
- Opened new workspace
- Tabs seem intuitive
- Uses search
- Uses newly opened workspace
- Problems with disappearing “ambiguous popup” context menu when clicking somewhere else
- Used search to find struct declaration
- Confused by workspace browser expandable list
- Tried searching for files
- Praised that touch interface is an intuitive choice to manipulate the interface
- Added new project
- Found annotation - find opening intuitive
- Annotation close “randomly”

B.3.2 Suggestions

- Search embedded in action bar for easy access
- Panning canvas should work on fragments, especially when large fragments are on screen - might result in inability to move anywhere else
- Expandable lists should have icons indicating state (collapsed/expanded)
- Code browser: sorting of entries by name, extension, etc.
- Project screen needs a title or description of functionality
- Long click on tabs should show menu to close tab, close all tabs, close all but this

Pre-Test Questionnaire

Your Name WALTER

Your Age 24

Your Major COMPUTER SCIENCE

Years of experience (Coding) 8

Known Languages

C, C++, JAVA, PHP

IDE/Editor proficiency ECLIPSE, IAR, ATMEL STUDIO, MPLAB

IDE/Editor preference ECLIPSE

Familiarity with CodeBubbles NONE

Android proficiency USER

Tablet proficiency LOW

Are you familiar with cscope? NO

How many usability tests have you participated in? FEW

Post-Test Questionnaire

What where your biggest issues?

ZOOM, RESIZE, SCROLLING

What parts of the interface are you dissatisfied with?

EXPANDABLE LISTS, PROJECT SELECT SCREEN

What parts of the interface do you like?

WORKSPACE, TOUCH INTERFACE

Do you have any advice for the future development of CodeMap?

MOVE FILES, RENAME FILES, ETC...
SEARCH ALWAYS VISIBLE
EASIER TO OPEN THE WHOLE FILE

How do you feel about CodeMap overall? (1 = Very unfavorably, 5 = very favorable)

[1 2 3 (4) 5]

Considering that the issues are ironed out, would you find CodeMap a useful addition to your daily coding routine? (1 = not useful, 5 = very useful)

[1 2 3 (4) 5]

Optional comments

B.4 Georgious

B.4.1 Observations

- Uses fingers instead of stylus
- Wants to display global variable declarations
- Re-arranges windows close to each other to exploit screen space
- Notices that dragging on fragment body doesn't work
- Found search
- Found new tab
- Uses search to find struct
- Found and understood project browser
- "Feels like" bubbles should be expandable
- Confused whether workspaces are project specific
- Confused by cross button in action bar
- Found workspace browser
- Leaves project browser open throughout test
- Praises application several times throughout test

B.4.2 Suggestions

- Line numbering in function fragments
- Folding or "minimizing" of fragments

Pre-Test Questionnaire

Your Name Georgios Katsoris

Your Age 31

Your Major MsC in Electrical Engineering, MsC in Computer Science

Years of experience (Coding) 6

Known Languages

English, Greek, C, C++, Java, VHDL

IDE/Editor proficiency Vim

IDE/Editor preference Vim

Familiarity with CodeBubbles Zero

Android proficiency Zero

Tablet proficiency Zero / don't own one

Are you familiar with cscope? No

How many usability tests have you participated in? None

Post-Test Questionnaire

What where your biggest issues?

None in particular

What parts of the interface are you dissatisfied with?

None in particular

What parts of the interface do you like?

Workspace, search feature,

Do you have any advice for the future development of CodeMap?

can already open code fragment ~~code fr~~ should not be opened more than once! Line numbering (as an option), resizing of code fragments

How do you feel about CodeMap overall? (1 = Very unfavorably, 5 = very favorable)

[1 2 3 (4) 5]

Considering that the issues are ironed out, would you find CodeMap a useful addition to your daily coding routine? (1 = not useful, 5 = very useful)

[1 2 3 (4) 5]

Optional comments

B.5 Lars

B.5.1 Observations

- Uses stylus
- Leaves browser window open
- Tried clicking on structs, even though they don't have link markup
- Tries double clicking on files in code browser to view entire file
- Moves fragments by dragging
- Tried moving fragments by dragging body
- Clicks on workspace browser to create new workspace
- Uses search
- Uses multiple workspaces
- Clicks on functions without hightlight
- Understands popup menu for ambiguous functions
- Understands workspace browser
- Didn't find zoom

B.5.2 Suggestions

- Function should be centered when adding them
- Clicking on variables could highlight all their uses in current/all fragment(s)
- Focusing functions by clicking on symbols in workspace browser should center fragments on screen

Pre-Test Questionnaire

Your Name Lars Barrichsen

Your Age 25

Your Major CS

Years of experience (Coding) 8

Known Languages
C, C++, java, sml, Makefile, avs, basic, Delphi, NSIS

IDE/Editor proficiency Netbeans, Suite, eclipse, Winamp

IDE/Editor preference Netbeans. Suite

Familiarity with CodeBubbles Saw intro video

Android proficiency Almost no use

Tablet proficiency 11

Are you familiar with cscope? Heard of

How many usability tests have you participated in? 2

Post-Test Questionnaire

What where your biggest issues?

Ambigous func ref close too fast
Open whole file is "hidden"

What parts of the interface are you dissatisfied with?

~~Zoom~~ Finding type definitions

What parts of the interface do you like?

Supports flipping the screen. Uncluttered

Do you have any advice for the future development of CodeMap?

Make it easier to collapse files in cx browser. Reflow => auto arrange fragments, possibly using relaxing over time or with slider

How do you feel about CodeMap overall? (1 = Very unfavorably, 5 = very favorable)

[1 2 3 (4) 5]

Considering that the issues are ironed out, would you find CodeMap a useful addition to your daily coding routine? (1 = not useful, 5 = very useful)

[1 2 3 (4) 5]

Optional comments

Bibliography

[AB10] Robert Zeleznik Andrew Bragdon, Steven P. Reiss. Code bubbles: Rethinking the user interface paradigm of integrated development environments. *Communications of the ACM*, 10, 2010.

[Bra] Andrew Bragdon. Code bubbles. `http://www.andrewbragdon.com/codebubbles_site.asp`.

[Cor] Oracle Corporation. Netbeans. `http://netbeans.org`.

[Dew] Eric Van Dewoestine. Welcome to eclim: The power of eclipse in your facorite editor. `http://eclim.org/`.

[DFB96] Peter F. Sweeny David F. Bacon. Fast static analysis of c++ virtual function calls. *Proccedings of the ACM*, 10, 1996.

[Foua] Eclipse Foundation. Eclipse mylin. `http://www.eclipse.org/mylyn`.

[Foub] The Eclipse Foundation. Eclipse. `http://www.eclipse.org`.

[Fouc] The Eclipse Foundation. Jgit - java git. `http://eclipse.org/jgit`.

[FSF] Inc. Free Software Foundation. Ncurses. `http://www.gnu.org/software/ncurses/`.

[Git] Inc GitHub. Github.com. `http://www.github.com`.

[GNU] GNU. Ddd - data display debugger. `https://www.touchdevelop.com`.

[Gooa] Google. Android user interface guidelines. `http://developer.android.com/guide/practices/ui_guidelines/index.html`.

[Goob] Google. Google analytics. http://www.google.com/intl/da/analytics.

[III] Harry H. Porter III. Smalltalk overview. http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html.

[JD95] Craig Chambers Jeffrey Dean, David Grove. Optimization of object-oriented programs using static class hierarchy analysis. *ECOOP*, 1, 1995.

[Jet] JetBrains. Jetbrains intellij idea. http://www.jetbrains.com/idea.

[Mica] Microsoft. Code canvas. http://research.microsoft.com/en-us/projects/codecanvas/.

[Micb] Microsoft. Debugger canvas. http://msdn.microsoft.com/en-us/devlabs/debuggercanvas.

[Micc] Microsoft. Touchdevelop. https://www.touchdevelop.com.

[Micd] Microsoft. Visual studio. http://www.microsoft.com/visualstudio.

[Ste] Joe Steffen. Cscope. http://cscope.sourceforge.net/.

[Uni] Texas A&M University. The pivot. https://parasol.tamu.edu/pivot/.

[Vir92] R.A. Virzi. Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors*, 34, 1992.