

Sketchassisted Development

Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University



by
Torben Schulz

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Horst Lichter

Registration date: 04.03.2014

Submission date: 04.09.2014

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, September 2014
Torben Schulz

Contents

Abstract	xvii
Überblick	xix
Acknowledgements	xxi
1 Introduction	1
1.1 Motivation	2
1.1.1 Attachable to widely used IDE	5
1.1.2 Usability	8
1.1.3 Minimum of Additional Files	9
2 Related Work	11
2.1 Exploring Source Code History	11
2.2 Source Code Visualization Approaches in Terms of Time and Space	13
2.3 Source Code History	16
2.4 Code Understanding & Knowledge	17

2.4.1	Mining	19
2.4.2	Refactoring Detection	20
2.4.3	Whiteboard and Sketches	20
3	Initial study	25
3.1	Meeting with Development Heads	25
3.2	Meeting with Design Team	27
3.3	Meeting with Developer	28
3.4	Daily Meeting	30
3.5	Meeting with Quality Manager	31
3.6	Meeting with Development Assistant	31
3.7	Summary	31
4	Prototypes	33
4.1	Implementation	33
4.1.1	IDE Extension	33
4.1.2	Sketch History	34
4.1.3	AST	36
4.1.4	IBOutlet Connection	38
4.1.5	Syntax Highlighting	40
4.1.6	Source Code History and Diff	42
4.2	Layout and Functionality	43
4.2.1	CodeShape	43

IBOutlet alike Connections	47
4.2.2 Azurite	56
Modifications	59
4.2.3 Chronos	60
5 Evaluation	65
5.1 Preparation	65
5.2 Execution	69
5.3 Analysis	73
5.3.1 Participants	73
5.3.2 Tactics to Find a Version	74
5.3.3 Task Completion Times	75
5.3.4 Initial Navigation Time	82
5.3.5 Sketch Inspection Time	93
5.3.6 Feedback and Suggestions	95
6 Summary and Future Work	105
6.1 Summary and Contributions	105
6.1.1 Limitations	107
6.2 Future work	108
6.2.1 Improve the prototypes	108
6.2.2 Sketch and source code entity consistency	108

Parallax	109
iPad App	109
A Informed Consent Form	111
B User Study Form	115
C Task Completion Time Graphs	127
D Navigation Time Graphs	143
E Source Code of decorateCommit	153
F Initial Study Questionnaire	157
G Sketch Inspection Time Graphs	161
H Sketches	169
Bibliography	173
Index	183

List of Figures

1.1	Software Prototype Functionality Brainstorming	2
1.2	ASCII Sketch	6
2.1	GraphTrail image se	15
3.1	Mobile App Wireframe	29
4.1	Valid Method Implementation	37
4.2	IBOutlet Connection	38
4.3	Determine IBOutlet Connection Rectangle	40
4.4	IBOutlet Connection sketch V2	41
4.5	GitX Application	44
4.6	Mockup of IBOutlet Connection	45
4.7	Xcode application	45
4.8	Ipad Prototype	46
4.9	IBOutlet Connection Sketch V1	47
4.10	Connect Sketch to a Source Code Entity	48

4.11	CodeShape Sidebar	50
4.12	CodeShape Explorer - right click context menu	51
4.13	CodeShape Explorer - Timeline Highlighting	51
4.14	CodeShape Explorer Diff - Addition, Deletion, Update Selected	53
4.15	CodeShape Explorer Diff - Addition, Deletion Selected	53
4.16	CodeShape Explorer Diff - Addition Selected	54
4.17	CodeShape Explorer Diff - Nothing Selected .	54
4.18	CodeShape Single Click Interaction	55
4.19	CodeShape Explorer	57
4.20	Azurite - Eclipse Plugin	58
4.21	Extended Azurite	59
4.22	Chronos History Slicing GUI	61
5.1	Git Tree	67
5.2	Screenshot of CodeShape Plugin	69
5.3	Screenshot of Azurite Plugin	70
5.4	Screenshot of Chronos Plugin	71
5.5	Screenshot of Chronos Plugin and Sketch . .	72
5.6	Task 1.2.2 Tukey HSD	78
5.7	Task Completion Graph Task 1.2.2	78
5.8	Task Completion Graph Task 2.1.1	79

5.9	Task 2.1.1 Tukey HSD	80
5.10	Task Completion Graph Task 2.2.2	81
5.11	Task 2.2.2 Tukey HSD	82
5.12	Task Completion Graph Task 3.3.3	83
5.13	Task 3.3.3 Tukey HSD	83
5.14	Task Navigation Graph Task 2.1.1	86
5.15	Task 2.1.1 Tukey HSD	87
5.16	Task Navigation Graph Task 2.2.1	87
5.17	Task 2.2.1 Tukey HSD	88
5.18	Task Navigation Graph Task 3.1.1	89
5.19	Task 3.1.1 Tukey HSD	90
5.20	Task Navigation Graph Task 3.2.1	91
5.21	Task 3.2.1 Tukey HSD	91
5.22	Task Navigation Graph Task 3.3.1	92
5.23	Task 3.3.1 Tukey HSD	92
5.24	Eclipse Compare Editor	97
5.25	NSSlider vs CodeShape Slider	98
5.26	Azurite Rectangle Popup	99
5.27	Modified Azurite Screenshot	100
5.28	Azurite Vertical Slider	100
5.29	Chronos information loss issue	102

5.30 Chronos History Slicing Connection Image Section	103
C.1 Task Completion Graph Task 1.1.1	128
C.2 Task Completion Graph Task 1.1.3	129
C.3 Task Completion Graph Task 1.2.1	130
C.4 Task Completion Graph Task 1.2.2	130
C.5 Task 1.2.2 Tukey HSD	131
C.6 Task Completion Graph Task 1.3.1	131
C.7 Task Completion Graph Task 1.3.2	132
C.8 Task Completion Graph Task 1.3.3	132
C.9 Task Completion Graph Task 2.1.1	133
C.10 Task 2.1.1 Tukey HSD	133
C.11 Task Completion Graph Task 2.1.2	134
C.12 Task Completion Graph Task 2.2.1	134
C.13 Task Completion Graph Task 2.2.2	135
C.14 Task 2.2.2 Tukey HSD	136
C.15 Task Completion Graph Task 2.3.1	137
C.16 Task Completion Graph Task 2.3.2	137
C.17 Task Completion Graph Task 2.3.3	138
C.18 Task Completion Graph Task 3.1.1	138
C.19 Task Completion Graph Task 3.1.2	139
C.20 Task Completion Graph Task 3.2.1	139

C.21 Task Completion Graph Task 3.2.2	140
C.22 Task Completion Graph Task 3.3.1	140
C.23 Task Completion Graph Task 3.3.2	141
C.24 Task Completion Graph Task 3.3.3	141
C.25 Task 3.3.3 Tukey HSD	142
D.1 Task Navigation Graph Task 1.1.1	143
D.2 Task Navigation Graph Task 1.2.1	144
D.3 Task Navigation Graph Task 1.3.1	144
D.4 Task Navigation Graph Task 1.3.3	145
D.5 Task Navigation Graph Task 2.1.1	145
D.6 Task 2.1.1 Tukey HSD	146
D.7 Task Navigation Graph Task 2.2.1	146
D.8 Task 2.2.1 Tukey HSD	147
D.9 Task Navigation Graph Task 2.3.1	148
D.10 Task Navigation Graph Task 2.3.3	148
D.11 Task Navigation Graph Task 3.1.1	149
D.12 Task Navigation Graph Task 3.2.1	149
D.13 Task 3.2.1 Tukey HSD	150
D.14 Task Navigation Graph Task 3.3.1	150
D.15 Task 3.3.1 Tukey HSD	151
D.16 Task Navigation Graph Task 3.3.3	151

G.1	Sketch Inspection Time Graph - Task 1.1.3 . . .	162
G.2	Task 1.1.3 Tukey HSD	162
G.3	Sketch Inspection Time Graph - Task 1.3.3 . . .	163
G.4	Task 1.3.3 Tukey HSD	163
G.5	Sketch Inspection Time Graph - Task 2.1.2 . . .	164
G.6	Task 2.1.2 Tukey HSD	164
G.7	Sketch Inspection Time Graph - Task 2.3.3 . . .	165
G.8	Task 2.3.3 Tukey HSD	165
G.9	Sketch Inspection Time Graph - Task 3.1.2 . . .	166
G.10	Sketch Inspection Time Graph - Task 3.3.3 . . .	167
G.11	Task 3.3.3 Tukey HSD	167
H.1	Inkling sketch for commit 9bfccb5	170
H.2	Inkling sketch for commit bbeedd1	171

List of Tables

5.4	ANOVA of task completion time, comparing Chronos History Slicing, CodeShape and Azurite.	76
5.4	Tukey HSD post hoc test of task completion time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape	77
5.4	ANOVA of navigation time comparing Chronos History Slicing, CodeShape and Azurite	85
5.4	Tukey HSD post hoc test of initial navigation time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape	86
5.4	ANOVA of sketch inspection time comparing Chronos History Slicing, CodeShape and Azurite.	94
5.4	Tukey HSD post hoc test of sketch inspection time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape	95
5.4	Compares ratio of mean sketch inspection time and mean task completion time(A)zurite, Chronos (H)istory Slicing and (C)odeShape	95

Abstract

Every software developer has been confronted with unknown source code. Several resources are consulted to obtain a deeper understanding and in order to get the rationale. The web and the source code itself, with its history of versions and API's serve as a primary resource.

Sketches are often used to depict overall ideas, assist discussions and reduce complexity. Commonly they are drawn on whiteboards, but are rarely found in source code repositories. Digitizing and retrieving those sketches with today's tools is tedious. So far hand-drawn sketches have been added in form of image files, which only allow visual interpretation. A possible link to source code has to be established manually. General software development environments do not support hand-drawn sketches, assisting the development process.

The ability to connect hand-drawn sketches to versions of source code entities was added to the development environment Xcode. I tested three software prototypes for sketch assisted source code history exploration against each other, but only in a few tasks significant results in task completion and navigation time have been observed. Every participant stated that they are most convinced of CodeShape, after they tried all three on different tasks.

The thesis begins with the motivation to connect sketches to versions of source code entities. Preliminary studies in the form of an interview were held at a software company. Next I developed and extended three software prototypes, which are evaluated in the subsequent chapter. The thesis ends with a summary and some suggestions for future work.

Überblick

Jeder Software Entwickler wurde bereits mit Quelltext oder gesamten Projekten konfrontiert, die nicht von ihm selbst entwickelt wurden. Um bei einem derartigen Projekt Verständnis zu erlangen, greift man in der Regel auf alle zur Verfügung stehenden Informationen zu. Seien es beispielsweise Kommentare, API's, Dokumentationen, Skizzen, Suchmaschinenergebnisse und die Historie die einem weiterhelfen können.

Das Problem bei Skizzen ist, dass sie meist nur lokal vorliegen. Zudem können Skizzen für außenstehende Entwickler zu komplex sein und eine Zuordnung von mehreren Zeichnern bei einer Skizze ist im Nachhinein auch nicht mehr möglich.

Ohne visuelle Interpretation einer Skizze lässt sich keine oder nur schwer eine Verbindung zu einem Projekt oder einzelnen Entitäten eines Projekts ziehen. Skizzen landen beispielsweise oft in einem Sammelordner und verlieren ihre Verbindung.

In meiner Arbeit werden zwei modifizierte Prototypen und eine eigene Implementierung vorgestellt und miteinander verglichen. Die Prototypen wurden um Skizzenfunktionalität erweitert und ermöglichen Navigation und Interaktion mit der Historie. Beim Vergleich der Prototypen in einer Benutzerstudie konnte nur in seltenen Fällen ein signifikantes Ergebnis bei der Bearbeitungsdauer und initialer Navigationszeit festgestellt werden. Die Umsetzung von CodeShape hat allen 17 Teilnehmern am besten gefallen, nachdem alle drei Prototyten getestet wurden.

Die Arbeit beginnt mit den Beweggründen Skizzen mit Versionen von Quelltextentitäten, zu verbinden. Danach folgt die Betrachtung und Auswertung von bereits existierender Forschung in diesem Gebiet. Anschließend folgt eine Zusammenfassung einer externen Studie, die zu einem frühen Zeitpunkt meiner Forschungsarbeit durchgeführt wurde. Als Nächstes folgt die Entwicklung von drei Prototypen, die im darauf folgenden Kapitel getestet und verglichen werden. Als Letztes folgt eine Zusammenfassung und ein Ausblick auf weitere mögliche Schritte.

Acknowledgements

I thank all developers who participated in my study and I want to thank my parents for supporting me during my whole studies, their considerations and endurance. Next I want to thank my girlfriend who had to be able to spare me from time to time.

Chapter 1

Introduction

Based on the work by Lukas Sychalski “Communication Of Source Code Designs Through Sketching” [2013], I came up with the idea of integrating sketch functionality into an integrated development environment (IDE) and connect sketches to a particular version of a source code entity.

Figure 1.1 shows a brainstorming of functionality that should be further studied and combined in order to develop a software prototype for sketch-supported source code history exploration.

I performed a
functionality
brainstorming

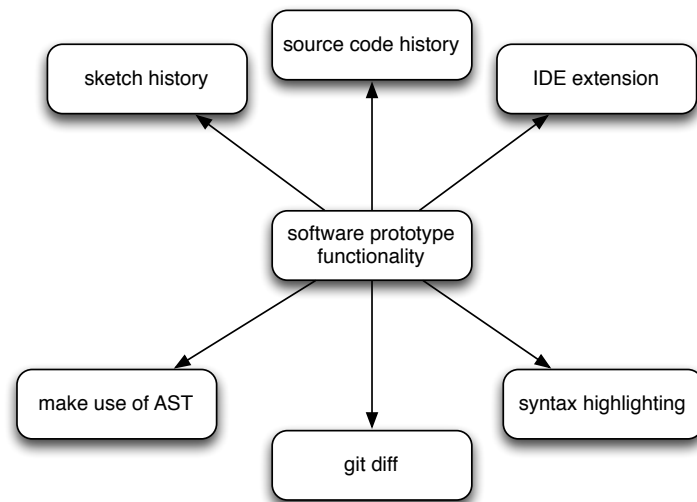


Figure 1.1: Software Prototype Functionality Brainstorming

1.1 Motivation

My work is based on
“Communication Of
Source Code Designs
Through Sketching”

During the process of finding a thesis topic, I participated in a user study, conducted by Lukas Spychalski. Thereafter I came up with some thoughts on how to improve his work, forming the basis for my thesis. What follows is a list of ideas/questions that arose.

List of upcoming
questions after I
participated in a user
study

- How can sketches be drawn naturally, without technical limitations, but still offer possibilities of their digital counterparts?
- When was a sketch committed and which version does it belongs to?
- Who removes a non valid sketch?
- How can a sketch be integrated into source code history?
- Is there a possibility for detecting a non valid/relevant sketch automatically?

- How can the process of connecting a sketch to a source code entity or vice versa be optimized?
- Is it possible to attach a prototype into a widely used IDE?
- Why does committing sketches with code not suffice?

ENTITY:

According to Godfrey and Zou [2005] a “software entity” can be a “... function, class, or file [...] that occurs in a particular version of a software system”. Or an even smaller part like “a declaration or body part of a method” [2007].

Definition:
entity

Two of those questions were embedded in the following paragraphs, revealing problems and needs of someone exploring the source code history.

When was the sketch committed and which version does it belong to? This question would be asked by someone, who is unfamiliar with a repository. Let us call this someone Dave (the developer) and imagine the following situation. Dave looks through an unknown repository for the first time and finds some image files containing sketches.

Developer Dave is confronted with an unknown repository

SKETCH:

“...simplified and structured visual representation that show entities and relationships representing the architecture or implementation of a software system.” Cherubini et al. [2007]

Definition:
sketch

He observes functionality related to the project and the timestamp reveals that they are one-year-old.

The meaning of all identifiers in the sketches and their link to other files remain indistinct in this early phase.

Dave performs a checkout of the commit to which one of the sketches was added, with the intention to find the files and identifiers.

Definition:
commit

COMMIT:

A commit [Chacon, 2009, p.5ff] represents a snapshot of files (here source code files) and is unmistakably identified with a SHA-1 (secure hash algorithm, whereas 1 indicates that it is the second iteration of this algorithm). A SHA-1 is calculated by using a file or directory structure and consists of 40 hexadecimal characters [Chacon, 2009, p.6]. In a single project SHA's are often referenced in a shortened form with a minimum of 6 characters instead of 40.

After failing to find any of those, he only examines the source code.

Sketch was added to
an "unsuitable"
commit

The reason that Dave could not find any identifiers is that the sketches had been added to an "unsuitable" commit. The sketch includes functionality and identifiers that were already deleted or renamed in the source code. Someone decided that the sketches contain worthwhile information and added them to this "unsuitable" commit, but did not check for validity of the identifiers. Although the identifiers have been renamed the sketch still makes sense.

A connection
between sketch and
source code file is
missing

What is also missing are connections from the sketch file to source code files, beside the visual representation in the sketch.

If the sketch had been added to a commit together with the source code files, where the identifiers match, it would have become more explanatory and maybe he would not have given up. But just adding sketches to commits seems to be a non ideal solution.

If there had been a textual hint in a documentation file, it would still require interpretation and manual maintenance.

Linking a sketch to a certain revision of a source code entity is bijective and would be an improvement.

Establishing this connection, should be as simple as dragging a connection from the sketch to the line(s) of interest or vice versa.

Who removes the sketch, if it is no longer valid?

Another problem about a connectionless sketches is that you do not notice when it becomes less important. The file has to be opened and the sketch reinterpreted. Next it has to be checked manually if the identifiers referring to source code entities still exist.

When does a sketch becomes invalid?

In the source code that was used in my user study (see chapter 5), I found a comment line which suffers from a similar issue:

connection issue

```
// ^^ I don't know what that means anymore :(
```

This line was introduced in an early commit in 2008 and still can be found in the current commit. The line it referred to was long gone, but nobody deleted the above comment. The issue is that both the *sketch to source code* and the *comment to source code* situation suffer from a missing connection. For both situations there exist circumstantial solutions to recover.

remaining artifacts

Issues to tackle I conducted an initial study (see chapter 3) and a subsequent prototype should tackle the following:

- Attachable to a widely used IDE (e.g., Xcode, Android Studio, Visual Studio), no additional tool/IDE
- Easy to use, i.e. a few mouse clicks should enable user to connect a sketch to source code entities of a particular version
- Sketches from whiteboard, notebook or any other medium should be usable, without redrawing and without much extra effort
- Meta files, listing the connection between sketches and files, should not clutter the working directory

List of issues to tackle in the implementation

1.1.1 Attachable to widely used IDE

Sketch capabilities should be offered without extra software, which was a requirement mentioned in an initial

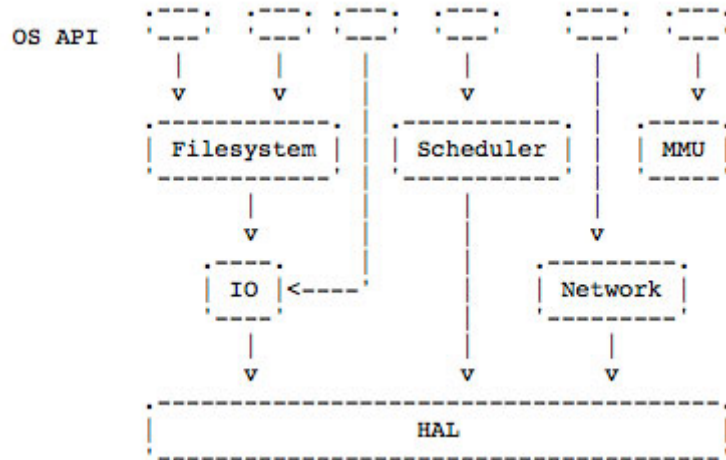


Figure 1.2: The figure shows an ASCII sketch of the operating system API and its layers created with GUIIO plugin for Microsoft Visual Studio

Attachable into IDE
(e.g. Xcode, Android
Studio, Visual
Studio), no
additional tool/IDE

study (see chapter 3). My goal was to develop a plugin which can be attached to a widely used IDE and no additional software is required. Simpson and Terry [2011] used the American Standard Code for Information Interchange (ASCII) to draw sketches inside code, trying to dispose of the requirement of additional sketch software (see Fig.1.2). I anticipate that this approach closes the connection gap, because the sketch can be placed directly to the position where it belongs, without depending on additional files. It requires a plugin, which was developed for Microsoft Visual Studio and it is restricted to a fixed set of UI components (buttons, windows etc.) that are converted to ASCII. It still misses the freedom of natural drawing and limits the user to predefined UI elements.

eclipse is frequently
used in research,
whereas Xcode is
rarely used

The plugin was integrated into Xcode [2014e], because I am most familiar with it. IDE's like eclipse are often used and extended in research studies and Xcode is less common. For example, Azurite [2013b] is an eclipse plugin and will be discussed in 4.2.2. Entering the keyword *eclipse* in the digital library of the Association for Computing Machinery ACM returns ~13.000 results, whereas for example the keyword *Microsoft Visual Studio* returns 4.464 and *Xcode* only

434.

As a source code language Java is often the language of choice (just like Azurite [2013b] and Chronos [2013], see 4.2.2 and 4.2.3). Again the ACM results return 100.147 entries for *Java* and only 1.182 for *Objective-C*. The main reason for those results is may be the fact that eclipse is open source and Java often used in programming classes, although the tools and languages are equally powerful. I decided to use Xcode and Objective-C, because I wanted to further explore those relatively unknown IDE's and languages in respect of sketches and source code history research.

Java is the language of choice in research

I was curious if it is possible to extend Xcode with my own functionality and thus I explored the prior research about extending it, which is subsequently described.

Is it possible to extend Xcode?

Apple does not formally support plugin development for Xcode and there is no official source showing how to do it. Two drawbacks arise from this fact. The first one is that whenever Apple updates Xcode it is possible, that a plugin developed for Xcode stops working. The second drawback is that there is no public documentation about the internals of Xcode, which makes it hard to attach a plugin to it.

There is no official documentation on how to attach a plugin to Xcode

It was developed respecting the fact that the plugin might not be usable in a further version of Xcode, so less programming effort is needed to integrate it into open source editors like *Textmate 2* [2012].

CodeShape was built in such a way that it can be used with other tools, beside Xcode

As Apple does not provide any information on plugin development for Xcode, I found websites like *nshipster.com* [2014] and *maniacdev.com* [2014] that attested the existence of third party plugins for Xcode and provide hints on how to implement one on your own [2013].

I studied the source code of some plugins in order be able to implement and integrate my own plugin into Xcode. Next I implemented a test plugin, which worked inside Xcode. After I had been assured, that it was possible to write a plugin for Xcode, I had started to sketch some prototypes incorporating the required key points of my initial study.

Plugins can be attached to Xcode

1.1.2 Usability

easy to use, i.e. few mouse clicks should enable user to connect sketch to source code

It was emphasized during the initial study (see chapter 3), that connecting a sketch to source code should be possible with only a few clicks. This aspect was also respected during the process of developing a prototype.

Keep mental effort at a minimum

On the one hand we have a set of sketches and on the other hand we have a set of source code lines. In order to connect a sketch to some lines the number of clicks and the mental effort should be as low as possible.

The source code lines and the sketches should be arranged in a way that they are visible together, so that the user is not required to remember the lines or sketch selected.

It would be possible to connect both entities, source code and sketch, with keyboard commands. This has the disadvantage that one has to type a range of line numbers, which identify the source code lines that should be connected with a sketch, which is prone to error.

Using a keyboard to connect sketches to line numbers is prone to error

A user can mistype the line number, which can only be detected by comparing the line number with the typed one. No feedback is given, because only the user knows, if the line numbers are correct.

Typing a range of line numbers requires additional mental effort, but the user is only interested in the source code, not the line number.

Provide guidance and feedback to reduce error

The conclusion is to offer some guidance and feedback during the process of connecting sketches to source code, in order to keep mental effort and possibility of error at a bare minimum.

sketches from whiteboard and notebook should be usable without redrawing them and without much extra effort

Sketch Digitizing Developers draw sketches on whiteboards and notebooks and less often digital ones [2007]. This was confirmed during my initial study (see chapter 3),

where all offices had a whiteboard attached to the wall and they were used on a regular basis.

I also found evidence that sketches are drawn into notebooks or on a piece of paper. These sketches are not digitized, because additional effort is required. In case of the whiteboards it was mentioned that they sometimes take a photo of them and store the file in the project folder.

digitizing
hand-drawn
sketches requires to
much effort

The challenge with sketches is to reduce the extra effort required to digitize them, but to not restrict the drawing process. Drawing should be as natural as before, without or a minimum of technical limits required for digitizing.

1.1.3 Minimum of Additional Files

Project directories already contain a lot of files: “source code, interface, documentation and several other”. The requirement for the software prototype is to keep the number of additional files, which are needed to connect sketches to source code, as low as possible.

sketches and files
listing the
connection should
not clutter the
working directory

To address this requirement it needs to be considered how to hide and separate the files, which are needed to record the connections that have been established.

Chapter 2

Related Work

2.1 Exploring Source Code History

I will look at existing approaches to explore source code history to integrate sketches and link them to a source code version. Several GUI's have been proposed. Bradley and Murphy [2011] compare two interfaces, Deep Intellisense [2008] and Rationalizer [2011].

Deep Intellisense vs
Rationalizer

Deep Intellisense presents historical information in different views separated from the source code view.

Structural elements under the cursor can be compared, whereas the most specific one is chosen. Elements are for example methods, fields and class declarations.

Deep Intellisense
considers structural
elements under the
cursor

Deep Intellisense uses source code history, bug reports, emails or other documents related to the source code element.

Opposed to display information in separate views, Rationalizer displays information integrated into the source code view. Three columns, "When?", "Who?" and "Why?" are added to the right hand side of the source code. Information is line based instead of element based and the "When?"-column tells the modification date of a line and the "Who?"-column specifies the author responsible. The

Rationalizer
integrates
information columns
in the source code
view

	<p>"Why?"-column displays the check-in note of the corresponding revision (a bug report is displayed instead, if available).</p>
<p>Rationalizer in breadth vs Deep Intellisense in depth information display</p>	<p>Rationalizer extends information in the breadth, displaying them together for all lines, whereas Deep Intellisense displays information in depth of one source code element.</p>
	<p>Their results show that each tool has its strengths and weaknesses depending on the task, also user satisfaction does not clearly state a winner. The idea of integrating information into the source code view is adapted in my versioning approach. In the source code view of CodeShape (see 4.2.1) the user can interact with lines, while the mouse cursor switches to a hand cursor (see Fig.4.18 and 4.2.1)</p>
<p>A longitudinal study measured the number of source code revert actions</p>	<p>A longitudinal study conducted by Yoon and Myers [2014] measured the quantity of reverting source code to an earlier state. It involves exploring the history, in which the version or lines of code to revert to, have to be found.</p>
	<p>The process of reverting to an earlier version is called <i>backtracking</i>. They analyzed 1,460 hours of fine-grained code editing logs, gained from 21 participants. Fine-grained means that every character-change, all copy & paste actions and delete operations are logged.</p>
<p>15,095 backtrackings in 1,460 hours have been observed eclipse plugin with selective undo</p>	<p>In total 15,095 backtrackings have been performed, which results in an average rate of 10.3 per hour. A backtracking can span a character change up to thousands of characters.</p>
	<p>Their eclipse plugin provides selective undo. Meaning the user can pick parts that should be undone and not selected parts are left untouched. It offers more flexibility opposed to ordinary undo.</p>
<p>ratio of different backtracking types</p>	<p>In 34% of all recorded backtracking instances, backtracking was performed manually by deleting or typing code. In another 20% a sequence of changing code, running it and later backtracking it, was observed. The selective undo feature was used in 9,5% of the cases.</p>
	<p>As a result developers backtracked every six minutes on</p>

average, including ordinary undo. The undo statistic does not include minor typo corrections, as the data was cleaned beforehand. Programmers need better backtracking tools, because of situations which are not well supported by existing programming tools.

2.2 Source Code Visualization Approaches in Terms of Time and Space

A large amount of different source code visualization approaches exist. Most of them constitute the visualization based on statistics calculated from existing source code. The tool Seesoft by Eick et al. [1992] analyzes each source code line by its importance.

Source code visualization is often based on statistics

Each line is represented as a row and colored according to its age in rainbow color. New lines are colored red and the oldest lines are colored in blue.

Each row is stored in a column, representing the height of a file the row belongs to. The y-coordinate of a row represents the relative position of the row in the column/file.

This visualization technique is similar to Azurite by Yoon et al. [2013a], where a column represents the source code file and a row represents a source code line. The rainbow color represents the time dimension and the y-coordinate the space dimension.

Seesoft's rainbowcolor represents time dimension

Azurite focusses on more fine grained changes, where every character change is taken into account. A rectangle represents the number of characters involved.

Azurite colors the change according to its type, instead of coloring a change in rainbow color. Green means one or several characters have been inserted, red means one or several characters have been deleted and blue means that at least one character was updated.

Azurite's character based changes are colored according to its type

The length of a rectangle representing the change equals

the number of characters involved.

The y-coordinate positioning of Azurite is equal to Seesoft, except that the height of the container (represents a file) stays fixed in Azurite. In addition the x-coordinate of a rectangle is defined by the modification. The x-axis represents a timeline of changes.

Chronos utilizes
History Slicing

Another implementation is called Chronos by Servant and Jones [2013] and is based on the technique of history slicing by Servant and Jones [2012]. Chronos provides a horizontal timeline, where every history slice is positioned according to the date of a commit beneath the timeline.

In contrast, to Seesoft and Azurite, the lines which have changed or are new in a commit are highlighted in blue. It is not differentiated between an addition or an update, whereas deletions are not visualized at all.

Chronos allows for
comparing history
slices close together
in time

The GUI can show several history slices side by side, if they are in dense chronological order. This means that two revisions of a file must be close to each other in time, which would be at most some days in between two revisions. If there is a year or more in between, they both would not fit together on the screen of an ordinary monitor, which was observed during my user study (see 5.3.6). It is hard to compare two versions, if they do not fit on the screen.

Chronos misses
compressing of
slices

The compact mode in Azurite removes time intervals, in which no changes have occurred (see 4.2.2). It compresses all revisions together. Chronos has nothing comparable.

History Slicing's
selection allows
most degrees in
freedom

As opposed to tools that work on a per file basis using diff [1976], the user can select slices spanning one or several files of a revision or some lines of files. Compared to Azurite, Deep Intellisense and SeeSoft, Chronos offers the most degrees of freedom in terms of selection.

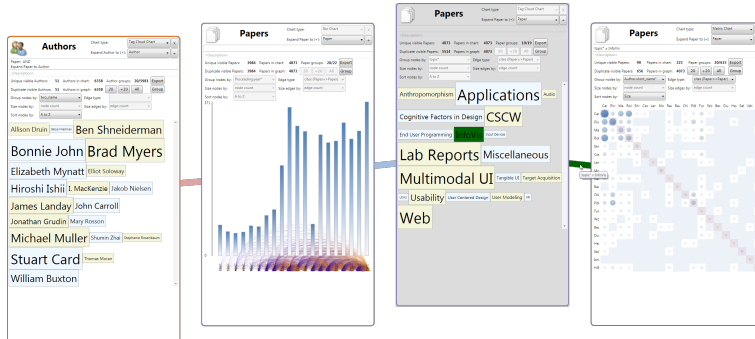


Figure 2.1: Image section of GraphTrail showing some aggregated papers and their statistics like the frequency of keywords

HISTORY SLICE:

“We model in history slicing the process through which developers select the subset of the history of the software project which is relevant for the lines of interest. The history slice for a set of lines of code of interest (i.e., slicing criterion) contains all their equivalent lines of code in all the past revisions of the software project in which they were modified. The goal of a history slice is to provide a reduced amount of information about the history of a set of lines of code Servant and Jones [2012].”

Definition:
history slice

GraphTrail [2012] takes a completely different approach. A user interacts with a canvas and is not restricted to source code entities, but can place anything e.g., research papers on the canvas (see Fig.2.1).

GraphTrail is not comparable to previously mentioned approaches

Research papers can be aggregated and one can choose between several diagram types representing e.g., citations between the aggregated set.

User interaction is persisted and visited elements are annotated with a star. As a result a visual history exploration can be performed, where one can follow a path connecting aggregated entities.

2.3 Source Code History

Bird et al. [2009] compare SCM (source code management) systems like svn with DSCM (decentralized source code management) systems like git [2014a]. DSCM's contain more history information compared to SCM's.

No matter which one is chosen, they build the basis for source code history exploration. As they provide the mechanisms to take snapshots of the current working directory, containing source code or any other files. The snapshots together form the history of a project and SCM's and DSCM's allow to interact with this history.

SCM's contain less information

The history of SCM's usually contains only validated code, without incorrect beginnings or experiments as opposed to DSCM's, which could contain code that will never be merged into the stable version. As a result git's history contains a larger amount of information.

git allows for changing the history without traces

The authors address git's possibility to alter history for example with the *git rebase* command. *git rebase* enables to modify history without traces. The reason is that a commit does not contain the information in which branch it has been created and the fact that *git rebase* command itself is logged only in the private log of a developer.

They also describe the use and evaluation of history information can indicate the progress of a project and analyzed the history of source code during a semester project [2004]. In this case the history information was gained from the Concurrent Versions System (CVS) of each group.

Is there a correlation between grade and degree of collaboration

Goal of analyzing this history information was to find out if there exists a correlation between grade and degree of collaboration. By creating diagrams based on historic information the functioning and amount of work was compared. As a result only one team out of five constantly worked on their project, whereas the rest cumulated all their work prior to a deadline. The authors conclude procrastination causes poor software design.

An automated approach of analyzing source code history based on CVS data is performed by Marmoset [2004]. Marmoset is an eclipse plugin, which automatically creates snapshots of a project, when one saves or adds files to the project, instead of manually evaluating the student information as in the scenario above[2004]. Marmoset users can upload a repository to a central server, and get feedback about how far away the project is from being finished. The server compiles the project and tests it with JUnit, statical analysis, a bug finder and a style checker and returns the results.

2.4 Code Understanding & Knowledge

Fritz et al. [2007] as the question: “Is the activity performed by a developer useful to determine the knowledge he has about a source code basis?” Based on how often and contemporary the source code was visited, significant differences could be determined. They conducted a user study with 19 professional Java programmers. To ascertain knowledge they used an eclipse monitoring in combination with questionnaires, after a certain amount of interaction was performed. An interaction is the selection and edit of source code, open/close of editors, windows and perspectives. The frequency and recency with which parts of the source code have been visited indicate the knowledge a programmer has about it.

Fritz et al. conducted a user study to determine level of knowledge about source code

In a study conducted by Hansen et al. [2013] it was evaluated how experience and notation influences comprehension of a developer.

He tested about 150 subjects with an average python-experience of 2 years, using 10 python programs with mostly less than 20 lines each and a cyclomatic complexity [1976] of 8. The cyclomatic complexity describes the number of linearly independent paths, which are less than 8. Accordingly the programs are well understandable.

How much is comprehension influenced by experience and format of source code?

Vertically and horizontally formatting the python code influences readability and interpretation.

vertical and horizontal changes in format affect readability and interpretation

In terms of vertical formatting, line breaks have the effect that code fragments are mistakenly considered as not belonging together, whereas blank lines increase readability. By determining if a line belongs to a loop, the vertical positioning of a line largely influences the interpretation.

In consideration of horizontal formatting, blank characters between arithmetic operators affect the interpretation of execution sequences.

Also knowledge level influences code understanding. In situations solving specific errors, experience helps, but experience can also be a disadvantage in unusual situations.

If a standard operator like “+” is overloaded, an experienced programmer expects standard behavior and not that it was overloaded, which often leads to misinterpretation.

If something behaves the way it is expected, an experienced programmer can more efficiently interpret reoccurring patterns.

plugin to
automatically
generate source
code examples

Ginosar et al. [2013] developed an IDE extension for Processing, which is capable of creating multistage code examples. Multistage means that several consecutively versions of a code example form an example with increasing complexity. More and more concepts are incorporated from stage to stage. This approach is used in programming books, tutorials and online videos, but the creation of such examples is tedious. Their extension should assist authors to create multistage code examples. The idea of back-propagation of new changes to older versions of a file can be beneficial in order to understand old versions. Back propagation can be used in case there is a new sketch attached to a new source code entity. An old version of this source code entity would use the identifiers used in the new the sketch, because they would be back-propagated from the new source code entity. Same holds true for the other direction and thus back-propagation would ensure consistency of source code and sketch.

2.4.1 Mining

In contrast to human code analysis and understanding, as described in the previous section, mining deals with machine interpretation. Static, dynamic and automated procedures try to understand source code, comments or identifiers. Vinz and Etzkorn [2008] combined all those procedures.

In their approach, comments and identifiers are tokenized to get domain knowledge, which can contain human concepts and cannot be gained from ordinary source code analysis methods.

Only half of the available information is used by just identifying comments and identifiers and they combine the result of code analysis and tokenization for improvement.

Due to the combination the *code to comment metrics* can be calculated, indicating how much a comment concept matches a source code concept, but it only considers comments with a distance of at least 5 lines to a code range.

code to comment
metrics try to match
their concepts

Another automated approach is carried out by Sourcerer[2008], which analyzes open source repositories in comparison to each other.

The user benefits from identifying, exploring and reusing existing open source implementations with improved software search and retrieval performance.

All mining approaches have in common that they can only filter information and make assumptions, which match certain patterns. The metric cannot be calculated, if for example the distance in the *code to comment metrics* is too large or the comments can not be found at the anticipated location. In these cases they are not considered and bias the result.

if something does
not match a pattern
it is not considered

2.4.2 Refactoring Detection

Refactoring detection is another automated approach, besides mining and attempts to gain knowledge from source code by automatically applying rules to detect refactoring.

In his dissertation about historical data from source code revision histories, Williams et al. [2006] describe a tool that was implemented to detect refactorings.

The tool was tested on student projects, apache, httpd and wine. The number of detected refactorings for each of the projects is given, but it is not said if this number is equal to all refactorings that occurred.

market basket
problem

Annie Ying delved into association rule mining on a per file basis. The idea is based on the *market basket problem* [1993] [2002]. The problem occurs when an ordinary consumer buys a product in a supermarket: “When a customer purchases item x , the customer is likely to also purchase item y ”. One tries to determine all association rules with the form $x \Rightarrow y$, lying above a given threshold. This scenario is transferred to source code modification and it is determined how likely it is, when a source code artifact x is changed, another artifact y is changed, too.

Zimmermann et al.
try to detect
co-changes

Zimmermann et al. [2004] follow a similar but more detailed approach, but more fine grained. They explored which files/functions also have to be edited after modification to one file/function Zimmermann [2009].

Their tool ROSE can predict some files and functions that have to be changed also, after initially changing a file. This is done by mining all changes for rules. Similarly to Ying et al. [2002] their rules have the following form: “code entity a implies that b and c have also to be changed”.

2.4.3 Whiteboard and Sketches

My work is based on “Communication Of Source Code Designs Through Sketching” [2013], which has its main focus

on sketches, as ideation tools, depicting several levels of abstraction and which support communication about source code. The value of sketches for developers has been stated in the work and thus I will not focus my main attention on it.

In the study conducted by Cherubini et al. [2007] the use of drawings in software development is evaluated with developers from Microsoft. The authors analyzed the quality and quantity of sketches on different drawing surfaces (e.g. whiteboard, paper, notebook, drawing tool) and during different scenarios/phases of development, as well as *how* and *why* sketches are drawn.

frequency, quality and quantity of sketching is measured

The content of sketches can take many forms, e.g., class inheritance, data flow, flow charts, state machines, sequence diagrams, database tables and relationships between servers and clients, etc. Price et al. [1993]. Rarely modeling languages like UML are used, which confirms to the statements in my initial study (see 3).

UML can be rarely found in sketches

It was observed that developers use their source code editor for design in favor of sketches on a physical medium, like whiteboard or paper, even it is considered less effective [2006a]. This was also observed in my initial study (see 3), where the developers favored IDE tools compared to hand-drawn sketches. Whiteboards are used for creating for example presentation brainstormings and during their daily meetings. Instead of sketches, wireframes are used for discussion and implementation of a project.

Developer prefer IDE tools as opposed to free-hand sketches

Four different reasons for creating sketches[2006a]:

1. Sharing thoughts with others
2. Grounding circumstances which can be ambiguous without a sketch
3. Manipulating cognitive processes by externalizing mental model
4. Free “space” for other thoughts and brainstorming which can lead for example to new ideas

whiteboards are
widely used

Except for customer presentation, office whiteboards predominated as a medium to draw sketches on, in all scenarios (understand, ad-hoc, refactor, design review, on boarding, 2nd stakeholders, customer, hallway art, documentation), which have been identified in their survey carried out with 350 Microsoft employees.

sketches are drawn
to improve code
understanding

In chapter 16 “I Don’t Understand the Code Well Enough to Change It” of the book by Feathers [2004], the author encourages the reader to sketch simple diagrams in order to get an understanding of the code and mentions similar aspects like the above four different reasons for creating sketches by LaToza et al. [2006a]. He also states that basic shapes like lines and blobs are sufficient and UML is not necessary. If someone else is trying to understand the source code, discussions based on sketches facilitate code understanding.

sketching takes
place in
collaboration

The idea of working together on and with sketches was taken up by Sangiorgi et al. [2012] and extended to support collaboration. They implemented a sketch collaboration tool called Gambit. Several people can use their preferred device (tablets, mobile devices, graphical tables etc.) to draw sketches together. A projector visualizes the sketches together on a canvas. The advantage of such an approach opposed to IDE tools is, that a user is not restricted to a limited set of shapes. A disadvantage is that a hardware setup, including a beamer and a canvas is required. Another disadvantage is that tablets and mobile devices still cannot imitate the naturalness of pen and paper. Pen and paper

are usable everywhere and can be used instantly, whereas in the Gambit approach certain conditions have to be fulfilled to be able to use it and setup is needed. For transient ideas coming to mind during work Gambit seems to be non ideal.

Chapter 3

Initial study

To get in touch with work processes of a company in the mobile IT sector, I visited a company based in Dortmund. The main company is operating internationally, has 1112 employees and yearly sales of 120,5 Mio EUR.

I started the day with a meeting I had with the head of application development, holding a PhD in computer science, *Henry*, the team leader of iOS development, *Iva* and the team leader of Android, *Aaron*. In the next meeting I talked to *Donna* of the design team, who shares the office with other designers. Afterwards I attended a developer team which works together on a project, starting two years ago. Next I could join the developers daily meetings, which are separated by platform respectively. After that I was introduced to *Quinn*, who is responsible for the quality management. Last I encountered the development assistant *Dean*.

I visited a mobile IT company with 1112 employees and yearly sales of 120,5 Mio EUR

During my visit I could talk to representatives of all departments

3.1 Meeting with Development Heads

During the meeting with Henry, Iva and Aaron I presented the idea and topic of my thesis and my motivation to visit the company they work for. My intention was to get some insights on the work processes they go through during their

motivation and thesis topic explanation

daily work. The focus was on the use of sketches and to get answers on the questions I noted. I brainstormed some questions before the meeting (see Appendix F). These questions served as a reminder. What follows are some of the questions I asked and the corresponding answers.

Every office has a whiteboard in use

“Are sketches drawn and when, what they are drawn onto?” Henry told me that they do use whiteboards to draw sketches, but he thinks that they could make more excessive use of it.

Henry said that every office has a whiteboard mounted to the wall and the number of workers in one office lies somewhere between two to eight.

In this context Iva mentioned “whiteboards, capturing input with cameras” and “it would be nice if they had any of them”.

pictures of whiteboard sketches are taken

“Are sketches persisted somehow?” They sometimes take pictures of the whiteboard sketches and archive them in a repository, being separated from the development repository, where mainly source code is stored.

Connecting sketches must be possible with a few clicks

“Does linking sketches with a certain state and part of source code or a particular file makes sense?” All three agreed. Iva said that a tool that enables you to connect a sketch with a source code file in a certain state, must be easy to use in an IDE. “It should be possible to connect a sketch with one or two clicks.” “The best would be if it is as easy as connecting IBOutlet Connections in interface builder” (see 4.1.4). The initial training of new developers probably would have been easier, if they had access to sketches linked to source code.

new developers could benefit of sketches linked to source code

A sketch must be connectable to any source code entity

“Should you just be able to link per method?” Henry, Aaron and Iva answered that it should be possible to connect a sketch to an adjustable part of a source code file. It

should be possible to link a sketch to a complete file, several methods or a range of lines.

“What do you think about drawing sketches on an iPad with an input device?” They could imagine to draw sketches with it, but they would prefer freehand shapes instead of sophisticated “limited” modeling languages like UML. Aaron added “...a global overview of the whole infrastructure” of a software project with classes and all its connections “there an iPad screen size is too small” and would prefer to draw such global views on a whiteboard. Iva: “drawing a sketch on a device like an iPad makes only sense in terms of sketches related to methods or parts of a source code file” and the others agreed on it. They mentioned digital pens, recording the shapes you draw, but they do not use them yet.

The screen size of an iPad does not suffice for an overview sketch

digital pens are not used

“Do you also draw database schemes?” Iva: “if a database scheme is required, we use the provided IDE tools, like Xcode’s internal tool to build up the core data model and our schemes are not complex, so that no sketch iterations are required.”

database schemes are directly produced in an IDE

3.2 Meeting with Design Team

In a meeting with Donna of the design team, she told me that their main work consists of producing wireframes, which are iteratively optimized in consultation with the developers (see Fig.3.1).

Definition:
wireframe

WIREFRAME:

“...wireframes are simply a representation of the skeletal structure of a mobile application, very often compared to a building’s blueprints. Being an application’s backbone, they lay out the structure, hierarchy and relationship between the elements that make up a mobile application. The intention of these structured drawings is to focus mainly on what the screen does, and not exactly what it looks like. Wireframes are supposed to lack color, graphics, or typographic styles; they are not meant to be viewed as final designs and are certainly intended to be a part of an iterative process. Mobile application design could be a long process, and wireframes play a key role in defining the structural foundation of the product, making it easier to understand and refine in the long run.”
(Definition by [Robles, 2013])

wireframes are
iteratively optimized

Donna crafts these wireframes with her colleagues and hands them over to the developers. If something remains unclear to the developers or something is impossible to implement, a new iteration of wireframes is produced.

3.3 Meeting with Developer

I had the chance to talk to a developer team of eight, consisting of two iOS developers, three Windows Phone developers and three Android developers. They shared one office with a whiteboard mounted to the wall.

Linking sketches to
source code must be
possible with all IDE’s
and without
metadata

If the developers work on the same project they usually are settled in the same office, which should facilitate communication. The development ranges over three mobile platforms: iOS, Android and Windows Phone. The number of developers depends on the size of the project and on the platforms the customers want to have supported. A tool which is capable of linking sketches to source code must run on all platforms. One of them said “I do not want to have the software repository cluttered with additional metadata”, referring to metadata as a byproduct of linking.



Figure 3.1: Example wireframe of a mobile app, which underwent some iterations and already contains color

For the first development step they use the documents produced during an initial customer meeting and wireframes of the design team.

They told me that during the entire development process they do use and annotate wireframes. These differ from platform to platform, because of the native UI elements. As an example, they mentioned the different Navigation elements of iOS compared to Android.

In case the wireframes are not sufficient and to ease com-

wireframes are the basis for discussion during the entire development process

Besides wireframes IDE visualization are used

munication with other developer teammates, tools like Xcode’s Interface Builder and Storyboard or Android Studio’s Design View are used to discuss development questions.

The developer team rarely uses the whiteboard. Some of them have a notebook to draw sketches into, focussing on wireframes.

Database models are
directly conceived in
the IDE

One of the developers said “if the database model is complex, something went badly wrong”. He added “our schemes are very simple and the main data source are JSON files, requested from a server”. They directly use IDE tools to come up with a database model. Hence, neither they draw database models on the whiteboard nor sketch it in a notebook.

UML is not used by
the developers

Concerning the question if modeling languages like UML are used in their sketches, they said that they do not use UML, rather free-form shapes connected by lines.

conservative
towards new tools

The developer team would like to keep using the tools they got accustomed to and have been skeptical about using an iPad and about linking sketches to source code in general.

3.4 Daily Meeting

platform dependent
daily meetings

During daily developer meetings, which are separated by platform i.e. Android meetings are lead by Aaron and iOS meetings are lead by Henry. A whiteboard is used to keep a statistic of what each of them was occupied with and for how long.

A meeting starts with Henry and Aaron asking about any difficulties being encountered, what they have been working on and what has been finished.

3.5 Meeting with Quality Manager

I talked to Quinn, who also appreciates linking sketches with source code. He could even imagine to use an iPad for sketching.

Quinn commented a drawing on the whiteboard in his office: “this was drawn for structuring an external presentation”. It visualizes a brainstorming, in which connected bubbles guide the presentation flow.

He was the only person I talked to, using UML. He uses MagicDraw to build UML representations of the system architecture and could imagine to use a drawing device like a pen for the iPad for sketching.

Only person using
UML

3.6 Meeting with Development Assistant

At the end of my visit, Dean presented some general information about the company, how customer meetings and projects proceed and which customer orientation they have. Developers sometimes attend those customer meetings, but at least the head of each platform is present.

3.7 Summary

Visiting the company in Dortmund revealed the workflow of mobile developers and other individuals involved in the development process. My assumption that sketches are used for visualization was confirmed by the utilization of whiteboards, wireframes and sketches. The potential of sketches is not completely exhausted.

Freeform shapes are preferred, as opposed to tools like UML. This was already ascertained in chapter 2. The problem of persisting whiteboard sketches became apparent during the interviews. They mentioned the potential benefit of linking sketches to source code and persisting them.

By persisting them, sketches can still be manipulated when the original whiteboard sketch no longer exists. New developers can take advantage of it.

Chapter 4

Prototypes

4.1 Implementation

This section will deal with implementation decisions. It precedes the 4.2.1 section, because some technical detail is given and it introduces some technical terms, which are used in the subsequent section.

In order to come up with a history exploration GUI, technical requirements had to be clarified.

4.1.1 IDE Extension

As described in the motivation part (see 1.1) I wanted to integrate a GUI plugin to support history exploration into Xcode. The plugin should act on top of Xcode and exchange information with it, extending its capabilities. Xcode already provides some information which are useful for my plugin. These are for example, the project(s) currently open, the source code file visible and of course the paths to each of them. Also the lines of the source code file, being in the visible screen section, can be obtained and are used in my plugin to control the amount of displayed information. The problem with these pieces of information are that they are not open to public and not easily accessible.

I wanted to
implement an Xcode
plugin for history
exploration

class-dump is used in order to get some information detail of Xcode

To be able to attach a plugin to Xcode's internal components, like the source code view, a tool like class-dump [2013] is helpful to get at least some implementation details about the underlying classes. It was a required step to get for example, variable names, because I wanted to be able to access the *Xcode view* containing the source code.

I needed to find a way to access Xcode's source code view

To link something with Xcode's source code view, one has to attach an observer to the notification *IDESourceCodeEditorDidFinishSetupNotification*. When this notification (triggered after Xcode has finished loading the IDE on startup) is sent to the Xcode application, the source code editor's *TextView* becomes accessible via the variable *textView*.

This variable of a *TextView* instance references a *DVTSourceTextView* object. *DVTSourceTextView* inherits from *NSTextView* and represents a class with some extra behavior implemented for Xcode only. *NSTextView* is a class which is publicly usable and documented [2014c]. In general *NSTextView* provides the basic functionality for developers implementing UI's using a *TextView* similar to Java's swing component *JTextArea* [2014]. I needed access to the *DVTSourceTextView* object in order to acquire the source code text, currently visible.

4.1.2 Sketch History

sketch digitizing is tedious

In general a sketches are digitized by taking a photo of it, scanning it or by redrawing it with the help of a computer software. All these forms of digitizing, except redrawing with sketch software, lack the ability to step back in the stroke history of a sketch or it is hard to do so. In case of taking a photo or scanning it, several photo sessions or scan iteration have to be performed during sketching. By using software commonly one has only the ability to undo or redo a limited number of performed steps.

The digital pen Wacom Inkling was used to digitize sketches

Here digital pens and their software comes into play, like for example, the Wacom Inkling [2014], which was used for the sketches in my user study (see chapter 5). The Inkling is a digital pen, tracking every stroke. Besides the pen it has

a small receiver (operating on batteries) that can be clipped at paper. After sketching it can be plugged into a computer, in order to open the sketches (stored in WPI files).

Inkling's file format WPI contains the coordinates of every stroke made *wac* [2014] in a vector format. A stroke entry in a WPI file consists of a stroke pressure value, tilt value, a timestamp and a clock *valuewpi* [2014].

The data in a WPI file is ordered chronological, so the order of "when a stroke was drawn" is set, but a single stroke entry does not contain a timestamp.

The accompanying software shipped with the Inkling is the Inkling Sketch Manager *Wacom* [2012]. This software provides a GUI for browsing all drawn sketches and edit them. An additional feature is the ability to step through every single stroke in chronological order.

Three limitations exist with this software in terms of sketch assisted source code history exploration.

- Again an additional application has to run.
- Inability to link sketches with source code.
- No timing information is used/accessible, except for the creation date of a sketch.

I implemented a parser that can read in the WPI files binary data, with the help of some websites [2014][2014] and got a set of coordinates. These coordinates can be used to draw Bezier curves[2014], which also exist e.g., in Java. With the help of those coordinates I was able to draw the sketch or a thumbnail version of it in every resolution, without the loss of quality.

The binary data of a WPI file contained a timestamp, describing the time, when a WPI file is generated. It is generated, when the first stroke is drawn. Beside the timestamp there was also *clock* data for every stroke entry, which counts, how many seconds have passed, until a stroke was drawn. I used this seconds counter as an offset for the

I implemented a parser for Inkling files

timestamp. This enabled me not to just get the order of all strokes, but also determine, when a single stroke was drawn with an exact timestamp. Now the duration of drawing parts of a sketch can be measured.

obtain a history of strokes

As a result, I receive historic information of all sketches and their strokes. With this information one is able to overcome the above described issues of the accompanying software. The ability to get the stroke and timing information of the WPI file, allows to have access to them from inside the plugin and draw for example, the sketch in Xcode. This provides the first step to link sketches with source code.

Revision control would be possible with the obtained information

A git alike functionality of branching and merging sketches can be built up on the history of strokes, because the basic requirements as described in the work by Chen et al. [2011] are fulfilled. The timing information for individual strokes can be used to further analyze the data. It would be possible to implement the capability of selecting strokes for a time range. As for example, the goal “select all strokes drawn on the 8th of June 2014 between 8 and 10 o’clock and delete them” would be achievable, but I did not investigate this any further.

4.1.3 AST

In order to achieve IBOutlet like Connection’s (see 4.1.4), my first idea was to use a regular expression to determine the screen coordinate bounds of source code entities like methods. These bounds are required to drag a connection from a sketch to a source code entity.

regular expression are not powerful enough

The problem with a regular expression approach is, that it is unlikely to come up with one, matching every possible situation/form a method can be implemented in. If another source code entity will be required in a future version, then a new regular expression has to be constructed.

The method implementation in figure 4.1 is a valid way to implement a method and a regular expression must be able to cope with multi line signatures, which could themselves

```

-(void)aMethod
:(NSString
*)aStringParam
{

}

```

Figure 4.1: One can see an unusual but valid Method implementation. A regular expression matching all unusual situations is hard to implement

contain lines with comments etc. Because of the difficulty finding a regular expression matching the great variety of valid method implementation formats, another approach had to be considered.

A solution to handle the unusual, but still valid implementations of e.g., methods, the AST (Abstract syntax tree) [2014] of the source code document can be consulted to detect the necessary information of the line numbers a method starts and ends.

The AST of source code documents written in C, C++, Objective C and Objective C++ can be obtained by using libclang [2007a]. It is a C interface library that allows for parsing and traversing the AST of source code files [2007b]. The library is able to parse a file containing source code and can output/access the AST of it. One can get the distinct line numbers, where a method implementation starts and ends as well as every valid syntactical element in a file with its type.

To accomplish IOutlet alike Connection's and syntax highlighting (see 4.1.5) I conducted the AST of each version of a source code file. Then, I obtained the line number, where for example, a method begins and ends, which is a required information to be able to achieve IOutlet Connection's.

In order to provide syntax highlighting, tokenization of the AST is required and then from every token the type can be determined and they can be highlighted accordingly.

information of the AST can help to overcome the limitations of regular expressions

Obtain the exact line numbers of e.g., methods from the AST

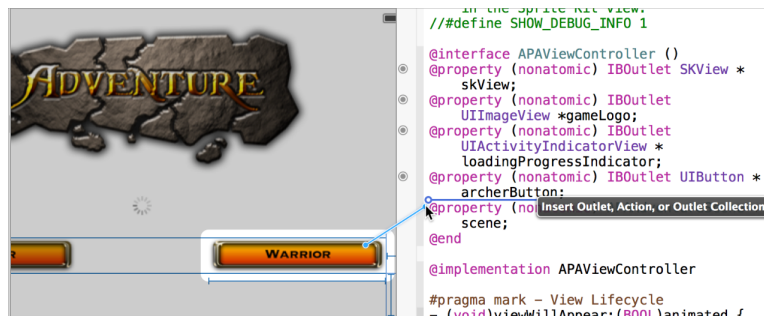


Figure 4.2: The user drags an IBOutlet connection from the button in Interface Builder, labeled with “WARRIOR”, to the source code view on the right. The blue line indicates this connection.

4.1.4 IBOutlet Connection

Xcode’s Interface Builder provides the ability to link interface objects like a Button to outlets in your source code, called IBOutlet connections. This procedure is visualized in figure 4.2. I transferred this idea to the context of sketches and source code versions. As I wanted to connect sketches to source code entities of a particular version, the technique of IBOutlet Connection’s provides a simple way of connecting elements with source code and the visualization guides the user throughout this process. Simplicity in form of a few clicks was a requirement noticed during my initial study (see 3).

I mimicked the behavior of IBOutlet Connection’s implementing them from scratch

My implementation of IBOutlet alike Connection had to start from scratch, because no source code of IBOutlet connections is publicly available. It has the benefit of using my implementation with other IDE’s or text editor’s, without Xcode dependencies.

The implementation required the following steps:

1. Get the mouse position coordinates
2. Convert mouse coordinates between screen, windows and view coordinates.

3. Get the pixel range of e.g., a method from its declaration to the last closing bracket

The items 1. and 2. can be implemented straightforward with Cocoa's provided functionality. "Cocoa is Apple's native object-oriented application programming interface (API) for the OS X operating system" [2014].

The last item 3. deals with *source code character to pixel position conversion*. In order to get the method extension in pixels, I first needed to convert the start-line and end-line of the method to pixel positions. This is possible via libclang. With the help of libclang I can parse the source code and access its AST. Where as libclang's AST feature is able to report the line number(s) of every source code entity, I am interested in [2014] (see 4.1.3 for details).

Line numbers had to be converted to pixel positions

The line numbers, obtained from the AST, can be converted to pixel coordinates and compared to the position of the mouse cursor. This process is depicted in figure 4.3.

As I wanted to imitate the IBOutlet Connection as similar as possible to its original counterpart, a transparent window on top of Xcode was required to draw/drag the IBOutlet connection to every possible position on the screen. The frame of the window is drawn in the sketch of figure 4.4. It should cover the whole screen while being transparent. Xcode and all other apps currently running should thus be "covered".

A transparent window filling the entire screen was required

The transparent window comes to the front and in focus as soon as the user left-clicks and hold on a sketch, triggering the routine for drawing an IBOutlet connection. The starting point for the connection is this particular left-click position and its endpoint is determined by the position where user lifts his finger. With such a design the fact that the plugin should be integrable into other editors is respected, because there is no deeply coupling and dependence on Xcode's core functionality.

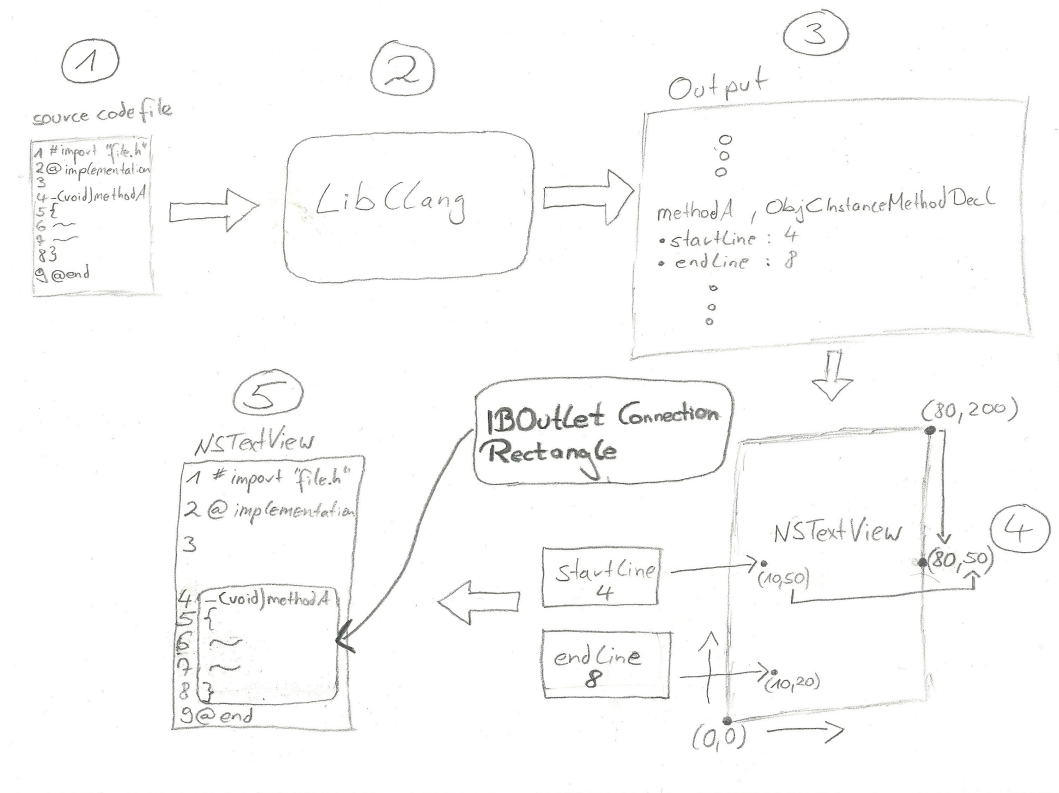


Figure 4.3: Determine IBOutlet Connection Rectangle - On the upper left one can see a source code file. This source code file is parsed with libclang and then libclang's output contains the start- and end line of e.g., the method. In the next step, labeled with 4 on the bottom right, the start- and end line numbers are converted to pixel positions. These pixel positions are used in step 5 to draw a rectangle on top of the method, when the user hovers it during the connection process.

4.1.5 Syntax Highlighting

I had to come up with syntax highlighting for source code entities

To enhance readability of source code syntax highlighting is used, which marks language keywords and other syntactical elements like brackets, in different colors. As I have access to the currently visible version of source code visible in Xcode, I could use the coloring of this version for the identical version in a source code exploration view. This would have of course be possible, but for the other versions of the source code file, which are not visible in Xcode at the moment, my own syntax highlighting is required.

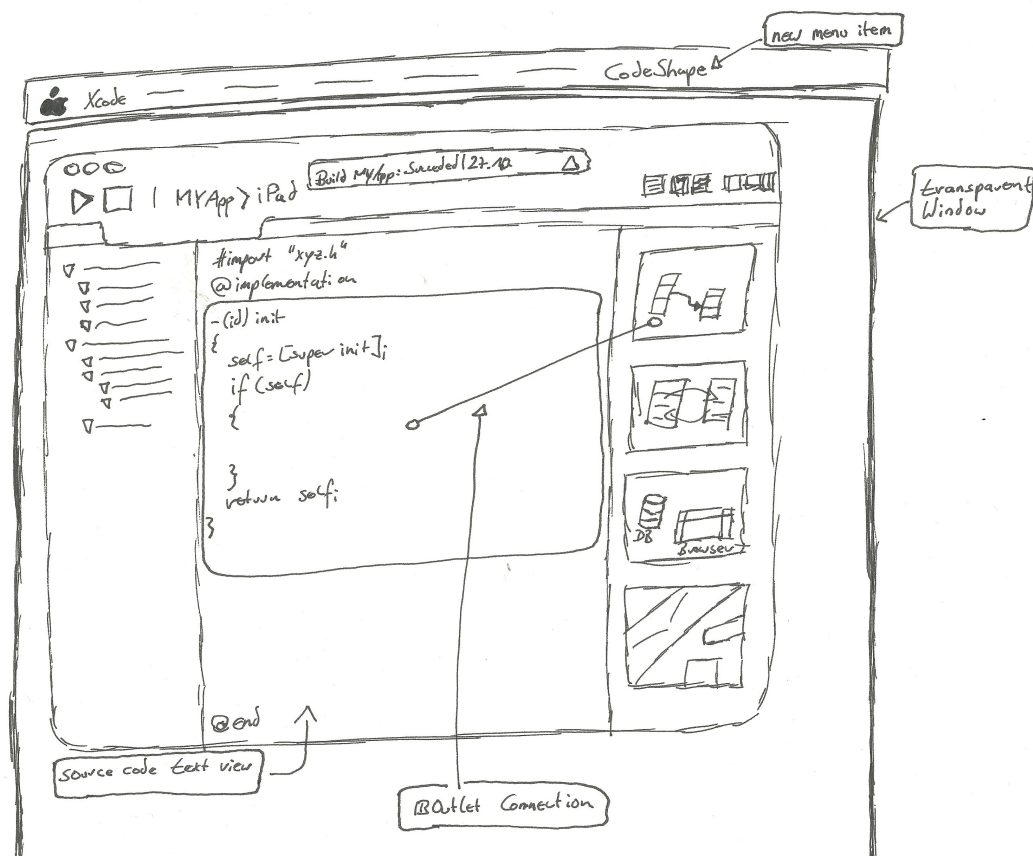


Figure 4.4: Early IBOutlet Connection sketch - The schematic window of Xcode can be seen, extended with a sidebar on the right of the application window, containing sketch thumbnails. An IBOutlet Connection is dragged from a thumbnail to the source code. As the connection endpoint is on top of a source code method, the entire method is encompassed in a rectangle, indicating to which entity the connection will be established. On the top a menu item was added beside Xcode's ordinary menu items. The whole screen is covered with a transparent window, serving as drawing canvas for the IBOutlet connections.

To achieve syntax highlighting I could have used one of several javascript libraries like *et. al* [2012] *et al.* [2012]. The drawback is, that a view displaying html would be required *WebView Inc.* [2010], which drastically decreases the interaction capabilities opposed to *NSTextView Inc.* [2014c].

I could not find any open source project on for example, github for syntax highlighting, so I inspected how *Codeine Gadina* [2013] achieves syntax highlighting by looking at

the source code of this project. It achieves syntax highlighting with the help of clang. The source code is tokenized and the type of the AST entities can be determined and differentiated, as for example a token of type *CETokenTypeString* represents a String and can be colored accordingly.

4.1.6 Source Code History and Diff

git is the version control system used

To be able to create or access a version of source code a version control system (VCS) is required. I decided to use git [2014a], a distributed version control system (DVCS), allowing the user to have a local repository, which can exchange information with remote repositories. Git was initially developed by Linus Torvalds in 2005 for developing the linux kernel. It is now one of the most used versioning control system according to an eclipse community survey Skerrett [2014].

objective-git is a library to call git functions in your own source code

To integrate git functionality into my software prototype CodeShape, I used the library libgit [2014] for which an Objective-C porting/binding *objective-git* exists [2014]. Libgit and objective-git provide access to the core methods of git. They allow for better performance and easier usage as opposed to calling git's command line methods.

To access a repository or individual commits amongst other things, libgit provides an API.

I encountered the limitations of the *git log* algorithm

revision listing issue I encountered the problem that *objective-git* could not fetch all revisions of *PBGitGrapher*-file in the GitX repository used for my user study (see chapter 5). I noticed that some revisions were missing in the revision listing produced by *objective-git*. First I thought this problem was related to *objective-git* only, but by executing *git log -follow Classes/git/PBGitGrapher.mm* in console the same revisions were missing. The *-follow*-parameter tells git to list files beyond renames. The problem is that the file *PBGitGrapher.m* is deleted in commit *cef35ac7* and the file

PBGitGrapher.mm was added with nearly identical content. The *git log* algorithm is not able to detect such changes, although the content remains nearly the same.

git diff The *objective-git* library allows to call git functions from within Objective-C and we call git diff functions in order to calculate file differences.

The git diff functionality is based on the original algorithm from Douglas McIlroy and James Hunt, who published their work in 1976 [2014]. Eugene W. Myers and Webb later improved the algorithm. Miller [2014]. The functionality is best described by: “Show changes between the working tree and the index or a tree, changes between the index and a tree, changes between two trees, changes between two blob objects, or changes between two files on disk” [Torvalds et al. 2014b]. It works on a per line basis and uses a special standardized format for the output [Foundation 2014].

git diff was used in order to calculate version differences

4.2 Layout and Functionality

4.2.1 CodeShape

The design of my source code history exploration GUI with sketch capabilities was inspired by GitX [de Bie and James 2014] and Xcode Inc. [2014e].

GitX served as design inspiration

In figure 4.5 GitX is shown, it brings git’s command line interface to a GUI.

The prototype I designed has a similar functionality of highlighting lines, but I further extended it. It is thus possible to toggle between the states of showing deletions, additions and updates and line interaction is provided. Further details are given at the end of this section.

I invented toggling of diff types

In this version of the plugin (see Fig.4.6) there is no thumbnail preview of the sketch.

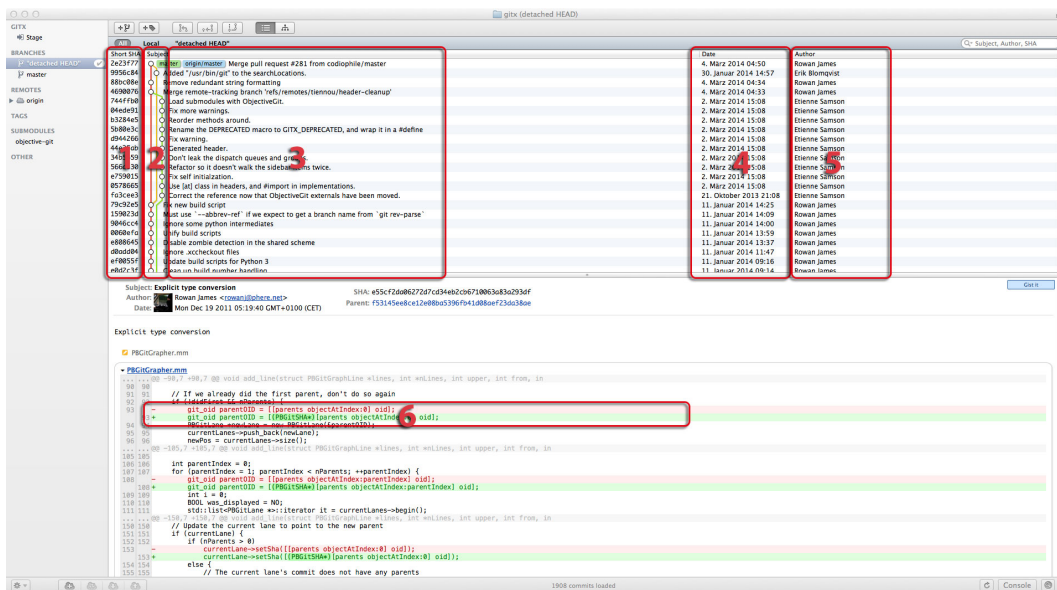


Figure 4.5: GitX application - In the red rectangle labeled with 1 the short SHA of each commit is given. This rectangle is followed by rectangle 2, which is a visualization of git's undirected acyclic graph, where each circle represents a commit. The lines connecting a commit show which commit follows and which one was before in time. Ordering normally starts from bottom to top, whereas the top most is the latest commit. The rectangle with label 3 contains the first line of the commit message of each commit and lies exactly on the same level as the circle. The next rectangle labeled with a 4 contains the date's of each commit on a per row basis. Label 5) contains the author's first- & last name. Rectangle with label 6 encloses a line diff, similar to *git diff*. The first line, highlighted with a light red, represents a deletion. This line was present in the commit before, but not in this particular commit. To further indicate that it refers to a deletion, a red plus precedes the row. This is also similar to git diff [2014a]. The next line, is highlighted in green to mark it as an addition and again a green plus precedes the line.

Tablet Prototype I also thought of a software prototype usable on a tablet device like the iPad. In figure 4.8 one can see an early design prototype.

The advantage of a tablet device would be that it is possible to combine sketch creation and exploration of source code history as well as sketch history on one portable device.

Further the user can manipulate any already drawn sketch and it allows for merging & branching of sketches or parts of it [2011]. This would be harder to achieve with a device

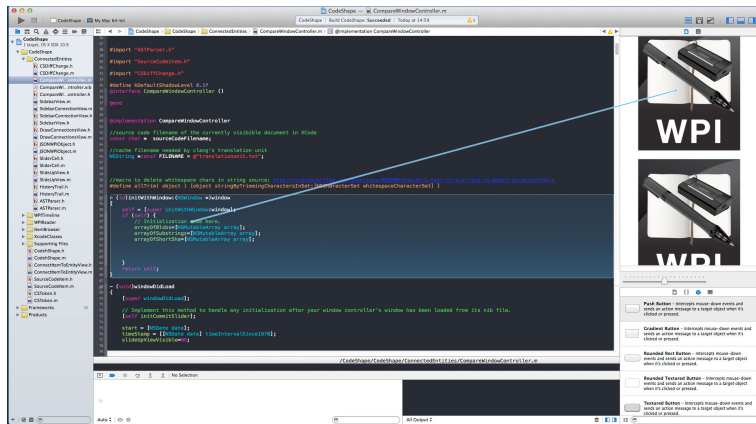


Figure 4.6: Mockup of IBOutlet Connection used to connect a WPI file to a source code method

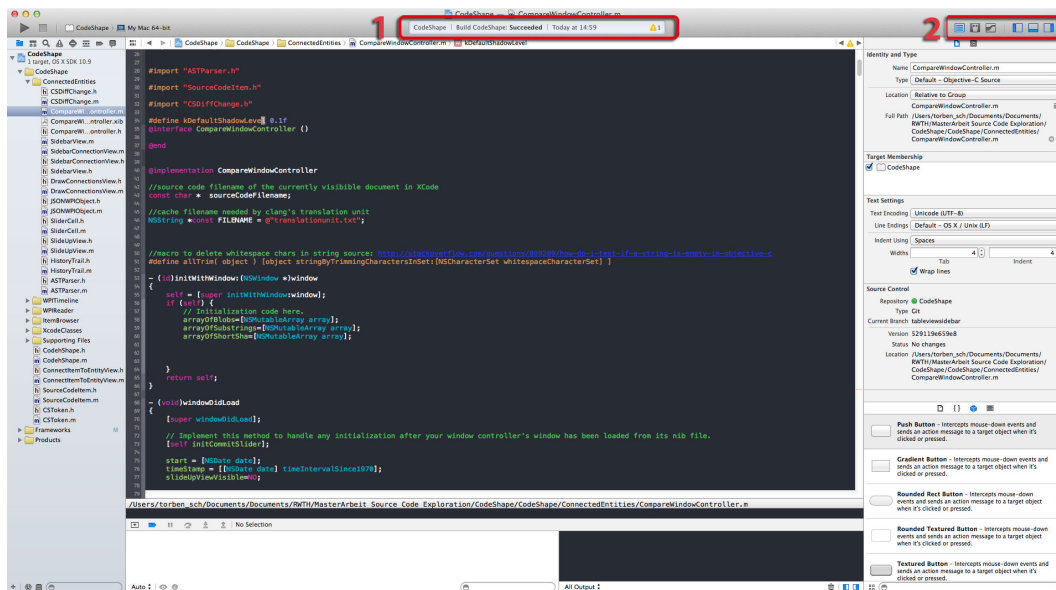


Figure 4.7: Xcode application - 1,2 were used as an inspiration for CodeShape

like the Inking. With an Inking continuing a sketch, requires the receiver to be attached to the exact same position as it was before. A person who has no physical access to the original drawing must also print the sketch and is then able to continue it, when the exact same position of the receiver is satisfied. It is thus more inconvenient to continue a sketch using a digital pen, as opposed to opening your own sketch or the one drawn by someone else on the iPad

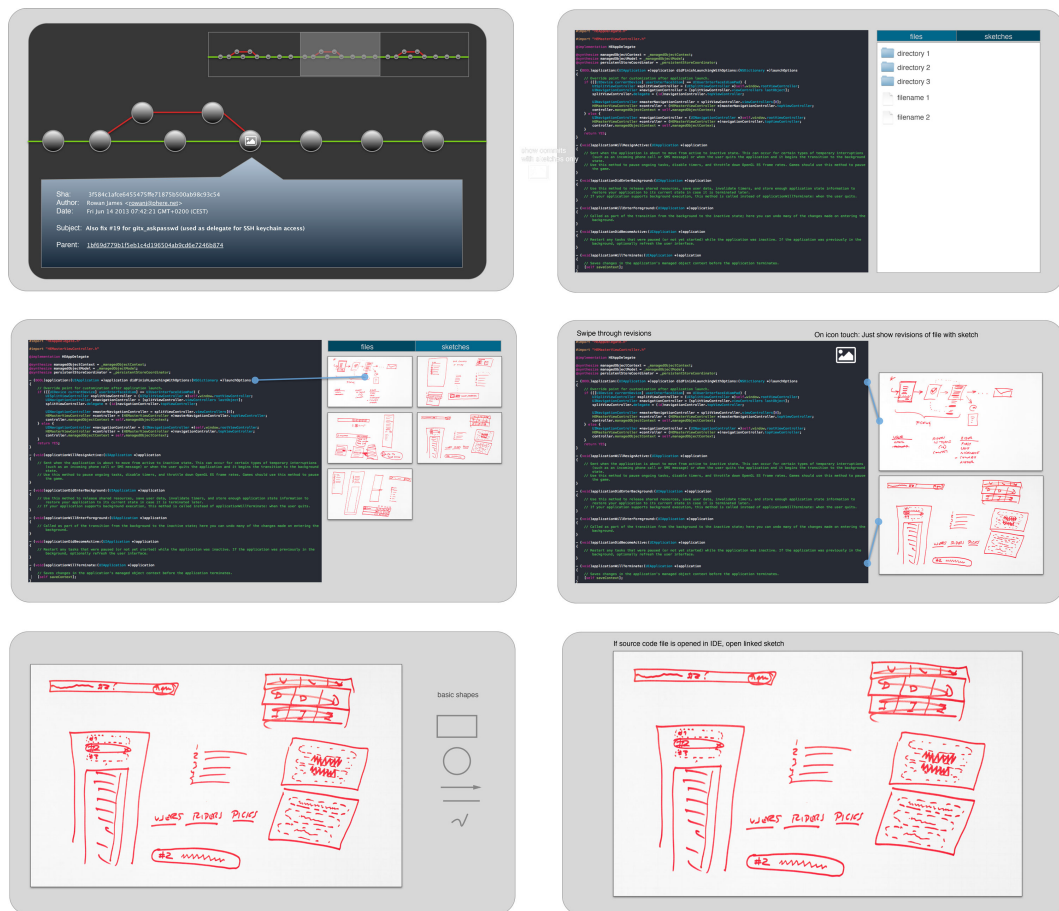


Figure 4.8: The mockup screen (gray rectangle) on the upper left, shows a git tree in which a commit circle has been touched, showing a popup beneath the touched circle. On the upper right of the mockup an overview map is shown, visualizing which part of the history we currently consider. In the upper right mockup a source code is shown and on the right there is a directory listing of the files in the repository. The two mockups in the middle show how connecting thumbnail sketches to source code is achieved. In the two mockups on the bottom a drawing canvas for drawing or extending existing sketches is shown. Basic shapes on the right hand side, support free-hand drawing.

and just continue work.

a tablet device can
become a
programming
assistant

Suggesting there would be a mobile app with sketch and source code history capabilities, then digitizing a sketch or drawing it already in the first place on a tablet device, would allow a developer to browse source code history and sketch history on a device like the Apple iPad or Mi-

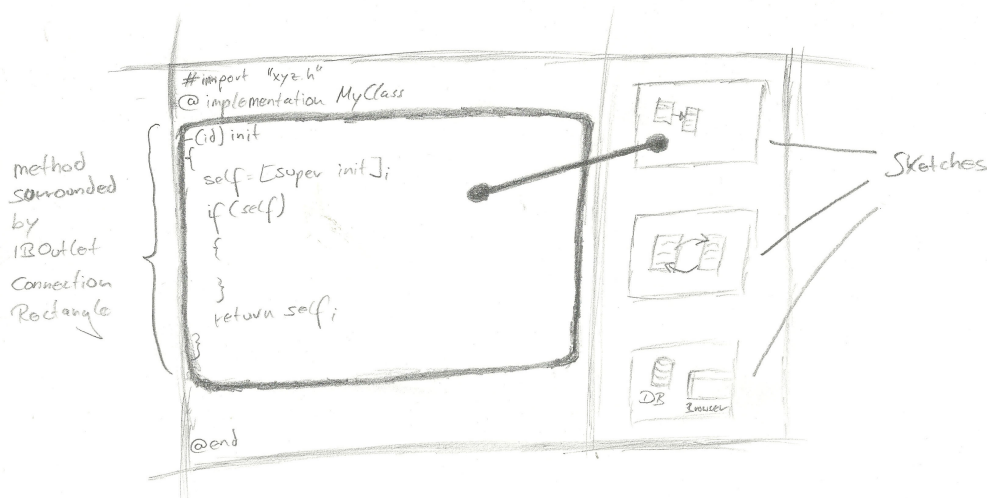


Figure 4.9: IBOutlet Connection sketch v1 - The sketch visualizes the situation in which a user might want to connect a sketch to a source code entity or vice versa. The right side of the sketch should represent a sidebar containing thumbnails of sketches.

crosoft Surface. A developer could use his normal screen of his desktop PC or laptop for development and the iPad serves as a development assistant, where the developer can browse through source code and sketch history besides his usual programming.

I gave up on the idea of developing an iPad app, because drawing on it does not feel as natural as with a digital pen, the screen is very small-sized and a resulting sketch is less accurate and precise as using a digital pen. No matter an iPad app would better support collaboration and history interaction of sketches.

drawing one an iPad does not feel natural

IBOutlet alike Connections

Related to the prototype I came up with the idea of connecting sketches with source code, like IBOutlet Connection's in Xcode's Interface Builder (see 4.1.4) [2014d]. This idea was inspired by a comment during a meeting in my initial study (see chapter 3).

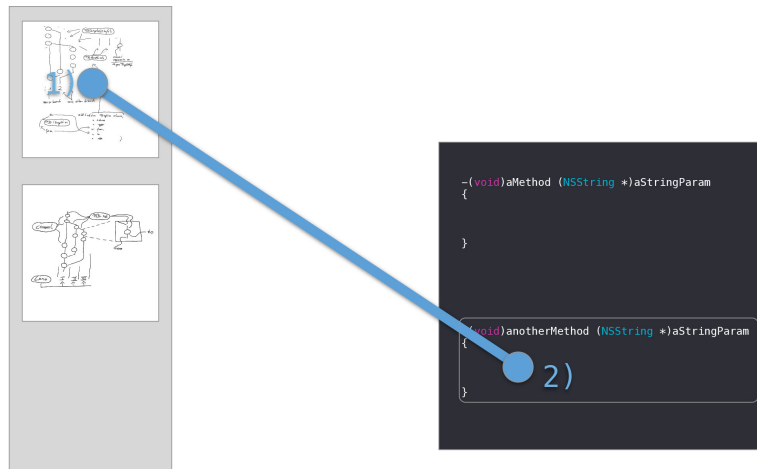


Figure 4.10: connect sketch to a source code entity - a user starts by clicking and holding the mouse button on top of a sketch thumbnail *1)*, he would like to connect to a source code entity (here a method). A dot appears, indicating that the connection will start from the clicked position. When the user now moves the mouse, a connection line is drawn. It starts at the first mouse-down position and ends at the current mouse position. In the next step *2)* the user has to release the mouse-down on top of a method, she would like the sketch to be connected to. It means that the mouse cursor has to be somewhere in the bounds of the method, which corresponds to a rectangle starting from its signature's leftmost starting character, to its closing bracket. Whereas the width is equal to the width of the source code's text view.

In figure 4.9 an early draft of a possible interface of the plugin can be seen.

In the figure 4.10 the process of connecting a sketch to a source code entity is explained.

Versions / Commits To jump between versions of a source code entity (a method in this case), I implemented a sidebar that becomes visible by clicking on a toggle button. The sidebar contains a table of versions (see Fig.4.11).

Another possibility to jump between versions are the blue circles on the bottom of figure 4.11. I choose this particular size and shape to make them easy clickable, as the size is a few pixels larger than an ordinary mouse pointer. If a sketch is attached to a version, it is indicated by a white icon (widely used symbol for a picture or in a broad sense artwork in any kind of format, containing an outline of a sun with a mountain [2013]). To be able to recognize this icon its size also influenced the size of the circle.

a blue circle
represents a commit

CodeShape has a source code line interaction context menu (see Fig.4.12). Selecting one of the menu options, results in circles that have changed their color to red, blue or green (see Fig.4.13). The colors indicate that non blue circles contain a change related to the source code line for which the context menu was activated. Further details to the colors and the context menu are given in the version differences paragraph 4.2.1.

CodeShape has a
context menu that
allows for line
interaction

version differences To be able to see what lines are added, deleted or updated the output of *objective-git* functions were called in order to highlight lines in which changes occurred.

I came up with the idea, of toggling between the different diff change types.

Originally a diff only distinguishes between addition and deletion. I noticed that when a line is updated, the diff contains a deletion directly followed by an addition, whereas the line numbers are the same, but they are placed in two rows. The new change type can be used to introduce or differentiate between three change types: addition, deletion and update.

I consider a new
change type in the
diff visualization

The update change type can be further improved using the Levenshtein distance Levenshtein [1965], which calculates the character differences of a string sequence. When a line completely differs from its previous version, one might consider to show the line diff in its original format, a deletion followed by an addition.

The update change
type leaves room for
improvement

Date	Short SHA	Author	Commit Message
12.09.2008 20:29:09	78e45bd	Pieter de Bie	Add support for --left-right This draws rectangles instead of circles when someone supplies --left-right as a GitX argument
17.09.2008 23:13:05	eeb3309	Pieter de Bie	Multithread test
18.09.2008 01:27:05	6e978dc	Pieter de Bie	Refactor cellInfo structure This makes the PBGitRevisionCell a bit nicer by retrieving all values from the PBGitCommit object itself, and using another NSTextFieldCell to draw the text. This mean that PBGitGrapher now stores its information in the PBGitCommit's, rather than in a custom grapher array. Also, because we don't need the grapher to display refs anymore, the ref labels are also displayed when using path limiting (for example, 'gitx -- Makefile').
24.11.2008 21:55:38	bb5696c	Pieter de Bie	GitGrapher: Limit the maximum number of lanes This limits the maximum number of lanes to 32, making the graphing a lot faster. For example, linux-2.6.git takes only 30 seconds now, rather than >400
24.11.2008 22:32:42	cef35ac	Pieter de Bie	GitGrapher: Rewrite looping code to C++ This makes the revwalking faster by not creating as many NSObjects as before.

Figure 4.11: CodeShape Explorer image section: In this figure CodeShape's sidebar with its version history is shown (becomes visible by pressing the show commit info sidebar button on the top). The leftmost column shows the date and time, the next column a short SHA of each version, next the author and in the last column the whole commit message is visible. The bottom of the figure shows blue circles, each of them represents a commit and it is another visualization of the table, more compact and it shows only the information on hovering a circle with the mouse. A circle with a white icon inside represents a version to which a sketch has been attached to.

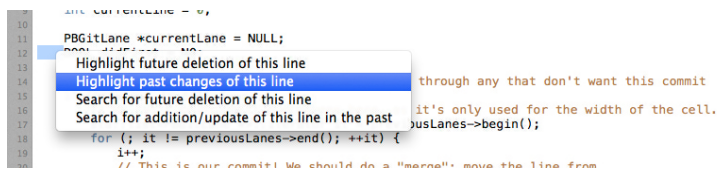


Figure 4.12: CodeShape Explorer - right click context menu

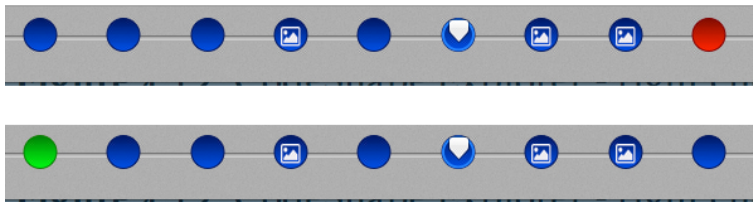


Figure 4.13: CodeShape Explorer - timeline highlighting - The upper half of the figure shows the timeline of CodeShape, when we have selected the context menu “Highlight future deletion of this line” (contextmenu step is not shown here). It marks the commit represented by a circle with red color and thus we know that there is a deletion of the selected line in this future commit.

The grey triangle alike selector, which is the same as the one used for NSSlider Inc. [2014b], indicates which commit is selected at the moment. In this case the commit, which is located 3 commits before the deletion of a line (red circle), is the currently selected.

In the other direction on the lower half of this figure one can see, when we have selected “Highlight past changes of this line” on a line of the current commit, that there is a green circle. This means that this line is added in this commit. The situation of a line, which is updated, is not shown in this figure.

The highlighting of updated lines is already visualized by Kaleidoscope [2012], which provides a GUI for diff operations. It also highlights the words that changed in updated lines, which is impossible with git diff, but the unix tool *wdiff* Foundation [2010] can calculate differences on a *word per word basis*. Thus it would be possible to produce an out-

put similar to Kaleidoscope, where lines containing an update are highlighted and also the individual updated words of this line.

The problem with highlighting a line as an update is, that only the new line is shown, so that the old version of the line is not visible in the diff visualization.

I solved this drawback by providing the aforementioned toggling states. Another solution would be to show the previous version together with the currently selected one. In figure 4.14 an assembled image section of the CodeShape Explorer view is shown. Lines of the source code are visible, whereas some of them are highlighted.

one can now tell
apart deleted lines
from updates more
easily

The advantage of the “newly” introduced update type and its highlighting is, that one can visually differentiate between an update and for example, a deleted line, followed by a line which was added by comparing their coloring.

The original diff output indeed noticed an update, by annotating updated lines with the same line number. A disadvantage is that one has to read and compare the line numbers, to recognize an update.

As opposed to the
original git diff,
update highlighting
reduces number of
required lines

Update highlighting decreases the number of shown lines, so that the number of lines containing an update is halved, because only the new content of the line is shown. This results in that the total number of lines in the source code document diff decreases by $n - \frac{n_{update}}{2}$. Where n is the total number of lines in the diff and n_{update} the old number of lines required to depict updates. Each individual update consisted of the old line followed by a new line. The denominator 2 results from the fact, that now only the new line is displayed. In figure 4.14 all states are toggled, including the update state. If one is interested in how the line looked before the update, one just has to deselect the update button figure 4.15. The output is now equal to the original git diff, except for the line numbering in this prototype. The prototype contains an issue and lacks to show equal line numbers in case of an update, instead the number is increased in every line. This case has to be taken into account in a feature version of the prototype.

```

3 int i = 0, newPos = -1;
4 std::vector<PBGitLane >> *currentLanes = new std::vector<PBGitLane >>;
5 std::vector<PBGitLane >> *previousLanes = (std::vector<PBGitLane >> *)pl;
6 std::list<PBGitLane >> *currentLanes = new std::list<PBGitLane >>;
7 std::list<PBGitLane >> *previousLanes = (std::list<PBGitLane >> *)pl;
8
9 int maxLines = (previousLanes->size() + commit.nParents + 2) * 3;
10 struct PBGitGraphLine *lines = (struct PBGitGraphLine *)malloc(sizeof(struct PBGitGraphLine) * maxLines);
11 int currentLine = 0;
12
13 PBGitLane *currentLane = NULL;
14 BOOL didFirst = NO;
15
16 // First, iterate over earlier columns and pass through any that don't want this commit
17 if (previous != nil) {
18     // We can't count until numColumns here, as it's only used for the width of the cell.
19     std::vector<PBGitLane >>::iterator it = previousLanes->begin();
20     for (; it < previousLanes->end(); ++it) {
21         std::list<PBGitLane >>::iterator it = previousLanes->begin();
22         for (; it != previousLanes->end(); ++it) {
23             ++i;
24             // This is our commit! We should do a "merge": move the line from
25             // our upperMapping to their lowerMapping

```




Figure 4.14: CodeShape Explorer Diff - addition, deletion, update selected - On the upper right an enlarged overlay, where all buttons are selected, is shown. The “+” sign represents addition, so when it is selected, all lines of the source code which have been added in this version, are highlighted green. It is similar in case of the deletion button (“-” sign on the right of the “+” sign button), which highlights a line in red, when this line does not exist in this version any more, but in the version before.

The button on right of the “-”-button, a circle on which two arrowheads are drawn, was re purposed, as it normally stands for refreshing in for example, the web context. A better suiting icon may be deliberated in future, but I could not come up with another symbol.

```

3 int i = 0, newPos = -1;
4 std::vector<PBGitLane >> *currentLanes = new std::vector<PBGitLane >>;
5 std::vector<PBGitLane >> *previousLanes = (std::vector<PBGitLane >> *)pl;
6 std::list<PBGitLane >> *currentLanes = new std::list<PBGitLane >>;
7 std::list<PBGitLane >> *previousLanes = (std::list<PBGitLane >> *)pl;
8
9 int maxLines = (previousLanes->size() + commit.nParents + 2) * 3;
10 struct PBGitGraphLine *lines = (struct PBGitGraphLine *)malloc(sizeof(struct PBGitGraphLine) * maxLines);
11 int currentLine = 0;
12
13 PBGitLane *currentLane = NULL;
14 BOOL didFirst = NO;
15
16 // First, iterate over earlier columns and pass through any that don't want this commit
17 if (previous != nil) {
18     // We can't count until numColumns here, as it's only used for the width of the cell.
19     std::vector<PBGitLane >>::iterator it = previousLanes->begin();
20     for (; it < previousLanes->end(); ++it) {
21         std::list<PBGitLane >>::iterator it = previousLanes->begin();
22         for (; it != previousLanes->end(); ++it) {
23             ++i;
24             // This is our commit! We should do a "merge": move the line from
25             // our upperMapping to their lowerMapping

```


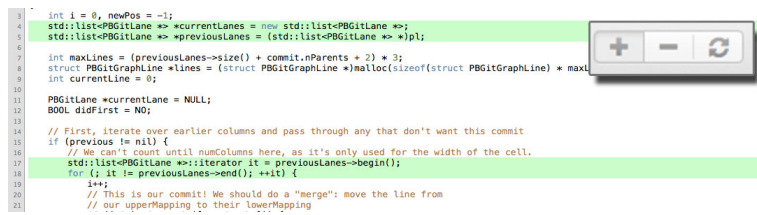


Figure 4.15: CodeShape Explorer Diff - addition, deletion selected

If one is interested in lines which are new in this version, one has to select only the “+”-button (similar for the other cases, see Fig. 4.15, 4.16, 4.17). In case one does not want to be distracted by the diff, unselecting all toggle buttons will disable diff-highlighting entirely.

history of lines In contrast, to common git GUI clients like Tower [2014], CodeShape allows to track the changes

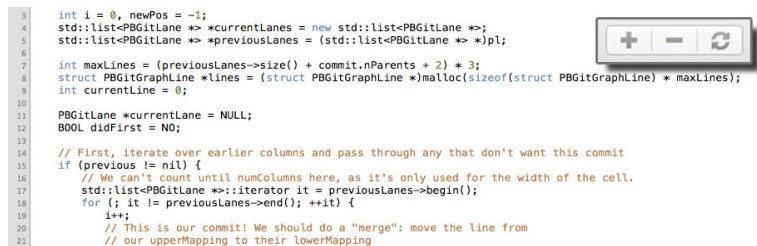


```

3 int i = 0, newPos = -1;
4 std::list<PBGitLane *> *currentLanes = new std::list<PBGitLane *>;
5 std::list<PBGitLane *> *previousLanes = (std::list<PBGitLane *> *)pl;
6
7 int maxLines = (previousLanes->size() + commit.nParents + 2) * 3;
8 struct PBGitGraphLine *lines = (struct PBGitGraphLine *)malloc(sizeof(struct PBGitGraphLine) * maxLines);
9 int currentLine = 0;
10
11 PBGitLane *currentLane = NULL;
12 BOOL didFirst = NO;
13
14 // First, iterate over earlier columns and pass through any that don't want this commit
15 if (previous != nil) {
16 // We can't count until numColumns here, as it's only used for the width of the cell.
17 std::list<PBGitLane *>::iterator it = previousLanes->begin();
18 for (; it != previousLanes->end(); ++it) {
19     i++;
20 // This is our commit! We should do a "merge": move the line from
21 // our upperMapping to their lowerMapping

```

Figure 4.16: CodeShape Explorer Diff - Addition Selected



```

3 int i = 0, newPos = -1;
4 std::list<PBGitLane *> *currentLanes = new std::list<PBGitLane *>;
5 std::list<PBGitLane *> *previousLanes = (std::list<PBGitLane *> *)pl;
6
7 int maxLines = (previousLanes->size() + commit.nParents + 2) * 3;
8 struct PBGitGraphLine *lines = (struct PBGitGraphLine *)malloc(sizeof(struct PBGitGraphLine) * maxLines);
9 int currentLine = 0;
10
11 PBGitLane *currentLane = NULL;
12 BOOL didFirst = NO;
13
14 // First, iterate over earlier columns and pass through any that don't want this commit
15 if (previous != nil) {
16 // We can't count until numColumns here, as it's only used for the width of the cell.
17 std::list<PBGitLane *>::iterator it = previousLanes->begin();
18 for (; it != previousLanes->end(); ++it) {
19     i++;
20 // This is our commit! We should do a "merge": move the line from
21 // our upperMapping to their lowerMapping

```

Figure 4.17: CodeShape Explorer Diff - Nothing Selected

of one or multiple lines.

A single click on a line of code brings up a popup above the commit circle and indicates where the line was changed, along with commit information (see Fig.4.18).

Jumping to the
commit a line was
implemented,
updated or deleted is
now possible

Double-Clicking a line leads to the commit where the line was implemented first or updated. You can only go back in time step by step.

To go to a future version of a line a right click context menu is given in figure 4.12. This menu allows to highlight the next or previous commit 4.13.

It was considered to highlight every past commit, where this line was changed, in a “recursive”-manner. Visually this would result in several highlighted circles, also only one line was selected. Recursive here means that every version containing a change, looks itself for a version, where the line selected also changed in a prior or future version now starting from this version. This suggestion was also made during the study (chapter 5) by one of the participants. This behavior has some drawbacks. Imagine the sit-

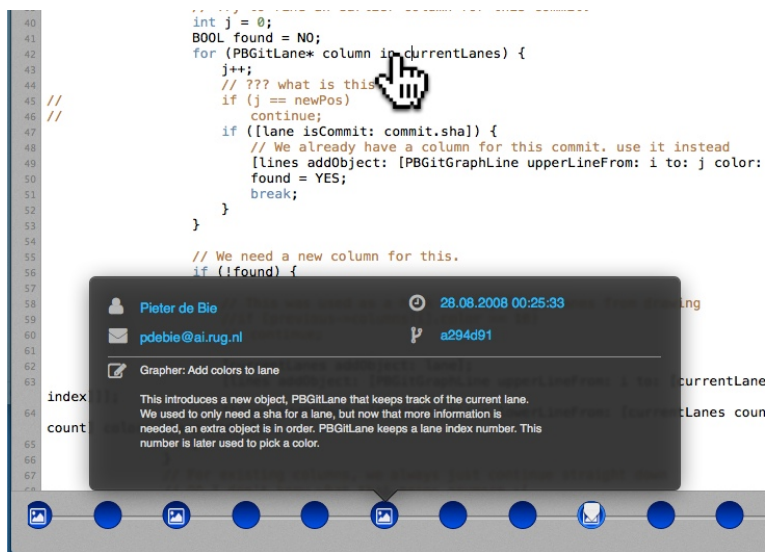


Figure 4.18: In the figure one can see CodeShape a cursor (stretched to see it better in this figure). A user clicked on the source code line and a commit circle, 3 commits before the current one, shows a popup above, indicating that in this version the line has changed.

uation, where the user selects “Highlight past changes of this line” on for example, a line

```
int i=0;
```

when the line of a previous version looks like

```
NSLog("has nothing todo with int i=0");
```

this version would be highlighted too, but it has nothing in common with the line we selected in first place, except for the same line number. Another version even further in the past, which is unrelated to both lines, would also be highlighted. The situation would be the same for “Highlight future deletion/update of this line”. As a consequence of such a situation, where several commits are highlighted, having nothing in common with the selected line(s), we limited highlighting to one prior or future commit per line. This does not solve the issue of unrelated line content, but it ensures that at most only one version, being unrelated, is

only the next or previous change is highlighted

selected.

With the help of the Levenshtein distance and a tool like *wdiff* it would be possible to come up with an upper bound threshold. It could be used to highlight only commits with a word difference below a certain threshold. Thus only changes which are related to the originally selected line, would be highlighted. This might solve the issue, but I did not investigate this any further.

Another thing to test would be only one context menu entry or two, combining future and past change. As it is indicated which current commit we are on, we can tell apart which of the highlighted circles/commits lay in the past and which in the future.

sketches are visible
together with the
source code

In contrast, to the view and sketch arrangement in Lukas' work [2013], I arranged the sketches beside or beneath the source code view. This is due to the fact that my design should focus on sketch assisted source code history to get the rationale behind the code with the help of the sketch.

In Lukas work, where the sketch is presented fullscreen the source code is not visible. I do not consider the ability to navigate with the help of the sketch, which requires the connection points in a sketch (see [2013]) to have certain size to be easily clickable and to focus on the sketch only.

Instead comprehension of source code with the help of a sketch and vice versa are of primary importance. Sketch and source code are complementary, communicating knowledge in different forms and thus I considered it as important to have them side by side. You do not have to keep information of either the sketch or the source code in mind, as it would be the case if only one of them is visible at a time (see Fig.4.19).

4.2.2 Azurite

Azurite tracks every
character change

Another plugin considered during my research is Azurite [2013b] [2013c]. Azurite is a selective undo plug-in

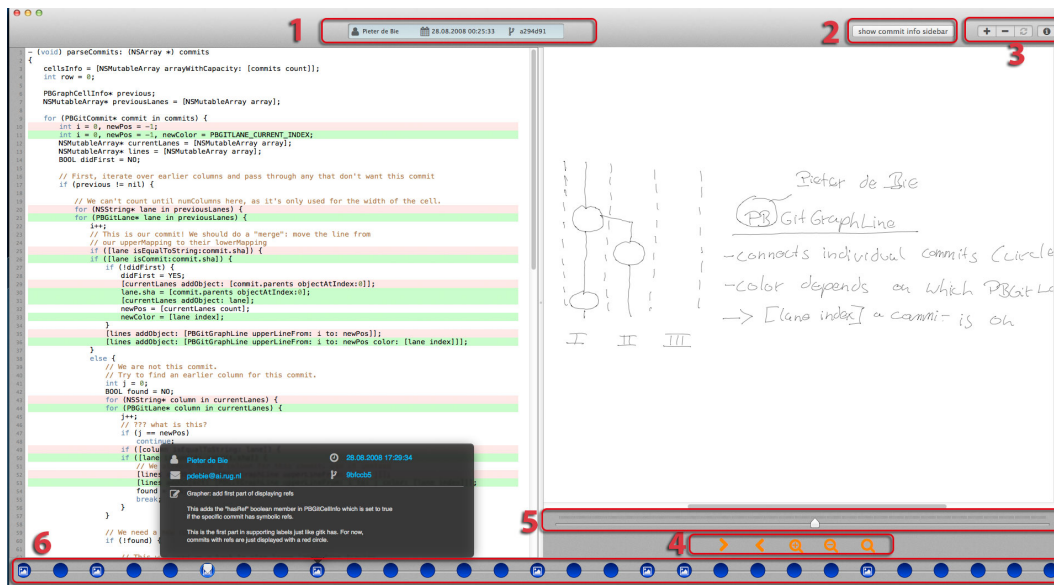


Figure 4.19: CodeShape Explorer: Number 1) shows the Author, date and time, as well as a short SHA. Number 2) is a toggle button to show a sidebar, containing a table of commits and versions (sidebar itself is not visible in this figure). Number 3) shows three toggle buttons for controlling the diff highlighting and the button on the right is for switching on/off the mouse over highlighting on commit circles in number 6). Number 4) shows a sketch control, whereas the left arrow is for stepping forward one stroke at a time, the next one vice versa, the magnifier with a plus symbol inside is for zooming into to the sketch, without quality loss and the other magnifier for zooming out. The last magnifier is for resetting to original size. Number 5) shows a slider to control the strokes in the sketch, allowing to step through the strokes of the sketch in order of creation order. Number 6) shows the timeline slider for stepping through commits, hovering one of the blue circles shows a popup (here centered above the 9th blue commit circle, which contains a sketch, indicated by the white icon inside).

for eclipse and tracks changes on a per character basis, being more fine grained than for example, the line based approach of git.

The individual changes are visualized in a timeline view (see Fig.4.20).

Azurite differentiates between additions (green rectangles), deletions (red rectangles) and updates (blue rectangles). If the user types for example, a "{", then a green rectangle is appended on to the timeline. Deleting this results in an

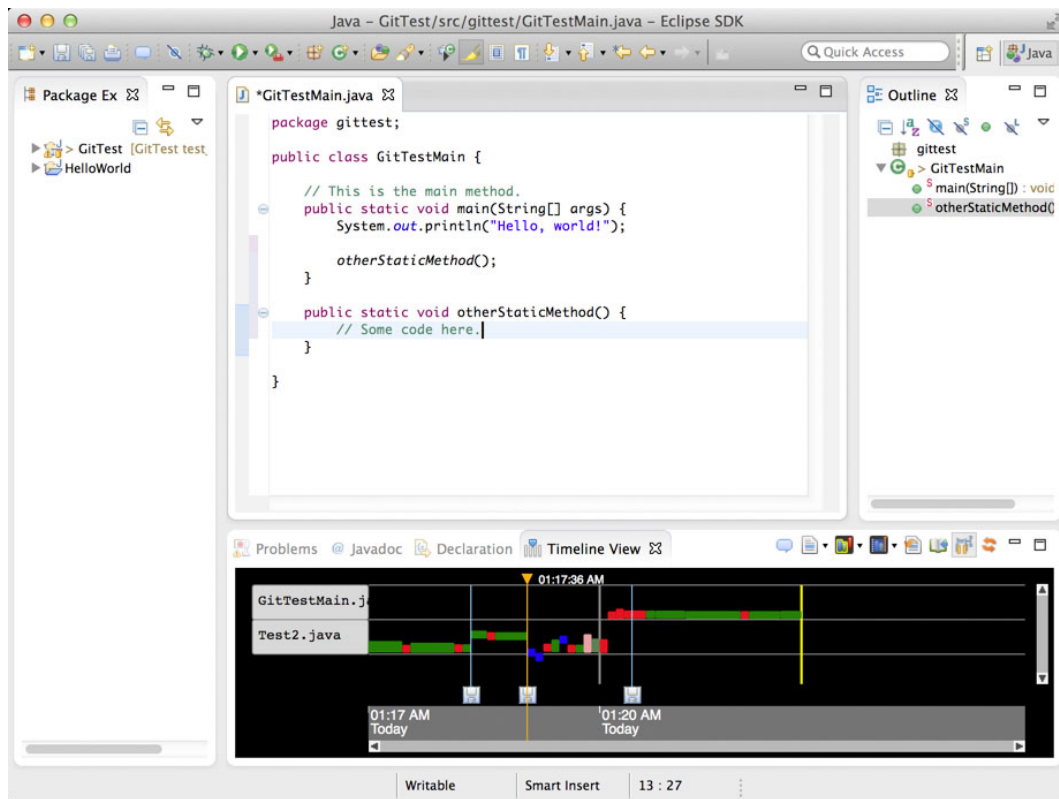


Figure 4.20: Azurite - eclipse Plugin - the black area at the bottom represents the timeline view. One can see green, red and blue shapes, representing addition, deletion and update

additional red rectangle.

Every time eclipse is started, a new session is created, which is visualized with a vertical bar and a date below on the timeline.

Modified Azurite highlights past changes in the timeline

Similar to CodeShape it is possible to select/highlight past changes. This can be triggered from within the source code view by selecting several lines and right-click on them. It brings up a context menu, which contains a menu item named Azurite. When the user selects this menu item, he is presented a submenu. The submenu contains an item "Select Corresponding Timeline View Rectangles" amongst others. As its label indicates it highlights rectangles in the timeline view for the lines that have been selected.

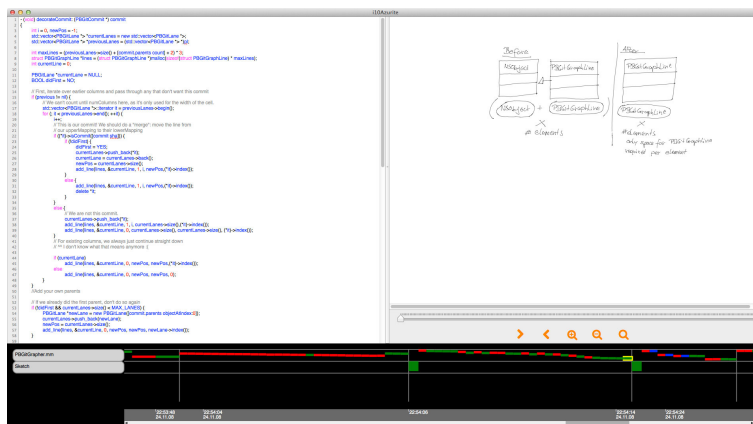


Figure 4.21: Extended Azurite

Modifications

I omitted some features of the original Azurite plugin. As for example, the selective undo feature, because the task set of my user study B did not require to undo anything. The tasks did not require any programming at all, but instead the participants should explore the source code.

selective undo was not needed during my user study and thus not implemented

The original Azurite was extended to support sketch capabilities. The source code view is split in halves, when there is a sketch available for a particular session and a sketch is displayed on the right of the source code view. The sketch view can be seen on the right part of figure 4.21. It is visually and functional identical to the sketch view of CodeShape.

The modified Azurite has the same sketch capabilities attached as CodeShape

To the original timeline view an additional row, labeled “Sketch”, was added. The row is required to be able to show rectangles, indicating that a sketch is available. The rectangles vary from the original ones, as they are of fixed size, quadratic and larger than its original counterparts. This was done to be able to tell them apart from rectangles representing source code changes and to make them more simply selectable than the small rectangles.

It was observed in the user study, that the original rectangles are hard to select (5). This was one reason to make the

Azurite’s rectangles a hard to select with the cursor

sketch-rectangles larger, so that at least the whole cursor fits into it. The width denominator W in Fitts's law Fitts [1954] is increased in this way $T = a + b \log_2 \left(1 + \frac{D}{W} \right)$ so that the average movement time T is decreased.

No matter if the user clicks on a sketch-rectangle or on one representing a source code fragment, the sketch view appears, when there is a sketch available for a session.

I had to decrease the level of detail from character based changes to line based changes, because git operates on line based changes and thus there were no character based changes available for the example git repository used.

The unique visualization of the original timeline can not be preserved by using git

Another drawback resulting from this fact is that git does not track the time for a line change. It is only possible to get the time of the commit the line change belongs to. Thus on the level of an Azurite session it is impossible to sort the line changes in chronological order. Instead I ordered them according to their line number, starting from the lowest line number to the highest and the distance between changes is the same for all. This can be seen in figure 4.21, where the changes of a session look like steps of a stair without gaps in between each change, which is equal to the compact mode of the original plugin. The compact mode removes gaps between any two changes.

4.2.3 Chronos

In this section the history exploration GUI named Chronos will be presented. Chronos is a tool visualizing slices of source code history obtained by the approach named history slicing Servant and Jones [2012] (see definition in 2.2).

A history slice can be anything from a single line and contiguous or disparate sets of lines from one or more files. The selected set of lines is then highlighted in Chronos with the history from all revisions containing changes in the set of lines.

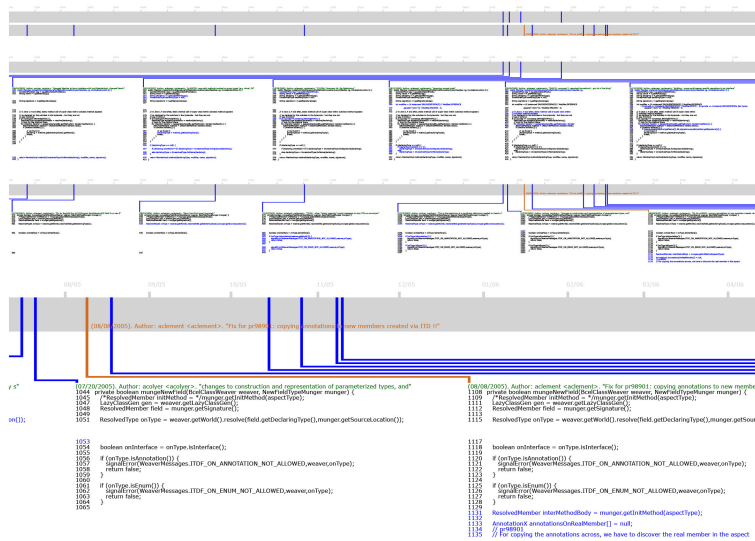


Figure 4.22: Chronos History Slicing GUI

Inside the GUI zooming, panning and highlighting is possible and the content is displayed in form of a scalable vector graphic (svg) (see figure 4.22).

modification Again the plugin is extended to provide sketch functionality. This time though sketches are shown beneath the version of source code they belong to. In its original version Chronos is - like Azurite - an eclipse plugin usable with Java source code. To provide sketch functionality and to make all plugins uniform, I built Chronos from scratch using Objective-C. The problem was that the only description and screenshots can be found (see Servant and Jones [2012]), so my prototype is only based on this information.

Like its original counterpart it allows for zooming, panning and highlighting, but does not use scalable vector graphic's. This was done on purpose, because of the fact that the sketches are not available in scalable vector graphic's, also they could have been converted into svg's, because the coordinates for the strokes are scalable to every resolution. All others views would have to be converted to svg's, too. I consider this as unnecessary, because everything is on its own

I changed the implementation and did not use svg's in my implementation of Chronos

already scalable. For example, the font size of the source code can be multiplied with a scaling factor and the same holds true for the other view like the timeline. This has the advantage - when the views are not converted to svg - that interaction with the NSTextView, which is again used to display the source code (like in CodeShape), like selecting text and searching inside the textview is still possible.

Chronos does not provide line interaction

For the user study selecting and searching inside the source code was disabled, to make the prototype as equal as possible to its original. It would thus be possible to add a context menu and search capabilities like in CodeShape and Azurite to our Chronos implementation. Not being able to search and interact with single lines was reported as critique during my user study.

Before I conducted the user study, we tested the individual prototypes and during this testing period we decided to change hovering to clicking for displaying meta-information about a version of source code. The reason for this was that when one moves the mouse from one version to another displaying meta-information was triggered, which was reported as distracting and annoying. To overcome this issue a single left click is required to show meta-information about a version.

Due to a bug I had to change the design of Chronos

A first prototype separated the whole window into two horizontally split scrollable views. One of the scrollviews was identical to the original plugin and the other scrollview contained the sketches for each source code, whereas the appropriate sketch, when there is one available, was displayed beneath the source code.

Chronos source code and GUI are not available online

When the user scrolled in one of the views the other one scrolled the same amount automatically. This prototype had the advantage that no matter which lines of source code the user looks at, the sketch is visible. The problem was, that when the user zoomed in one view, the application crashed and I could not find the reason for this. To bypass this problem, I removed the bottom scrollview and displayed sketches beneath the source code. This comes with the disadvantage that a sketch is only visible for at most the lower part of source code lines, but when the user

scrolls to the top of source code lines, it results in that the sketch becomes invisible in this part.

Chapter 5

Evaluation

I conducted a user study to evaluate three software prototypes. Therefore I asked graduate and undergraduate computer science students and three external software developers to participate in my study, which was carried out at RWTH Aachen University.

5.1 Preparation

The three software prototypes (Chronos History Slicing, (see 4.2.3), Azurite (see 4.2.2) and CodeShape (see 4.2.1)) are tested with three main tasks and one task per prototype (see Appendix B for the task set). Each of the tasks consists of two or at most three subtasks and every main task with its subtasks were scheduled an operating time of approximately 15 minutes. Each main task was tested with every condition and the number of participants per main task and condition was kept balanced. For example, one study run could look like: *participant A* tested *Azurite* on *main task 1*, *CodeShape* on *main task 2* and *Chronos History Slicing* on *main task 3*. Another run could look like: *participant B* tested *Chronos History Slicing* on *main task 1*, *CodeShape* on *main task 2* and *Azurite* on *main task 3* etc.

3 software
prototypes were
tested on different
tasks within 15
minutes per
prototype

In a pre-study the time needed to complete a task was approximated and some minor adjustments were imple-

Tasks are based on “Hard-to-Answer Questions about Code”	mented to both, the user study form (see Appendix B) and the prototypes.
most frequently asked questions about source code, with the categories: rationale, intent and implementation, debugging, refactoring and history of source code	The tasks were constructed following LaToza’s work “Hard-to-Answer Questions about Code” [2010] in which 179 developers reported 371 questions on source code. In a previous work they already reported some of these questions [2006b]. Similar questions, occurring frequently during history exploration have been observed by Holmes and Begel [2008].
	In their work LaToza and Brad Myers tried to find the most frequently asked questions about source code. I divided them into the following categories: rationale, intent and implementation and history.
	We defined our tasks based on the following LaToza’s questions:
	Rationale Why wasn’t it done this other way?
	Intent & Implementation What does this do in this case?
	History When, how, by whom, and why was this code changed or inserted?
	History How has it changed over time?
	History Has this code always been this way?
GitX was used as an example code base	I choose a source code repository hosted on <i>github</i> called GitX [2014] and used the questions above for creating specific tasks (see Appendix B). The repository contains a git GUI client for MacOS X similar to git’s own GUI <i>gitk</i> [2007]. It was written in Objective-C and partly in C/C++, due to performance reasons.
	The development (1st commit) started in June 2008 and several developers (75+) contributed to this project.
only the history of the file PBGitGrapher.m(m) was considered during the study	In particular, I constrained the example on a specific part of the repository, the file <i>PBGitGrapher.m(m)</i> and its <i>parseCommits</i> method. Later it was (in a subsequent

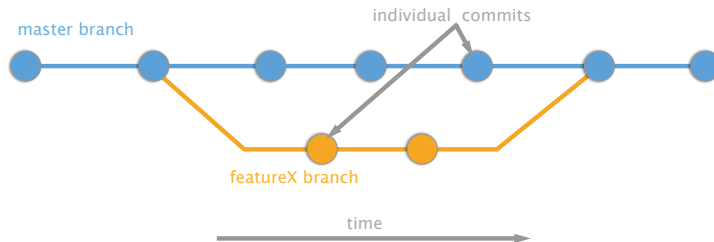


Figure 5.1: Git tree: One can see a schema of a git tree. It belongs to the set of directed acyclic graphs, with the addition that a child can only have one parent. Direction arrows are not shown in this figure.

commit) renamed to *decorateCommit* (see Appendix E for current version of *decorateCommit* method). The entire history of the method used has been fetched from the commit history with the help of *objective-git* (see 4.1.6). It is responsible to provide the dependencies and structure needed to draw git's directed acyclic graph (a visualization connects individual commits with lines and branches) (see Fig.5.1) [2007]. I choose the *PBGitGrapher* class, as it is responsible for the tree structure. It suits as a good example to draw sketches for, because it already has a visual representation. In the case a participant does not know much about git, he is at least familiar with graphs and trees in particular, because everyone of my participants has a computer scientist background and thus had been confronted with graphs.

As I could not find any sketches related to this project, I have drawn these eight sketches myself using the Wacom Inkling (see 4.1.2), with which I produced a digital version of each sketch (see 4.1.2) (see Appendix H for two of the sketches used: Fig.H.1, Fig.H.2). Sketches have been connected to versions of the *parseCommits/decorateCommit* method, where larger refactoring changes took place in their source code and where the commit message gives

8 self hand-drawn sketches have been connected

some hints about these changes.

The word “commit” is used in the following for versions of a file or in particular versions of a method. So, the state of a file or method is equal to the state it had in a particular commit. I connected eight sketches to eight commits of the *parseCommits/decorateCommit* method before the execution of the study.

29 versions of the
method were used

I considered 29 commits in the setup. Each of them contained changes to the method *parseCommits/decorateCommit* and fulfilled the criteria of larger refactoring and a commit message with some hints about the changes. Only the number of lines the method spans are considered during the study, no context was given like the whole PBGitGrapher file or any other file of the repository.

obfuscate versions
of interest

Only four out of the eight sketches visualized information that can be used to solve some of the tasks. The remaining sketches have been connected to make it less obvious, that a certain commit/sketch is required for a task. These remaining sketches also describe the source code they are connected to, but no task requires to consider them. Four additional sketches have been drawn, because a user would less likely come up with the idea to look for commits only, sketches have been connected to. The setup fulfilled this intention, because none of the participants clicked through all commits containing a sketch only, not clicking on a commit without a sketch in between. It seems like a proper strategy to avoid such a behavior.

In figure 5.2 you can see the GUI of our prototype CodeShape. This is how the screen looked like, when a participant had to solve a main task with the help of CodeShape (in Fig.5.3 the same is shown for Azurite and in Fig.5.4 and Fig.5.5 for Chronos).

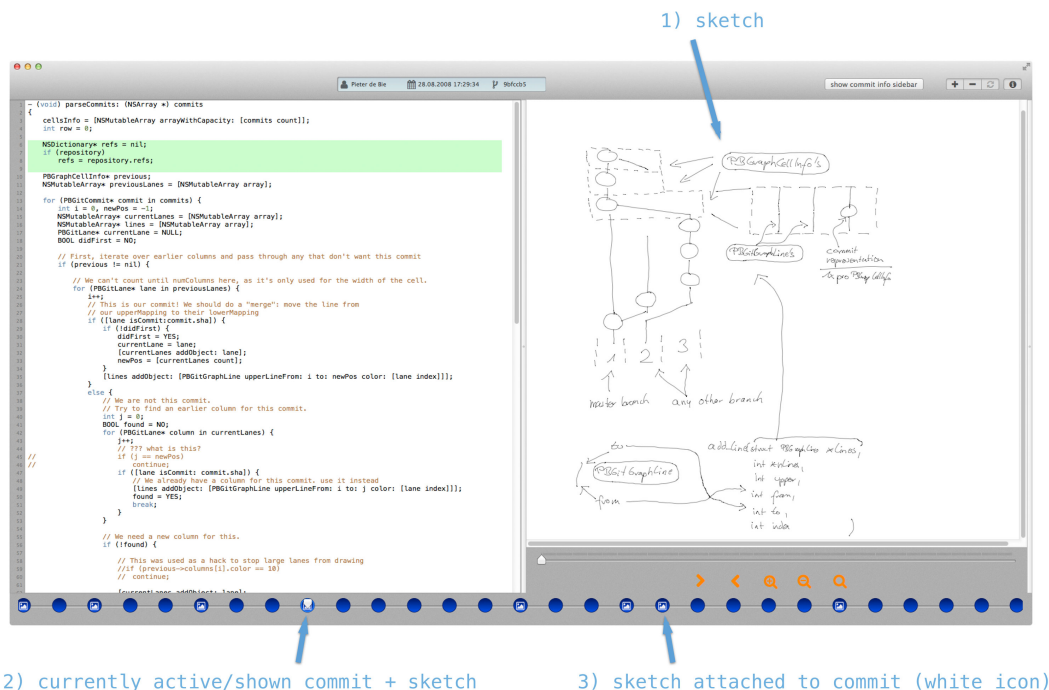


Figure 5.2: Screenshot of CodeShape Plugin - The arrow of 1) points to a sketch, which has been connected in a previous step (not shown) to the source code method `parse_commits` on the left. On the bottom one can see a chronological timeline (blue dots) of all 29 commits. The arrowhead in 2) points to the currently active commit, visualized with the gray triangular alike indicator. A white icon 3) on top of it, indicates that a sketch is connected to this commit.

5.2 Execution

The study was structured as follows: After introducing myself, the participants were told that my thesis is about sketch assisted source code history exploration and some details about the prototypes (CodeShape, Azurite and Chronos History Slicing), the GitX repository and the study in general are given.

During the user study every participant evaluated three software prototypes in a within-group design. We indicated that the study has the purpose to evaluate these three prototypes and in order to determine its strengths and weaknesses in comparison to each other and to test usability in general. It was stated that the study is not

test usability in a within-group design

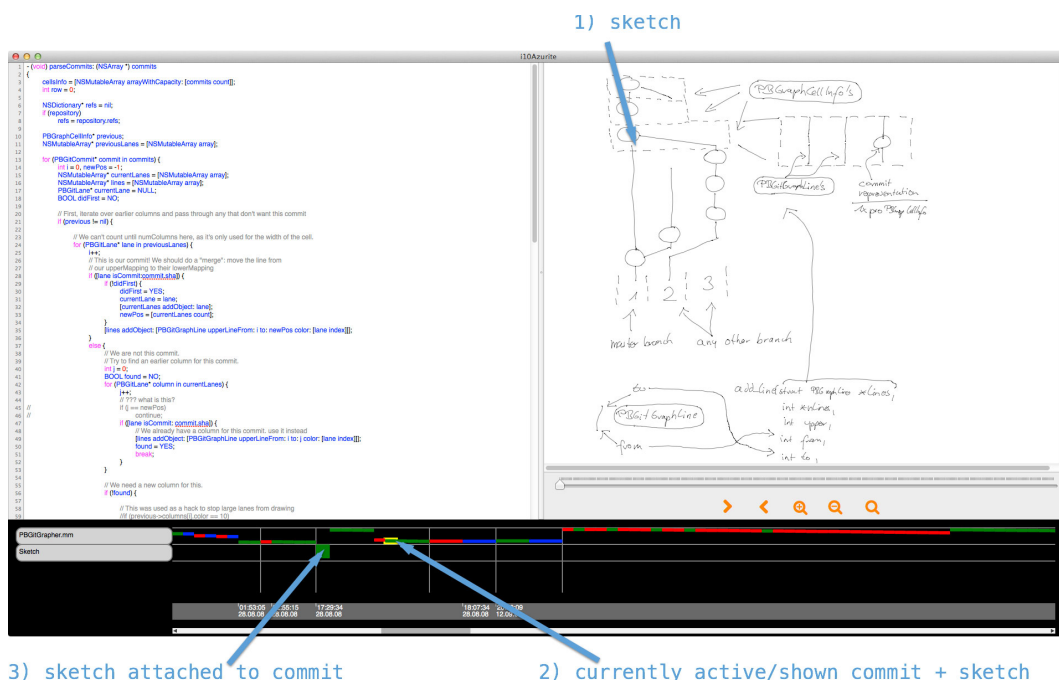


Figure 5.3: Screenshot of Azurite Plugin - The arrow of 1) points to a sketch, which has been connected in a previous step (not shown) to the source code method `parse_commits` on the left. On the bottom one can see a black timeline. A commit is separated by two vertical gray bars. In between those bars are small green, red and blue rectangles, whereas green represents an addition, red a deletion and blue an update of a line. The arrowhead in 2) points to the currently active commit, visualized with the yellow outline around a change rectangle, which belongs to the current commit. A larger green square 3), indicates that a sketch is connected to this commit.

meant to test the users performance.

study duration ~50
minutes

Every participant was told that the study will take up to approximately 50 minutes (3 * 15 minutes + 5 minutes for filling out a consent form and a discussion about all three prototypes).

hardware and study
setup

I performed the whole study in a room in which only one participant and me as the investigator of the study were present. The hardware setup consisted of a Macbook Pro attached to an Apple Cinema Display with an external mouse and keyboard. The Macbook screen was totally dimmed during the study and every participant sat directly

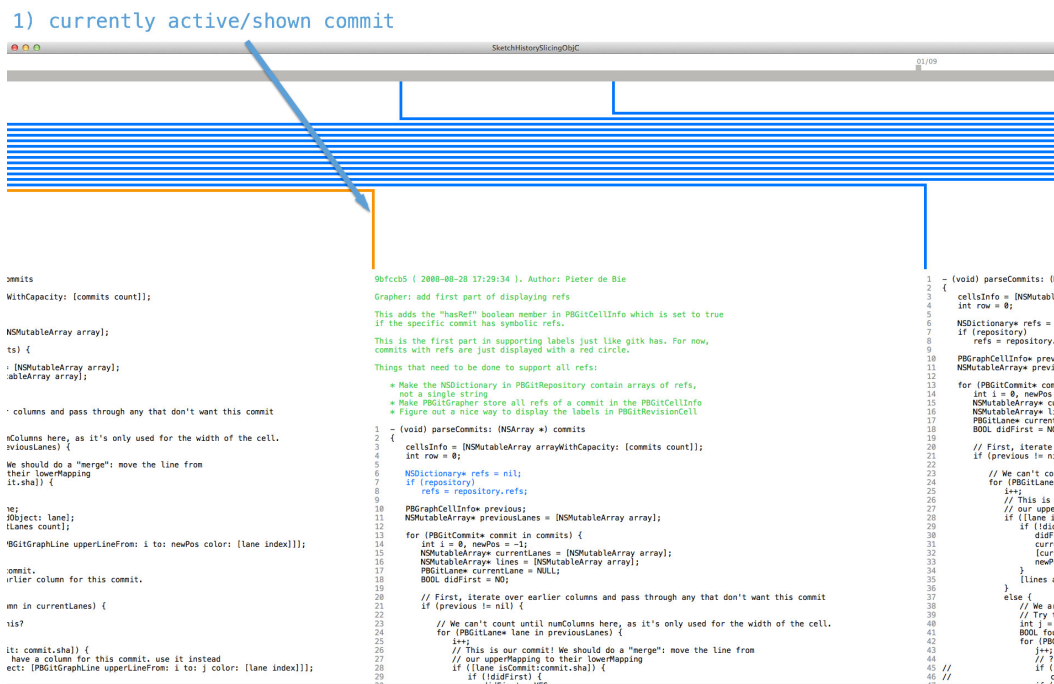


Figure 5.4: Screenshot of Chronos Plugin - The arrowhead in 1) points to the currently active commit, visualized with the orange connection line. The sketch is not visible in this screenshot, because it is below the implementation of the method, outside of the visible part of the window.

in front of the external screen, with me sitting next to them.

Before a subject was presented to the first set of tasks, he was asked to fill out an *Informed Consent Form* (see Appendix A) to inform him among others that the session is screen captured and a microphone is used to record the voices of the subject and the study manager.

In the next step I explained the first prototype and its functionality by showing each of the available features (e.g., context menu to navigate in version history, highlighting in case of comparing (see diff 4.1.6) two versions). The new prototype that should be used for this set of tasks was shown, before a new main task. The functionality and usage of this new prototype was explained. Afterwards it was asked if something remained unclear.

The participants were not told, which prototype was the

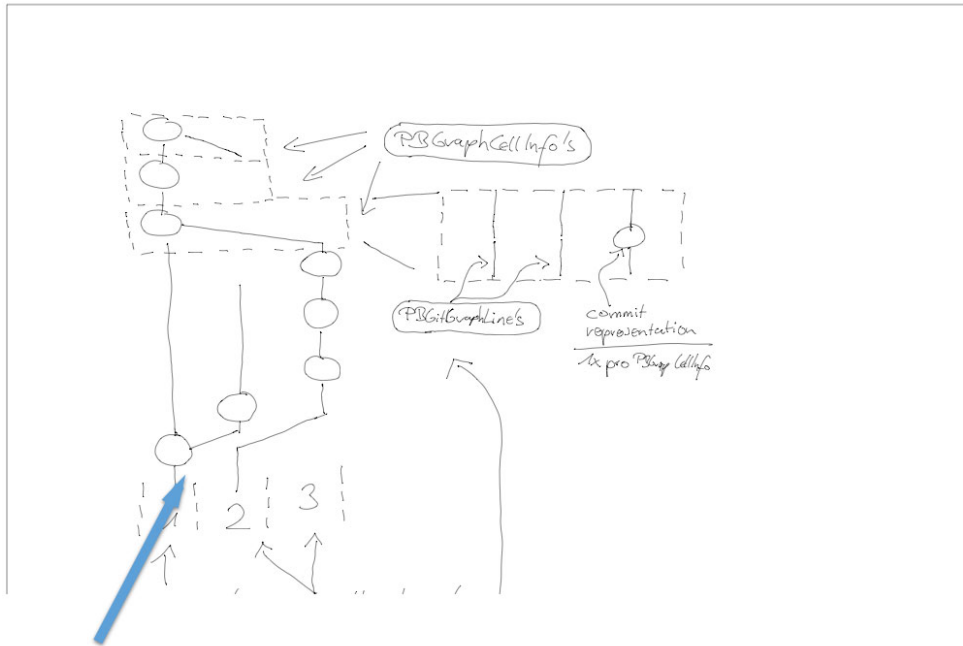
informed consent
form had to be filled
out

functionality
introduction to each
participant

```

121
122 if ([commit.parents count] > 0 && ![commit.parents objectAtIndex:0] isEqualToString:@"")
123   currentLane.sha = [commit.parents objectAtIndex:0];
124 else
125   [currentLanes removeObject:currentLane];
126
127 previousLanes = currentLanes;
128 [cellsInfo addObject: previous];
129 }
130 }

```



1) sketch

Figure 5.5: Screenshot of Chronos Plugin and Sketch- The arrow of 1) points to a sketch, which has been connected in a previous step (not shown) to the source code method *parse_commits* above.

one I designed from scratch and which one was extended by me in order not to influence them in any way during the study. This becomes obvious when several participants mistakenly identified for example Azurite as the prototype designed by me. In case of Azurite (see Fig.5.3) the timeline at the bottom (a javascript implementation used together with HTML5), was taken from the original Azurite plugin with some minor adjustments. I implemented the rest and connected it with a timeline. As Chronos has no open source version available, the whole prototype was implemented based on the textual and visual information of the paper by Servant and Jones [2013].

one task sheet was handed out at a time

After the prototype introduction I handed out the first set

(1.1.1-1.1.3) of tasks and told the participant that these tasks are meant as a “warm-up phase” to get familiar with the prototype they were assigned to for this task.

I orally communicated that the source code, the commit message and the sketches, can be used to answer the task or contain information to solve the task.

In addition if something still remains unclear, not behaves the way expected or any other kind of feedback, were supposed to be communicated at the moment it comes to mind. The method used was mainly *think aloud*, but sometimes *constructive interaction* was used [1993, p.195ff], when the subject needed some guidance, had no idea how to continue, or had walked into a dead end.

After a participant has finished all tasks (three main tasks with a different prototype per main task), he was asked which prototype he prefers using, compared to its competitors. Then, each individual prototype was again shown to the user and it was discussed what was apparent, what he likes about and dislikes about it in comparison to the other ones and how working with it diverges from his usual workflow.

Participants were orally reminded that commit message and sketch can contain valuable information

think aloud and seldom constructive interaction were used

inquire feedback comparing prototypes

5.3 Analysis

5.3.1 Participants

The study included 17 participants, 6 undergraduates and 11 graduated students. To get some general information about them a *questionnaire* was handed out before the study (see appendix A). Programming experience in years was 9.8 years on average. Five subjects reported that they did not know about Objective-C and one out of 15 had no knowledge about git. On average the participants were 27.5 years old, whereas the youngest was 20 and the oldest 34.

17 participants, 6 undergraduate and 11 graduate students with 9.8 years programming experience on average, participated

Two screen capturing video files were corrupted, so that only 15 *samples* can be found in the statistical analysis. In

file corruption issue

order to keep the samples balanced, I carried out another two studies, because the corrupted files could not be analyzed and only my discussion notes of those two screen capturings remained.

There is a low
sample rate per
condition

Although I will report some results as being significant, it has to be considered that there are only five samples per condition. Thus there is a tendency of having a significant effect, but for a more robust result further samples have to be considered.

5.3.2 Tactics to Find a Version

tactic description
how participants
tried to find a certain
commit

The overall tactic to find a commit was very similar among all participants and looked like the following:

1. Coarse grained navigation on per date basis
2. Fine grained navigation per hour
3. Even more fine grained per SHA

using Chronos,
finding the right time
and SHA was most
difficult

A search pattern like a binary search was neither reported by the subject nor be observed by myself. Finding the year of the date was the first sub step taken and then the month & day were searched in order to find a particular version. Here Chronos causes the most difficulties, because in cases the date was similar in hour and day, users clicked through every connection, drawn between a timeline entry and the source code belonging to it (see Fig.5.30) and compared the time and SHA to find the correct one, which were only visible by clicking on a connection. This is opposed to Azurite, in which the whole date with its time is given, as its the case with CodeShape. In addition, CodeShape directly presents the SHA and the commit message in a list.

CodeShape users
used sidebar table
for navigation

When the users tested CodeShape, nearly all, except two, used the sidebar, containing a list of chronological ordered commits by date (see Fig.4.11 in 4.2.1).

The other two used the horizontal circle timeline of commits to find the version they were looking for. Only in case

a commit close to the current one has to be considered or when they used the context menu, then all of the participants used the lower horizontal timeline.

5.3.3 Task Completion Times

In this section the time needed to solve a task will be analyzed. I will not report the task completion time of task 1.1.2, because every participant immediately responded with the answer. The main mental work to solve this task was already done in task 1.1.1 (see Appendix B) and so the participants almost instantly replied. All other tasks are included in the statistic.

Task 1.1.2 was not statistically covered on purpose

To measure the task completion times screen capturing together with audio recording were used. As soon as a participant started reading the task the time was taken and stopped when the correct answer was given.

screen capturing and audio recording

The time needed to solve the task includes the *initial navigation time*, which is evaluated in 5.3.4.

consistent date format In order not to influence the time measurement, because of different date formats, I specified a consistent format among all conditions. Many of the tasks (12 out of 22) required the user to navigate to a certain commit. Where the date in the format DD.MM.YYYY (DIN 5008) followed by hh:mm and a short SHA (7 characters) was given. This was chosen, because in a pre-study I found out that the users had difficulties interpreting the date, when it was given in ISO 8601 format JJJJ-MM-TT and hh:mm:ss, due to the fact that it is rarely used in Germany. So, the task and the GUI format had been adjusted to conform the DIN 5008 date format. Except for the timeline-labels of Chronos, where the short date format, the month following the year with two digits each, was kept like in the original version of the plugin.

date format across prototypes

The consistent date format was chosen in order not to bias the timing among the prototypes.

	df_M	df_R	F	p
task 1.1.1	2	11	2.61	0.118
task 1.1.3	2	11	1.1	0.366
task 1.2.1	2	11	1.1	0.366
task 1.2.2	2	11	6.63	0.013*
task 1.3.1	2	11	1.15	0.352
task 1.3.2	2	11	2.97	0.093
task 1.3.3	2	11	0.17	0.849
task 2.1.1	2	11	5.62	0.021*
task 2.1.2	2	11	0.55	0.591
task 2.2.1	2	11	0.61	0.559
task 2.2.2	2	11	15.2	0.001**
task 2.3.1	2	11	2.03	0.177
task 2.3.2	2	11	3.09	0.086
task 2.3.3	2	11	2.61	0.118
task 3.1.1	2	10	3.57	0.068
task 3.1.2	2	10	0.04	0.961
task 3.2.1	2	10	1.23	0.332
task 3.2.2	2	10	0.45	0.647
task 3.3.1	2	11	1.37	0.293
task 3.3.2	2	11	0.7	0.517
task 3.3.3	2	11	4.91	0.03*

Table 5.4: ANOVA of task completion time, comparing Chronos History Slicing, CodeShape and Azurite.

Alternative hypothesis H_A At least one mean of the time needed to solve a task with the help of one prototype is significantly different to the time needed using another prototype.

ANOVA was used to observe statistical significance

A *one-way within subjects ANOVA* was conducted to compare the effect of three software prototypes on time needed to solve a task, CodeShape, Chronos History Slicing and Azurite conditions.

Task 1.2.2 There was a significant effect of task completion time at the $p < .05$ level for the three conditions [$F(2, 11) = 6.628, p < .05$].

	σ_A	μ_A	σ_C	μ_C	σ_H	μ_H	p_{C-A}	p_{H-A}	p_{H-C}
task 1.1.1	24.93	55.2	37.35	34.25	33.52	82.6	0.604	0.394	0.104
task 1.1.3	60.75	103.0	56.48	91.25	65.69	147.8	0.956	0.504	0.388
task 1.2.1	60.75	103.0	56.48	91.25	65.69	147.8	0.956	0.504	0.388
task 1.2.2	52.26	239.2	45.64	121.75	49.57	162.8	0.012*	0.078	0.458
task 1.3.1	311.47	257.2	30.92	74.75	30.52	124.2	0.357	0.528	0.92
task 1.3.2	75.9	165.2	33.16	92.75	27.5	96.8	0.137	0.137	0.993
task 1.3.3	61.79	173.0	66.41	161.0	69.75	186.4	0.96	0.945	0.837
task 2.1.1	19.82	50.5	10.14	22.75	17.62	58.0	0.089	0.769	0.018*
task 2.1.2	64.14	151.25	40.87	143.75	82.97	109.0	0.987	0.619	0.72
task 2.2.1	74.26	130.75	42.74	135.25	14.02	105.83	0.989	0.685	0.594
task 2.2.2	15.64	95.0	20.47	59.75	13.4	37.33	0.026*	0.0***	0.126
task 2.3.1	91.64	168.75	26.85	93.25	42.7	106.33	0.198	0.256	0.934
task 2.3.2	89.52	129.5	40.55	95.0	11.76	47.67	0.628	0.077	0.368
task 2.3.3	26.17	140.0	25.51	137.5	46.77	184.17	0.995	0.197	0.168
task 3.1.1	18.46	42.75	4.1	20.0	19.14	34.67	0.063	0.723	0.319
task 3.1.2	82.01	97.75	58.73	86.67	37.81	94.33	0.961	0.997	0.984
task 3.2.1	106.54	131.75	16.4	74.0	3.51	79.33	0.33	0.505	0.991
task 3.2.2	178.16	208.5	40.92	148.33	95.2	140.33	0.685	0.706	0.994
task 3.3.1	54.47	98.4	23.91	60.17	24.88	77.0	0.264	0.729	0.81
task 3.3.2	58.25	88.8	14.67	73.33	108.65	122.67	0.903	0.719	0.487
task 3.3.3	42.55	146.8	32.04	105.33	125.8	245.33	0.544	0.128	0.024*

Table 5.4: Tukey HSD post hoc test of task completion time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 121.75, SD = 45.64$) was significantly different from the Azurite condition ($M = 239.2, SD = 52.26$). However, the Chronos History Slicing condition ($M = 162.80, SD = 49.57$) did not significantly differ from the Azurite condition and the CodeShape condition.

Tukey HSD was used to spot significant differences between a pair of prototypes

The results of Tukey HSD are visualized in figure C.5 .

Task 2.1.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 5.621, p < .05$].

Post hoc comparisons using the Tukey HSD test in-

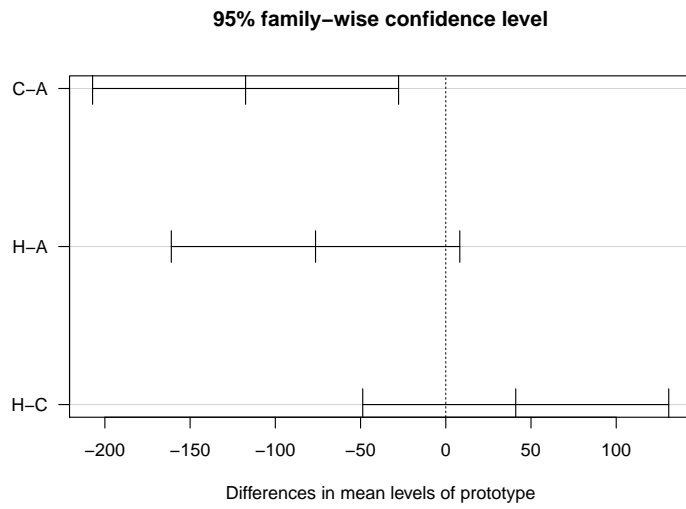


Figure 5.6: Task 1.2.2 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result

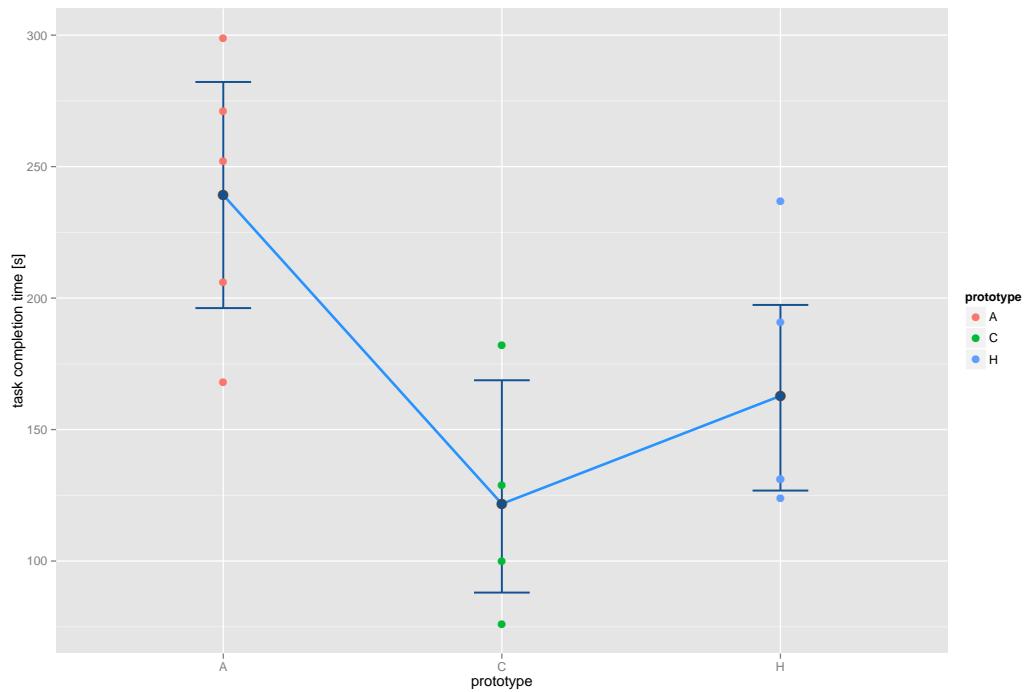


Figure 5.7: Task completion graph task 1.2.2

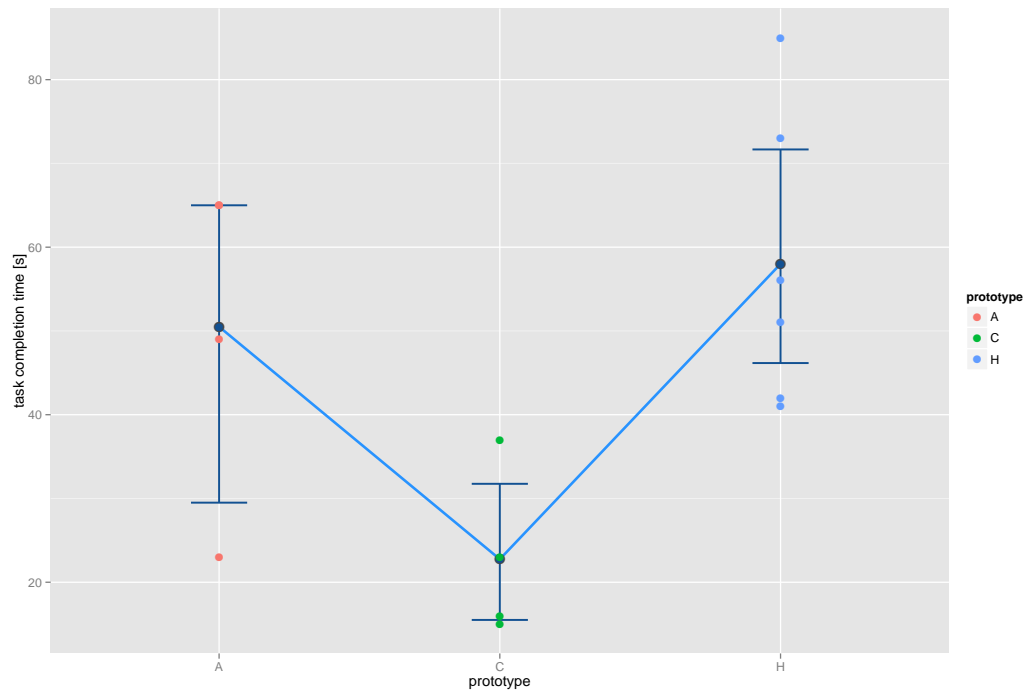


Figure 5.8: Task completion graph task 2.1.1

indicated that the mean score for the CodeShape condition ($M = 22.75, SD = 10.14$) was significantly different from the Chronos History Slicing condition ($M = 58.00, SD = 17.62$). However, the Azurite condition ($M = 50.50, SD = 19.82$) did not significantly differ from the Chronos History Slicing condition and the CodeShape condition.

The results of Tukey HSD are visualized in figure C.10.

Task 2.2.2 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 15.20, p < .05$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 59.75, SD = 20.47$) was significantly different from the Azurite condition ($M = 95.00, SD = 15.64$). The mean score for the Chronos History Slicing condition

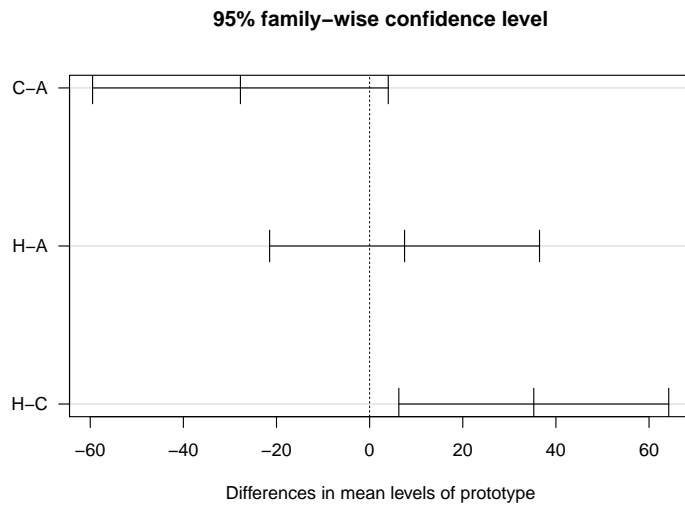


Figure 5.9: Task 2.1.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory Slicing result.

($M = 37.34, SD = 20.48$) was significantly different from the Azurite condition. However, the History Slicing condition did not significantly differ from the CodeShape condition.

The results of Tukey HSD are visualized in figure C.14.

Task 3.3.3 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 4.908, p < .05$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 105.33, SD = 32.04$) was significantly different from Chronos History Slicing condition ($M = 283.00, SD = 125.80$). However, the Azurite condition ($M = 146.80, SD = 42.55$) did not significantly differ from the Chronos History Slicing condition and the CodeShape condition.

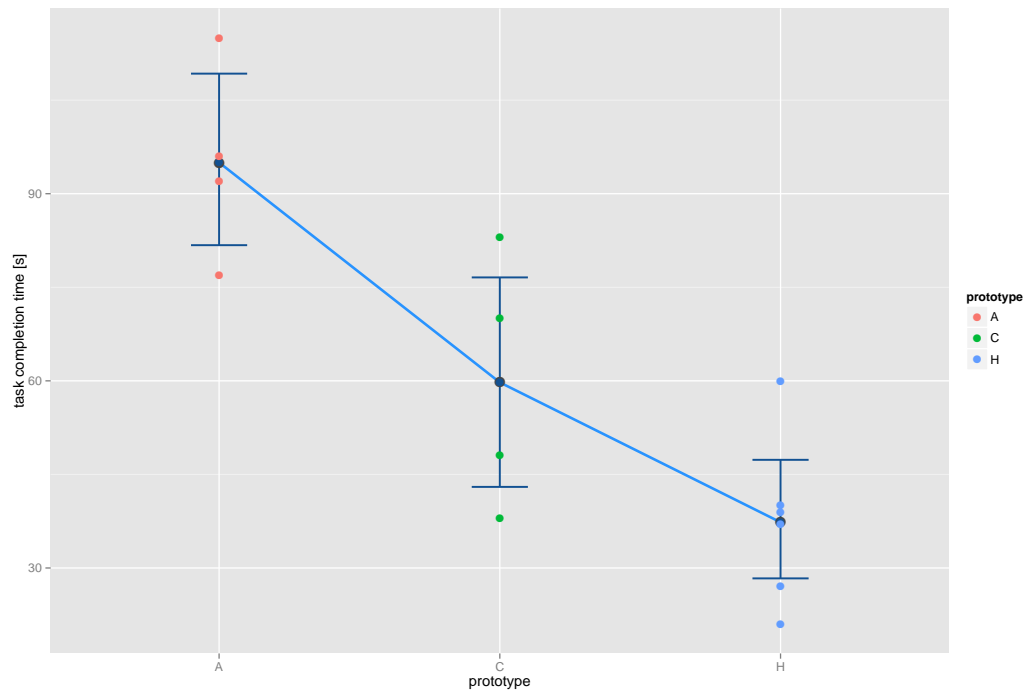


Figure 5.10: Task completion graph task 2.2.2

The results of Tukey HSD are visualized in figure C.25.

General result Taken together, these results suggest that the prototype does have an effect on task completion time. Specifically, our results suggest that when we navigate using CodeShape, the task completion time is lowest in general. In two out of 21 cases it is significantly lower than Azurite and in another two cases it is significantly lower than using Chronos History Slicing. For the tasks that have not been reported above, the graphs in Appendix C show that the mean of CodeShape is lower than the mean of Chronos History Slicing and Azurite, but it is not significantly lower in general (7 out of 11).

Task completion time in general lowest using CodeShape

Interestingly in task 2.2.2 and task 2.3.2 Chronos History Slicing achieved the best results, also they are not significantly better than Codeshape's results. According to my screen capturing observations these results occur, because when the user has to consider just one line that has changed

Finding single change is fastest using Chronos History Slicing

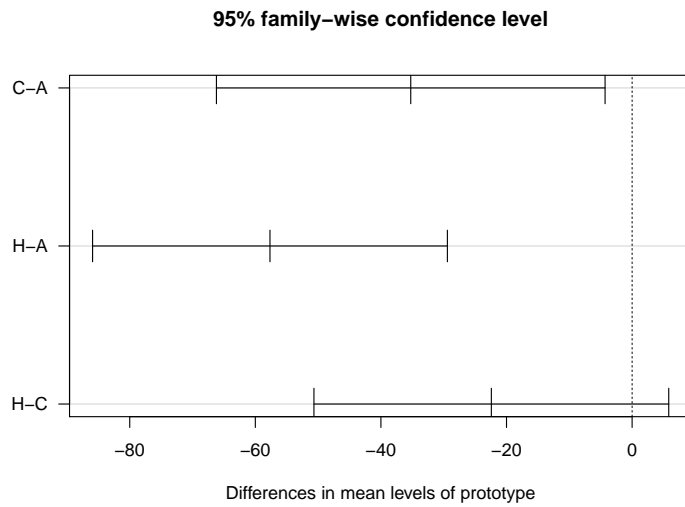


Figure 5.11: Task 2.2.2 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory Slicing result is significantly lower than the (A)zurite result

(highlighted in blue in case of Chronos History Slicing), it can be found fastest using Chronos History Slicing. When the user has to consider two or more changes in each version, than Chronos is not superior any more and the context menu of CodeShape outperforms it.

5.3.4 Initial Navigation Time

The task completion time is sometimes composed of initial navigation time, which is the amount of time spend to navigate to a certain commit and focus time, which is the time spend on a single commit.

We here concentrate on tasks, where the user has to navigate to a certain commit and then solve the rest of the task, which I call focus time. It is not included in the initial navigation time measurement.

Task selection
criteria

I only consider tasks, where it is possible to clearly

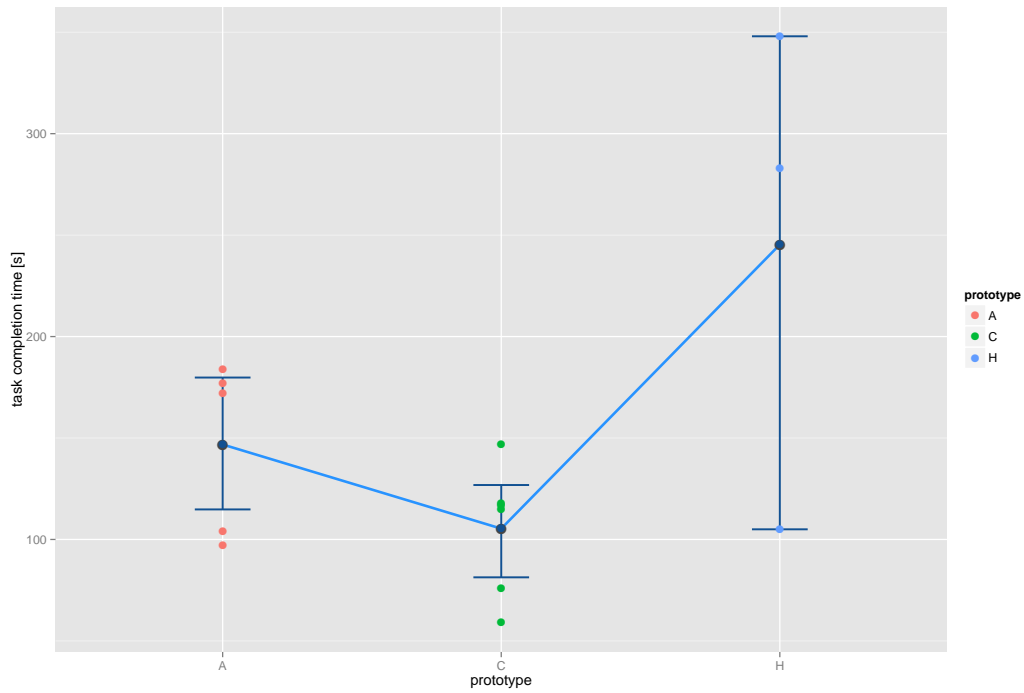


Figure 5.12: Task completion graph task 3.3.3

95% family-wise confidence level

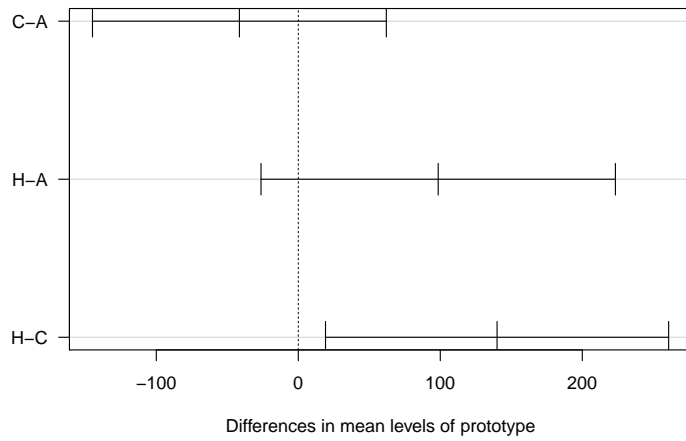


Figure 5.13: Task 3.3.3 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory Slicing result

tell apart navigation time, from focus time. These are the tasks, where the user has to navigate to only one commit and the information provided by this particular commit should suffice to answer the question. Or tasks where navigation to one commit is considered as initial navigation time and represents the basis. Additional navigation time required for this task is not added up to this basis.

Set of considered tasks task 1.1.1, task 1.2.1, task 1.3.3, task 2.1.1, task 2.2.1, task 2.3.1, task 2.3.3, task 3.1.1, task 3.2.1, task 3.3.1, task 3.3.3

The initial navigation time can supply evidence to the hypothesis: with the help of which of the three prototypes, finding a particular commit is fastest.

Alternative hypothesis H_A At least one mean of the initial navigation time, needed to find a particular version with the help of one prototype, is significantly different to the initial navigation time of another prototype.

within subjects
ANOVA comparing 3
software prototypes

Again *one-way within subjects ANOVA* was conducted to compare the effect of three software prototypes on time needed to navigate to a particular version of source code, CodeShape, Chronos History Slicing and Azurite conditions.

Task 2.1.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 7.284, p < .001$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 22.00, SD = 10.23$) was significantly different from the Chronos History Slicing condition ($M = 58.67, SD = 18.32$) and Azurite condition

	df_M	df_R	F	p
task 1.1.1	2	11	2.39	0.137
task 1.2.1	2	11	2.42	0.134
task 1.3.1	2	11	3.47	0.068
task 1.3.3	2	11	2.94	0.095
task 2.1.1	2	11	7.28	0.01*
task 2.2.1	2	11	5.3	0.024*
task 2.3.1	2	11	1.26	0.32
task 2.3.3	2	11	1.91	0.194
task 3.1.1	2	10	4.6	0.038*
task 3.2.1	2	11	7.13	0.01*
task 3.3.1	2	11	4.07	0.048*
task 3.3.3	2	11	1.41	0.286

Table 5.4: ANOVA of navigation time comparing Chronos History Slicing, CodeShape and Azurite

($M = 52.50, SD = 14.15$). However, the History Slicing condition ($M = 58.67, SD = 18.32$) did not significantly differ from the Azurite condition ($M = 52.50, SD = 14.15$).

The results of Tukey HSD are visualized in figure D.6.

Task 2.2.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 5.30, p < .05$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 12.00, SD = 2.94$) was significantly different from the Azurite condition ($M = 35.20, SD = 18.21$). The mean score for the Chronos History Slicing condition ($M = 15.50, SD = 8.19$) was significantly different from the Azurite condition. However, the History Slicing condition did not significantly differ from the CodeShape condition.

The results of Tukey HSD are visualized in figure D.8.

	σ_A	μ_A	σ_C	μ_C	σ_H	μ_H	p_{C-A}	p_{H-A}	p_{H-C}
task 1.1.1	24.32	53.6	34.91	38.0	33.75	82.2	0.74	0.347	0.13
task 1.2.1	14.06	36.6	6.85	24.75	21.79	48.4	0.533	0.498	0.116
task 1.3.1	12.1	29.0	5.68	22.25	23.32	49.0	0.81	0.168	0.073
task 1.3.3	15.96	42.4	7.62	21.0	17.38	42.0	0.123	0.999	0.131
task 2.1.1	14.15	52.5	10.23	22.0	18.32	58.67	0.042*	0.811	0.009**
task 2.2.1	18.21	35.25	2.94	12.0	8.19	15.5	0.032*	0.046*	0.878
task 2.3.1	7.97	37.25	7.66	28.0	9.22	30.83	0.31	0.493	0.865
task 2.3.3	23.77	60.75	9.81	29.75	26.7	46.33	0.17	0.595	0.509
task 3.1.1	8.92	37.25	3.87	19.17	19.05	35.0	0.049*	0.955	0.12
task 3.2.1	8.84	15.2	4.09	10.5	4.62	27.33	0.462	0.056	0.008**
task 3.3.1	16.29	29.6	3.74	13.0	5.57	28.0	0.056	0.976	0.149
task 3.3.3	11.61	37.2	23.19	25.83	4.51	45.33	0.54	0.798	0.286

Table 5.4: Tukey HSD post hoc test of initial navigation time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape

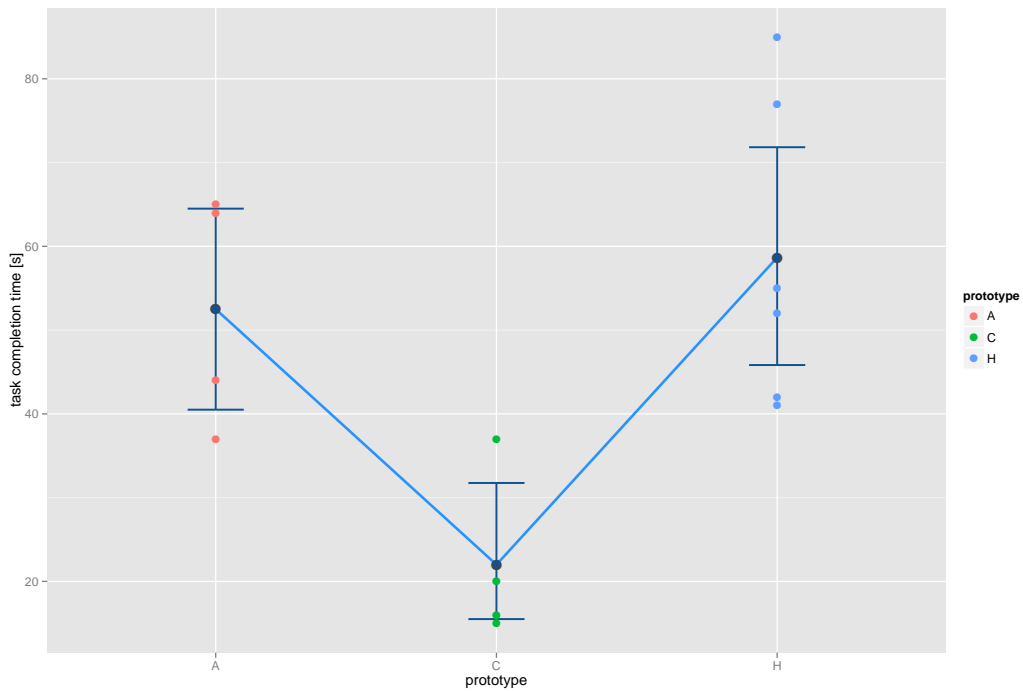


Figure 5.14: Task navigation graph task 2.1.1

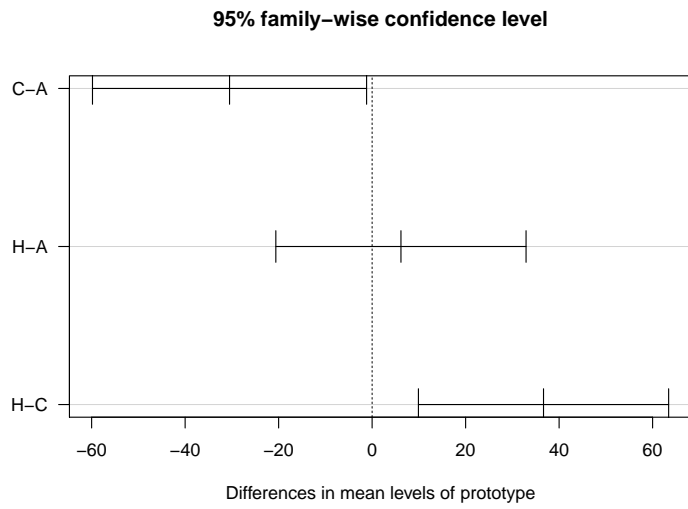


Figure 5.15: Task 2.1.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory Slicing result

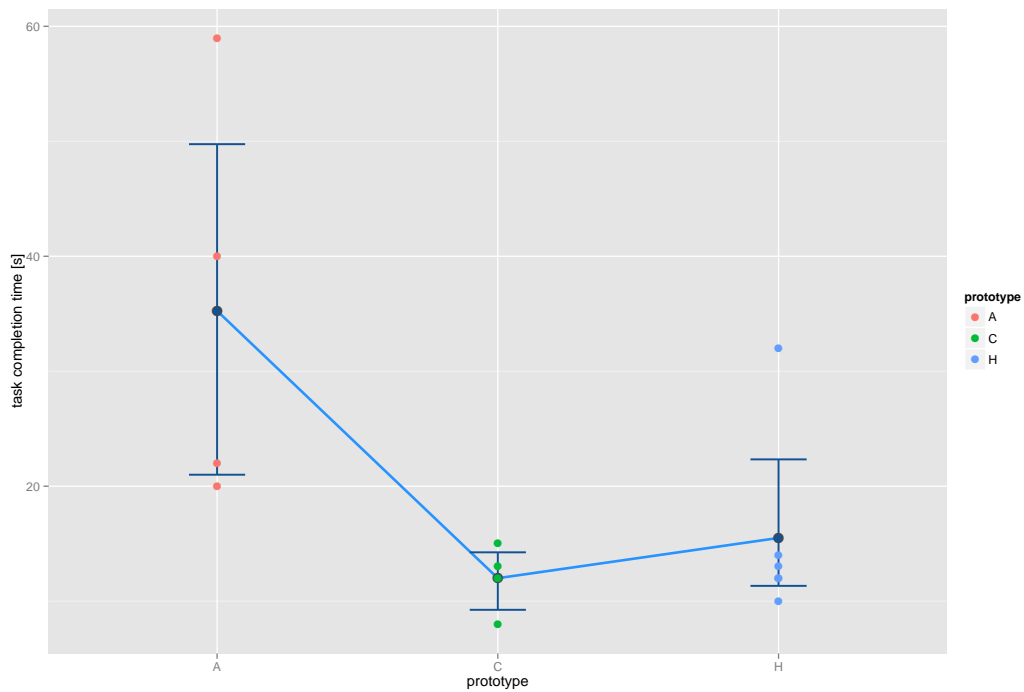


Figure 5.16: Task navigation graph task 2.2.1

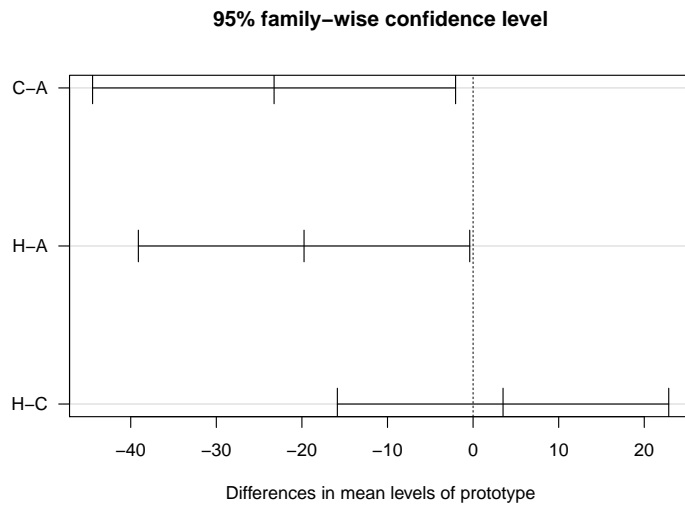


Figure 5.17: Task 2.2.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory slicing result is significantly lower than the (A)zurite result

Task 3.1.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 10) = 4.60, p < .05$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 19.17, SD = 3.87$) was significantly different from the Azurite condition ($M = 37.25, SD = 8.92$). However, the History Slicing condition ($M = 35.0, SD = 19.05$) did not significantly differ from the CodeShape condition and the Azurite condition.

The results of Tukey HSD are visualized in figure 5.19.

Task 3.2.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 7.134, p < .05$].

Post hoc comparisons using the Tukey HSD test indi-

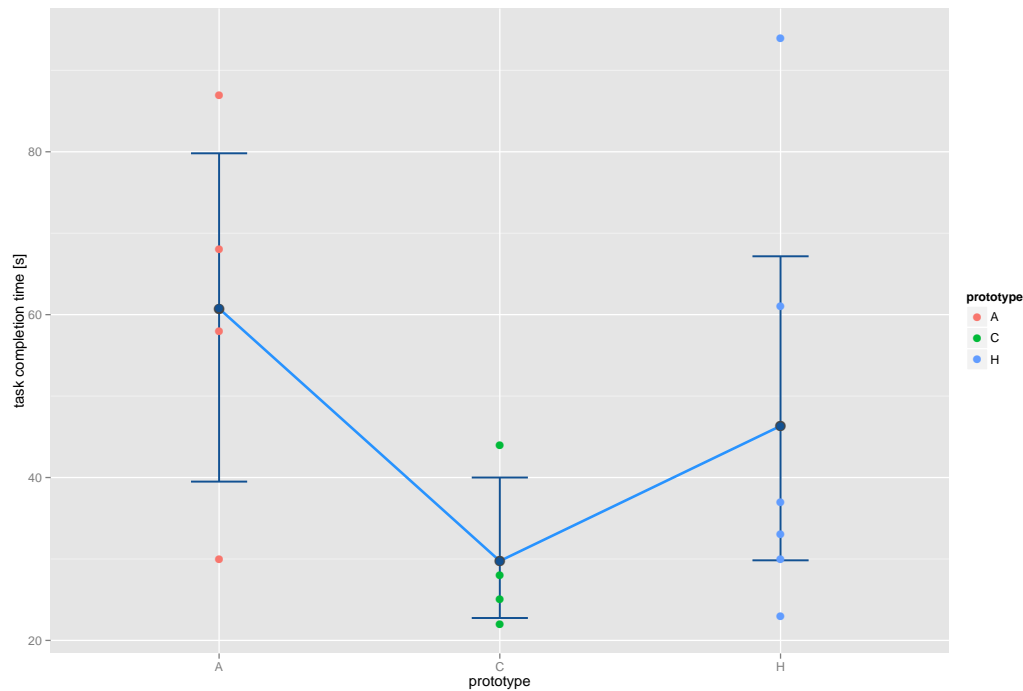


Figure 5.18: Task navigation graph task 3.1.1

cated that the mean score for the CodeShape condition ($M = 10.50, SD = 4.09$) was significantly different from the Chronos History Slicing condition ($M = 27.34, SD = 4.62$). However, the Azurite condition ($M = 15.20, SD = 8.84$) did not significantly differ from the Chronos History Slicing condition ($M = 27.34, SD = 4.62$) and the CodeShape condition ($M = 10.50, SD = 4.09$).

The results of Tukey HSD are visualized in figure D.13.

Task 3.3.1 There was a significant effect of navigation time at the $p < .05$ level for the three conditions [$F(2, 11) = 4.07, p < .05$].

Post hoc comparisons using the Tukey HSD test indicated that the mean score for the CodeShape condition ($M = 13.00, SD = 3.74$) did not significantly differ than the Chronos History Slicing ($M = 28.00, SD = 5.57$) and the Azurite condition ($M = 29.60, SD = 16.29$).

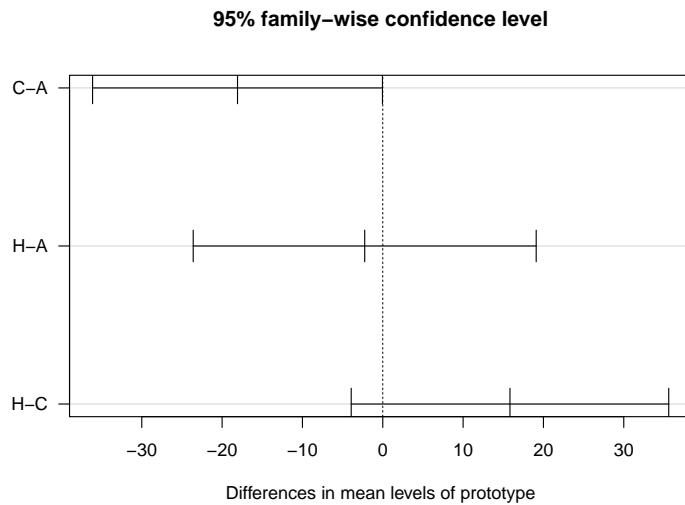


Figure 5.19: Task 3.1.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory slicing result is significantly lower than the (A)zurite result

The results of Tukey HSD are visualized in figure D.15.

overall CodeShape
has the tendency to
be the fastest
prototype

General result Taken together, these results suggest that the prototype does have an effect on initial navigation time. Specifically, our results suggest that when we navigate using CodeShape, the navigation time is lowest. In one case navigation time is significantly lower than Chronos History Slicing and Azurite. In three out of 11 cases it was significantly faster than Azurite. Another time it was significantly faster than using Chronos History Slicing. For the tasks that have not been reported above, the graphs in Appendix D show that the mean of CodeShape is in general lower as the mean of Chronos History Slicing and Azurite, but it is not significantly lower (7 out of 11).

Chronos interface
design in case of
several versions per
day causes poor
navigation time
results

As described in 5.3.2 above, Chronos required the participant to click through several connections, if there are several commits at one day. Except for the case when the right commit was found by chance in first place. The situation

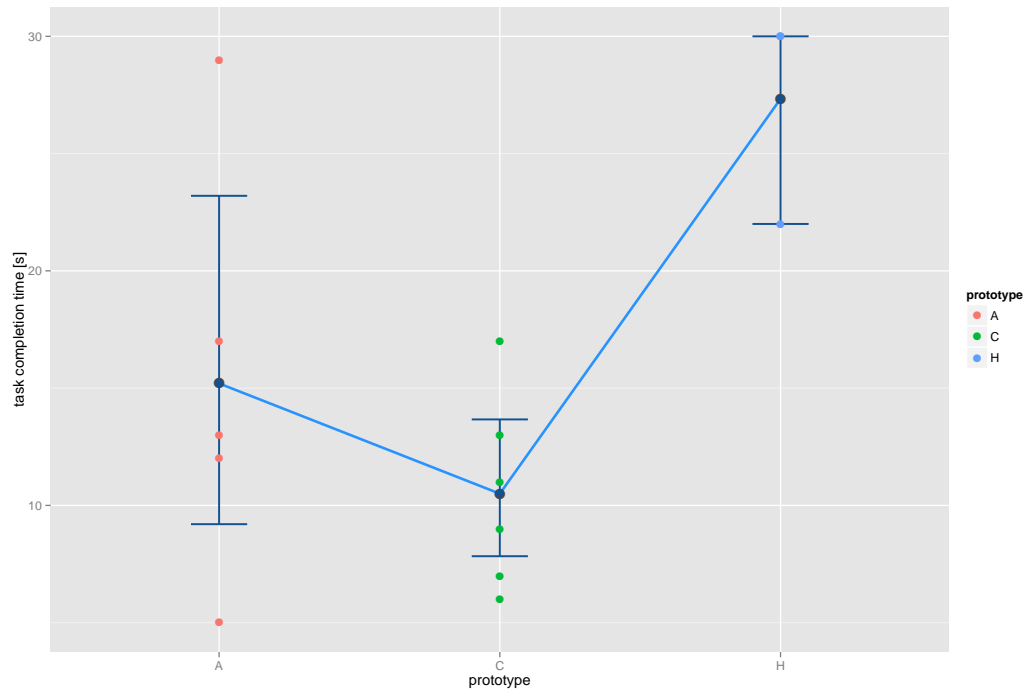


Figure 5.20: Task navigation graph task 3.2.1

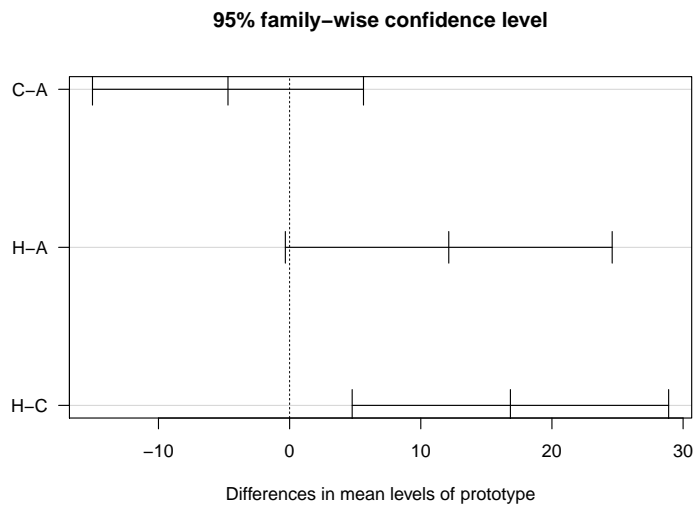


Figure 5.21: Task 3.2.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory slicing result

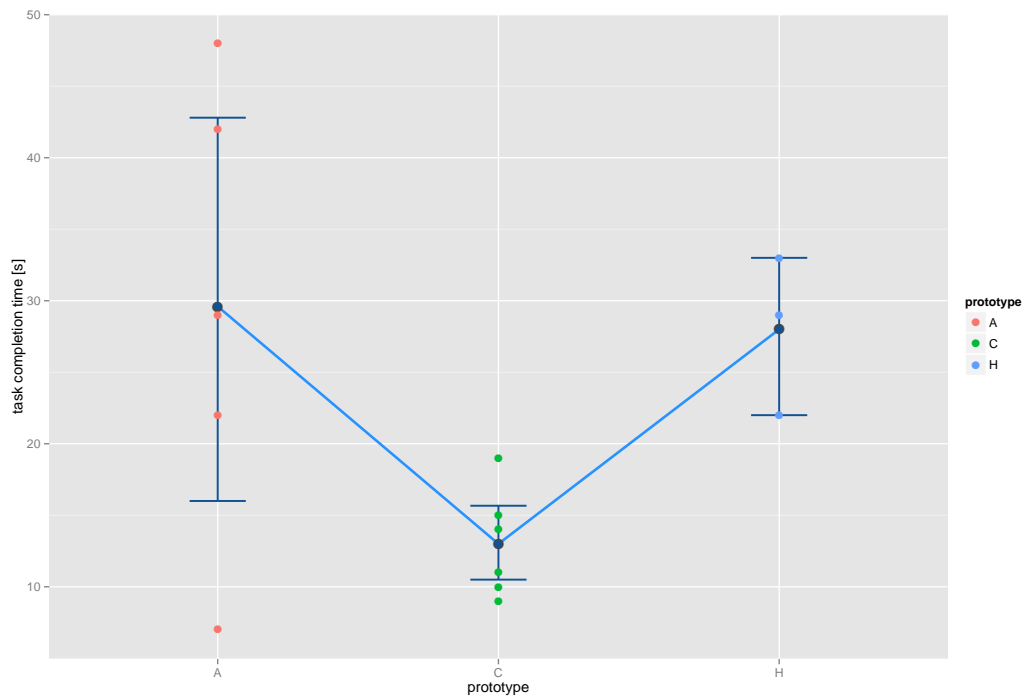


Figure 5.22: Task navigation graph task 3.3.1

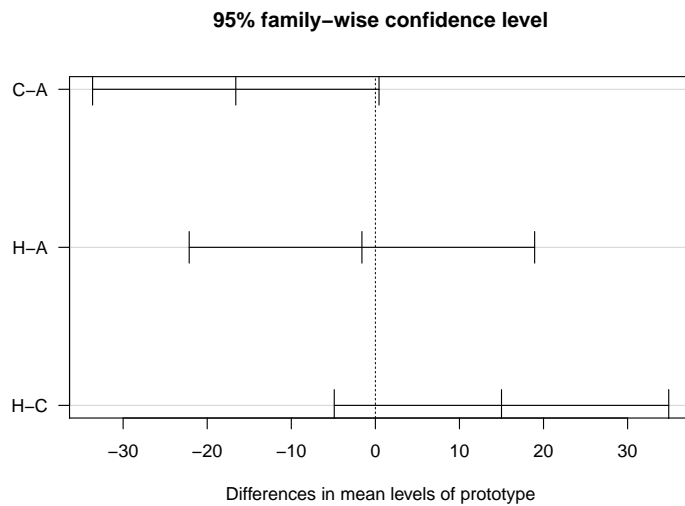


Figure 5.23: Task 3.3.1 shows the differences in mean of Tukey HSD post hoc comparison. The result does not differ significantly

where several connections have to be clicked through occurred in task 1.1.1, task 1.2.1, task 1.3.1, task 2.1.1, task 2.3.3 and task 3.3.3. Only in task 2.3.3 Azurite was even worse in all other cases Chronos History Slicing required the most amount of time to find a version.

5.3.5 Sketch Inspection Time

The sketch inspection time states the sum of seconds a participant considers a sketch.

We here concentrate on tasks only, where a sketch was attached to a commit.

I only consider tasks, where it is possible to clearly measure the sketch inspection time. These are in my opinion the tasks, where the user moves the cursor “on top of” the sketch and the cursor is not moved for a certain amount of time. When the cursor is moved out of the sketch, the time was stopped. This behavior could be observed among all participants.

Task selection
criteria

Set of considered tasks task 1.1.3, task 1.3.3, task 2.1.2, task 2.3.3, task 3.1.2, task 3.3.3 Whereas task 3.1.2 has only one sample per condition and is not considered in table 5.4 and table 5.4, because only one person in each condition looked at the sketch. In task 1.1.3 and task 1.3.3 and task 2.3.3. the ratio of sketch inspection to overall task completion time was more than 50% in case of CodeShape condition(see table 5.4), but the mean task completion time was the lowest among all conditions in these cases.

Alternative hypothesis H_A At least one mean of the sketch inspection time, needed to find a particular version with the help of one prototype, is significantly different to the sketch inspection time of another prototype.

task 1.1.3, task 1.3.3, task 2.1.2, task 2.3.3, task 3.1.2, task 3.3.3 There was no significant effect of sketch inspection

time for the three conditions among all tasks (see Appendix G for the result graphs).

General result I could not observe a significant effect of sketch inspection time. This was an expected result, because all sketches are the same among all tasks. For the Azurite and the CodeShape condition the sketch interaction functionality was identical, but Chronos History Slicing did not allow stepping through single strokes in order to reproduce the construction. That the stepping functionality did not have an effect is probably due to the fact that the sketches are rather trivial and consisted of one idea. It has to be further investigated, if more complex sketches, which consisted of several ideas and evolved over a longer period would affect the results. I did not consider more complex sketches, because of the short task completion time available in the user study. The sample rate of sketch inspection time was even lower as for task completion and initial navigation time. This does not insure authoritative statements about the effect of sketches on task completion time. In case of the task completion time in task 3.3.3, where a significant effect between CodeShape and History Slicing can be observed, the ratio of sketch inspection time to task completion time is nearly identical 0.30 and 0.33, but only one person inspected the sketch in this task for the History Slicing condition.

overall CodeShape
has the tendency to
be the fastest
prototype

	df_M	df_R	F	p
task 1.1.3	2	8	0.69	0.528
task 1.3.3	2	11	0.25	0.783
task 2.1.2	2	7	3.7	0.08
task 2.3.3	2	11	1.89	0.196
task 3.3.3	2	9	4.64	0.041*

Table 5.4: ANOVA of sketch inspection time comparing Chronos History Slicing, CodeShape and Azurite.

	σ_A	μ_A	σ_C	μ_C	σ_H	μ_H	p_{C-A}	p_{H-A}	p_{H-C}
task 1.1.3	11.27	28.0	33.26	45.33	20.02	27.0	0.631	0.998	0.533
task 1.3.3	43.75	68.6	10.01	85.75	40.35	77.4	0.766	0.923	0.938
task 2.1.2	39.72	75.67	7.9	28.5	19.76	32.67	0.087	0.145	0.973
task 2.3.3	15.18	59.5	32.17	74.25	10.31	49.33	0.562	0.714	0.172
task 3.3.3	14.39	53.0	18.32	31.83	0.0	80.0	0.146	0.346	0.06

Table 5.4: Tukey HSD post hoc test of sketch inspection time comparing (A)zurite, Chronos (H)istory Slicing and (C)odeShape

	μ_{A_s}	μ_{C_s}	μ_{H_s}	μ_{A_c}	μ_{C_c}	μ_{H_c}	$\frac{\mu_{A_s}}{\mu_{A_c}}$	$\frac{\mu_{C_s}}{\mu_{C_c}}$	$\frac{\mu_{H_s}}{\mu_{H_c}}$
task 1.1.3	28.0	45.33	27.0	103.0	91.25	147.8	0.27	0.5	0.18
task 1.3.3	68.6	85.75	77.4	173.0	161.0	186.4	0.4	0.53	0.42
task 2.1.2	75.67	28.5	32.67	151.25	143.75	109.0	0.5	0.2	0.3
task 2.3.3	59.5	74.25	49.33	140.0	137.5	184.17	0.43	0.54	0.27
task 3.1.2	20.0	21.0	38.0	97.75	86.67	94.33	0.2	0.24	0.4
task 3.3.3	53.0	31.83	80.0	146.8	105.33	245.33	0.36	0.3	0.33

Table 5.4: Compares ratio of mean sketch inspection time and mean task completion time(A)zurite, Chronos (H)istory Slicing and (C)odeShape

5.3.6 Feedback and Suggestions

This section reports the feedback that was given, including suggestions on how to improve some functionality. Most of the feedback was given after the participants have finished the tasks. One has to bear in mind that all three software artifacts are prototypes, so one cannot expect them to be as feature rich and polished as final products.

respect
prototype-status of
each prototype

General feedback All of the 17 participants stated that they are most convinced of CodeShape, after they tried all three on the different tasks. After that nearly all participants stated Azurite second behind CodeShape and Chronos was last.

They remarked that they are pleased with the appearance compared to the other two and that it is the fastest prototype for navigation between versions. Chronos inability to interact with lines and the resulting manual scrolling both-

ered them most.

implement
SHA-search in all
prototypes in future
releases

Nearly all participants wanted to have a functionality to filter/search for SHA's. We already thought about implementing it and it was also discussed in my pre-study. The problem is that all plugins must have had this feature then and this would have a drastic impact on performance compared to the original versions. Another problem with this approach is, that we would not have been able to observe a "real" search strategy. Users would just have typed in the SHA's and it would lead them directly to the version they wanted. The SHA was more thought as a help in order to know that one has found the right commit.

feature request:
history of
exploration itself

Participant: "an indicator where one was before would be nice" This comment was made twice. One person reported this as a wished feature for CodeShape and another for Azurite. I already thought of this feature for CodeShape beforehand, which provides browser alike functionality, meaning that the user can go to some previous commit and then step back and forth between commits. I implemented this feature in my prototype, in consequence of some issues and bugs related to this the buttons where hidden during the study, so that this feature was not usable in the study.

Users suggested
that version
highlighting has to
be more
eye-catching

CodeShape Feedback A participant did not notice that a commit circle was highlighted, after having triggered a context menu command. This was reported another two times and the subjects suggested, that the highlighting must be more eye catching. For example, the circles should blink for a short period and maybe increase the size in this period.

A diff view should
show two versions
side by side

One user requested a diff view like in the eclipse IDE (see Fig.5.24), placing two versions side by side. He is most familiar with this layout. In CodeShape he was not sure to which versions the diff output belongs. It compares the current version with the previous one. He suggested that you can select an arbitrary second file for side by side comparison. It would be superior to choose between side by side comparison and single version diff, as it is possible in Kaleidoscope [2012].

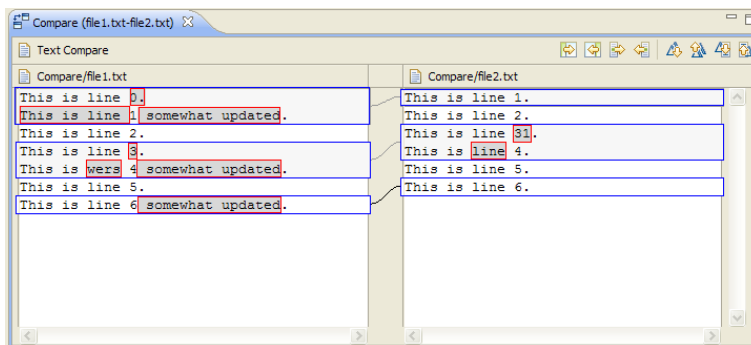


Figure 5.24: Eclipse compare editor - Compare two different files and show their differences (or two different versions of a file, which is not shown)

Two users did not expect that the sidebar automatically faded away, when a commit is selected.

Three user asked me, why the context menu command to highlight a future or past version, did not highlight all commits where something has changed. So, that you can step through them, without activating the context menu for every version. This is a technical limitation, which was already discussed before the user study. The problem is that a line that has nothing in common with the current one, would get highlighted, too. An improvement would be to calculate the Levenshtein distance for two lines and highlight only those beneath a given threshold, which is further described in 4.2.1.

Two participants requested an indication beneath the timeline. For example, a label written beneath intervals spanning a year.

Three participants reported: “how do I know which version I am looking at on the timeline” It indicates that redesigning native UI elements (here Cocoa’s NSSlider [2014b]) can lead to misinterpretation, because it was perceived as a completely new element. Redesigned tick marks caused this effect. The tick marks have been redesigned to “commit-circles”. The indicator, showing which commit one is currently looking at, was not modified. This should

Three user asked for a consecutive highlighting of changes

The timeline should have time labels

Partly reconstructing native UI elements leads to confusion

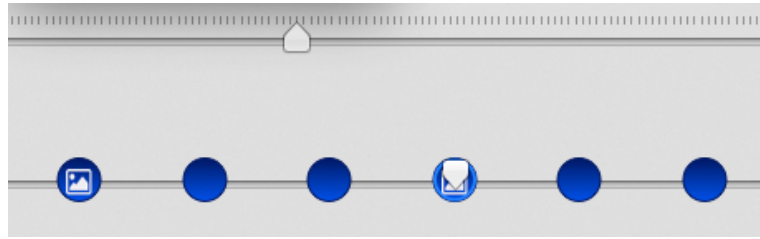


Figure 5.25: On top one can see the original NSSlider, without any modifications and on the bottom one can see the modified version used in CodeShape to navigate through commits. Blue circles are used with white icons on top of these, to visualize that there is a sketch available, instead of the original small gray tick marks.

be improved in a future version, making the slider knob more stand out. In figure 5.25 one can see them both in comparison, at the top there is the original NSSlider and at the bottom the modified one.

Azurite's small-sized rectangles affect usability

Azurite Feedback Most of the users said something like “the rectangles should be larger, because they are hard to hit with the mouse”. My observations showed that on average two to three clicks on a rectangle are required, until it was selected, which affirmed their statements. One problem is the small size of a rectangle, which can become as small as 20 pixel in width and 6 pixel in height. This roughly corresponds to one third of the MacOS X default mouse cursor (see Fig.5.26).

There are too many rectangles visible during navigation, requiring much scrolling

The level of detail should be reduced, because there are too many rectangles in every session. Relating to that one participant suggested to aggregate changes. Contiguous line numbers of the same change type can be aggregated into a single change. Indeed Azurite implements a compact mode (see 4.2.2), compressing the timeline by removing gaps in between each change, but the number of rectangles and its size stays the same. Azurite can be improved by integrating a filter that reduces the number and size of rectangles.

The popup should contain more self explanatory information

The information provided by the popup, which occurs

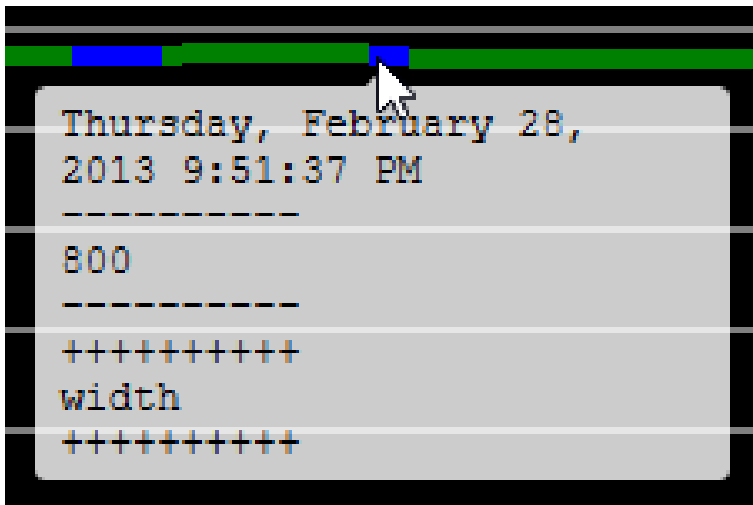


Figure 5.26: Azurite rectangle popup: The blue rectangle, where the mouse points on, is an edit operation performed at 9:51pm, 02/28/2013. It replaced a constant "800" with a variable named "width". The date formatting corresponds to the original one, not the one used in my study.

when one hovers a rectangle in Azurite, was not helpful to one user. Another one suggested to add the line number in front of the changed line, in order to make the popup more helpful.

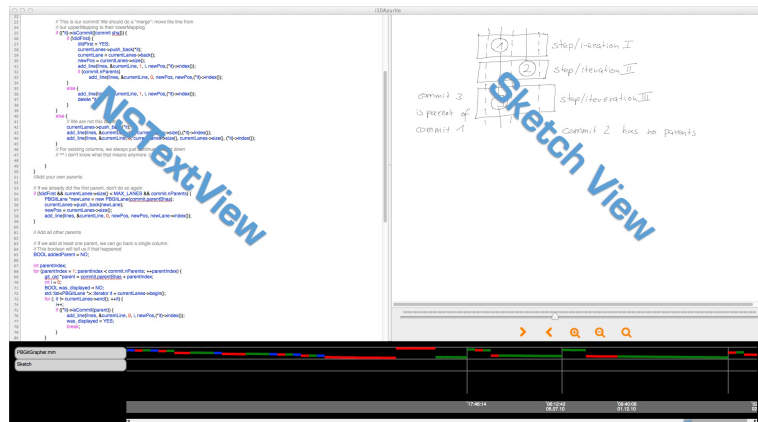
Sometimes the timeline scrollbar stopped working and in this case only restarting the plugin solved this problem. This happened twice and I also observed it during the preparation of a new study. It is probably an issue related to Azurite itself, as it only affects the original scrollview of Azurite (see Fig.5.27).

The scrollbar
sometimes stopped
working

Nearly all participants did not discover the vertical slider and thus its functionality remained unknown. The slider 5.28 appears by clicking on the bottom of the timeline. I found this slider by accident and two months later I could not remember how to bring it up. It took me some time to figure it out again.

It was not obvious
that there exists a
hidden vertical slider

Only one participant discovered the slider.



Azurite's original timeline scrollbar

Figure 5.27: In the screenshot the modified Azurite plugin can be seen, with its original timeline at the bottom.

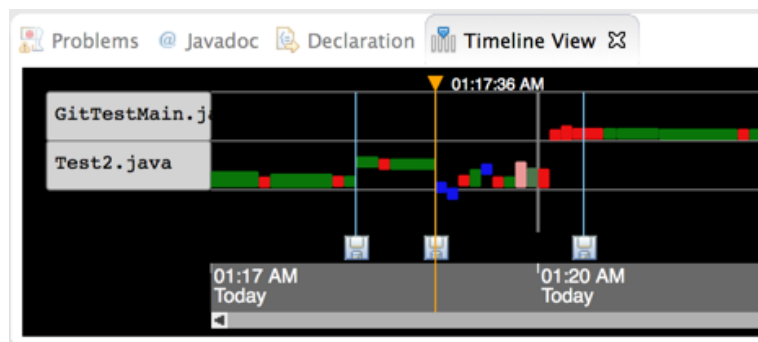


Figure 5.28: The figure shows an image section of Azurite. The orange vertical bar in middle, with an arrowhead alike top pointing downwards, represents the Azurite vertical slider. It only becomes visible by clicking on the gray area at the bottom.

Chronos needs
search and
interaction
capabilities like a
context menu

Chronos Feedback The majority of participants requested a search feature and a context menu, similar to the one of CodeShape and Azurite. In task 2.2.1 (see 5) all participants asked me, if there exists search functionality, but the variable row can only be found, by reading lines. In CodeShape and Azurite searching is possible by pressing “cmd + f”.

In my re-implementation of Chronos I used `NSTextView`, which allows for searching and line interaction. During the user study searching and line interaction was disabled, because it is impossible in the original plugin. The reason for that is, that the original implementation uses an `svg` instead of a `NSTextView`.

Another aspect, which was reported is to get rid of unnecessary and tedious scrolling.

Chronos should hide unnecessary information

Double clicking on a vertical bar in Chronos should automatically scroll to the source code view. Related to that five people said that time intervals, where no changes occur, should be stripped away. It has to be further investigated if this removes the visual uniqueness of every repository. A gap in development, could help the user to remember the repository more easily. The user should be able to toggle a compact mode like in Azurite.

Also the date labels have been explained to every participant, it led to confusion, because users did not remember, which part of the date, in for example, “07\08” represents the month and which one the year. During a study I was asked several times, if I can explain the format again. It would be more explanatory if the year is written in a 4 digit format and when the date is formatted according to the participants locale.

The unusual date format causes trouble

During the use of Chronos I observed that the participants just scrolled horizontally in task 1.3.2 to find out how the last two lines have changed. In doing so they skipped over some versions, because they are not in the visible part of the view. These versions have fewer lines than the other versions and would only become visible by also scrolling vertically.

Users unknowingly skipped useful information

This problem is visualized in figure 5.29.

By coincidence there were no change to the last two lines of task 1.3.2 in an “invisible” version. If there had been changes, chances would have been high that the user would have missed and scrolled over them.

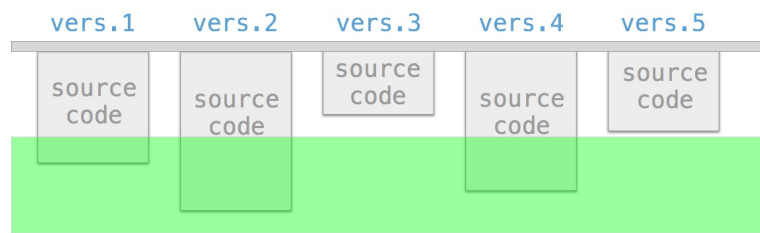


Figure 5.29: Chronos information loss issue - The green semitransparent rectangle represents the visible area of the screen, when a user scrolls horizontally. The third and the fifth gray rectangle from the left have no indication in the visible area that they exist.

There should be an indicator, which reminds the user of remaining, but currently not visible versions
Orientation change and hidden detail cause confusion

To overcome this issue Azurite needs some indication that there is an "invisible" version. An overview map like the one being used in Sublime Ceriu [2011], Xcode's mini map plugin Ceriu [2013] or another indication could help to solve this issue.

Two problems occurred among all participants. The first problem is that connections are revisited, because participants forget, that they already checked them.

The second problem is that participants do not know how vertical connections are ordered (from top to bottom or vice versa). The orientation is changed from vertical to horizontal. In case of several connections for the same day, one does not know if the earliest commit in terms of time is the up most or the one at the bottom. Several participants annotated this issue (see Fig.5.30).

"Sketches definitely help..."

Sketch Value "Did the sketches helped solving the tasks and did they help in general." "Could you imagine to use sketch functionality?" On answer was: "They would definitely help, but she or her colleagues would probably not redraw a sketch.". If sketches are drawn with a digital pen, no redrawing is required.

More than the half of the participants reported that the sketches helped a lot to solve the tasks and that they could see the potential benefit during their usual work as a devel-

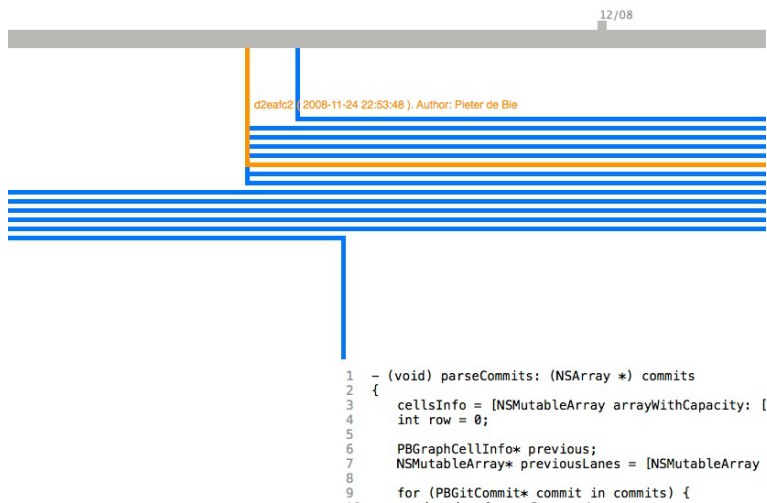


Figure 5.30: Modified version of Chronos History Slicing: The blue connections start at a date entry of the gray timeline at the top (two blue vertical bars reach the top gray bar). The position indicates, that they belong to November 2008, because on right one can see 12/08, which indicates that there the month December begins. One of the blue connections has been selected (highlighted in orange) and the date, time and SHA is only shown for this selection

oper. One participant reported that: “the sketches helped a lot, sometimes more then reading and understanding the code. Sketches are a good additional support”.

A few participants tried to move the sketch with the mouse, which is impossible in the prototypes. They suggested to integrate the ability to move sketches with the mouse instead of using the scrollbars.

Some users had difficulties reading some words in the sketch, when strokes were missing, because occasionally the Inkling did not track individual strokes. As a conclusion the quality and accuracy of digital pens like the Inkling should be improved and the digitized version should be exactly equal as the version drawn on paper.

I observed that experienced git users had fewer problems with sketch interpretation. A reason for this can be that

familiar mouse
interaction of
sketches for a future
version

The use of Inkling
resulted in some
quality drawbacks

they are more familiar with git terminology and thus can fill in the missing strokes/words. Same holds true for reserved Objective-C names in the sketch, where the level of experience in Objective-C resulted in improved readability, when something was hardly readable.

The sketches have been used to explain the answers. Some participants hovered with the mouse cursor over a part of a sketch and verbalized the part. This illustrates that sketches can support discussion.

Another aspect observed was that participants rarely switched back and forth between source code and sketch interpretation. Switching between them occurred at most three times. I observed that nearly all participants had a phase considering the source code in a first step and then a phase where the sketch was considered. A comparison of source code lines with sketch indentifiers was not observed.

Chapter 6

Summary and Future Work

In this Chapter I will briefly summarize my results and contributions. After that some ideas on future work are given.

I presented and compared three software prototypes, Azure, Chronos History Slicing and CodeShape, that support source code history exploration. All of them access the source code history of a git repository and provide different visualization approaches to navigate through history. An exploration is assisted with sketches, providing different sources of information.

6.1 Summary and Contributions

Based on the work by Spsychalski [2013], I came up with the idea of integrating sketch functionality into an integrated development environment (IDE) and connect sketches to a particular version of a source code entity and tried to improve process.

connect sketches to
source code entities

I brainstormed functionality and improvements that have to be included in a prototype, attaching to Lukas ideas. It

A mobile IT company was visited and interviews have been conducted	formed the basis for the analysis of related work, in which I explored source- and sketchhistory related topics and ways of automation. I visited a mobile IT company, which develops mobile applications mainly for the financial services industry. First I had a meeting with the head of application development and the team leaders of the different mobile platforms they support. Together we discussed sketches combined with source code. Next I talked to a person of the design team, which told me that wireframes are the primary source used for discussions with the developers. Thereafter I observed the developer team. They rarely draw sketches, because they receive the wireframes of the design team and if this does not suffice the interface builders of the respective platform is used for ideation. Every office has a whiteboard attached to the wall and all employees make use of it, except for the developers.
I implemented a Xcode plugin called CodeShape	Based on the impressions and conceptions from the initial meeting, I designed and implemented CodeShape from scratch. CodeShape is functional Xcode plugin and provides the ability to parse sketches drawn with the Wacom Inkling and connect them to source code entities like a method in a source code file. Next I implemented and repurposed IBOutlet connections [2014d], enabling the user to connect Inkling sketches to source code entities.
CodeShape provides a sketch- and source code history	A source code history exploration window of CodeShape, parses the repository in order to find the versions in which a source code entity changed. A chronological list of source code entities is then displayed to the user, including versions to which sketches have been connected to. The user can step through the history of a sketch, connected to a source code entity, or through the history of a source code entity. Also a user can interact with source code lines of an entity. A context menu provides the ability to show a past or future version of one or several lines.
two additional prototypes have been implemented and extended with sketch capabilities	The software prototype Chronos was implemented based on information of a research paper [2013]. My Azurite implementation makes use of its original timeline view, whereas the rest was implemented from scratch. Chronos and Azurite were extended with sketch capabilities, whereas the capabilities for Azurite are identical to

CodeShape and Chronos just allows zooming into a sketch.

In my user study I evaluated these three prototypes on a set of tasks, which I derived from LaToza and Myers [2010]. Only in a few cases a significant result was observed. Thus there is no overall evidence that CodeShape outperforms the other two prototypes, which might be due to the small sample size and larger differences among those samples, but it can also be the reason that CodeShape does not have a significant effect. All of the 17 participants stated that they are most convinced of CodeShape, after they tried all three on the different tasks. After that nearly all participants stated Azurite second behind CodeShape and Chronos was last.

a user study
compared the three
prototypes on a set
of invented tasks

6.1.1 Limitations

Our approach has some inherent limitations. One limitation arose from the fact, that the prototypes just consider its own functionality, without integrating the usual functionality like for example the ordinary version editor of Xcode or the one of eclipse.

usual IDE
functionality was not
usable during the
study

The participants were only presented with the history of a method, without its context. Other behavior in solving the tasks might have observed, when the rest of the file or the whole repository would have been available to the participants.

Another limitation is that my sketches have been drawn by myself, with the task set already in mind, as I could not find any sketches for the repository considered in my initial study. This might have biased the results. Same holds true for the task set itself.

self drawn sketches
might have biased
the result

6.2 Future work

6.2.1 Improve the prototypes

In section 5.3.6 several improvements for each of the prototypes were proposed, according to problems that occurred during the user study.

git alike functionality
for sketches

The sketch capabilities are rather limited at the moment, but research has shown that branching & merging of sketches would be possible [2011]. As an example, one can come up with collaborative sketching, offering more flexibility as Gambit [2012], and further explore the history of sketch data.

There are already tools that can detect refactorings of source code elements, but no comparable tools exist for sketches. Combining source code and sketch refactoring detection would provide the possibility to adjust one of the two entities in case of change. Also connections can be automatically removed in the case they become invalid.

6.2.2 Sketch and source code entity consistency

a tool by Ginosar et
al. can probably be
repurposed to
ensure sketch
consistency

It is hard to determine, when identifiers like method names used in a sketch have become invalid. Invalid means that the identifiers in the sketch have become inconsistent with the source code, because of refactored code. Ginosar et al. [2013] developed a tool to create multistage source code examples. The back propagation functionality used in their tool could be utilized in case there is a new sketch attached to a new source code entity. An old version of this source code entity would use the identifiers utilized in the new the sketch, because they would be back propagated from the new source code entity to the old one. Same holds true for the other direction and thus propagation can be a way to ensure consistency of source code and sketch. This could improve the lack of ordinary source code refactoring detection to determine if a connected sketch has become inconsistent with the source code.

Parallax

I considered an iPad prototype for drawing sketches during an early phase of my thesis (see 4.2.1), but it still has some disadvantages.

The problem with today's display surfaces like the one of the iPad is that they all suffer from a more or less intense parallax effect, resulting in a drawing experience that does not feel as natural as drawing with a normal pen.

There should be neither an electronic parallax nor a visual parallax effect. Visual parallax depends on the thickness of the screen cover glass. As a user it feels unnatural and it is harder to draw precise shapes and strokes touching each other.

Drawing on today's displays feels unnatural, because of the parallax effect

Electronic parallax is caused due to the fact that the electromagnetic (EM) digitizer is behind the display and its counterpart, the coils inside a pen, are not directly on the tip of it Rodriguez [2014]. This enforces the effect of unnatural drawing with a feeling of distance between the touching point of the pen tip and the actual drawing visualization.

If it becomes possible to reduce the parallax effect to a bare minimum, then an iPad prototype may become a superior replacement to ordinary or digital pens.

iPad App

Every stroke in a sketch, should be automatically persisted and editable. Editable in this context means that a stroke should be removable, transformable (change shape, color, z-index etc.) and it should be possible to label strokes, group them together to build up a polyline object.

An iPad app should offer a lot more degrees of freedom as opposed to pen & paper

Basic shapes should be available for selection. So, you can drag an ellipse, a rectangle and a triangle onto the canvas area of the surface. If strokes had been persisted, the user should be able to browse through and edit the history of his

set of strokes and shapes.

Appendix A

Informed Consent Form

INFORMED CONSENT FORM

Evaluation of history exploration assisted by sketches Principal Investigator: Torben Schulz, Media Computing Group, RWTH Aachen University

PURPOSE OF THE STUDY

This study is conducted in relation to my master thesis at RWTH Aachen University. The goal of the study is to evaluate different representations of history exploration software prototypes and the usefulness of sketches connected to certain history revisions in order to answer questions about the intention and history of source code entities of a software repository (i.e. file, class, method).

PROCEDURE

You will be asked to perform two tasks per condition in which you should navigate through source code history. You will be asked to answer some questions. The study will take up to 50 minutes.

RISKS / ALTERNATIVES TO PARTICIPATION

There are no risks associated with participation in the study. Participation in this study is voluntary. You are free to withdraw or discontinue the participation at any time.

CONFIDENTIALITY

All information collected during the study period will be kept strictly confidential. You will be identified through identification numbers. No publications or reports from this project will include identifying information on any participant.

- I have read and understood the information on this form
- I agree to being filmed during the study (screencapturing & audio recording)

_____ Date: _____
participant's signature

gender male female

age _____years

occupation _____

programming experience _____years

Do you know about Objective-C?

yes

no

Do you know about the version control system git?

yes

no

I want to take part in the lottery to win a 50€ itunes or play store gift card.

email address

Appendix B

User Study Form

Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department RWTH Aachen University

sketchassisted development

user study

Comparison Of Three Source Code Visualizations

Torben Schulz

DATE: *24th June 2014*

1 CONDITION I (15 MINS)

1.1 INTRODUCTION TASK (5MINS)

1. Please navigate to the code that was implemented at *DATE* **24.11.2008 22:54** (commit *SHA* **e570c3e**)
2. How do you know that you have found the right point in time?
3. The change introduced the use of **struct & malloc** (*LINE* 8). Why was that done?

1.2 FIRST TASK (INTENT AND IMPLEMENTATION, 5MINS)

1. Have a look at *Line* 61 to 66 in commit with *DATE* **18.09.2008 01:27** (commit *SHA* **6e978dc**). What happens when the number of parents of the current commit in the if-block is equal to 0?
2. Please find out if somebody fixed this and when it was fixed the first time?

1.3 SECOND TASK (HISTORY, 5MINS)

1. The line:

Listing 1: source code line

```
int maxLines = (previousLanes->size() + nParents+2)*2;
```

- can be found e.g. in the commit at *DATE* **02.12.2010 02:08** in *LINE* 9 (commit with SHA **9488afc**). Who added/introduced this line with some minor changes?
2. How were the last 2 lines of the method changed over time? Just consider textual changes before the last closing bracket(s).
 3. Why was **PBGitLane** at *DATE* **28.08.2008 00:25** introduced and what does current lane means in this context? It was introduced at commit with SHA **a294d91**.

2 CONDITION II (15 MINS)

2.1 INTRODUCTION TASK (5MINS)

1. Please navigate to the code that was implemented at *DATE 27.08.2008 21:51* (commit *SHA bbeedd1*)
2. The change introduced the use of *PBLine* (e.g. *LINE 29* or *43*). Why do we need *PBLine*?

2.2 FIRST TASK (INTENT AND IMPLEMENTATION, 5MINS)

1. Have a look at line 102 in commit with *DATE 27.08.2008 23:31* (commit *SHA 727e42f*). What happens when we delete this line or comment it? Does the software stop working or crash?
2. Please find out when `LINE 4 int row=0;` is not used anymore?

2.3 SECOND TASK (HISTORY, 5MINS)

1. The line:

Listing 2: source code line

```
PBGitLane *currentLane = NULL;
```

can be found e.g. in the commit at *DATE* **25.11.2008 18:12** in *LINE* 11 (commit with SHA **777f70f**). Who added/introduced this line?

2. How has the first line of the method changed over time until last commit?

3. Why was **git_oid** introduced? It was introduced at *DATE* **24.11.08 22:54** in *LINE* 68 (commit with SHA **8372aee**).

3 CONDITION III (15 MINS)

3.1 INTRODUCTION TASK (5MINS)

1. Please navigate to the code that was implemented at *DATE 22.03.2010 19:09* (commit SHA `6599ce1`)
2. The change introduced the use of:

Listing 3: source code *LINE 20 and 21*

```
if(!*it) // This is an empty lane, created when the lane
        // previously had a parentless(root) commit
    continue;
```

Why do we need these lines?

3.2 FIRST TASK (INTENT AND IMPLEMENTATION, 5MINS)

1. Have a look at *LINE 9* and *LINE 10* in commit with *DATE 05.07.2010* (commit *SHA 69827e9*). What happens when we set `maxLines` to 0? Does the software stop working or crashes?
2. Please find out when *LINE 9* `int maxLines = (previousLanes->size() + nParents + 2) * 2;` was slightly changed? So when was the size of the upper bound changed?

3.3 SECOND TASK (HISTORY, 5MINS)

1. The line:

Listing 4: source code line

```
PBGitLane *currentLane = NULL;
```

can be found e.g. in the commit at *DATE 01.12.2010 09:40* in *LINE 13* (commit with SHA `0edbc05`). Who added/introduced this line?

2. Consider the following line:

Listing 5: source code *LINE 50*

```
// ^^ I don't know what that means anymore :(
```

There exist commits where there is a line directly beneath this line. Try to find at least one of them.

3. Why was `[lane index]` introduced? It was introduced at *DATE 28.08.08 00:25* (commit with SHA `a294d91`).

Appendix C

Task Completion Time Graphs

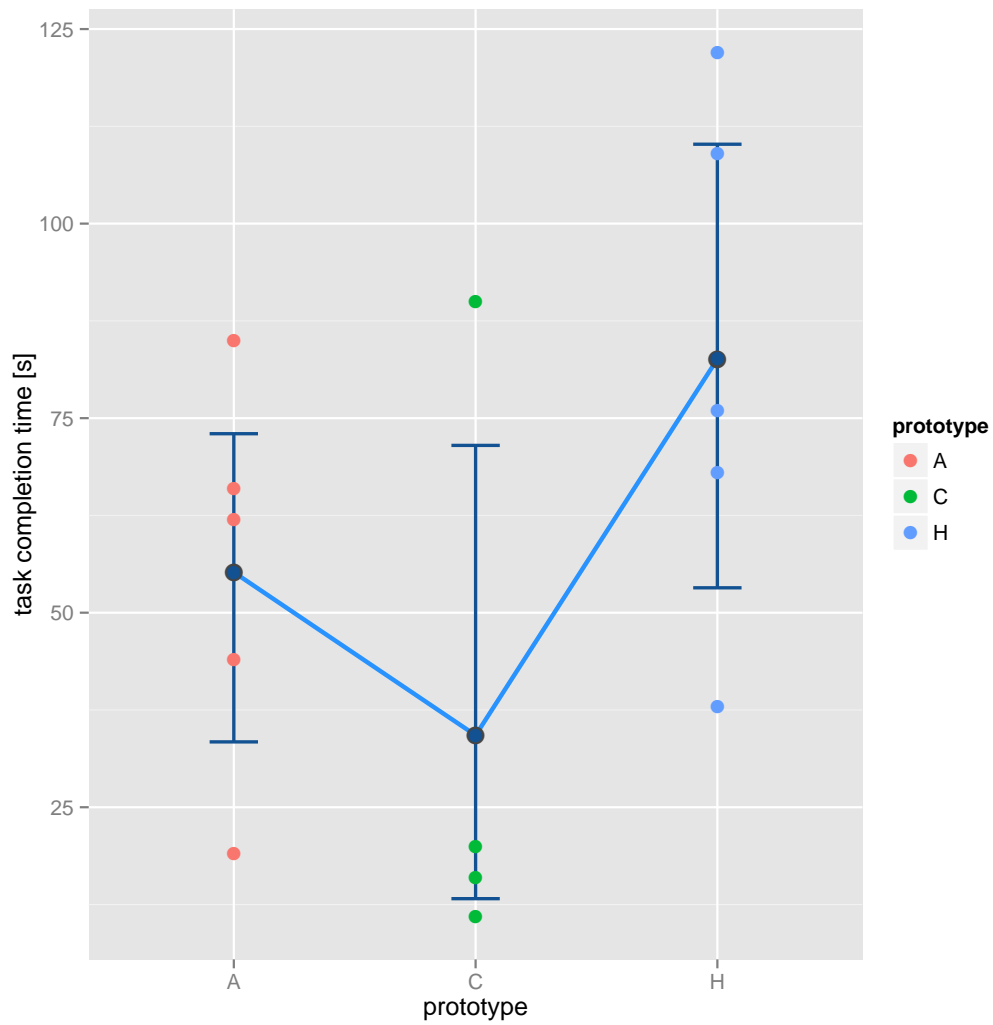


Figure C.1: task completion graph task 1.1.1

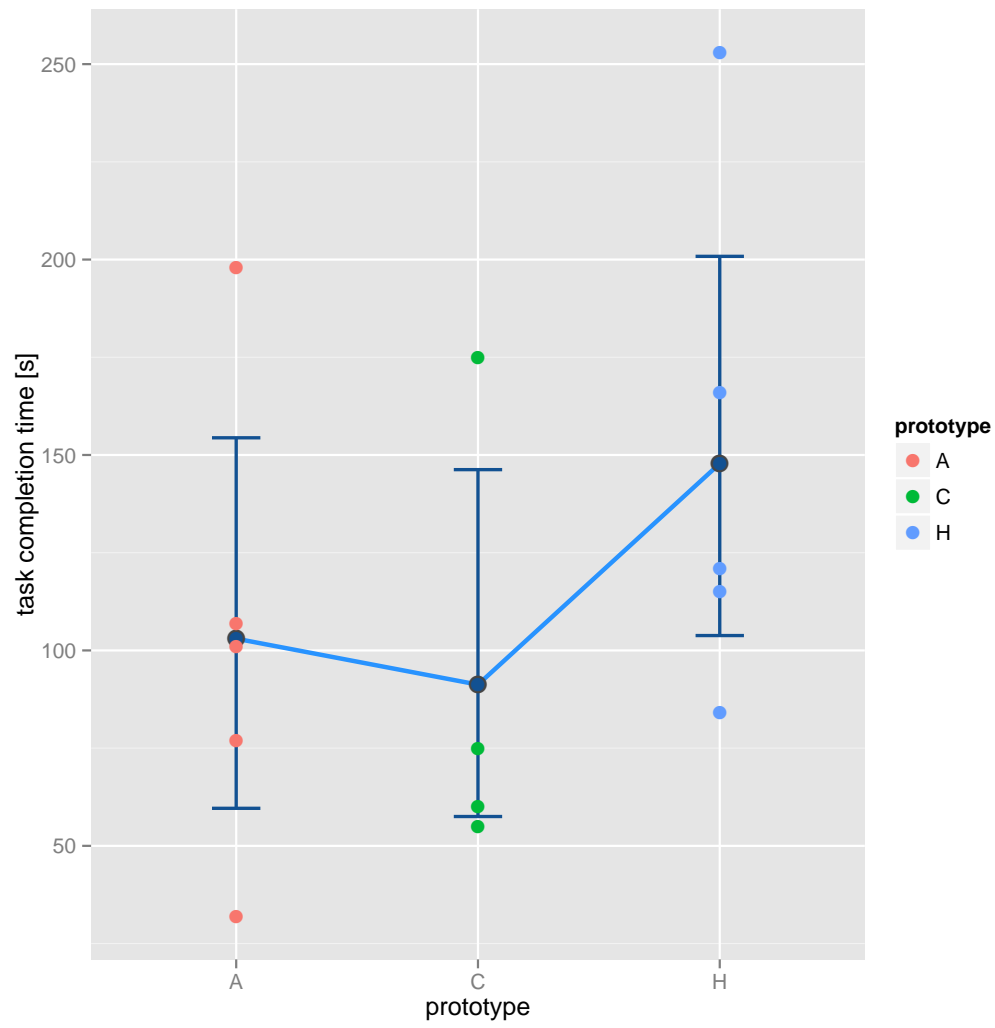


Figure C.2: task completion graph task 1.1.3

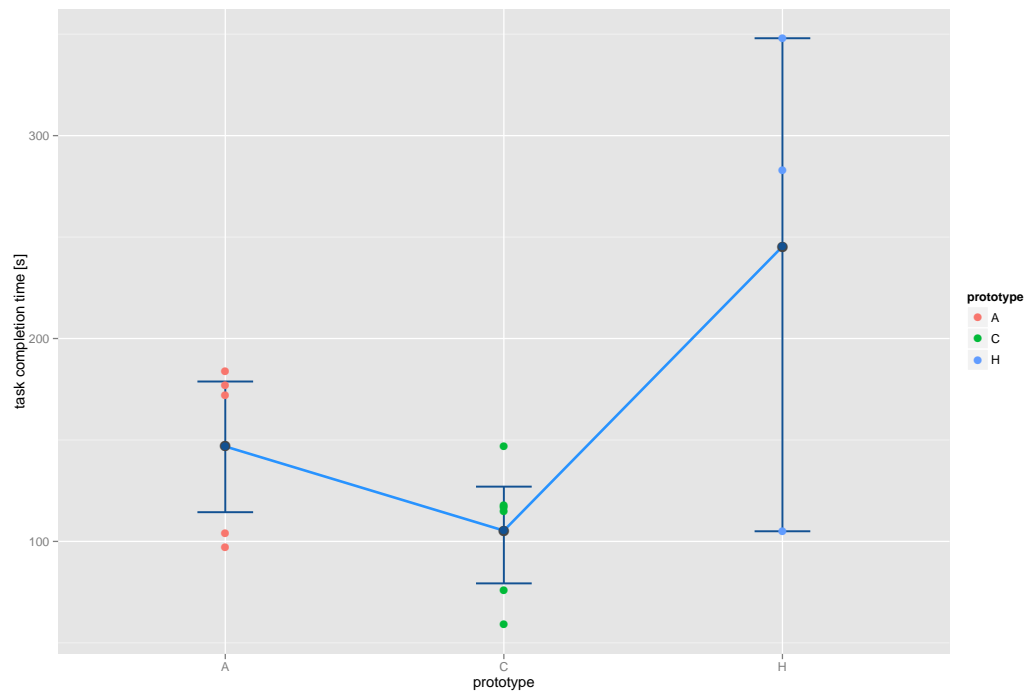


Figure C.3: task completion graph task 1.2.1

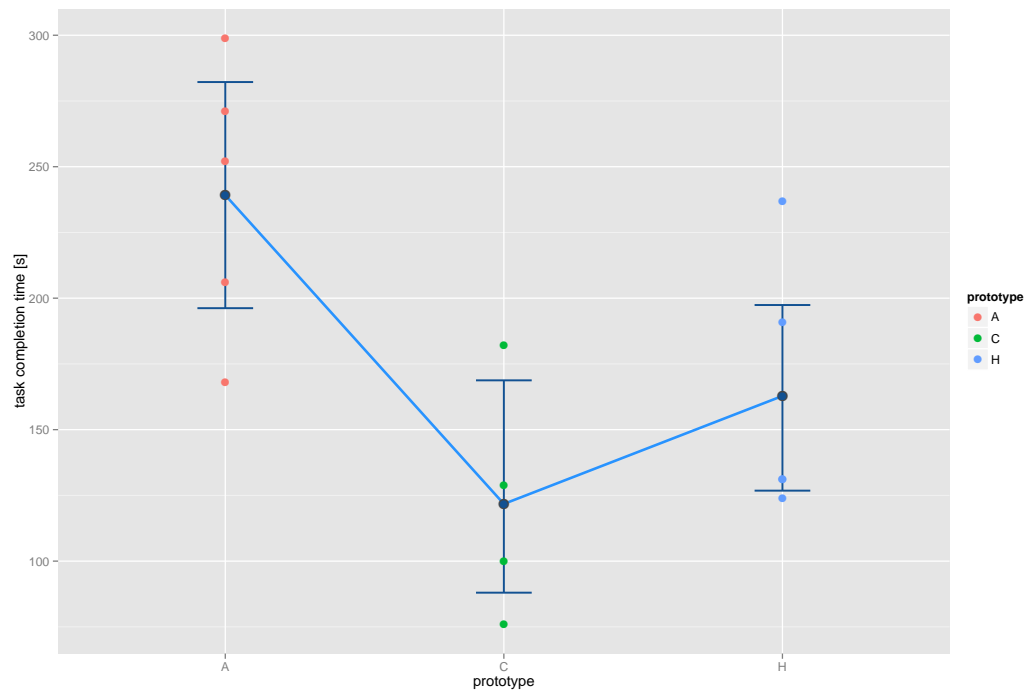


Figure C.4: task completion graph task 1.2.2

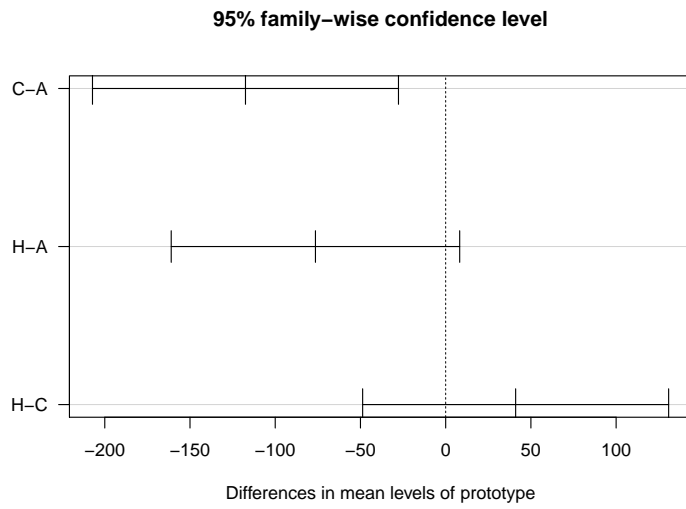


Figure C.5: Task 1.2.2 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result

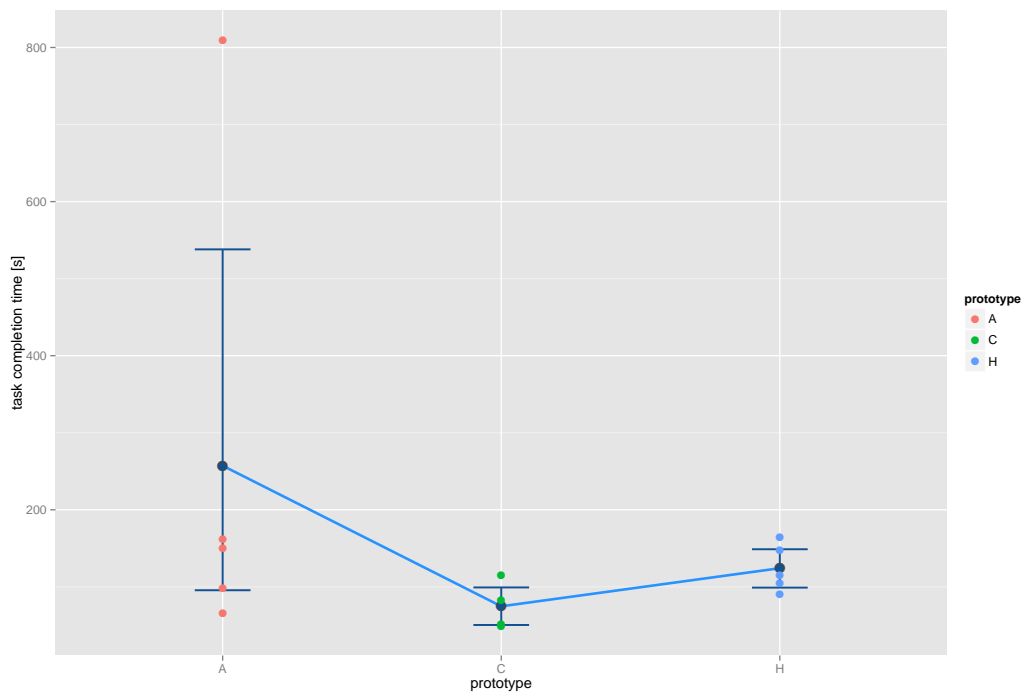


Figure C.6: task completion graph task 1.3.1

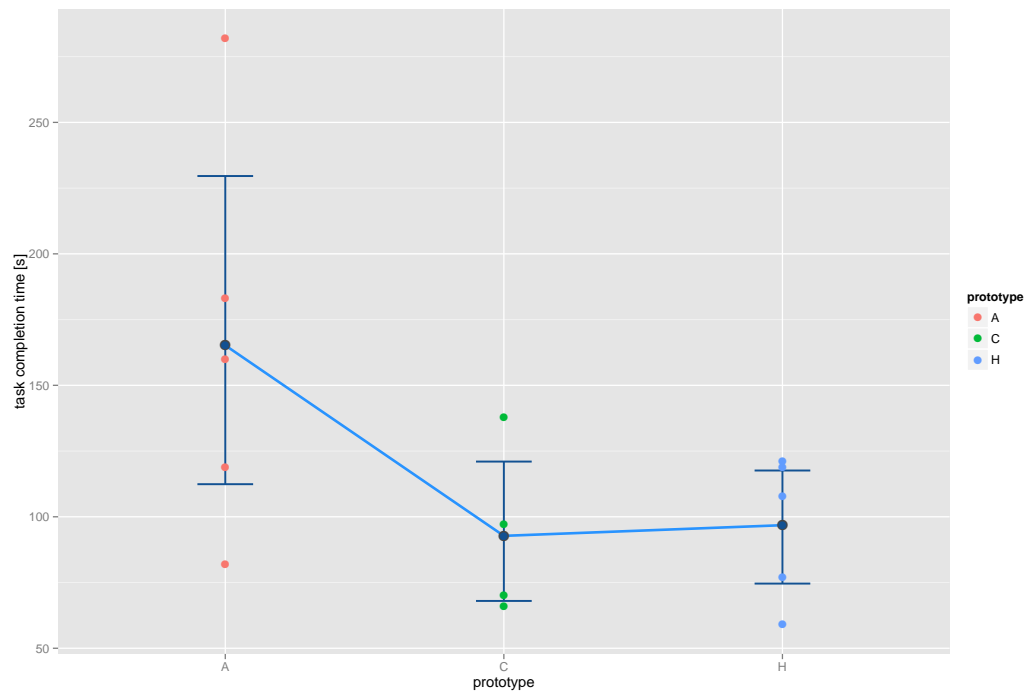


Figure C.7: task completion graph task 1.3.2

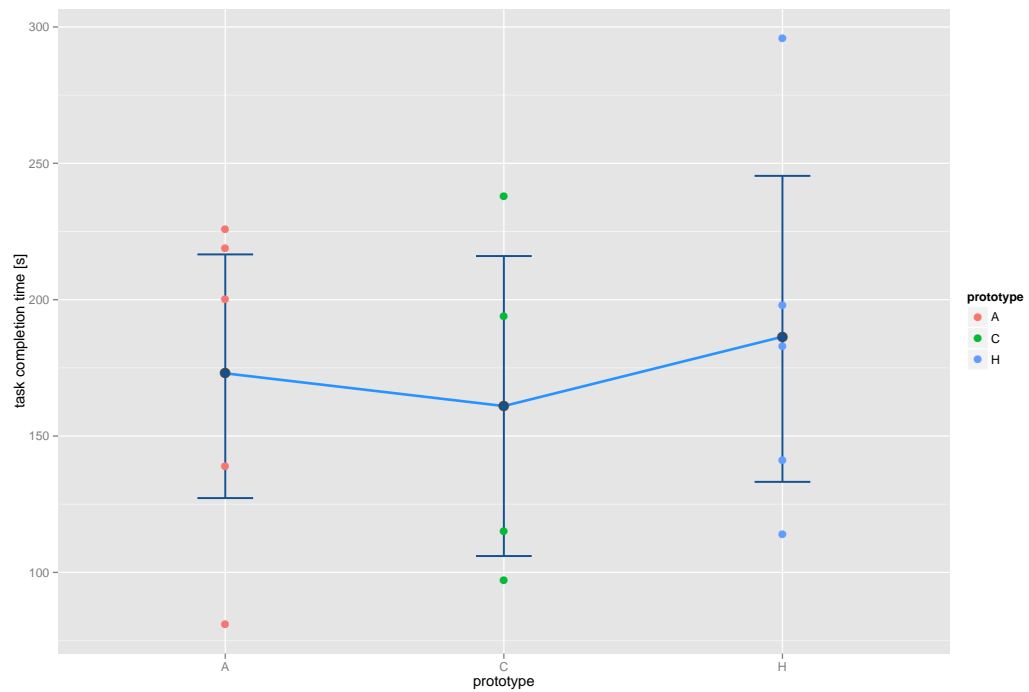


Figure C.8: task completion graph task 1.3.3

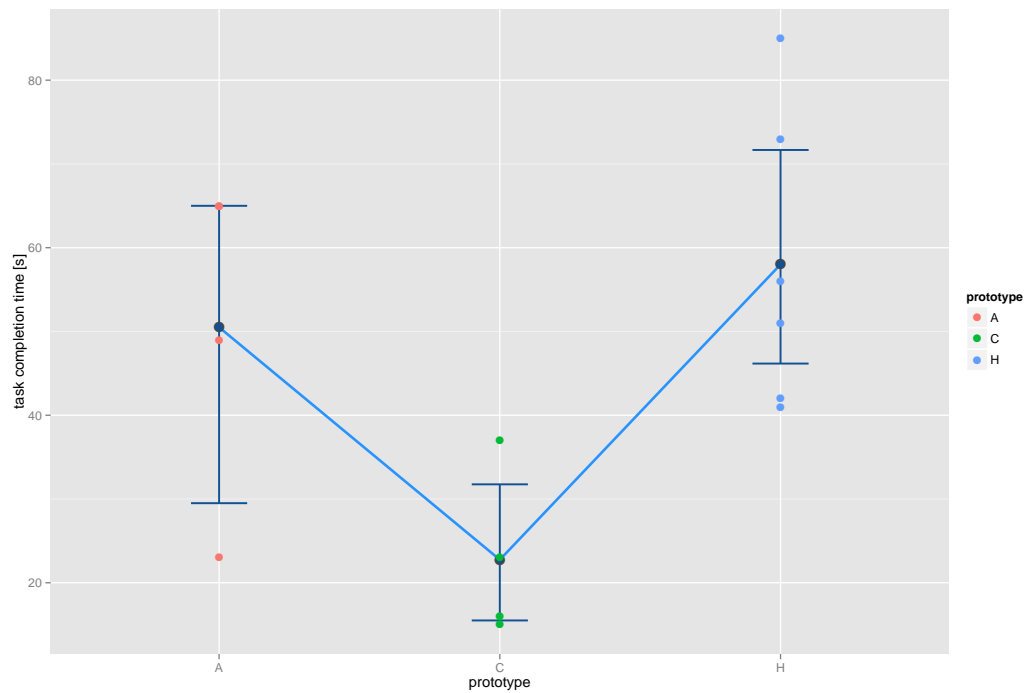


Figure C.9: task completion graph task 2.1.1

95% family-wise confidence level

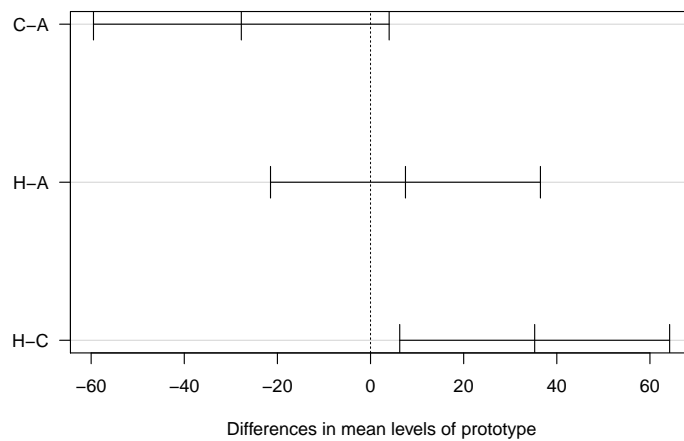


Figure C.10: task 2.1.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory Slicing result.

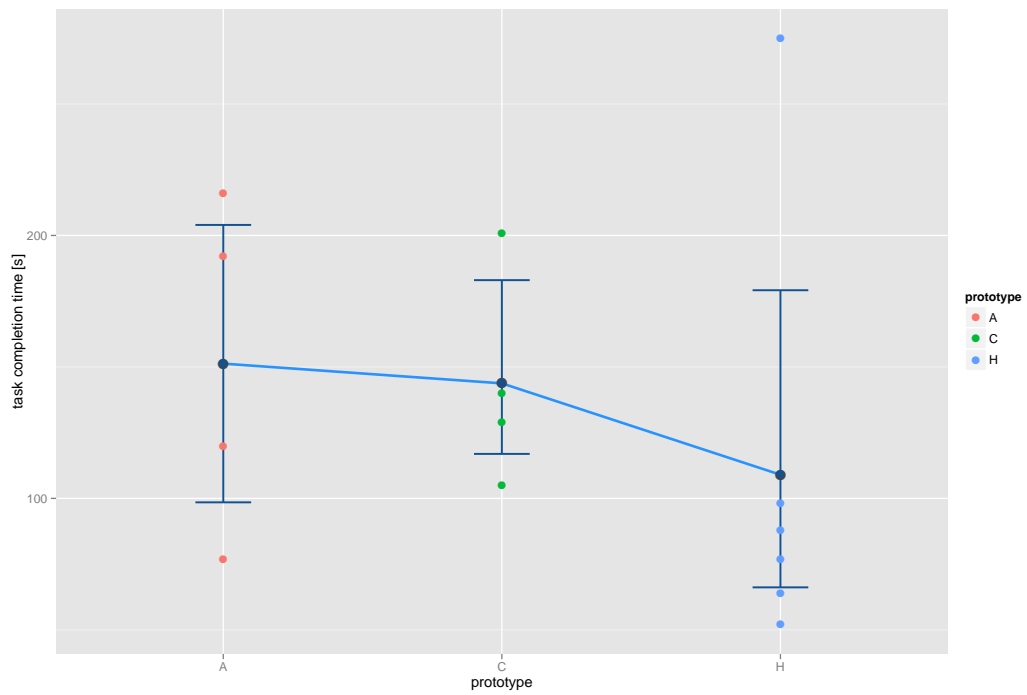


Figure C.11: task completion graph task 2.1.2

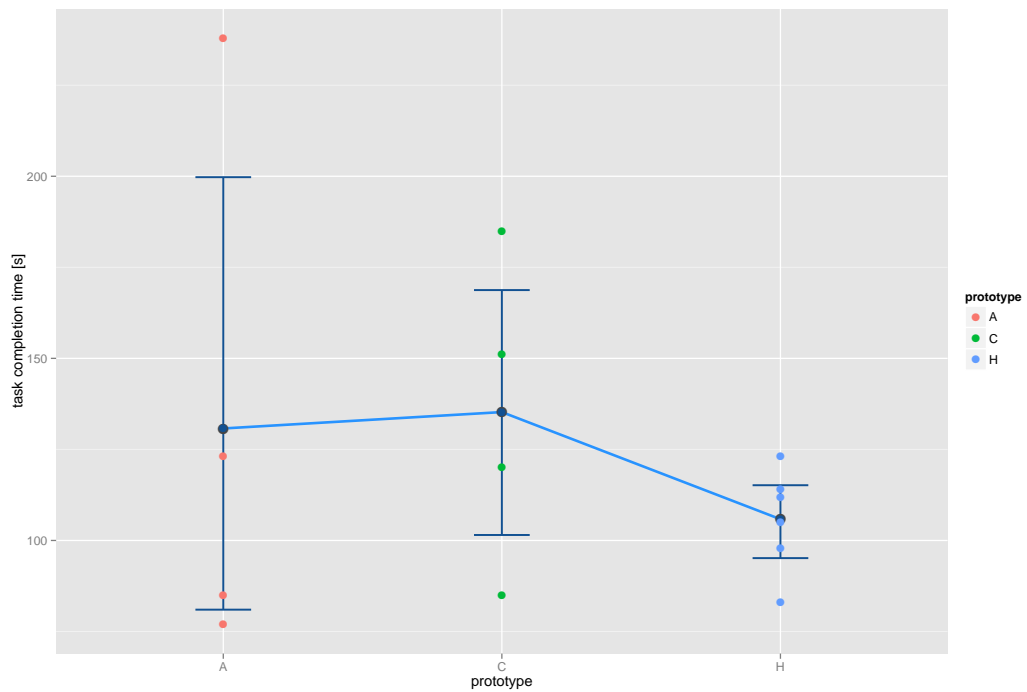


Figure C.12: task completion graph task 2.2.1

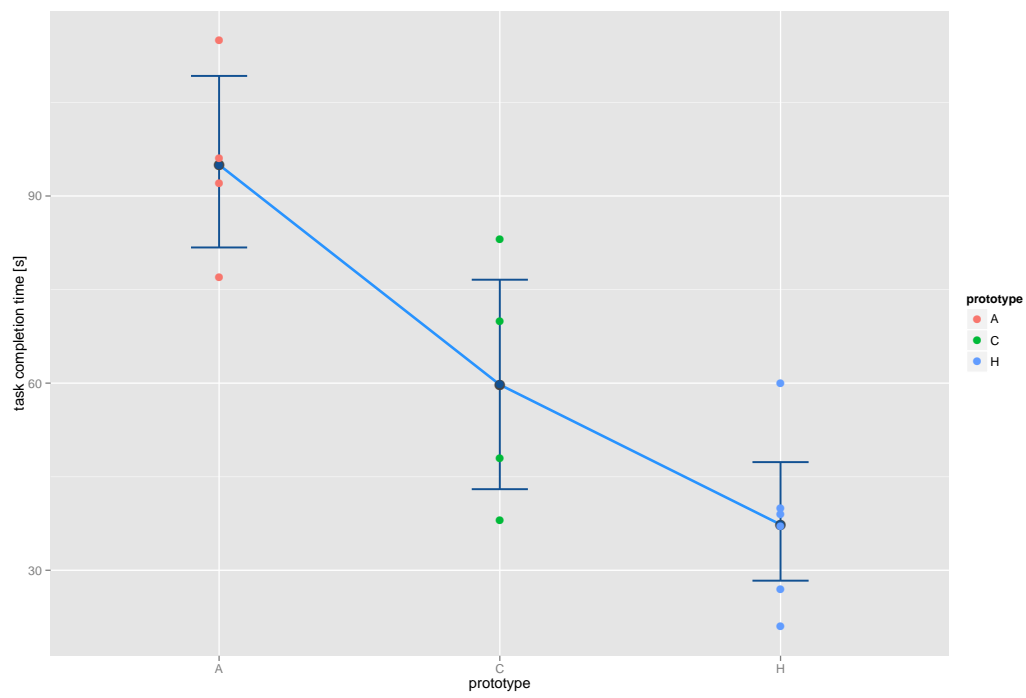


Figure C.13: task completion graph task 2.2.2

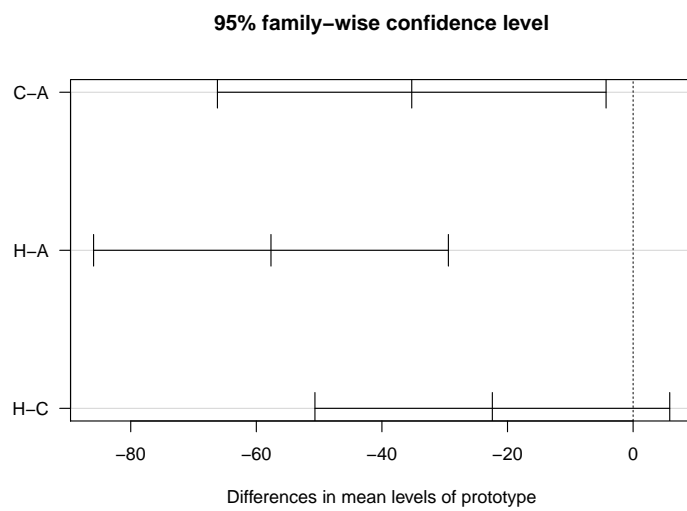


Figure C.14: task 2.2.2 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory Slicing result is significantly lower than the (A)zurite result

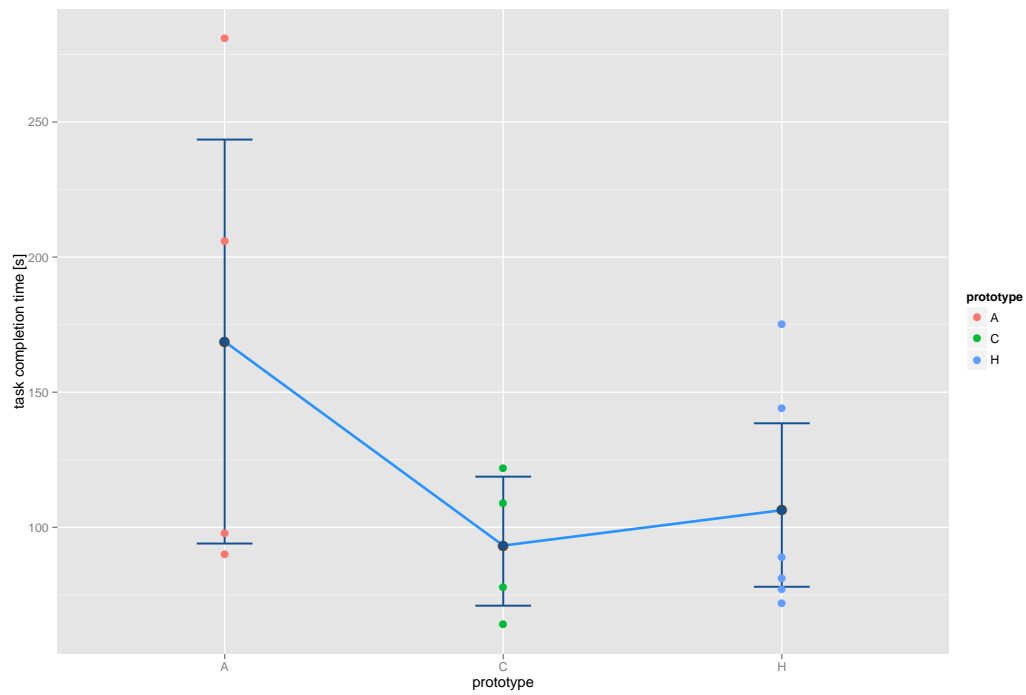


Figure C.15: task completion graph task 2.3.1

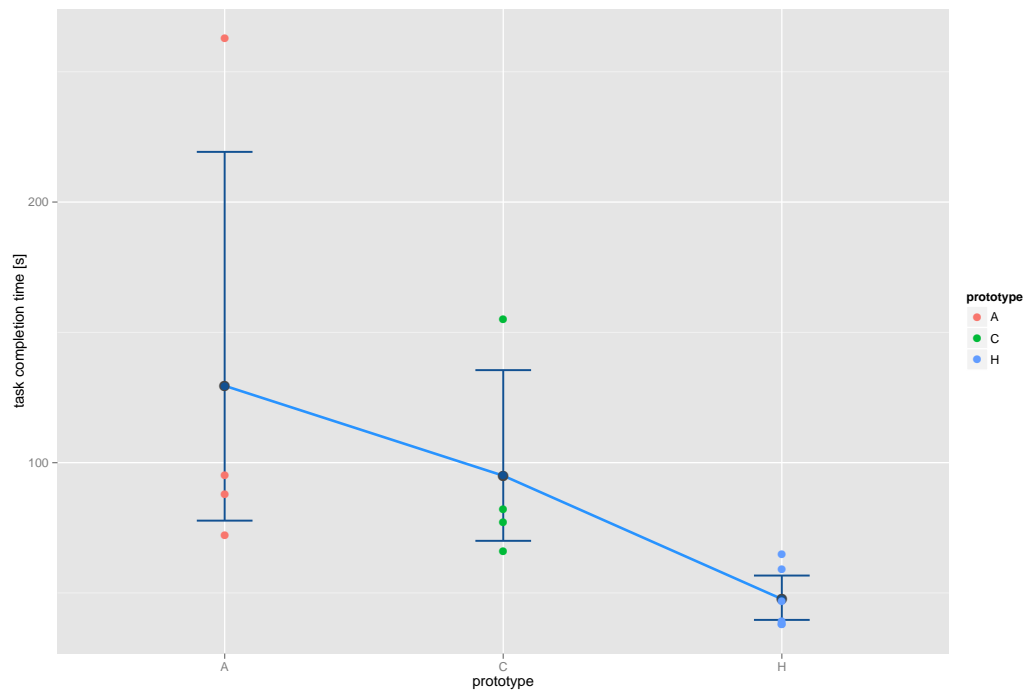


Figure C.16: task completion graph task 2.3.2

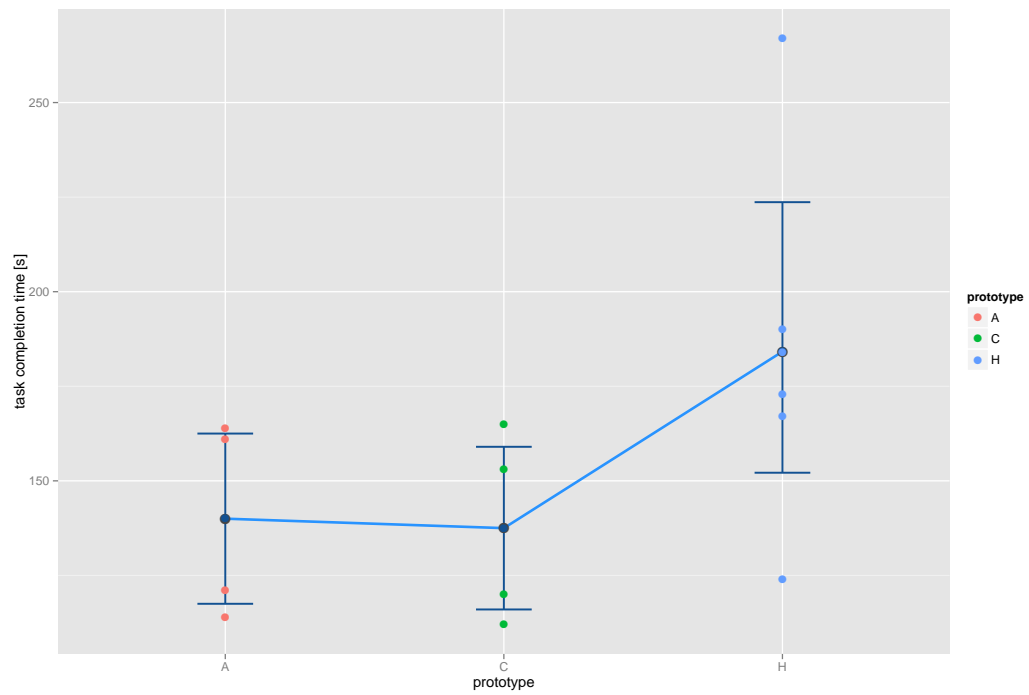


Figure C.17: task completion graph task 2.3.3

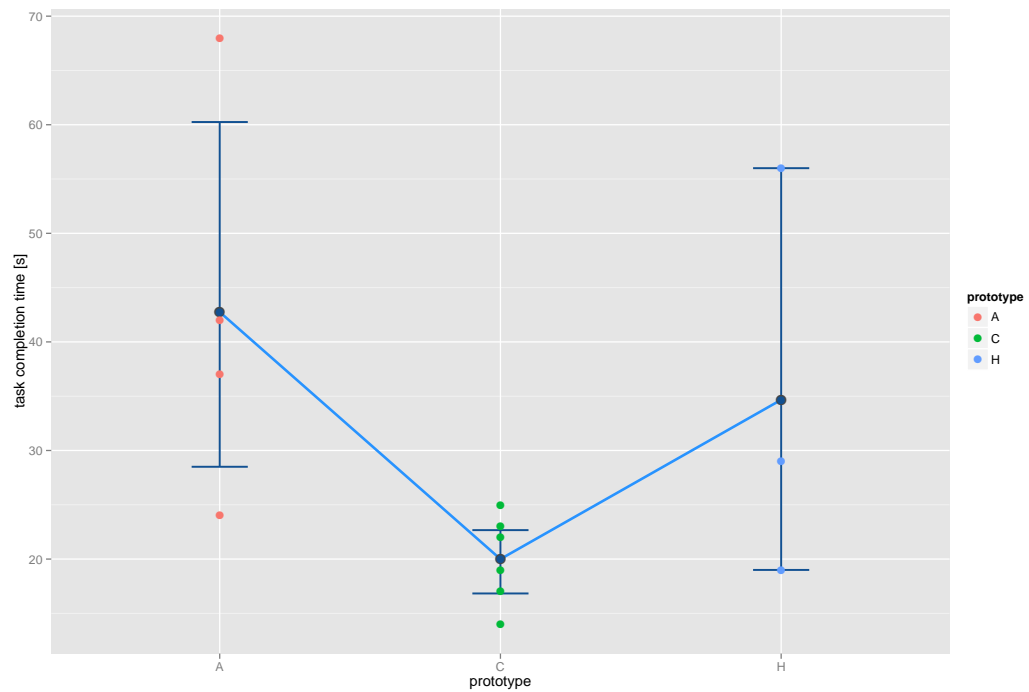


Figure C.18: task completion graph task 3.1.1

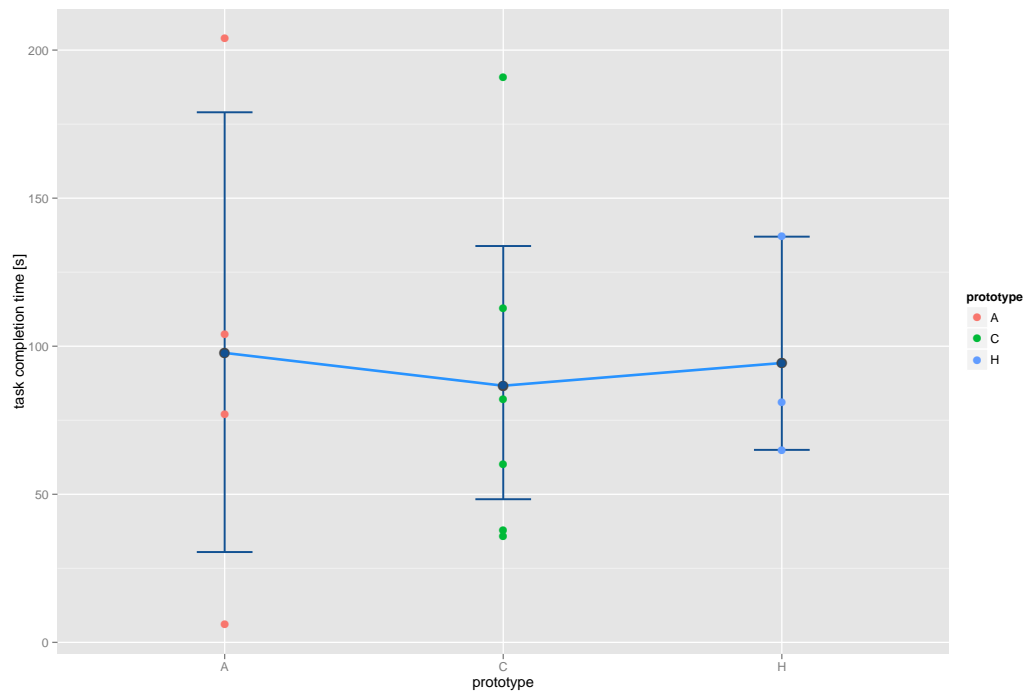


Figure C.19: task completion graph task 3.1.2

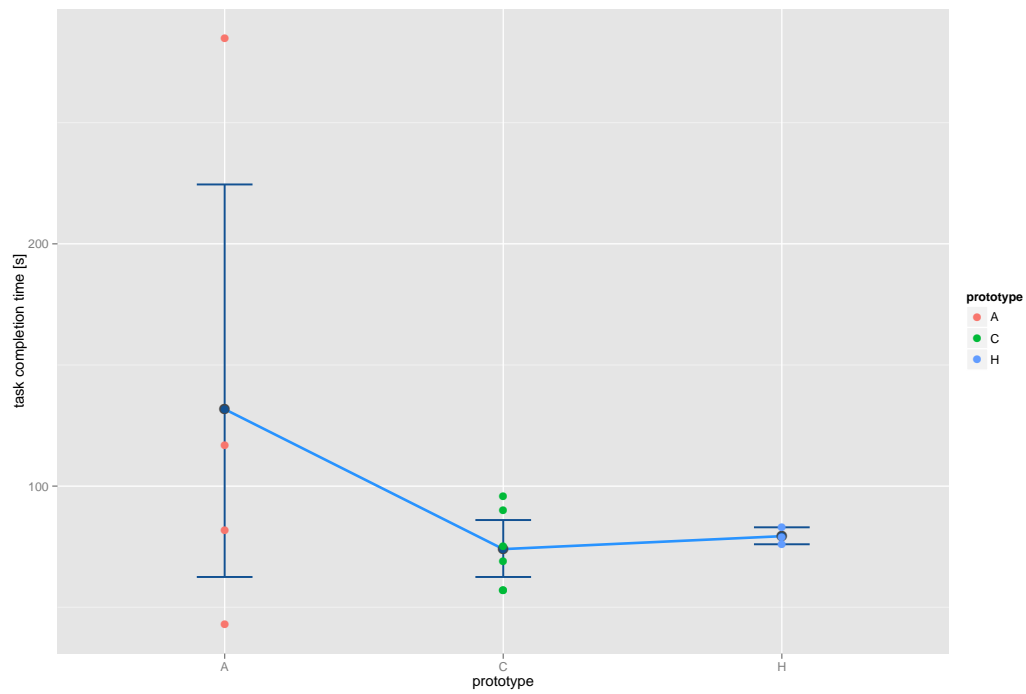


Figure C.20: task completion graph task 3.2.1

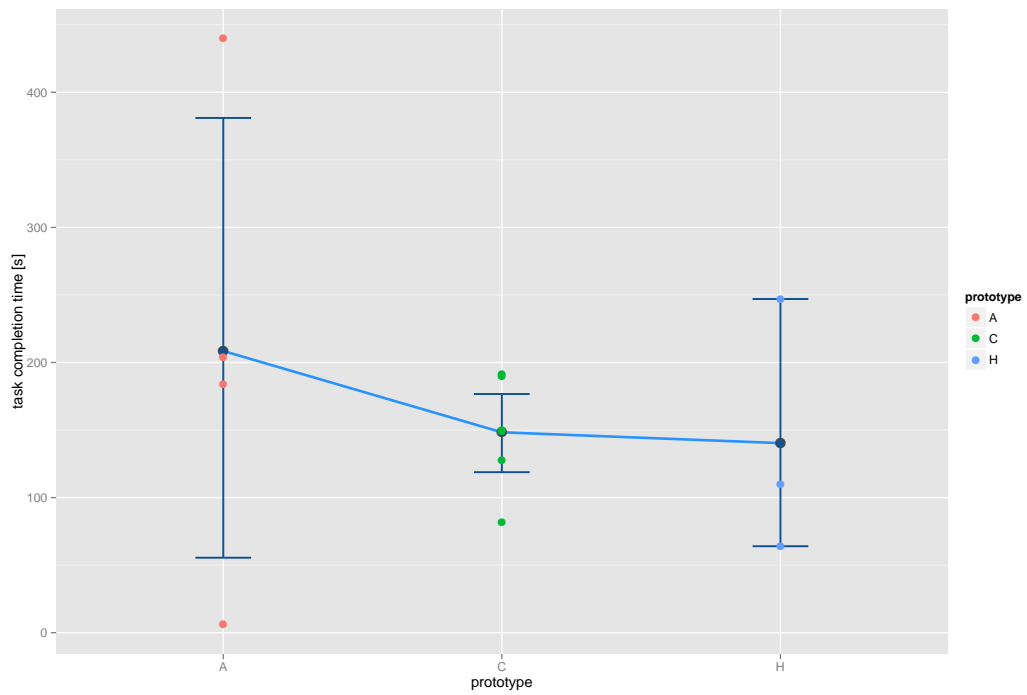


Figure C.21: task completion graph task 3.2.2

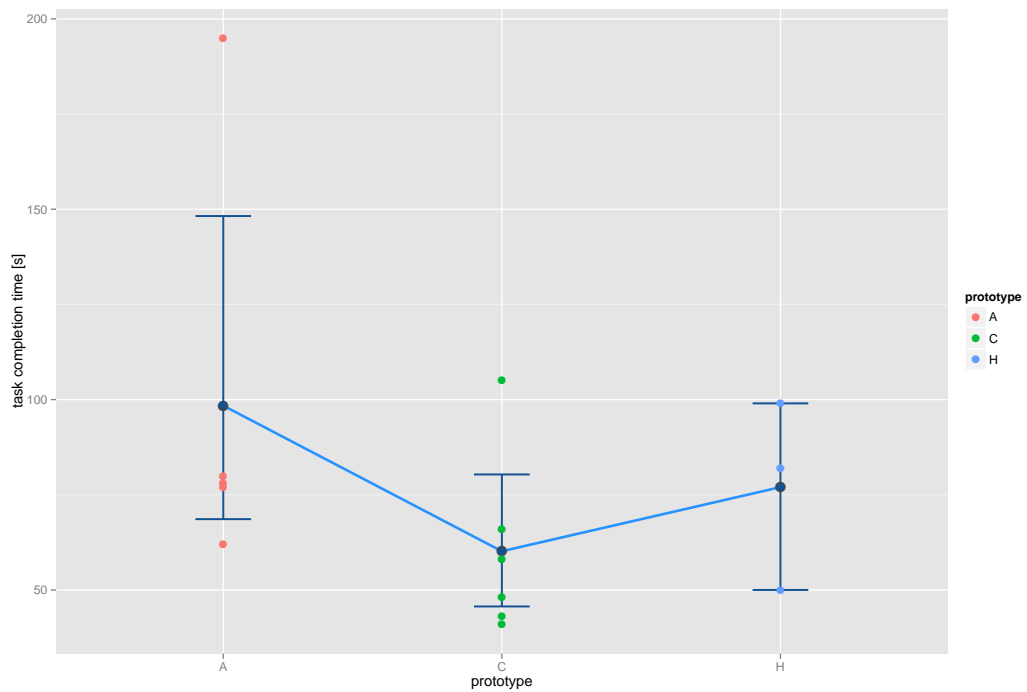


Figure C.22: task completion graph task 3.3.1

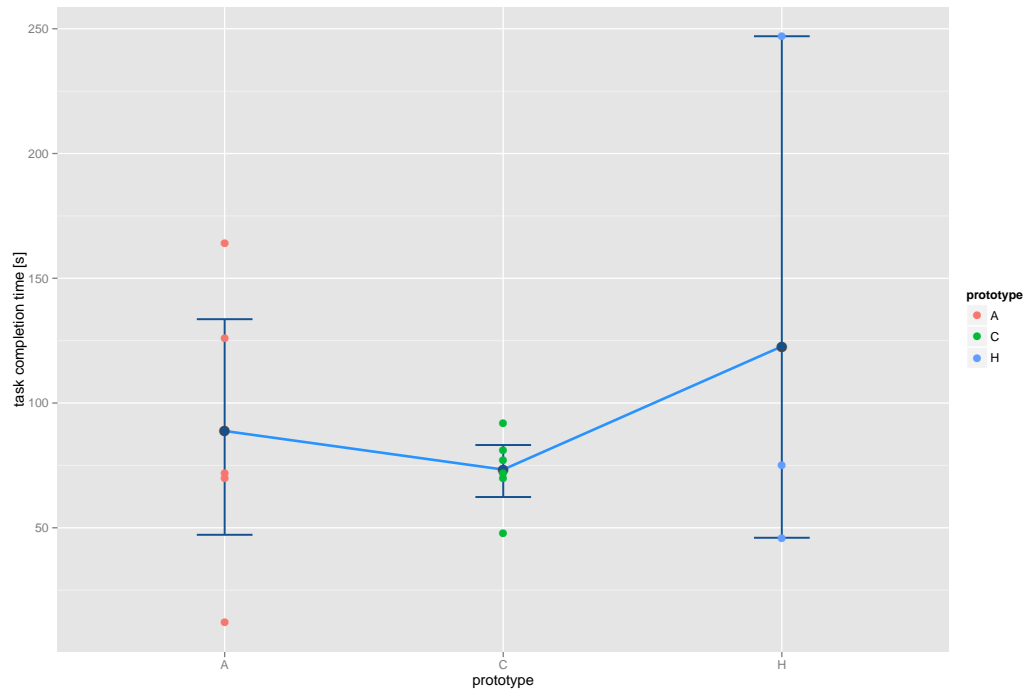


Figure C.23: task completion graph task 3.3.2

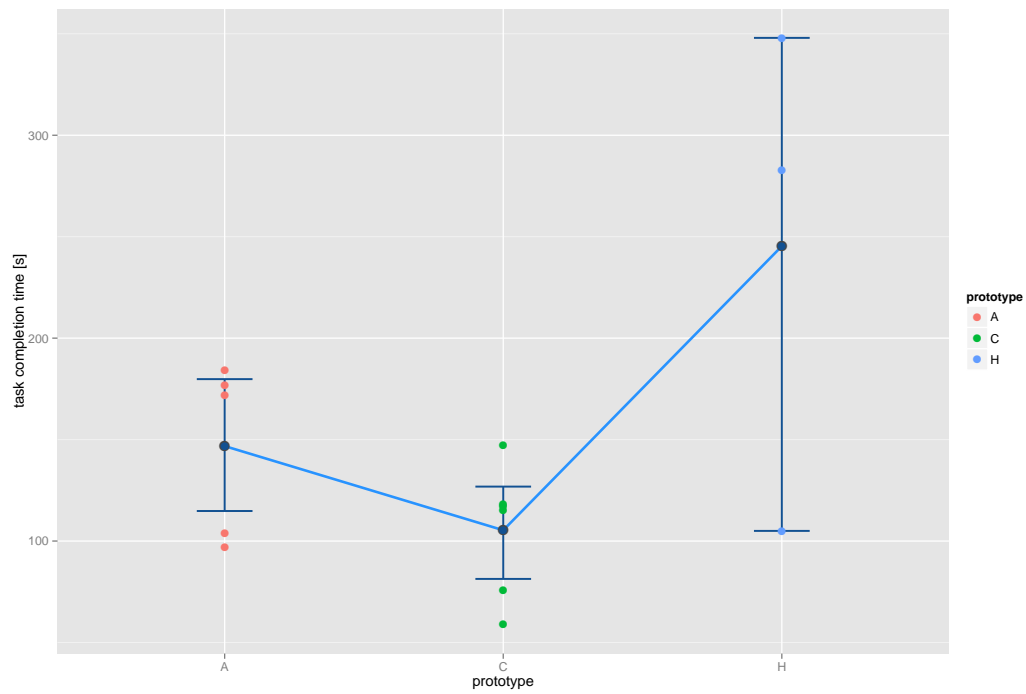


Figure C.24: task completion graph task 3.3.3

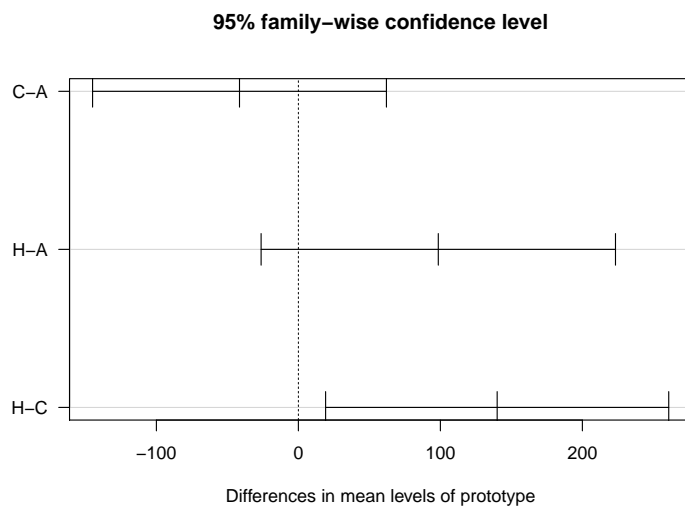


Figure C.25: task 3.3.3 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory Slicing result

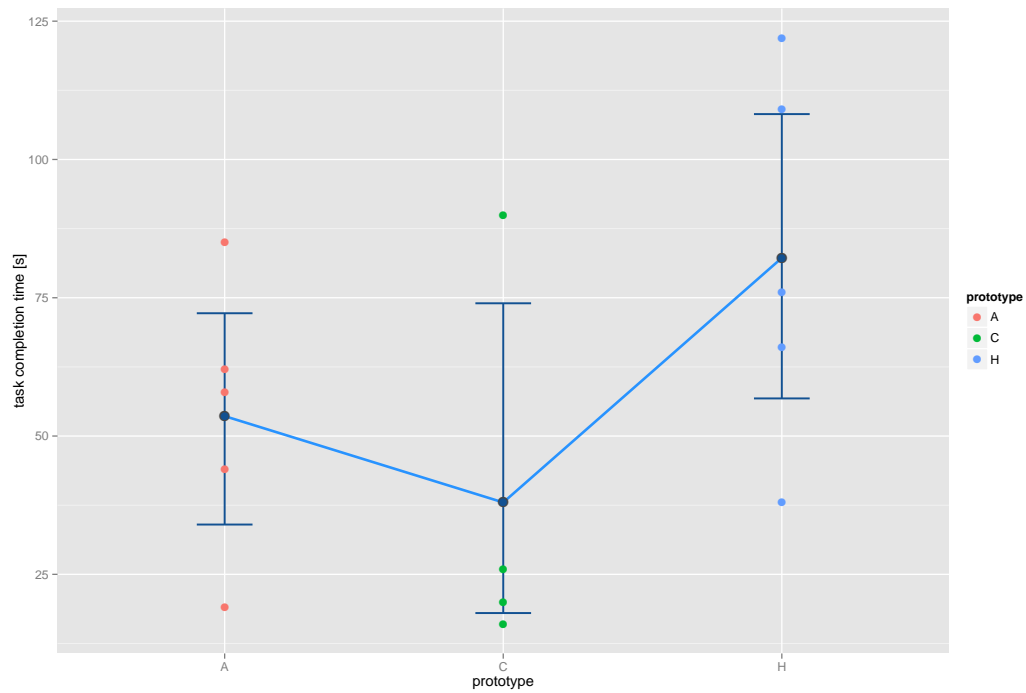


Figure D.1: task navigation graph task 1.1.1

Appendix D

Navigation Time Graphs

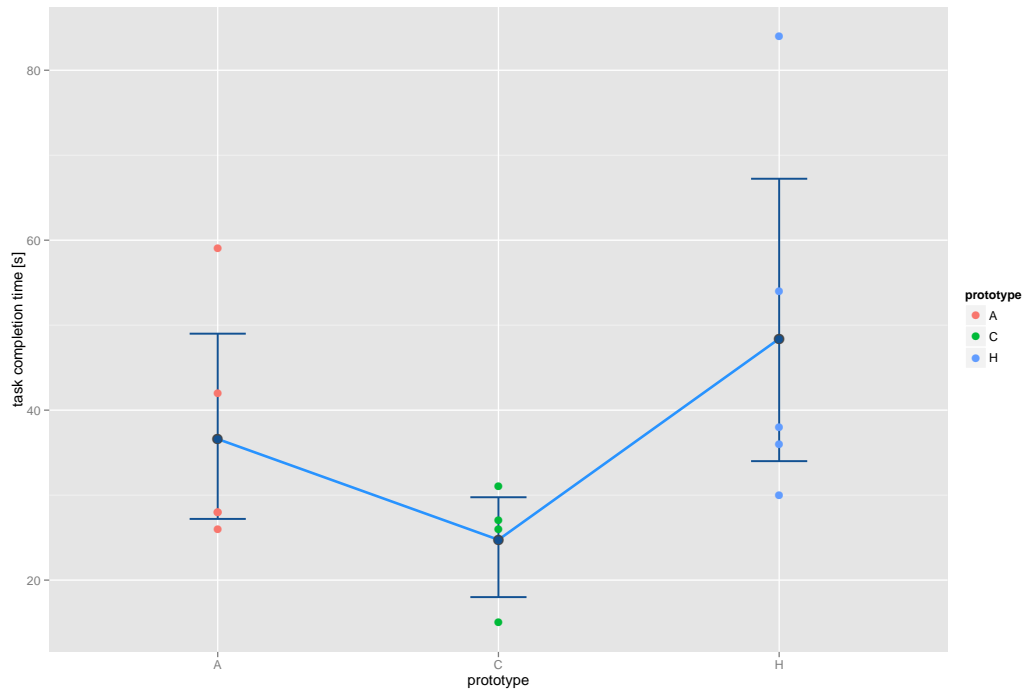


Figure D.2: task navigation graph task 1.2.1

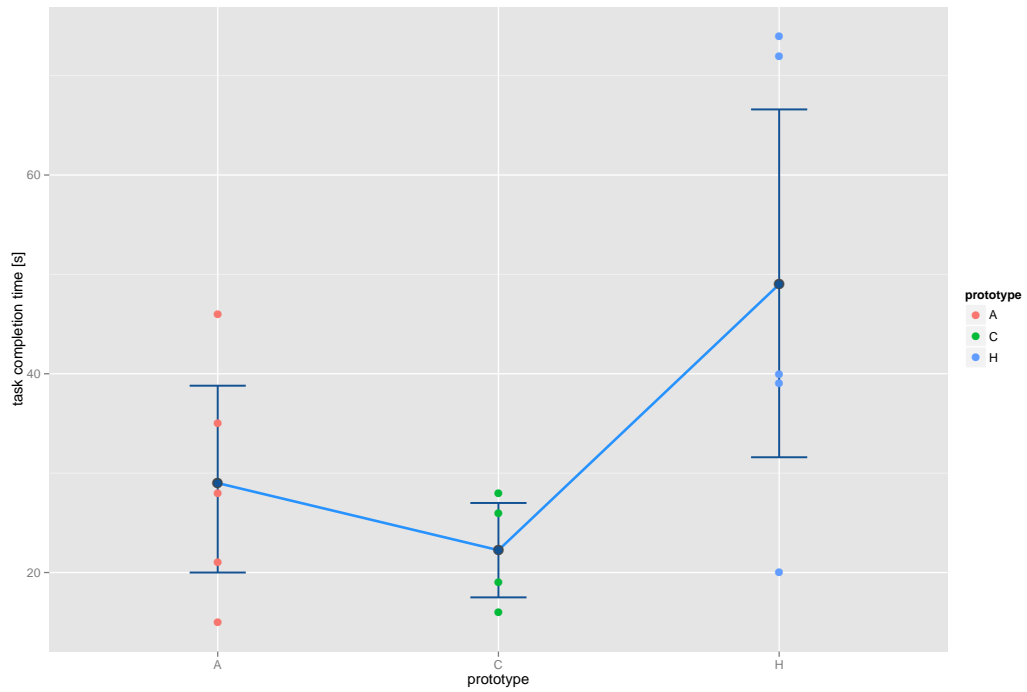


Figure D.3: task navigation graph task 1.3.1

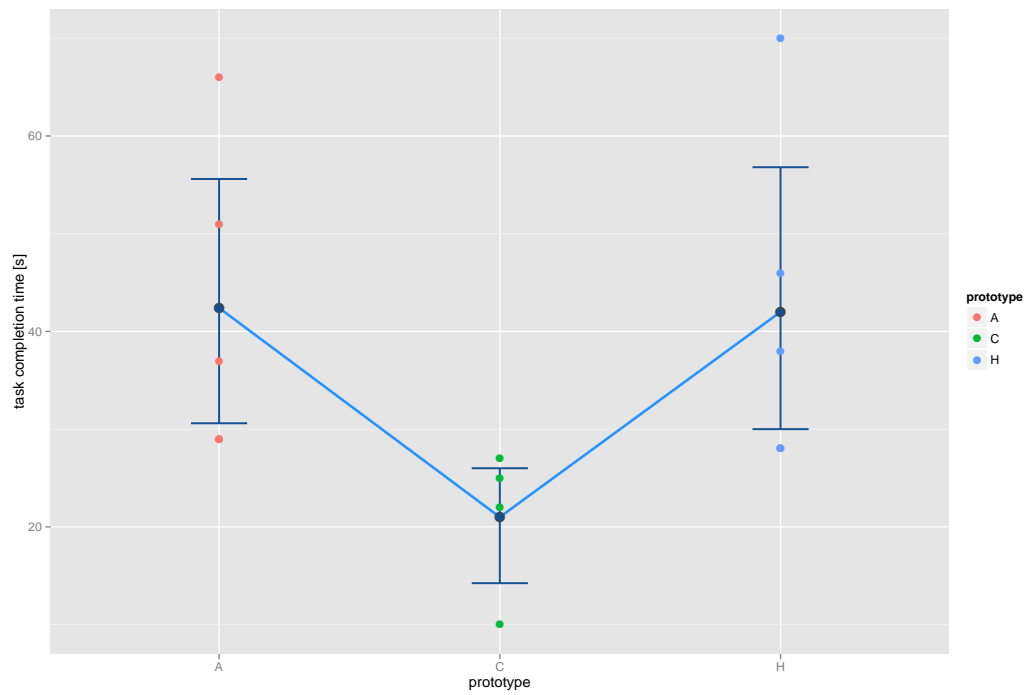


Figure D.4: task navigation graph task 1.3.3

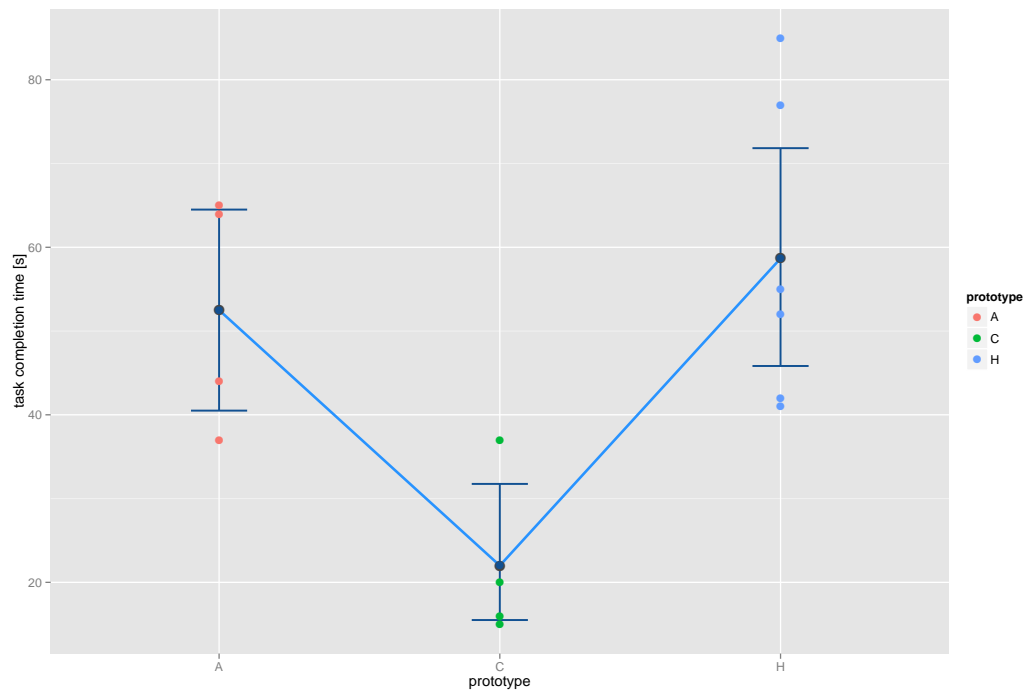


Figure D.5: task navigation graph task 2.1.1

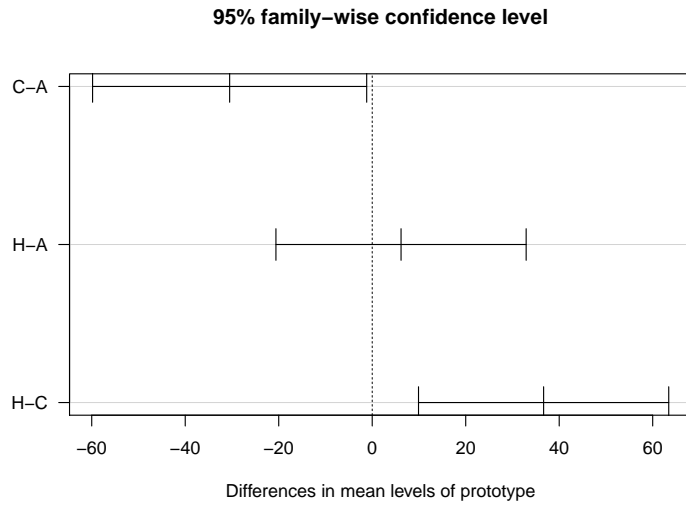


Figure D.6: task 2.1.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory Slicing result

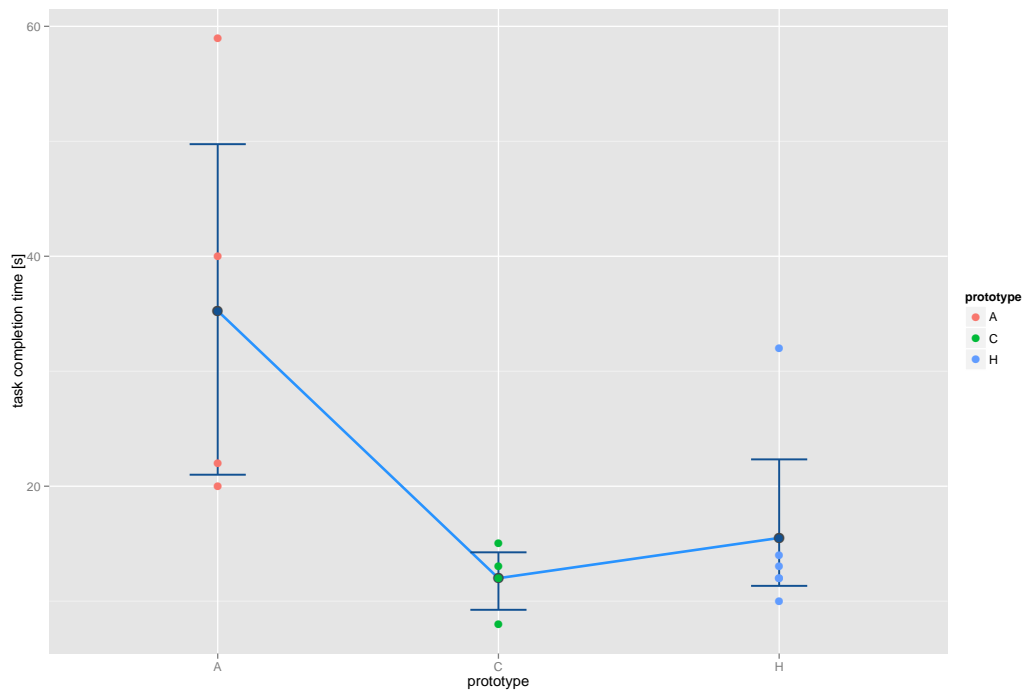


Figure D.7: task navigation graph task 2.2.1

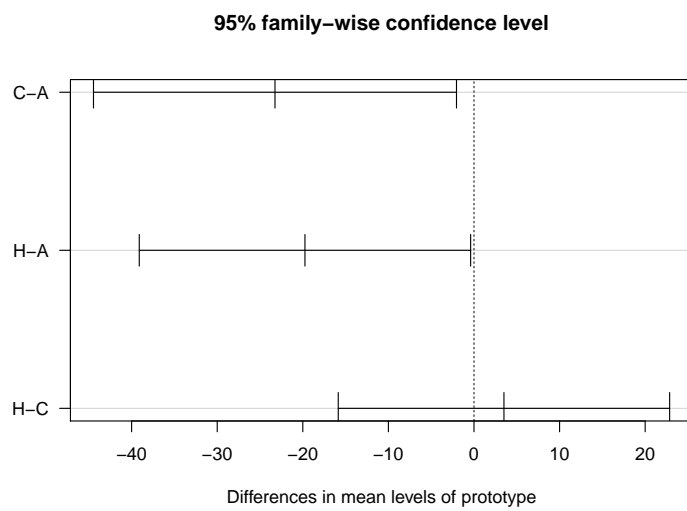


Figure D.8: task 2.2.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the (A)zurite result and the Chronos (H)istory slicing result is significantly lower than the (A)zurite result

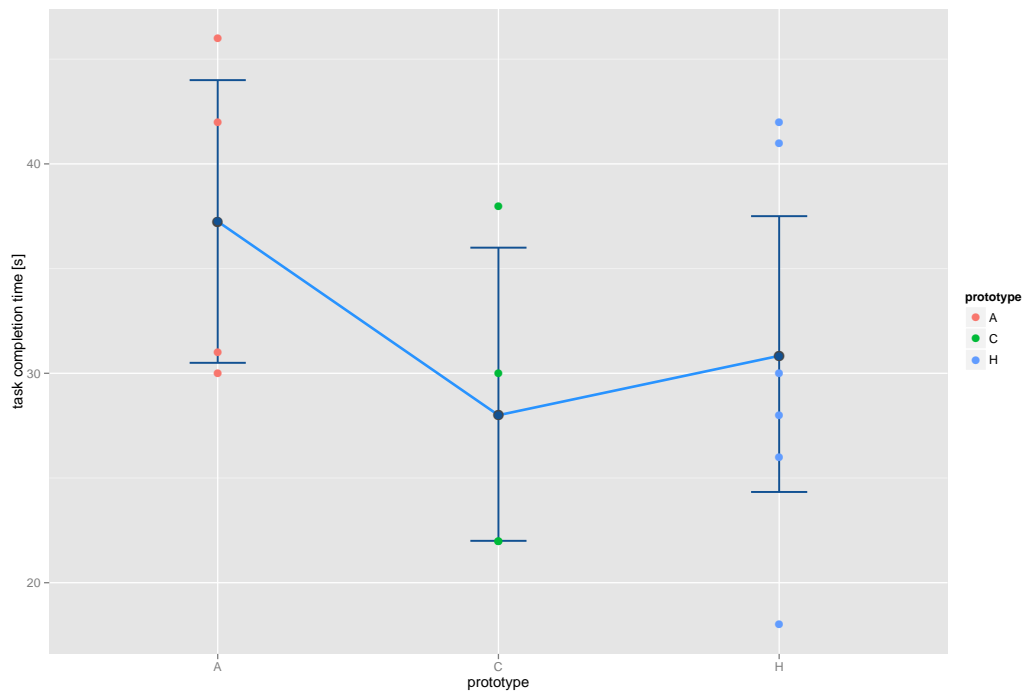


Figure D.9: task navigation graph task 2.3.1

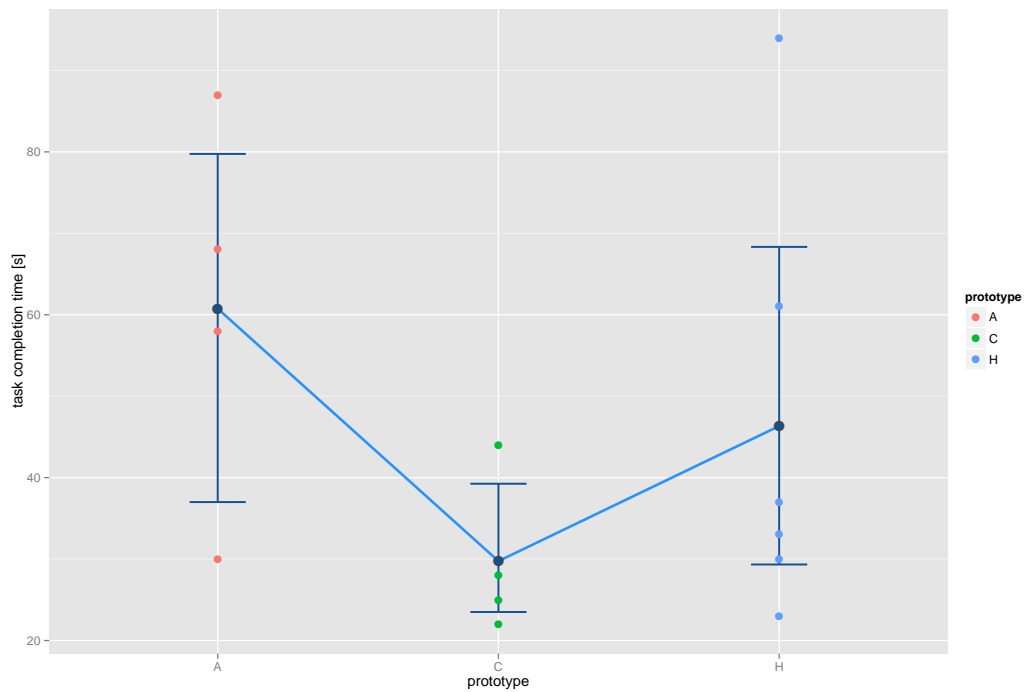


Figure D.10: task navigation graph task 2.3.3

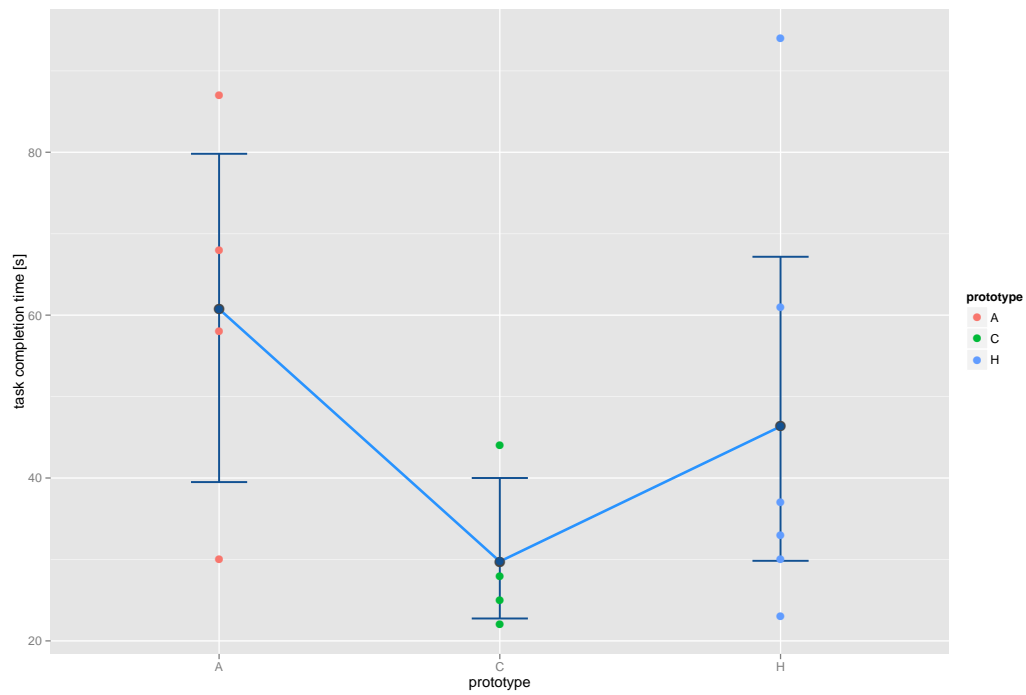


Figure D.11: task navigation graph task 3.1.1

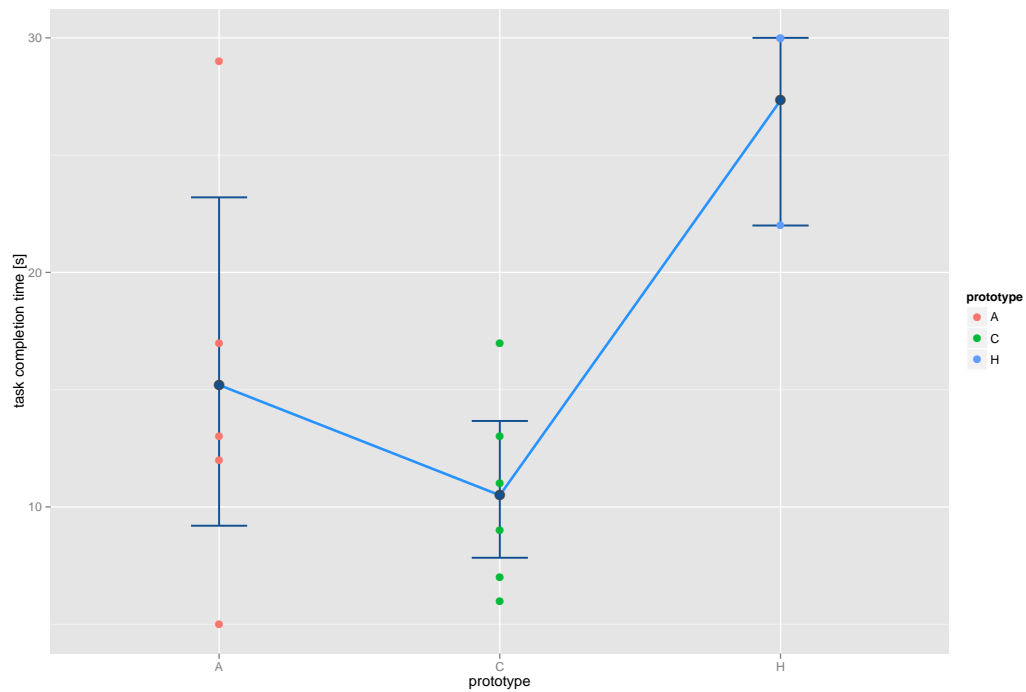


Figure D.12: task navigation graph task 3.2.1

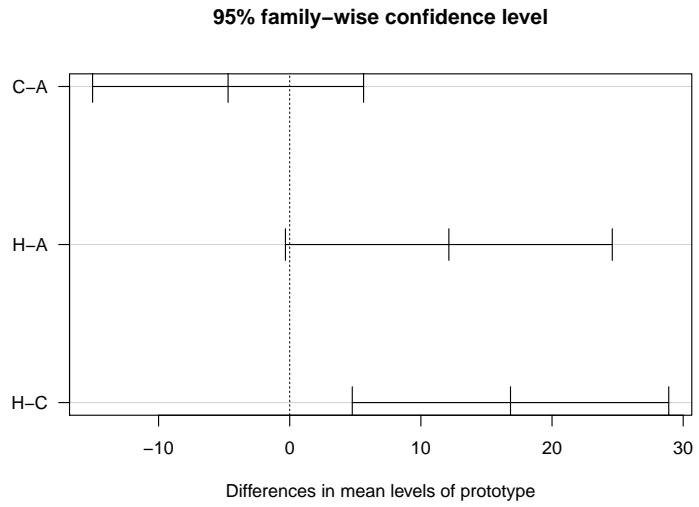


Figure D.13: task 3.2.1 shows the differences in mean of Tukey HSD post hoc comparison. The (C)odeShape result is significantly lower than the Chronos (H)istory slicing result

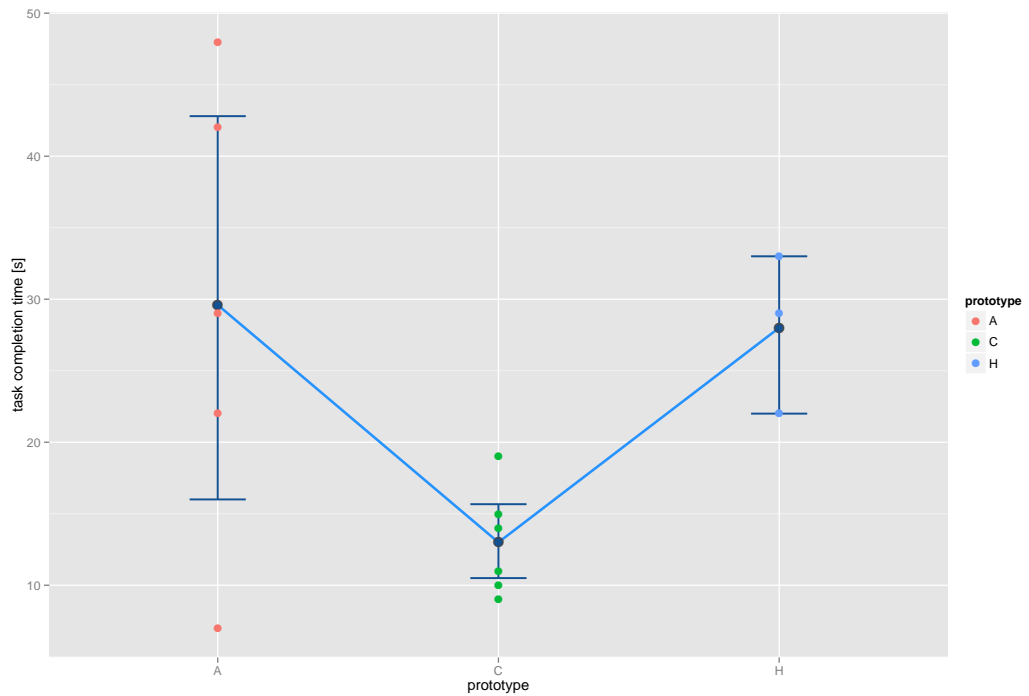


Figure D.14: task navigation graph task 3.3.1

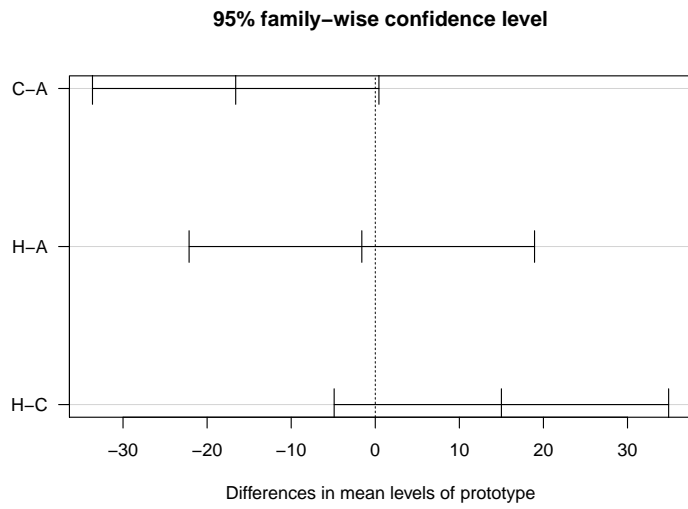


Figure D.15: task 3.3.1 shows the differences in mean of Tukey HSD post hoc comparison. The result do not differ significantly

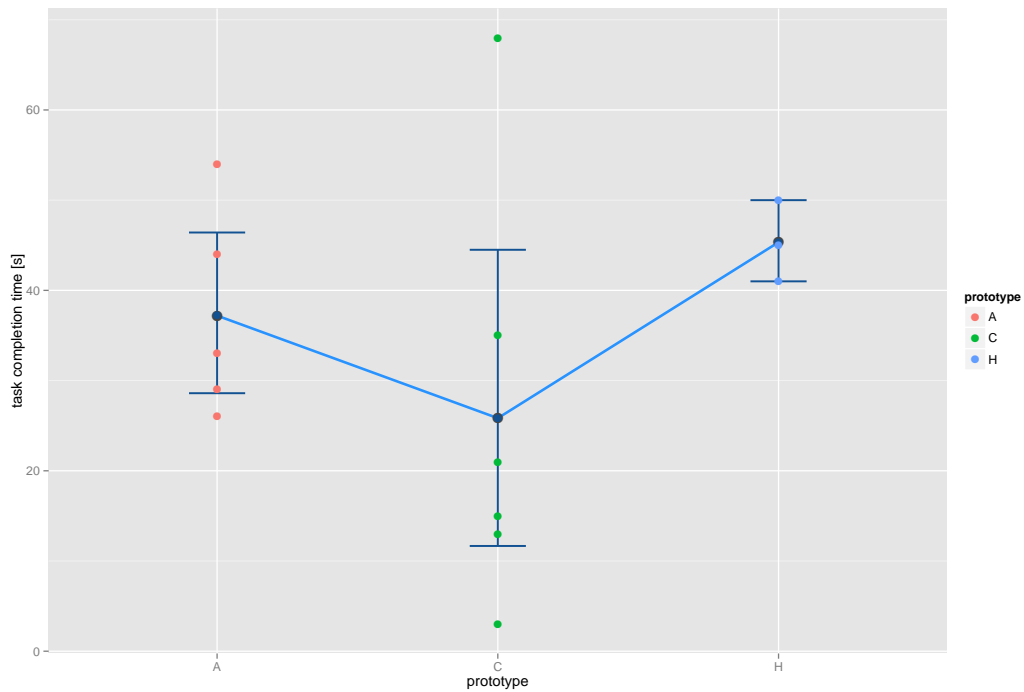


Figure D.16: task navigation graph task 3.3.3

Appendix E

Source Code of decorateCommit

```

- (void) decorateCommit: (PBGitCommit *) commit
{
    int i = 0, newPos = -1;
    LaneCollection *currentLanes = new LaneCollection;
    LaneCollection *previousLanes = self.pl;
    NSArray *parents = [commit parents];
    int nParents = [parents count];

    int maxLines = (previousLanes->size() + nParents + 2) * 2;
    struct PBGitGraphLine *lines = (struct PBGitGraphLine *)malloc(sizeof(struct PBGitGraphLine) * maxLines);
    int currentLine = 0;

    PBGitLane *currentLane = NULL;
    BOOL didFirst = NO;
    const git_oid *commit_oid = [[commit sha] git_oid];

    // First, iterate over earlier columns and pass through any that don't want this commit
    if (self.previous != nil) {
        // We can't count until numColumns here, as it's only used for the width of the cell.
        LaneCollection::iterator it = previousLanes->begin();
        for (; it != previousLanes->end(); ++it) {
            i++;
            if (!*it) // This is an empty lane, created when the lane previously had a parentless(root) commit
                continue;

            // This is our commit! We should do a "merge": move the line from
            // our upperMapping to their lowerMapping
            if ((*it)->isCommit(commit_oid)) {
                if (!didFirst) {
                    didFirst = YES;
                    currentLanes->push_back(*it);
                    currentLane = currentLanes->back();
                    newPos = currentLanes->size();
                    add_line(lines, &currentLine, 1, i, newPos, (*it)->index());
                    if (nParents)
                        add_line(lines, &currentLine, 0, newPos, newPos, (*it)->index());
                }
                else {
                    add_line(lines, &currentLine, 1, i, newPos, (*it)->index());
                    delete *it;
                }
            }
            else {
                // We are not this commit.
                currentLanes->push_back(*it);
                add_line(lines, &currentLine, 1, i, currentLanes->size(), (*it)->index());
                add_line(lines, &currentLine, 0, currentLanes->size(), currentLanes->size(), (*it)->index());
            }
            // For existing columns, we always just continue straight down
            // ^^ I don't know what that means anymore :(
        }
    }
    //Add your own parents

    // If we already did the first parent, don't do so again
    if (!didFirst && nParents) {
        const git_oid *parentOID = [(GTOID*)[parents objectAtIndex:0] git_oid];
        PBGitLane *newLane = new PBGitLane(_laneIndex++, parentOID);
        currentLanes->push_back(newLane);
        newPos = currentLanes->size();
        add_line(lines, &currentLine, 0, newPos, newPos, newLane->index());
    }

    // Add all other parents

    // If we add at least one parent, we can go back a single column.
    // This boolean will tell us if that happened
    BOOL addedParent = NO;

    int parentIndex = 0;
    for (parentIndex = 1; parentIndex < nParents; ++parentIndex) {
        const git_oid *parentOID = [(GTOID*)[parents objectAtIndex:parentIndex] git_oid];
        int i = 0;
        BOOL was_displayed = NO;
        LaneCollection::iterator it = currentLanes->begin();
        for (; it != currentLanes->end(); ++it) {
            i++;
            if ((*it)->isCommit(parentOID)) {
                add_line(lines, &currentLine, 0, i, newPos, (*it)->index());
                was_displayed = YES;
                break;
            }
        }
        if (was_displayed)
            continue;

        // Really add this parent
        addedParent = YES;
        PBGitLane *newLane = new PBGitLane(_laneIndex++, parentOID);
        currentLanes->push_back(newLane);
        add_line(lines, &currentLine, 0, currentLanes->size(), newPos, newLane->index());
    }

    if (commit.lineInfo) {

```

```

        self.previous = commit.lineInfo;
        self.previous.position = newPos;
        self.previous.lines = lines;
    }
    else
        self.previous = [[PBGraphCellInfo alloc] initWithPosition:newPos andLines:lines];

    if (currentLine > maxLines)
        NSLog(@"Number of lines: %i vs allocated: %i", currentLine, maxLines);

    self.previous.nLines = currentLine;
    self.previous.sign = commit.sign;

    // If a parent was added, we have room to not indent.
    if (addedParent)
        self.previous.numColumns = currentLanes->size() - 1;
    else
        self.previous.numColumns = currentLanes->size();

    // Update the current lane to point to the new parent
    if (currentLane) {
        if (nParents > 0)
            currentLane->setSha( [(GT0ID*)[parents objectAtIndex:0] git_oid]);
        else {
            // The current lane's commit does not have any parents
            // AKA, this is a first commit
            // Empty the entry and free the lane.
            // We empty the lane in the case of a subtree merge, where
            // multiple first commits can be present. By emptying the lane,
            // we allow room to create a nice merge line.
            std::replace(currentLanes->begin(), currentLanes->end(), currentLane, (PBGitLane *)0);
            delete currentLane;
        }
    }

    delete previousLanes;

    self.pl = currentLanes;
    commit.lineInfo = self.previous;
}
}

```


Appendix F

Initial Study Questionnaire

Fragebogen

Mitarbeiterinteraktion

Kommt es vor, dass Mitarbeiter bei anderen Mitarbeitern ins Büro kommen, um Fragen zu stellen?

Werden dabei Skizzen angefertigt, oder andere Hilfsmittel benutzt?

Wenn Skizzen angefertigt werden:

mit was für einem Medium (Whiteboard, Papier, Notizblock ?)

wird UML verwendet oder andere festgelegte Strukturelemente oder nur frei gezeichnete Formen?

Was für 3rd party tools werden neben XCode, Android Studio usw. eingesetzt (Photoshop , Skizzentool o.ä)?

Wie werden neue Mitarbeiter eingearbeitet, sprich wie werden sie an ein Projekt herangeführt und welche Hilfsmittel werden verwendet?

Was passiert:

wenn ein Mitarbeiter einen Bug o.ä. eines anderen Mitarbeiters findet?

wenn es zu einem Merge-Conflict kommt?

Meetings

Wie oft gibt es Meetings?

Wird bei Meetings ein Whiteboard verwendet oder eine Präsentation (was für ein Inhalt, Skizzen, Bilder, Wireframes, Statistiken)?

Mit wem werden Meetings abgehalten(alle Mitarbeiter, Team, Kunde)?

Entwicklungsunterstützung

Liefert Kunde Skizzen, Spezifikation o.ä.?

Werden nur Photoshop-Wireframes als Vorlage benutzt?

Wird eine Dokumentation, Wiki o.ä. angelegt?

Wie werden Datenbankschemata bei Android dargestellt, sprich Relationen, Zusammenhänge usw. von sqlite-Datenbanken?

Mit was für Ressourcen wird gearbeitet um Dinge nachzugucken, sprich Bücher, Websites (stackoverflow, API's) oder evtl. auch Videos (WWDC, Google IO)

Wird primär, sekundär Monitor um Dokumentationen, Skizzen o.ä. anzuzeigen, verwendet?

Werden Handskizzen gemacht?

Arbeitet jeder mit git?

Wenn ja auf Konsolenbasis, GUI-Client oder über die IDE?

Projekt

Wie viele Leute arbeiten an einem Projekt?

**Wie lang wird meist an einem Projekt gearbeitet?
mehrere Monate**

Werden mehrere Projekte parallel entwickelt, sprich was passiert beim context switching, wenn versucht wird das mental model zu „reaktivieren“.

Wird/Muss dokumentiert werden? Wenn ja, manuell oder mit z.b. doxygen?

Was bekommen die Mitarbeiter für die Umsetzung eines Projekts an Material?

Zukunftsfragen

Könntet ihr euch vorstellen Skizzen mit dem Quelltext zu verknüpfen und so zu archivieren?

Was für ein Medium würdet ihr am liebsten zur Erstellung verwenden (Whiteboard, Papier, Wacom Zeichenboard)?

Haltet ihr es generell für sinnvoll Skizzen mit Quelltext zu verbinden (Einwände, andere Idee?)

Wenn ihr wüsstest, dass die Fehlerrate sinkt und es weniger duplizierten Code gäbe, wäre dies ein Argument für euch, um Skizzen mit Quelltext zu verbinden?

Was müsste so ein Tool können?

Würdet ihr Photos von Skizzen machen, wenn diese sich dann mit dem Quelltext verbinden lassen?

Was wäre eure Idee/Wunsch um schneller Code zu verstehen bzw. schneller das mental model aufzurufen?

Appendix G

Sketch Inspection Time Graphs

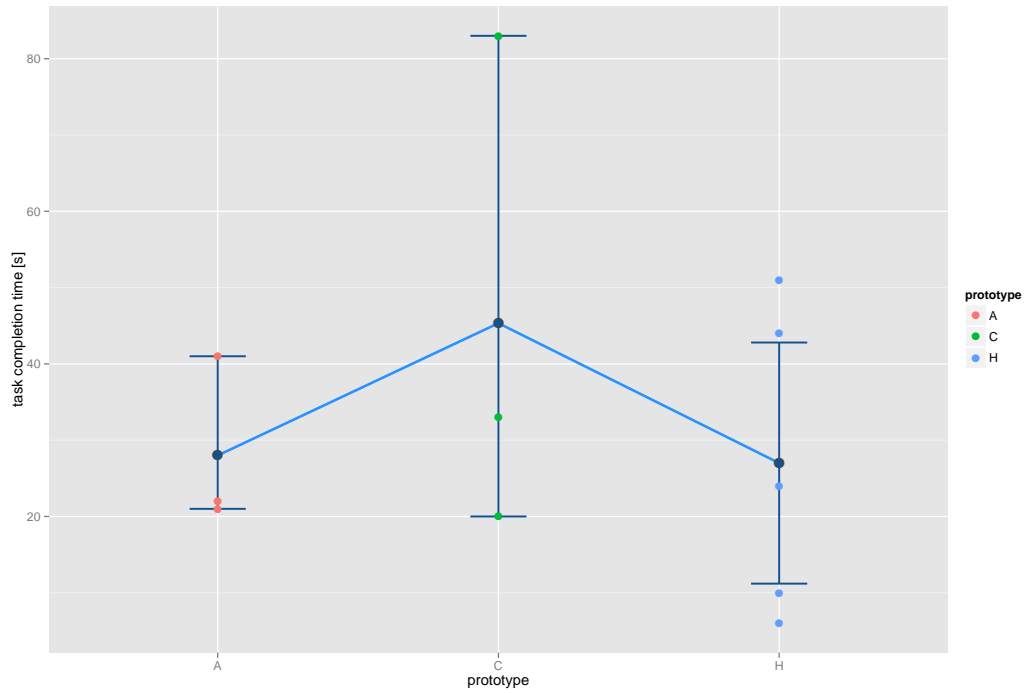


Figure G.1: Sketch Inspection Time graph - task 1.1.3

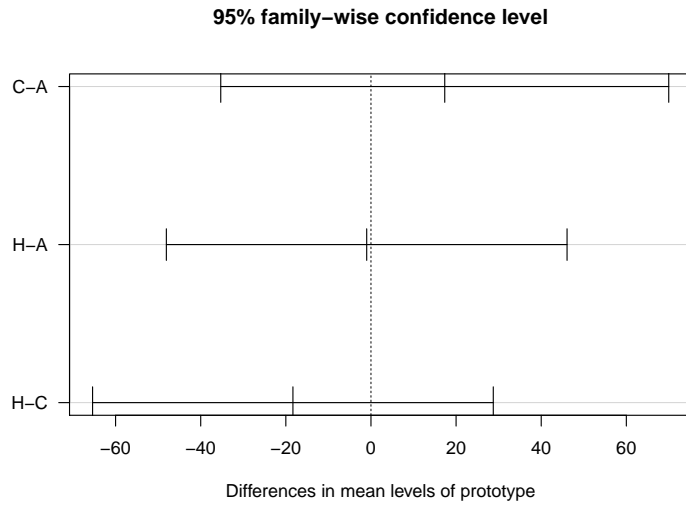


Figure G.2: The graph of task 1.1.3 shows the differences in mean of Tukey HSD post hoc comparison.

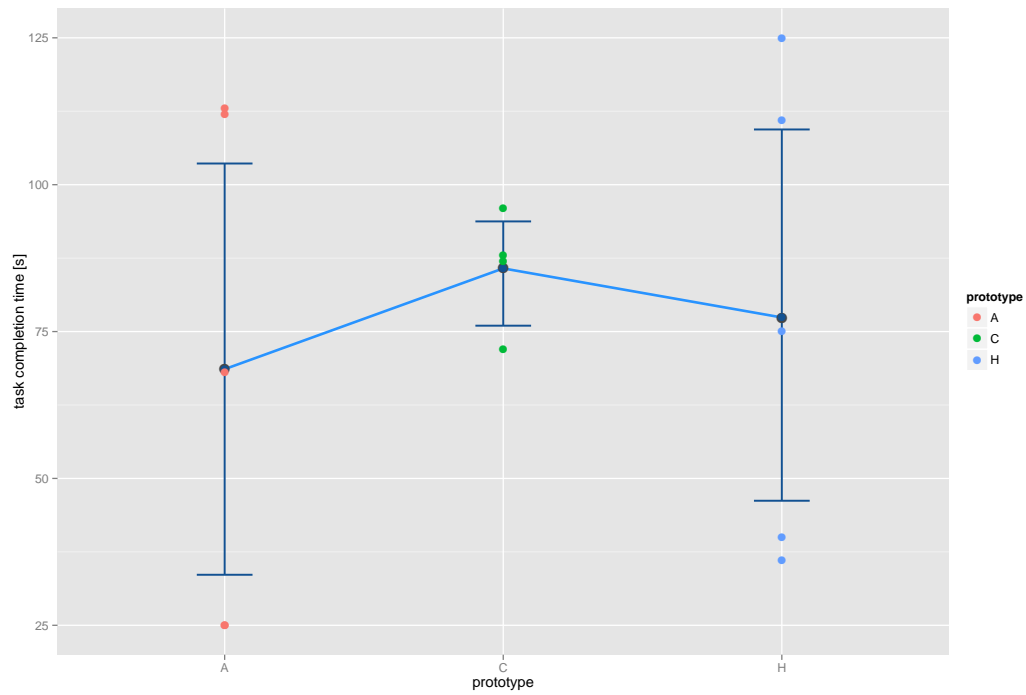


Figure G.3: Sketch Inspection Time graph - task 1.3.3

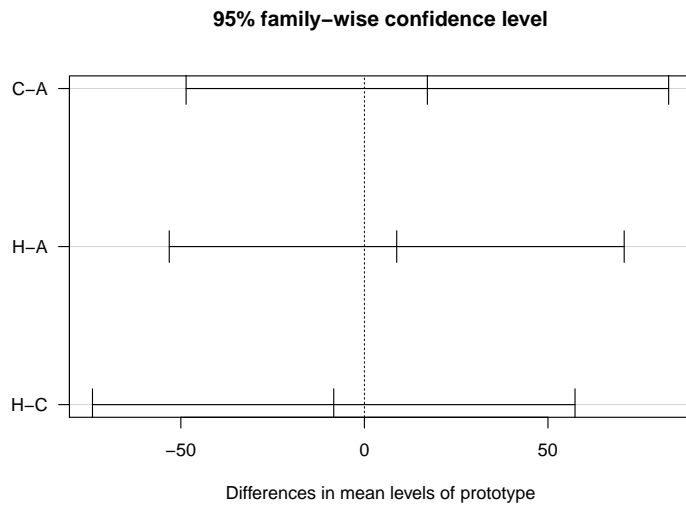


Figure G.4: The graph of task 1.3.3 shows the differences in mean of Tukey HSD post hoc comparison.

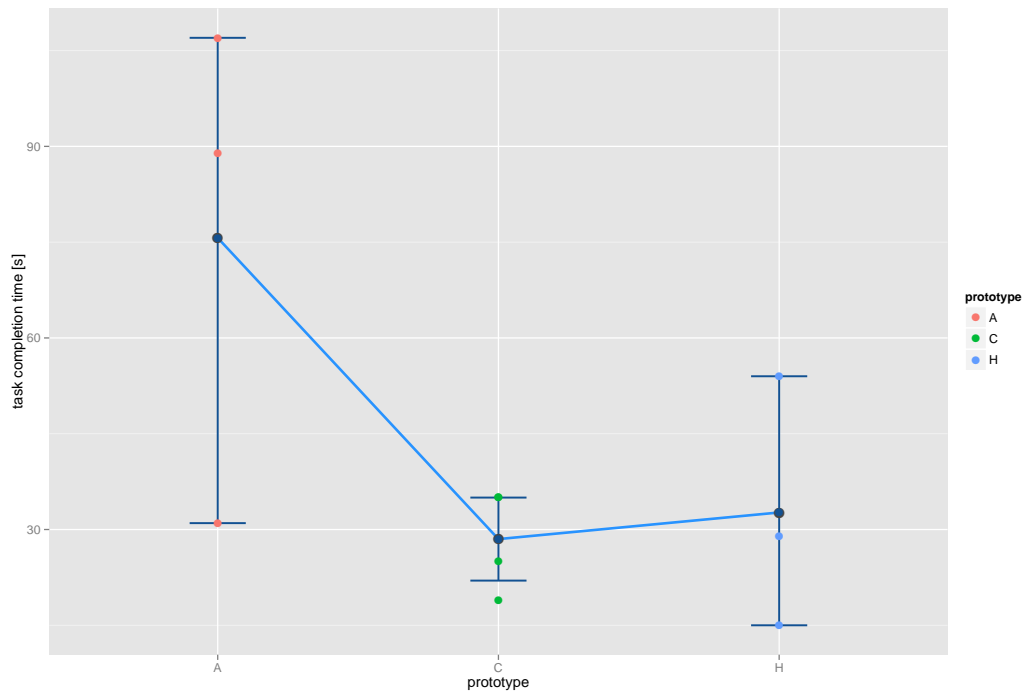


Figure G.5: Sketch Inspection Time graph - task 2.1.2

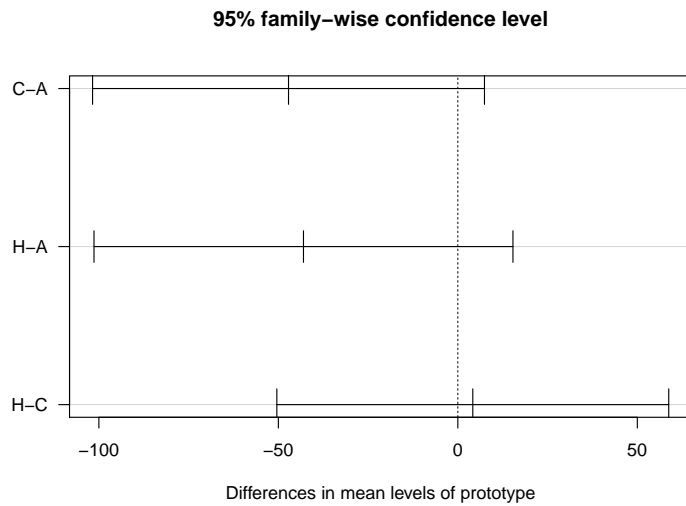


Figure G.6: The graph of task 2.1.2 shows the differences in mean of Tukey HSD post hoc comparison.

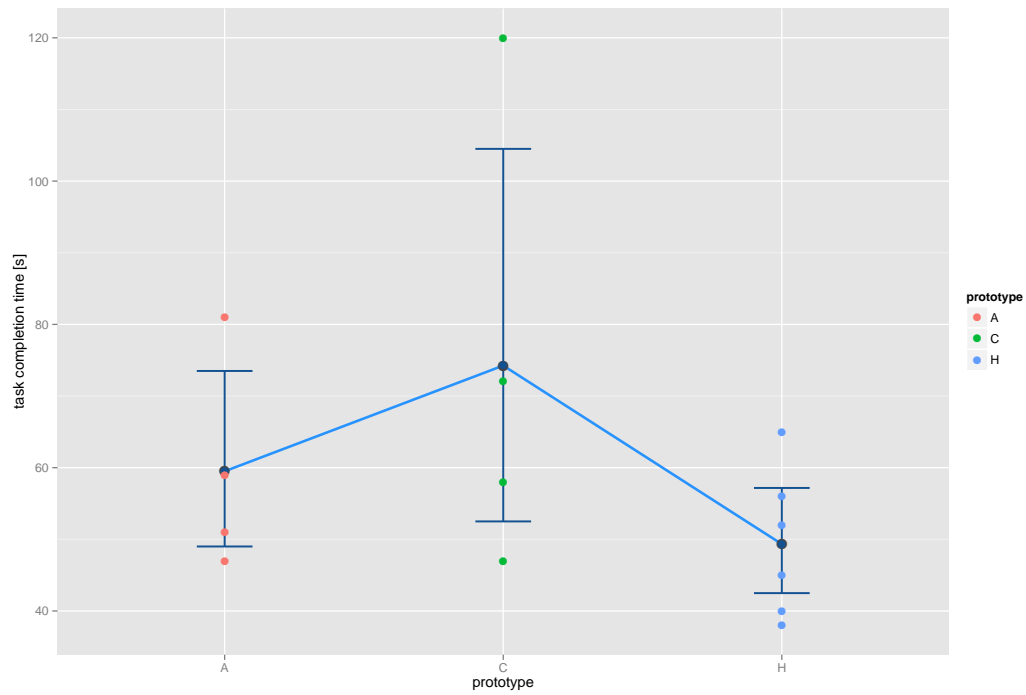


Figure G.7: Sketch Inspection Time graph - task 2.3.3

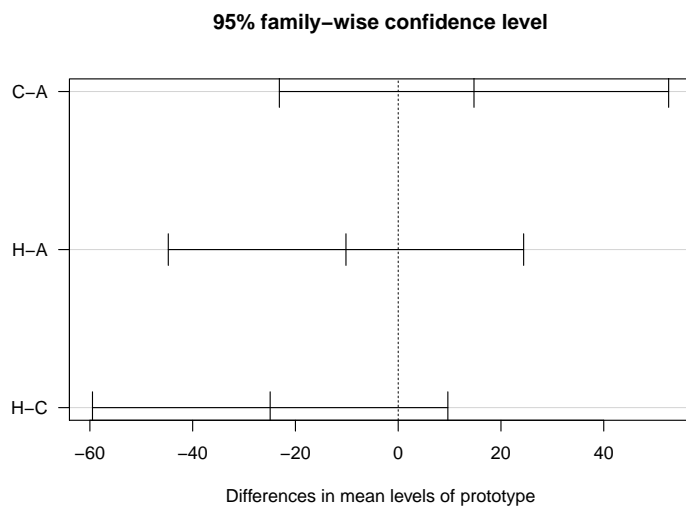


Figure G.8: The graph of task 2.3.3 shows the differences in mean of Tukey HSD post hoc comparison.

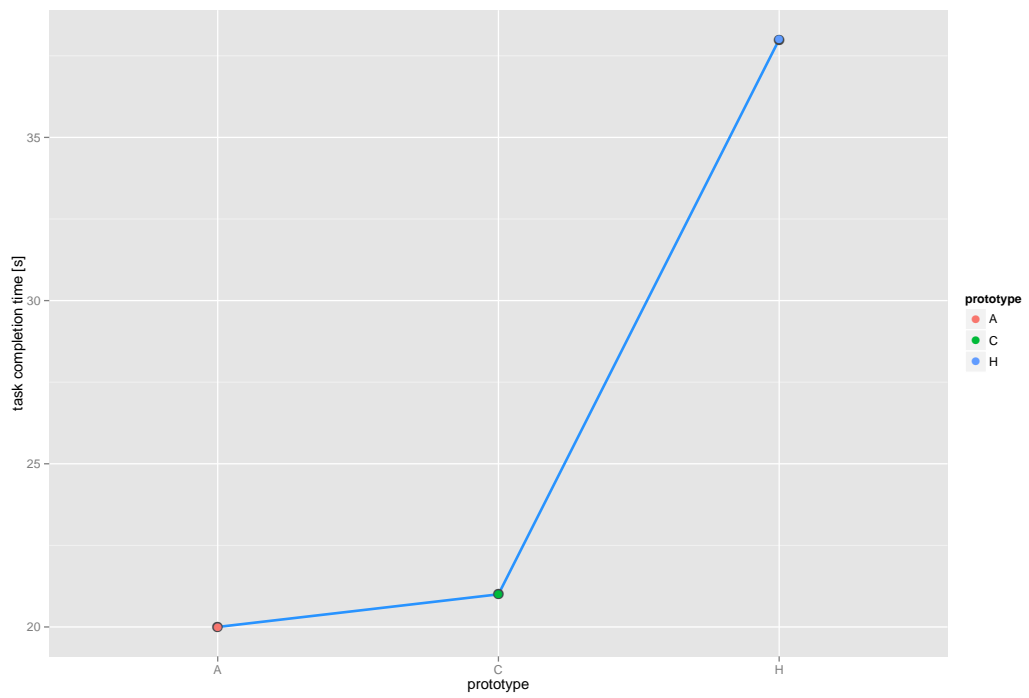


Figure G.9: Sketch Inspection Time graph - task 3.1.2

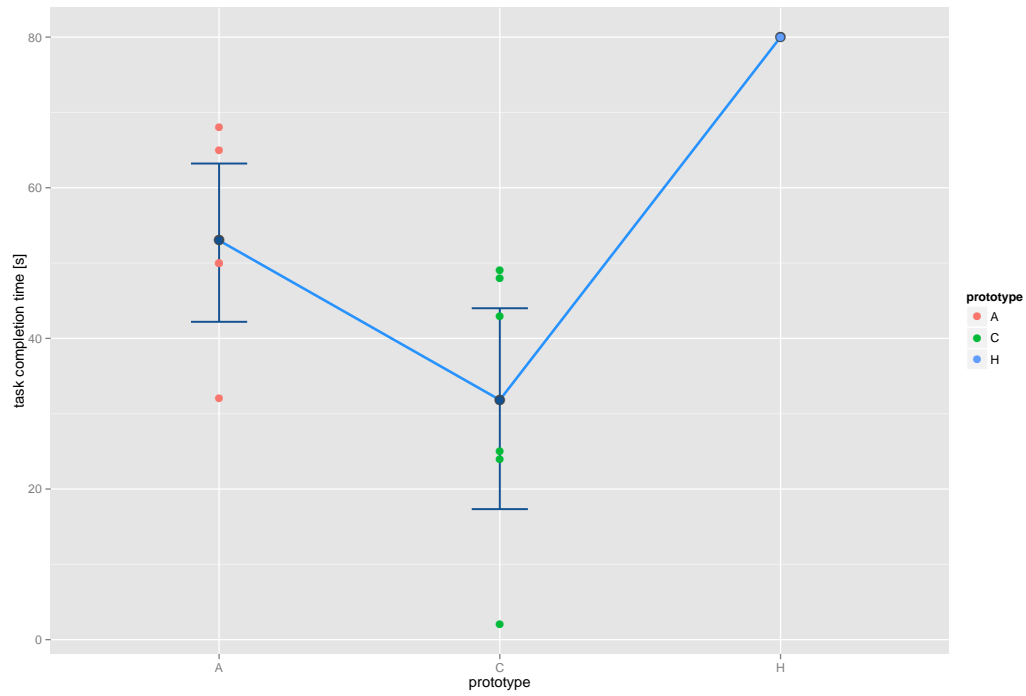


Figure G.10: Sketch Inspection Time graph - task 3.3.3

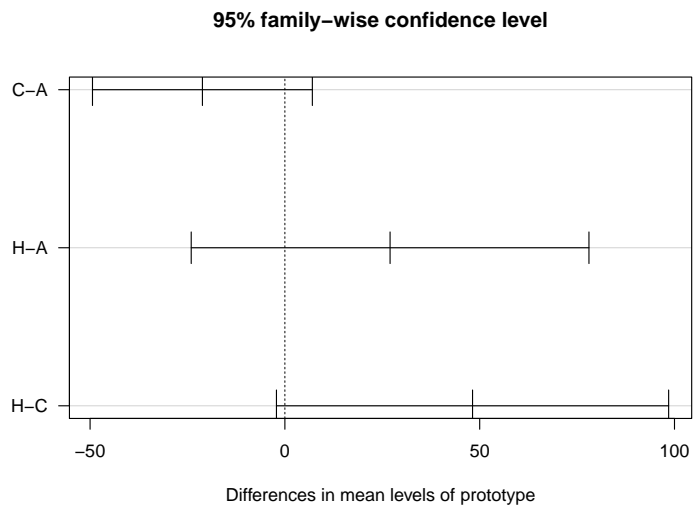


Figure G.11: The graph of task 3.3.3 shows the differences in mean of Tukey HSD post hoc comparison.

Appendix H

Sketches

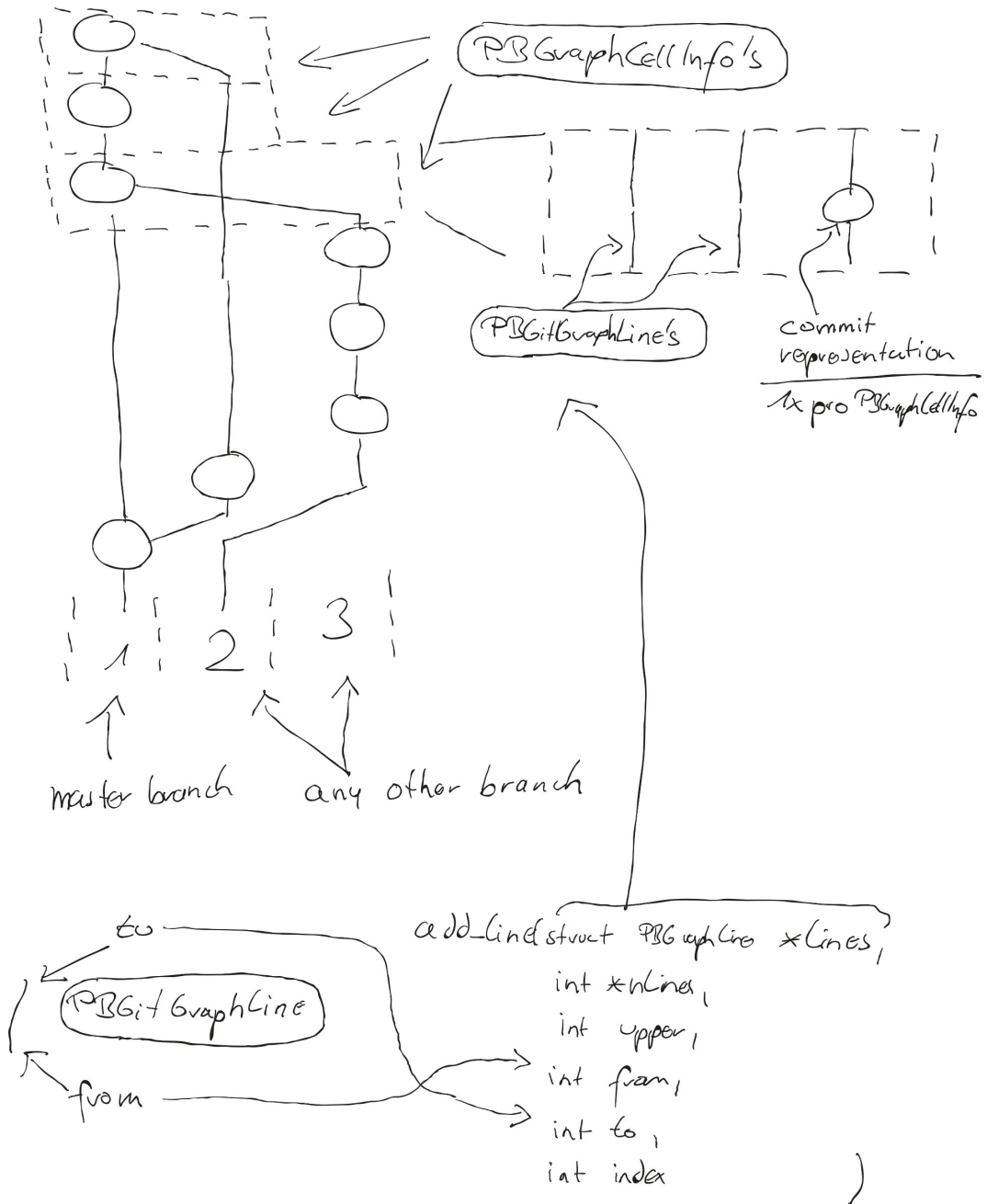


Figure H.1: Inking sketch for commit 9bfccb5

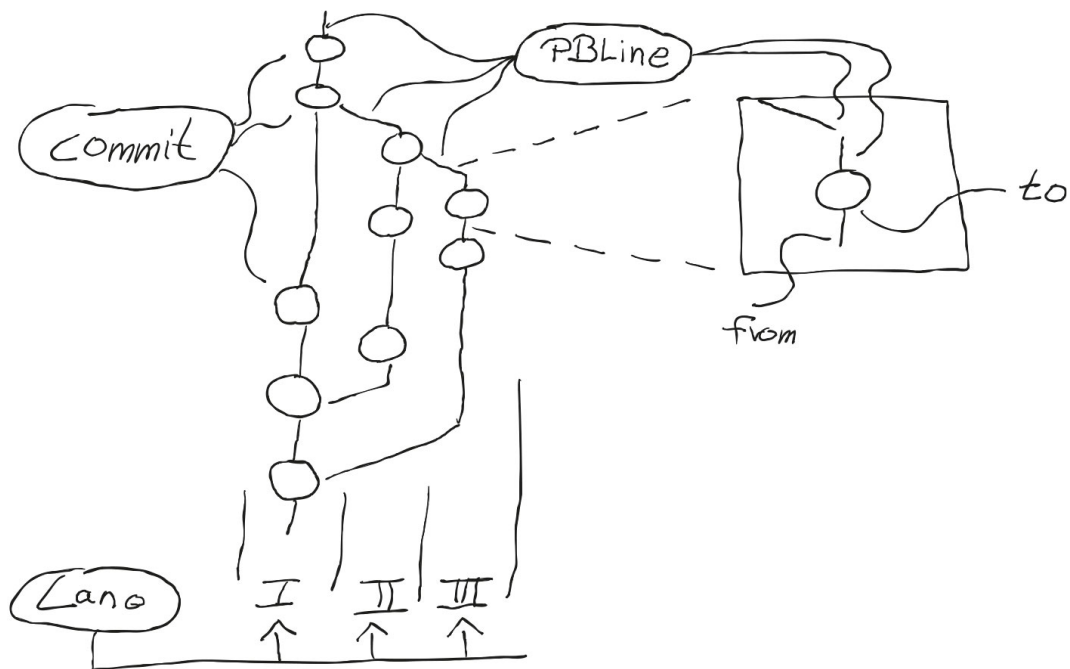


Figure H.2: Inking sketch for commit bbeedd1 used in task 2.1.3

Bibliography

- Paths. <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaDrawingGuide/Paths/Paths.html>, 2014. [Online; accessed 15-August-2014].
- libgit2 - the Git linkable library. <https://github.com/libgit2/libgit2>, 2014. [Online; accessed 15-August-2014].
- Xcode Plugins. <http://nshipster.com/xcode-plugins/>, 2014. [Online; accessed 15-August-2014].
- Objective-C bindings to libgit2. <https://github.com/libgit2/objective-git>, 2014. [Online; accessed 15-August-2014].
- Wacom Inklink. <http://www.wacom.com/de-de/de/creative/inkling>, 2014. [Online; accessed 15-August-2014].
- libdpen. https://github.com/bakercp/libdpen/blob/master/docs/ELI_FileSpec.md, 2014. [Online; accessed 15-August-2014].
- Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN 0163-5808. doi: 10.1145/170036.170072. URL <http://doi.acm.org/10.1145/170036.170072>.
- Mike Ash. Introduction to libclang. <https://mikeash.com/pyblog/friday-qa-2014-01-24-introduction-to-libclang.html>, 2014. [Online; accessed 15-August-2014].

- Font Awesome. fa-picture-o. <http://fortawesome.github.io/Font-Awesome/icon/picture-o/>, 2013. [Online; accessed 15-August-2014].
- C. Bird, P.C. Rigby, E.T. Barr, D.J. Hamilton, D.M. German, and P. Devanbu. The promises and perils of mining git. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 1–10, May 2009. doi: 10.1109/MSR.2009.5069475.
- Alexander W.J. Bradley and Gail C. Murphy. Supporting software history exploration. In *Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11*, pages 193–202, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0574-7. doi: 10.1145/1985441.1985469. URL <http://doi.acm.org/10.1145/1985441.1985469>.
- Stefan Ceriu. Sublime Text. <http://www.sublimetext.com/>, 2011. [Online; accessed 15-August-2014].
- Stefan Ceriu. Xcode MiniMap plugin. <https://github.com/stefanceriu/SCXcodeMiniMap>, 2013. [Online; accessed 15-August-2014].
- Scott Chacon. Pro git (expert's voice in software development), 2009. URL <https://github.s3.amazonaws.com/media/progit.en.pdf>.
- Hsiang-Ting Chen, Li-Yi Wei, and Chun-Fa Chang. Non-linear revision control for images. *ACM Trans. Graph.*, 30(4):105:1–105:10, July 2011. ISSN 0730-0301. doi: 10.1145/2010324.1965000. URL <http://doi.acm.org/10.1145/2010324.1965000>.
- Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07*, pages 557–566, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: 10.1145/1240624.1240714. URL <http://doi.acm.org/10.1145/1240624.1240714>.
- Oracle Corporation. Class JTextArea. <http://docs.oracle.com/javase/8/docs/api/javax/>

- swing/JTextArea.html, 2014. [Online; accessed 15-August-2014].
- Craig. Common Xcode4 Plugin Techniques. <http://www.blackdogfoundry.com/blog/common-xcode4-plugin-techniques/>, 2013. [Online; accessed 15-August-2014].
- Pieter de Bie and Rowan James. gitx. <https://github.com/rowanj/gitx>, 2014.
- Johann Dowa. Xcode Plugin Listing. <http://maniacdev.com/xcode-plugins>, 2014. [Online; accessed 15-August-2014].
- Cody Dunne, Nathalie Henry Riche, Bongshin Lee, Ronald Metoyer, and George Robertson. Graphtrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1663–1672, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208293. URL <http://doi.acm.org/10.1145/2207676.2208293>.
- Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr. Seesoft—a tool for visualizing line oriented software statistics. *IEEE Trans. Softw. Eng.*, 18(11):957–968, November 1992. ISSN 0098-5589. doi: 10.1109/32.177365. URL <http://dx.doi.org/10.1109/32.177365>.
- Herbert Ellebruch. Wacom Inklink WPI File Format. <http://www.useful-tools.de/EN-WPI-Format-Downloads.html>, 2014. [Online; accessed 15-August-2014].
- Ivan Sagalaev et al. Syntax highlighting for the Web. <https://highlightjs.org>, 2012. [Online; accessed 15-August-2014].
- Lea Verou et al. Prism is a lightweight, extensible syntax highlighter, built with modern web standards in mind. It's a spin-off from Dabblat and is tested there daily by thousands. <http://prismjs.com/>, 2012. [Online; accessed 15-August-2014].

- Michael Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131177052.
- Paul Fitts. Fitts's law. http://en.wikipedia.org/wiki/Fitts's_law, 1954. [Online; accessed 15-August-2014].
- B. Fluri, M. Wursch, M. Pinzger, and H.C. Gall. Change distilling: tree differencing for fine-grained source code change extraction. *Software Engineering, IEEE Transactions on*, 33(11):725–743, Nov 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.70731.
- Free Software Foundation. wdiff - front end to diff for comparing files on a word per word basis. <http://www.gnu.org/software/wdiff/>, 2010. [Online; accessed 15-August-2014].
- Free Software Foundation. Detailed Description of Unified Format. http://www.gnu.org/software/diffutils/manual/html_node/Detailed-Unified.html#Detailed-Unified, 2014. [Online; accessed 15-August-2014].
- fournova Software GmbH. Version control with Git - made easy. In a beautiful, efficient, and powerful app. <http://www.git-tower.com/>, 2014. [Online; accessed 15-August-2014].
- Thomas Fritz, Gail C. Murphy, and Emily Hill. titledoes a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 341–350, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi: 10.1145/1287624.1287673. URL <http://doi.acm.org/10.1145/1287624.1287673>.
- Jean-David Gadin. C / C++ / Objective-C code editor. <http://www.xs-labs.com/en/projects/codeine/overview/>, 2013. [Online; accessed 15-August-2014].
- Shiry Ginosar, Luis Fernando De Pombo, Maneesh Agrawala, and Bjorn Hartmann. Authoring multi-stage

- code examples with editable code histories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 485–494, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2268-3. doi: 10.1145/2501988.2502053. URL <http://doi.acm.org/10.1145/2501988.2502053>.
- M.W. Godfrey and Lijie Zou. Using origin analysis to detect merging and splitting of source code entities. *Software Engineering, IEEE Transactions on*, 31(2):166–181, Feb 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.28.
- Michael E. Hansen, Robert L. Goldstone, and Andrew Lumsdaine. What makes code hard to understand? *CoRR*, abs/1304.5257, 2013.
- Reid Holmes and Andrew Begel. Deep intellisense: A tool for rehydrating evaporated information. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 23–26, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-024-1. doi: 10.1145/1370750.1370755. URL <http://doi.acm.org/10.1145/1370750.1370755>.
- J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Technical Report CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.
- Apple Inc. WebView Class Reference. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/WebKit/Classes/WebView_Class/Reference/Reference.html, 2010. [Online; accessed 15-August-2014].
- Apple Inc. git-diff-files - Compares files in the working tree and the index. <https://developer.apple.com/library/mac/documentation/Darwin/Reference/Manpages/man1/git-diff-files.1.html>, 2014a. [Online; accessed 15-August-2014].
- Apple Inc. NSSlider Class Reference. https://developer.apple.com/library/mac/documentation/cocoa/reference/applicationkit/classes/NSSlider_Class/Reference/Reference.html, 2014b. [Online; accessed 15-August-2014].

- Apple Inc. NSTextView Class Reference. https://developer.apple.com/library/mac/documentation/cocoa/reference/applicationkit/classes/NSTextView_Class/Reference/Reference.html, 2014c. [Online; accessed 15-August-2014].
- Apple Inc. Creating an Outlet Connection. https://developer.apple.com/library/ios/recipes/xcode_help-interface_builder/articles-connections_bindings/CreatingOutlet.html#/apple_ref/doc/uid/TP4000971-CH15, 2014d. [Online; accessed 15-August-2014].
- Apple Inc. XCode - The complete toolset for building great apps. . <https://developer.apple.com/xcode/>, 2014e. [Online; accessed 15-August-2014].
- Thomas D. LaToza and Brad A. Myers. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0547-1. doi: 10.1145/1937117.1937125. URL <http://doi.acm.org/10.1145/1937117.1937125>.
- Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 492–501, New York, NY, USA, May 2006a. ACM. ISBN 1-59593-375-1. URL <http://research.microsoft.com/apps/pubs/default.aspx?id=74240>.
- Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, pages 492–501, New York, NY, USA, 2006b. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134355. URL <http://doi.acm.org/10.1145/1134285.1134355>.
- Chris Lattner. clang: a C language family frontend for LLVM. <http://clang.llvm.org/>, 2007a. [Online; accessed 15-August-2014].
- Chris Lattner. libclang: C Interface to Clang. [http:](http://)

- [//clang.llvm.org/doxygen/group__CINDEX.html](http://clang.llvm.org/doxygen/group__CINDEX.html), 2007b. [Online; accessed 15-August-2014].
- Vladimir Levenshtein. Levenshtein distance. http://en.wikipedia.org/wiki/Levenshtein_distance, 1965. [Online; accessed 15-August-2014].
- Y. Liu, E. Stroulia, K. Wong, and D. German. Using CVS historical information to understand how students develop software. In *MRS 2004: International Workshop on Mining Software Repositories*, 2004.
- MacroMates Ltd. textmate. <https://github.com/textmate/textmate>, 2012. [Online; accessed 15-August-2014].
- Thomas J. McCabe. A complexity measure. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. URL <http://dl.acm.org/citation.cfm?id=800253.807712>.
- Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.
- Steve Nygard. class-dump. <https://github.com/nygard/class-dump>, 2013. [Online; accessed 15-August-2014].
- Shawn O. Pearce. gitk - The Git repository browser. <http://git-scm.com/docs/git-gui/1.5.4>, 2007. [Online; accessed 15-August-2014].
- Black Pixel. Kaleidoscope - spot the differences. <http://www.kaleidoscopeapp.com/>, 2012. [Online; accessed 15-August-2014].
- Blaine A. Price, Ronald M. Baecker, and Ian S. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages & Computing*, 4(3):211–266, September 1993. ISSN 1045926X. doi: 10.1006/jvlc.1993.1015. URL <http://dx.doi.org/10.1006/jvlc.1993.1015>.
- Jorge Robles. <http://chaione.com/the-role-of-wireframing-in-mobile-app-design/>. <http://chaione.com/the-role-of-wireframing-in-mobile-app-design/>, 2013. [Online; accessed 15-August-2014].

- Rick Rodriguez. Microsoft addresses N-Trig concerns in Reddit response. <http://surfaceproartist.com/blog/2014/5/27/microsoft-addresses-n-trig-concerns-in-reddit-response>, 2014. [Online; accessed 15-August-2014].
- Ugo Braga Sangiorgi, François Beuven, and Jean Vanderdonck. User interface design by collaborative sketching. In *Proceedings of the Designing Interactive Systems Conference, DIS '12*, pages 378–387, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1210-3. doi: 10.1145/2317956.2318013. URL <http://doi.acm.org/10.1145/2317956.2318013>.
- FABIEN SANGLARD. GIT SOURCE CODE REVIEW: DIFF ALGORITHMS. http://fabiensanglard.net/git_code_review/diff.php, 2014. [Online; accessed 15-August-2014].
- Peter Savage. A Closer Look At Diffs and Tags. <http://cbx33.github.io/gitt/afterhours3-1.html>, 2014. [Online; accessed 15-August-2014].
- F. Servant and J.A Jones. Chronos: Visualizing slices of source-code history. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*, pages 1–4, Sept 2013. doi: 10.1109/VISSOFT.2013.6650547.
- Francisco Servant and James A. Jones. History slicing: Assisting code-evolution tasks. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12*, pages 43:1–43:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1614-9. doi: 10.1145/2393596.2393646. URL <http://doi.acm.org/10.1145/2393596.2393646>.
- James Simpson and Michael Terry. Embedding interface sketches in code. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology, UIST '11 Adjunct*, pages 91–92, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1014-7. doi: 10.1145/2046396.2046438. URL <http://doi.acm.org/10.1145/2046396.2046438>.
- Ian Skerrett. Eclipse Community Survey 2014 Results. <http://ianskerrett.wordpress.com/2014/06/>

- 23/eclipse-community-survey-2014-results/, 2014. [Online; accessed 15-August-2014].
- Jaime Spacco, David Hovemeyer, and William Pugh. An eclipse-based course project snapshot and submission system. In *3rd Eclipse Technology Exchange Workshop (eTX)*, Vancouver, BC, October 24, 2004.
- Lukas Spsychalski. Communication Of Source Code Designs Through Sketching. <http://hci.rwth-aachen.de/materials/publications/spychalski2013a.pdf>, 2013. [Online; accessed 15-August-2014].
- Linus Torvalds, Shawn O. Pearce, and Junio C. Hamano. git scm - distributed version control system . <http://git-scm.com/>, 2014a. [Online; accessed 15-August-2014].
- Linus Torvalds, Shawn O. Pearce, and Junio C. Hamano. git-diff - Show changes between commits, commit and working tree, etc. <http://git-scm.com/docs/git-diff>, 2014b. [Online; accessed 15-August-2014].
- Bradley L. Vinz and Letha H. Etzkorn. Improving program comprehension by combining code understanding with comment understanding. *Know.-Based Syst.*, 21(8): 813–825, December 2008. ISSN 0950-7051. doi: 10.1016/j.knosys.2008.03.033. URL <http://dx.doi.org/10.1016/j.knosys.2008.03.033>.
- Tommi Virtanen. Git for Computer Scientists. <http://eagain.net/articles/git-for-computer-scientists/>, 2007. [Online; accessed 15-August-2014].
- Wacom. Inkling Sketch Manager V1.1. <http://www.wacom.asia/en/inkling-sketch-manager-v11-1>, 2012. [Online; accessed 15-August-2014].
- Wikipedia. Cocoa (API). [http://en.wikipedia.org/wiki/Cocoa_\(API\)](http://en.wikipedia.org/wiki/Cocoa_(API)), 2014. [Online; accessed 15-August-2014].
- Wikipedia. Abstract syntax tree. http://en.wikipedia.org/wiki/Abstract_syntax_tree, 2014. [Online; accessed 15-August-2014].

- Chadd Creighton Williams, Doctor Of, and Chadd Creighton Williams. Using historical data from source code revision histories to detect source code properties, 2006.
- Annie TT Ying, Gail C Murphy, Raymond T Ng, and Mark C Chu-Carroll. Using version information for concern inference and code-assist, 2002. URL <http://www.cs.ubc.ca/~murphy/OOPSLA02-Tools-for-AOSD/position-papers/aying.pdf>.
- YoungSeok Yoon and Brad A. Myers. A longitudinal study of programmers' backtracking. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2014)*, 2014.
- YoungSeok Yoon, B.A. Myers, and Sebon Koo. Visualization of fine-grained code change history. In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pages 119–126, Sept 2013a. doi: 10.1109/VLHCC.2013.6645254.
- YoungSeok Yoon, B.A Myers, and Sebon Koo. Visualization of fine-grained code change history. In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pages 119–126, Sept 2013b. doi: 10.1109/VLHCC.2013.6645254.
- YoungSeok Yoon, Brad A. Myers, and Sebon Koo. Azurite - Adding Zest to Undoing and Restoring Improves Textual Exploration. <http://www.cs.cmu.edu/~azurite>, 2013c. [Online; accessed 15-August-2014].
- T. Zimmermann. Changes and bugs, mining and predicting development activities. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 443–446, Sept 2009. doi: 10.1109/ICSM.2009.5306296.
- Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering, ICSE '04*, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0. URL <http://dl.acm.org/citation.cfm?id=998675.999460>.

Index

abstract, xvii
ACM, 6
API, 39
Apple Cinema Display, 70
ASCII, 6
association rule mining, 20
AST, 37
audio recording, 75
Azurite, 6, 13, 56, 65

back propagation, 18
backtracking, 12
Bezier curves, 35
binary search, 74
blue circle, 49
branching & merging, 108
breadth, 12

C/C++, 66
CETokenTypeString, 42
Chronos, 14, 60
Chronos History Slicing, 65
clang, 42
class-dump, 34
Cocoa, 39
code to comment metrics, 19
Codeine, 41
CodeShape, 12, 42, 65
commit, 4, 68
compact mode, 14, 60
constructive interaction, 73
context menu, 49, 54, 58, 71
CVS, 16
cyclomatic complexity, 17

decorateCommit, 67, 68
Deep Intellisense, 11
dense chronological order, 14

- depth, 12
- DIN 5008, 75
- directed acyclic graph, 67
- DSCM, 16
- DVCS, 42
- DVTSourceTextView, 34

- eclipse, 6, 17
- Electronic parallax, 109
- entity, 3
- evaluation, 65–73

- fine-grained, 12
- Fitts's law, 60
- future work, 108–110

- Gambit, 22
- git, 42
- git diff, 43, 52
- git GUI client, 66
- git log, 43
- git rebase, 16
- gitk, 66
- GitX, 42, 43, 66
- GraphTrail, 15

- history slice, 14, 60

- IBOutlet Connection, 36, 37, 38, 39, 47
- IDE, 1
- IDESourceCodeEditorDidFinishSetupNotification, 34
- initial navigation time, 75, 82
- Inkling Sketch Manager, 35
- iPad, 27, 44
- iPad app, 47
- ISO 8601, 75

- Java, 7
- JTextArea, 34

- Kaleidoscope, 51

- Levenshtein distance, 49, 56
- libclang, 37, 39
- libgit, 42

- Macbook Pro, 70
- MagicDraw, 31
- market basket problem, 20
- Marmoset, 17
- merging & branching of sketches, 44

Microsoft Surface, 46
Microsoft Visual Studio, 6
mining, 19
motivation, 2
multistage code examples, 18
multistage source code examples, 108

notification, 34
NSSlider, 51
NSTextView, 34, 41, 101

Objective-C, 42, 61, 66
objective-git, 42, 49, 67
one-way within subjects ANOVA, 76
overloaded, 18

parseCommits, 66, 68
parser, 35
PBGitGrapher, 42, 66
pre-study, 65

rainbow color, 13
Rationalizer, 11
recursive, 54
Refactoring detection, 20
regular expression, 36, 37
ROSE, 20

SCM, 16
screen capturing, 75
Seesoft, 13
selective undo, 12
sidebar, 48
sketch, 3
sketch capabilities, 59
sketch inspection time, 93
source code visualization, 13
Sourcerer, 19
structural elements, 11
svg, 61

task completion times, 75
Textmate 2, 7
think aloud, 73
timeline, 70, 72
timeline view, 57
timestamp, 35
Tower, 53
tree, 67
Tukey HSD test, 77

UML, 31

VCS, 42

visual parallax, 109

Wacom Inkling, 34, 67

wdiff, 51, 56

WebView, 41

Windows Phone, 28

wireframe, 28

within-group design, 69

WPI, 35

Xcode, 6, 41

Xcode Interface Builder, 38, 47

