On the impact of automatic bookmarks for prediting navigation cost through sourcecode

Alexander Ertli

26. Dezember 2021

Bachelorarbeit - Informatik - Universität Bremen
Betreut von Martin Schröer
Erstgutachter: Prof. Dr. Rainer Koschke, Universität Bremen
Zweitgutachterin: Abir Bouraffa, Universität Hamburg

Erklärung

Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit[1] ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Hamburg, 26. Dezember 2021

Alexander Ertli

[1] Bei einer Gruppenarbeit muss die individuelle Leistung deutlich abgrenzbar und bewertbar sein und den Anforderungen entsprechen.

Inhaltsverzeichnis

1 Problemstellung

Programmverstehen in Software ist sehr zeitaufwändig. Laut der Arbeit von Fjeldstad und Hamlen verbrachten Entwickler in der Maintenance-Phase hiermit schon 1979 50% ihrer Zeit[1]. Heutzutage müssen immer mehr große Projekte sowie Frameworks genutzt und gepflegt werden, weshalb man annehmen kann, dass der Wartungsaufwand von Software massiv zugenommen hat. Nach einer neueren Studie ist der Anteil des Programmverstehens sogar 58% [2].

Insgesamt sollen Entwickler 35 bis 50% der Arbeitszeit damit verbringen durch Code zu navigieren. Von David Piorkowski, Austin, Henley und Tahmid Nabi wurde ebenfalls beobachtet, dass Entwickler nicht zuverlässig entscheiden können, ob eine Navigation für den gegebenen Zweck sinnvoll ist[3]. Viele Studienteilnehmer aus der Studie von Piorkowski verschätzen sich häufig bei dem Zeitaufwand einer Navigation. Analoge Beobachtungen wurden auch zuvor in einer Laborstudie von Amy J. Ko, et al. gemacht[4]. Wenn Entwickler relevante Code-Fragmente gefunden haben, folgten sie vielmals den Abhängigkeiten, um dort weitere für die Aufgabe relevante Code-Stellen zu suchen. Dabei seien sie mehrmals zu dem Ursprung zurückgekehrt. Viele Navigationen hatten die Entwickler in dieser Laborstudie sogar wiederholt getätigt. Anhand der Studienergebnisse von Amy J. Ko, et al. wurde auch bekräftigt, dass Navigationen durch den Quell-Code eine signifikante Komponente der insgesamt aufgewendeten Zeit von Entwicklern darstellt.

Ein Ansatz, um überflüssige Navigationen zu vermeiden, bzw. zu erleichtern, sind Bookmarks. Dennoch werden Bookmarks von Entwicklern nur selten bis gar nicht verwendet. Dies stellte Guzzi durch eine Befragung fest.[5].

Es scheint als, wenn die Notwendigkeit unnötige Navigationen zu vermeiden, ein sogenannter “blind spot” beim Entwicklungsprozess ist. Dies kann man als Analogie sehen zu der Festellung von Oliveira et al. 2014[6]. Hier wurde erkannt, dass Sicherheit-Aspekte und Verwundbarkeiten vom Quell-Code beim Entwicklungsprozess einen solchen “blind spot” darstellen. Damit könnte man auch das Paradox erklären, dass dieselben Entwickler aus der Befragung von Guzzi et al., die angegeben hatten, Bookmarks nützlich zu finden, diese nicht nutzen.

Um den Overhead des manuellen Setzens von Bookmarks zu eliminieren und damit die Nutzung ebendieser für Entwickler attraktiver zu machen, entwickelte Moritz Weinig das Konzept der Autobookmarks. Hierbei werden Bookmarks mithilfe eines Algorithmus für Entwickler automatisch gesetzt[7].

Das Konzept der Bookmark könnte unterstützend beim Navigieren durch unbekannten Quell-Code eingesetzt werden. Bessere Nachvollziehbarkeit eigener Navigationen könnte man mithilfe von Bookmarks erreichen. Diese würden somit bei einigen der Navigations-Probleme, die Amy J. Ko, et al. sowie David Piorkowski, et al. beobachtet hatten, unterstützend eingesetzt werden können. Konkret besteht die Möglichkeit relevante Code-Stellen zu markieren, um nach Feststellung, dass ein Navigationspfad irrelevant für die Aufgabe war, dort hin wieder zurückzukehren[7]. Dies sollte ein effizienteres Navigieren solcher Art ermöglichen und vermeiden, dass sich stetig der Ursprung eines Navigationspfades gemerkt werden muss.

Bei der Ausarbeitung von Moritz Weinig konnte beobachtet werden, dass die Akzeptanz von automatisch gesetzten Bookmarks hoch ist. Der Fokus seiner Masterthesis lag auf Entwicklung und Validierung des Konzeptes der Autobookmarks an sich, daher gab es keine ausreichende Untersuchung zu der Frage, ob Autobookmarks das Navigationsverhalten von Entwicklern sinnvoll beeinflussen können.

Diese Fragestellung ist daher der Fokus dieser Arbeit. Spezifisch ergibt sich die Prüffrage, ob mithilfe der automatischen Bookmark-Setzung, die Anzahl der Navigationen, durch den Code verringert werden kann. Entwickler beim Navigieren zu unterstützen erscheint sinnvoll, denn diese investieren viel Zeit in fehlerhafte Vermutungen und Navigationen.[3][4]. Die Vermutung liegt nahe, dass die Anzahl der Navigationen[2] eine geeignete Metrik zur Darstellung des Aufwandes der Entwickler dar, welche ein Verständnis für unbekannten Code bilden. Wie die Arbeit von Piorkowski et al. konnte bereits zeigte, können Entwickler nicht zuverlässig den Nutzen einer Navigation einschätzen. Eine Reduktion von der Navigationsanzahl erscheint daher als Metrik für eine Optimierung des Programmverstehens sinnvoll.

2 Autobookmark Eclipse-Plugin

Wie bereits erwähnt entstand das Konzept des automatischen Bookmark-Setzens, um mithilfe automatisch gesetzter Bookmarks Entwicklern die Ankerpunkte zu bieten, welche auch ein klassischer Bookmark geboten hätte. Autobookmarks werden an Code-Stellen gesetzt, denen ein Entwickler viel Aufmerksamkeit zugewandt hatte. Hierbei handelt es sich um Code-Stellen, mit denen der Nutzer viel im Eclipse-Editor interagiert. Um diese Stellen zu ermitteln wird das Autobookmarkmodel verwendet. Befragte Studienteilnehmer für die Masterthesis von Moritz Weinig hatten im Konzept der Auto-

[2]Im folgenden auch als Navigations-Performance referenziert.

bookmarks Potenzial gesehen[7]. Für diese Arbeit wurde dieses Plugin ausgebaut und rationalisiert. Hierzu siehe Kap. 2.1. Auf das Ermittlungsverfahren für die Code-Fragmente wird in den folgenden Abschnitten eingegangen, sowie in dem Kapitel 2.1. Für Details zu dem Autobookmarkmodel siehe folgende Abschnitte und das Kap. 2.2.

MIMESIS Das Plugin basiert auf dem Aufzeichnungsframework MIMESIS. Hierbei handelt es sich, um ein Framework, welches für eine Studie, zur Untersuchung vom Programmverstehen von Software Entwicklern in unbekannten Sourcecode[8], der DFG entwickelt wurde. Dazu wurden Software-Entwickler nach einer demografischen Umfrage gebeten, eine von zwei Programmieraufgaben in einer zuvor eingerichteten Remoteumgebung zu lösen. Während der Bearbeitung werden alle Interaktionen mit der IDE-Eclipse durch das Aufzeichnungsframework aufgezeichnet.

Degree of Interest Die Eclipse Erweiterung Autobookmarks ermittelt anhand der Interaktionen von den Entwicklern mit der IDE den "Degree of Interest"[9] für Code-Fragmente mit denen interagiert wurde. Dieser "Degree of Interest" wird für das Erzeugen und Verändern der "Autobookmarks" genutzt.

Der "Degree of Interest" ist ein Model, welches in Mylar angewendet wird. Hierbei werden Quellcode-Dateien, in denen der Entwickler Code editiert oder selektiert hat, mit einem "DOI-Score" versehen und mit weiteren Interaktionen dieser Art wird der Score gesteigert. Dies wird dem Nutzer durch farbliche Hinterlegung des Dateinamens im Eclipse-Paket-Explorer dargestellt[9]. In Autobookmarks wird im Gegensatz zu Mylar der "DOI-Score" je Code-Zeile berechnet und im Eclipse-Editor entsprechend repräsentiert. Zusätzlich gibt es eine tabellarische Übersicht von allen Autobookmarks mit denen man durch einen Doppelklick zu dem jeweiligen Codefragment gelangen kann[7]. Weiteres ist im Unterkapitel 2.1 Aufbau erläutert.

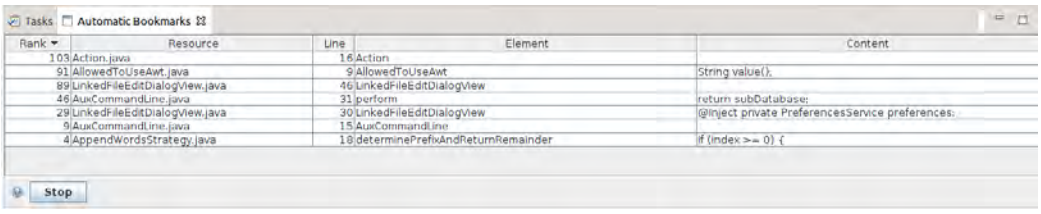

Abbildung 1: UI-Element mit der Auflistung aktiver Autobookmarks

Autobookmark Ein Autobookmark wird gezielt für eine Zeile gesetzt und hebt zusätzlich für den Entwickler dazugehörige Code-Blöcke visuell hervor. Dadurch ist eine Hervorhebung, der Codefragmente, mit denen dieser häufig interagiert wurde, gegeben.

Dies ist wahrscheinlich in folgenden Szenarium nützlich: Wenn ein Entwickler unbekannte Code-Blöcke nacheinander untersucht, also z.B. den Abhängigkeiten entlang navigiert[4], kann er mithilfe der Autobookmarks:

1. gezielt durch das Klicken eines Bookmarks aus der Auflistung von allen Autobookmarks zu vorher besuchten interessanten Code-Blöcken zurückkehren[7].
2. Irrelevante Codestellen in die man sich oft verlaufen hat, durch die visuelle Hervorhebung im IDE-Editor schneller wiederzuerkennen und im Weiteren meiden.

Beides sollte zu einer Reduzierung der Navigation insgesamt beitragen und somit auch einer gesteigerten Effizienz der Entwickler.

2.1 Evolution

Autobookmarks ist ein Konzept mit dem für Entwickler Code-Fragmente, denen viel Aufmerksamkeit zugewandt wurde, mit Bookmarks versehen und durch Hervorhebungen markiert werden. Alle vorgenommenen Änderungen und Erweiterungen oder Rationalisierungen an dem Quell-Code von Autobookmarks berücksichtigen diesen Grundsatz. Im Vergleich zu der Version von Moritz Weinig, wurde das User-Interface und die internen Datenstrukturen stark rationalisiert.

Durchgeführte Änderungen und deren Motivation:

1. Das Plugin wurde erweitert, um zuverlässige Protokollierung von jedem sichtbaren Autobookmark sicherzustellen.
2. Weiterhin wurde die neueste Version von Mimesis integriert, um den Vergleich zu den Daten aus der Mimesis-Studie zu ermöglichen.
3. Der Algorithmus zum Erzeugen der automatischen Bookmarks wurde überarbeitet. Dies war unumgänglich, um eine unbeaufsichtigte Studie durchzuführen, da sonst bei einer intensiveren Nutzung oder einer längeren Arbeitssitzung der Ressourcenverbrauch zu hoch wäre.
    (a) Auch war Autobookmark nicht für komplexe Projekte mit vielen Quell-Code Dateien ausgelegt. Das machte sich bemerkbar, indem

bei Interaktionen mit vielen verschiedenen Dateien, zu viele Autobookmarks angelegt wurden. Da zu den Autobookmarks in der ursprünglichen Version von Moritz Weinig dem Nutzer kein Score präsentiert wurde, war die Hervorhebung von besonders interessanten bzw. oft besuchte Code-Stellen nicht mehr gegeben.

(b) Außerdem wurde der Algorithmus erweitert, um alle sinnvollen Möglichkeiten auszunutzen, die Mimesis im Gegensatz zu Mylar[9] bietet, sodass für die Berechnung des DOI-Scores nicht nur die Selektion und Editierung von Codezeilen berücksichtigt werden, sondern viele weitere Interaktionsarten mit dem Eclipse-Editor. Dies sollte sicherstellen, dass auch für Nutzer, die generell Text beim Lesen nicht selektieren, auch Autobookmarks erzeugt werden.

Weitere vorgenommene Änderungen und Ergänzungen sind im folgenden Unterkapitel Aufbau und im Abschnitt Eventtypen beschrieben. Die Funktionsweise von Autobookmarks und des Berechnungsalgorithmus ist in den jeweiligen Unterkapiteln erläutert.

Aufbau Ein Autobookmark besteht aus folgenden Daten:

1. Rank: dies stellt den Score eines Codeelements dar
2. Codeelement: Dies ist ein Tupel aus:
   (a) Ressource: Die Quellcodedatei
   (b) Line: Die Zeile für welche der Score gilt

Der Score ist quantitativer Wert zur Bewertung der Relevanz eines Codeelements. Dies ist notwendig, um die Autobookmarks, nach der Relevanz sortiert, dem IDE-Nutzer zu präsentieren. Dieser wird stark angelehnt an den "Degree of Interest"[9] berechnet, daher ist der Score eine Zusammensetzung aus der gewichteten Anzahl von Events, die eine Interaktion mit dem jeweiligen Codefragment darstellen. Jedes Autobookmark hat einen eigenen Score. Design-Ziel war es, dass mithilfe des Scores folgende Abstufungen der Relevanz für jedes Autobookmark zuverlässig möglich sind:

1. Score > 40 sehr Relevant
2. Score > 20 Relevant
3. Score > 1 wenig Relevant
4. Score > 0 Neutral

Aufgrund der Funktionsweise vom Berechnungsalgorithmus und der Decay-Funktion gibt es keine Notwendigkeit den Score nach obenhin zu begrenzen. Der Berechnungsalgorithmus und die Decay-Funktion wird im hierauf folgenden Kapitel erläutert.

2.2 Autobookmark Model

Der Berechnungsalgorithmus basiert auf dem Autobookmark Model von Moritz Weinig[7], wurde aber an einigen Stellen angepasst, um die Abstufungen der Relevanz wie bereits dargelegt zu erreichen.

Folgende Anpassungen wurden vorgenommen:

1. Es wurde eine Score-Decay Funktion eingebaut.
2. Autobookmarks mit neutraler Relevanz werden dem Nutzer nicht angezeigt.
3. Der Score darf nicht unter den Nullpunkt 0 fallen.
4. Es wurden mehr Eventtypen für die Score-Berechnung hinzugezogen.
5. Die unabhängige Gewichtung der einzelnen Eventtypen wurde eingeführt.
6. Die Gewichtung der Eventtypen wurde iterativ ausbalanciert.

2.2.1 Eventtypen

Folgende Event-Typen wurden für die Berechnung berücksichtigt:

Eventtyp	Gewichtung
textSelectionEvent	1
viewEvent	0.5
editorTextCursorEvent	0.9
editorMouseEvent	0.8
scrollEvent	0.1
codeChangeEvent	0.9

Tabelle 1: Relevante Interaktionstypen

Die Art der Interaktion wird durch Event-Typen unterschieden. Diese konkreten Eventtypen wurden ausgewählt, da es bei diesen, um zeilenspezifische Interaktionen mit dem Eclipse Editor handelt.

Gewichtung Um zu gewährleisten, dass Autobookmarks zuverlässig beim Auftreten gewisser Interaktionsmuster gesetzt werden, mussten die Eventtypen gewichtet bei der Autobookmark Berechnung einfließen. Die eben erwähnten Interaktionsmuster werden im folgenden Abschnitt genauer beschrieben. Der Wert von den Gewichten ist hierbei nicht repräsentativ für die "Wichtigkeit" des jeweiligen Eventtyps. Da wie eben erläutert manche Eventtypen häufiger vorkommen als andere, ist auch für die häufigeren Eventtypen eine kleinere Gewichtung notwendig. Beispielweise wird das 'codeChangeEvent' pro Charakter erzeugt. Als Folge ist der Beitrag zur Steigerung des Scores vom 'codeChangeEvent' bei gleicher Gewichtung viel höher, als der eines Beispielweise 'textSelectionEvent's, da dieses Event nur einmalig für die Interaktion vorkommt. Daher wurde auch die Gewichtung entsprechend geringer gewählt.

Tuning der Gewichtung Beim Auftreten dieser Interaktionsmuster sollten entsprechend Autobookmarks für die jeweiligen Code-Fragmente erzeugt werden. Durch das Tunen zu einer möglichst Gleichberücksichtigung der Events für die Berechnung ist dies in der gewählten Konfiguration gegeben. Deshalb war notwendig, dass trotz der unterschiedlichen Häufigkeiten alle Eventtypen in der Berechnung gleich berücksichtigt werden. Hierzu wurde iterativ jeder Wert einzeln getunt bis das gewünschte Verhalten erreicht wurde. Dabei wurden 3 Entwickler gebeten die finale Konfiguration zu testen. Durch deren positive Rückmeldung wurde Zweckorientierung der gewählten Konfiguration bestätigt. Beim Tunen der Gewichte wurde mitberücksichtigt, dass die Decay-Funktion ausgelöst wird, um Autobookmarks mit denen länger nicht interagiert wurde, zu entwerten.

Interaktionsmuster Als aussagekräftige Interaktionen über die Relevanz eines Codefragmentes, halte ich:

- Gehäufte Mausbewegung, wie zum Beispiel dies bei kreisender Bewegung um einen Codeblock der Fall ist und die Bearbeitung des Codefragmentes an sich.
- Das Bewegen des Textcursors in dem Codefragment.
- Die Textselektion von Codefragmenten hat auch eine hohe Aussagekraft, daher ist diese auch mitberücksichtigt worden.
- Durch die Berücksichtigung des Viewevents wurde das einmalige Ansehen von Codefragmenten in die Berechnung des Scores miteinbezogen. Dies ist allerdings so gewichtet worden, das einfaches Überfliegen nicht

ausreicht, um zu diesem Codefragment ein wenig relevanten Score zu vergeben.

- Scrollen durch eine Datei wurde für die Berechnung des Scores nur sehr gering gewichtet. In Kombination mit dem 'viewEvent' ist dies aber durchaus sinnvoll. Da so auch beim schnellen hin und her Springen zwischen zwei weit auseinander liegenden Codefragmenten in derselben Datei die Autobookmarks sinnvoll gesetzt werden können.

2.2.2 Berechnungsalgorithmus

Für jede Interaktion mit der IDE wird ein Event mit allen relevanten Informationen vom Mimesis-Modul erzeugt. Alle erzeugten Events werden von Mimesis während der Laufzeit gesammelt. Periodisch wird der Autobookmark-Algorithmus alle neu hinzugekommenen Events relevanten Typs abarbeiten.

```
unprocessedEvents.stream().forEachOrdered(event -> {
  visitors.stream().forEach(visitor -> event.accept(visitor));
  });
```

Listing 1: Abbarbeiten von Events geschieht mithilfe des Visitor-Patterns

Der Algorithmus berechnet dann aus den relevanten Events für jede Code-Zeile einen Score, falls für dieses Codeelement bereits Score existiert, so werden beide addiert. Dies wird in dem Datenmodel der Klassen Instanz "DegreeOfInterest" zentral für die Anwendung festgehalten. Dabei wird die zusätzlich für alle weiteren Score Einträge, um den Decay-Wert von 0.0001 verringert. Die Integration der Decay-Funktion war aus denselben Gründen notwendig, wie die der Decay-Funktion in Mylar[9]. Im anderen Fall würden bei einer Interaktion-intensiven Arbeitssitzung in jeder besuchten Datei Bookmarks gesetzt sein und diese somit keine Aussagekraft bezüglich der Relevanz mehr haben.

```
private Map<IFile, Map<Integer, Double>> data = new HashMap<>();
```

Listing 2: Datenstruktur des DOI-Scores. Die erste Map hat als Schlüssel eine Datei. Die zweite Map enthält die Zuordnung des Scores zu jeder einzelnen Zeile dieser Datei

Masking-Verfahren In einem zweiten Schritt werden alle Score Einträge mit der Datenrepository "BookmarkRepository" nach einem besonderen Masking-Verfahren synchronisiert.

Hierbei passieren folgende Schritte:

- Sortiere die Zeilennummern-Datei nach dem Score.
- Hole den obersten Listeneintrag und mache dafür einen Autobookmark.
- Überspringe die nächsten 30 Einträge.
- Wiederhole das Verfahren für alle weiteren Zeilen der Datei bis die gewünschte Anzahl an Bookmarks angelegt wurde.

```
data.forEach(
(k,e)-> e.replaceAll((f,g) -> reduce(g))
);
```

Listing 3: Implementierung der Decay-Funktion, siehe hierzu die Datenstruktur des DOI-Scores im Vergleich: Listing 2.2.2

Das Masking-Verfahren wurde von Moritz Weinig entwickelt und verhindert, dass innerhalb derselben Codefragmente mehrere Autobookmarks gesetzt werden, was keiner sinnvollen Verteilung der Bookmarks entspricht.

Bis auf die vorher genannten Punkte funktioniert der Berechnungsalgorithmus weiterhin nach der Spezifikation wie in der vorhergehenden Arbeit[7] und wurde lediglich in der Implementierung zwecks Reduzierung der Kopplung zwischen den Klassen des Autobookmarkmoduls zu den Klassen aus dem Mimesis Modul, da diese in der neuen Version eine andere Schnittstelle haben oder Zwecks Steigerung der Kohäsion und Lesbarkeit, Testbarkeit, wie auch zur Eliminierung von aufgedeckten Bugs.

Es war wichtig, dass das Autobookmark Model sich sehr ähnlich bis gleich verhält, wie die für Moritz Weinigs Studie gewählte Konfiguration, damit auch die Korrektheit des Modells bezüglich der Hervorhebung von relevanten Codefragmenten die Moritz Weinig in seiner Arbeit festgestellt hat, auch für diese Arbeit weiterhin gilt.

3 Methode

Um die Fragestellung dieser Arbeit zu untersuchen, war eine Datensammlung notwendig. Es wurden Software-Entwickler gebeten an einer demografischen Umfrage teilzunehmen und gebeten mithilfe einer remote Desktopumgebung mit eingerichteter Eclipse-IDE eine Programmieraufgabe zu bearbeiten. Hierzu wurden Studierende und Mitarbeiter des Informatikfachbereiches über E-Mail-Aufrufe zur Teilnahme an der Studie über einen Weblink aufgerufen. Ebenso wurden gezielt E-Mails an Studierende aus fortgeschritte-

nen Informatik-Veranstaltungen verteilt. Die Bearbeitung der Aufgabe wurde aufgezeichnet[3].

Inhalt der Studie Im Vorfeld der Studie wurde auf Zweck und Funktionsweise des Bookmark-Tools hingewiesen. Im Abschluss ist eine weitere Umfrage durchgeführt worden. Bis auf die letzte Umfrage war der Ablauf, Inhalt der Umfrage und die Aufgabe identisch mit der Aufgabe B der Studie zu Mimesis. Für weiteres zur Aufgabe siehe im entsprechenden Unterkapitel 3.1.

Mimesis Mimesis ist ein Aufzeichnungs-Framework, welches im Rahmen eines DFG Projekts entwickelt wurde. Ziel des Forschungsprojektes ist, Arbeitsmuster von Entwicklern beim Bearbeiten von Programmieraufgaben an unbekannten Code aufzudecken. Dies geschieht mittels der Eclipse-IDE in welcher das Framework als Eclipse-Plugin eingebunden ist. Dies ermöglicht die Aufzeichnung aller Interaktionen von Entwicklern mit der Eclipse-IDE. Das Autobookmark-Modul ist eine Erweiterung des Mimesis-Frameworks. Daher ist es möglich dieses ebenfalls zur Aufzeichnung zu nutzen. Die Aufzeichnungsdaten werden durch das Autobookmark-Modul nicht verändert, was einen Vergleich aufgrund des gleichen Aufbaus von den beiden Studiendatensätzen mit und ohne Autobookmarks-Modul ermöglicht.

Technischer Aufbau Teilnahme an der Studie war mittels eines Webbrowsers über den Link der Studie möglich. Nach Öffnen der URL im Browser konnte man über einen Button eine Sitzung starten, die bei erneutem Besuch des Links Wiederaufnahme der Sitzung ermöglichte. Für die Durchführung der Studie wurde eine VM von der Uni-Bremen zur Verfügung gestellt mit 16GB RAM 160GB Festplatte und 4 Kernen der 6-ten Generation eines Intel Xeon Prozessors. Der technische Aufbau und die Umfrage-Anwendung war nicht mit der von der Mimesis Studie identisch, hatte dennoch denselben Ablauf simuliert. Es wäre wünschenswert gewesen dieselbe VM und denselben Aufbau für die Durchführung der beiden Studien zu nutzen, aber der technische Aufbau und die Webanwendung war hierfür nicht ausgelegt, daher war es weniger Aufwand alles neu zu entwickeln. Auch einzelne Komponenten auf einen anderen Server aufzusetzen war keine Option, da die Datenbank der Mimesis Studie zu mocken, zu viel Overhead mit sich gebracht hätte. Die wesentlichen Unterschiede zwischen den beiden Studien, welche ein Studienteilnehmer wahrnehmen konnte, waren, dass die Autobookmarkstudie nach

[3]Für Details siehe Abschnitt: 3.2

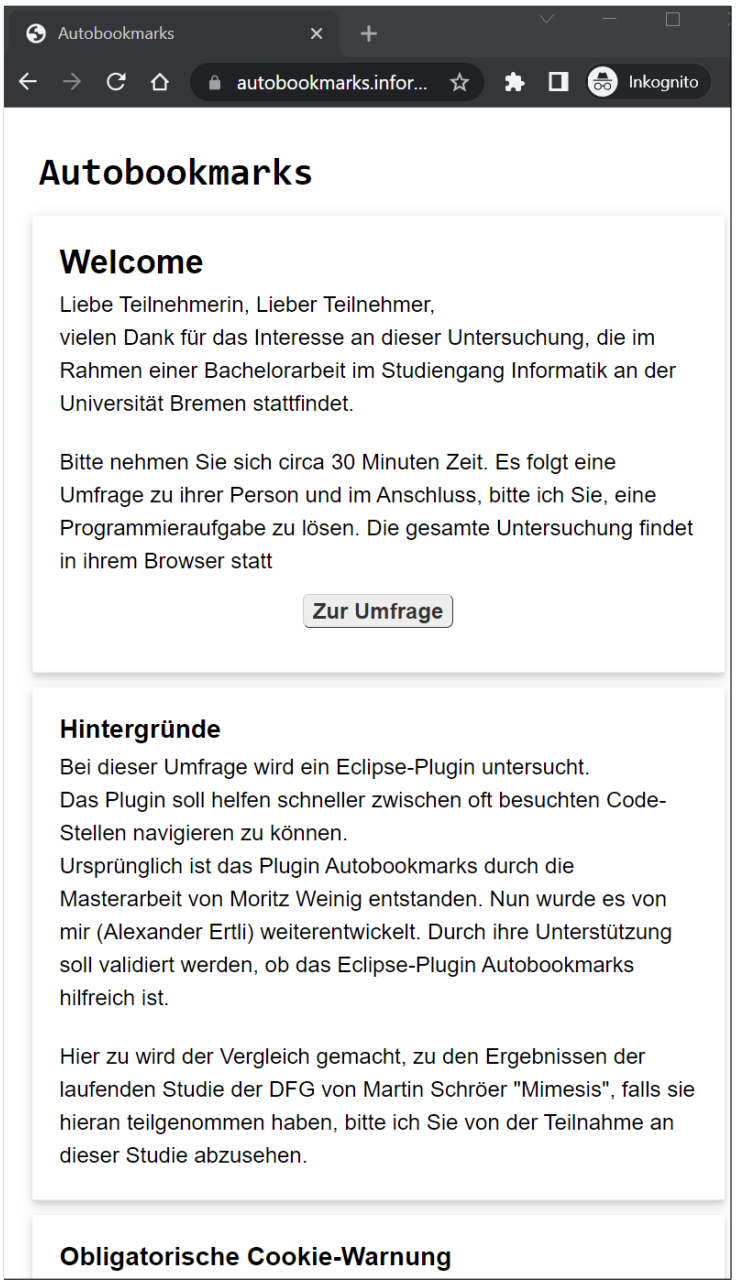

Abbildung 2: Landingpage der Autobookmark-Web-App

Abschicken der demografischen Umfrage sofort eine Desktopumgebung öffnete, statt hier eine 6 Minutenwartezeit zur Generierung der Desktopumgebung abzuwarten, sowie das die Desktopumgebung etwas performanter war.

Abbildung 3: Anleitung aus der Autobookmark-Web-App zum Autobookmarks Plugin

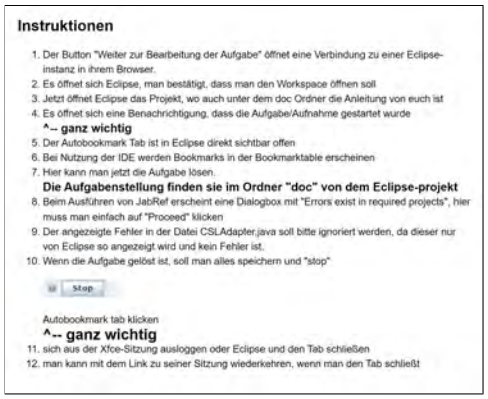

Abbildung 4: Anleitung aus der Autobookmark-Web-App zur Programmieraufgabe

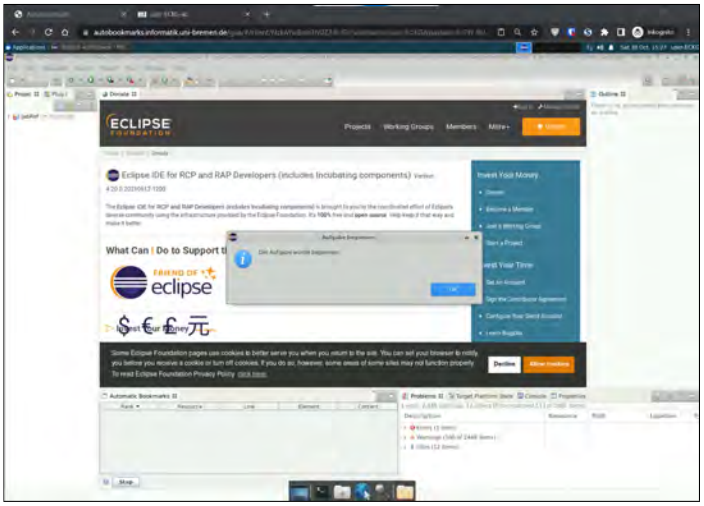

Abbildung 5: Desktopumgebung, wie ein Studien-Teilnehmer sie bei starten der Programmieraufgabe aus der Web-App heraus vorfindet

3.1 Programmieraufgabe

Wie bereits erwähnt wurde eine der Programmieraufgaben aus der Studie zum Programmverstehen der DFG übernommen. Hierbei sollte man einen Fehler in der Software JabRef beheben. Um diesen zu beseitigen, musste der Studien-Teilnehmer eine fehlerhafte Objektinstantiierung berichtigen.

```
@FXML
private void initialize() {
  viewModel = new AboutDialogViewModel(dialogService, null,
      buildInfo);

  textAreaVersions.setText(viewModel.getVersionInfo());
  this.setResizable(false);
  }
```

Listing 4: Die Fehlerhafte Objektinstantiierung ist in Zeile 3 dieses Code-Snippets. Um den Fehler zu beheben, muss man anstatt des null-Parameters die Instanz des 'clipBoardManager' übergeben, welcher weiter oben im Quell-Code initialisiert wurde

3.2 Aufzeichnungs-Daten

Durch das Mimesis-Modul wurden alle Interaktionen des Entwicklers mit der IDE aufgezeichnet und mit einer Erweiterung des Autobookmark- Plugin alle zum jeweiligen Zeitpunkt vorhandenen Bookmarks protokolliert.

Es gibt 2 Aufzeichnungsdateien. Die Recording-XML-Datei [4] von dem Mimesis-Modul und die Aufzeichnungsdaten des Autobookmark-Moduls.

Mimesis-XML Aus den Aufzeichnungsdaten vom Mimesis-Plugin werden unter anderem folgende Daten für jedes sample ermittelt:

filename	directory	opened time	switching-time	durration

Diese Aufzeichnungsdaten sind für die Untersuchung des Navigationsverhaltens zentral. Die Daten-Struktur in der Aufzeichnungs-XML-Datei ist für diesen Zweck aber nicht optimal. Daher sind einige Umwandlungen und Aggregation(en) notwendig, um das gewünschte Tabellenformat zu erhalten.

Die Aufzeichnungs-XML-Datei besteht aus 3 Tabellen, die erste enthält Informationen zur Arbeitsumgebung wie der Auflösung und die dritte Tabelle enthält die im Vordergrund geöffneten Dateien des Eclipse-Editors. Die zweite beinhaltet eine Liste aus allen Events, die das Mimesis-System aufnimmt jeweils im Tupel mit einem Schlüssel zu dem zugehörigen Eintrag aus der zweiten Tabelle. Dies sind nicht nur Interaktionen des Entwicklers, sondern

[4] Für eine Auflistung aller von Mimesis aufgezeichneten Daten siehe: `https://gitlab.informatik.uni-bremen.de/dfgpc/mimesis/-/wikis/events/Event-Table`

z.B. auch Hintergrundaufgaben die Eclipse ausführt, sogenannte 'RecourceEvents'. Aus der dritten Tabelle werden die Events mit dem Typ 'recordingEvent', 'ViewEvent' und 'launchEvent' eingelesen sowie mit den Einträgen aus der zweiten Tabelle gemergt. Hiervon ausgehend sind folgende Aggregation(en) notwendig, um das Zielformat zu erreichen:

- Als Start der Arbeitssitzung wird das erste 'recordingEvent' genutzt.
- Es wird ermittelt, wann es einen Wechsel der geöffneten Datei gab und die beiden Zeitpunkte wird als 'opened time' und 'switching-time' festgehalten mit den entsprechenden Einträgen 'filename' und 'directory'.
- Die zugehörige Differenz der Zeitpunkte wird als 'durration' festgehalten.
- Für das Ende der Arbeitssitzung wird das entsprechende recordingEvent genutzt.
- Von allen Einträgen in den Spalten 'opened time' und 'switching-time' wird der Eintrag der Spalte 'opened time' abgezogen und die Ergebnisse als neue Werte gespeichert. Dies hilft beim Visualisieren der Daten, da die Zeit Achse dadurch immer bei 0 beginnt.
- Falls ein Nutzer mehrere Arbeitssitzungen gebraucht hat, werden die beiden Datensätze gemergt.

Die Anzahl der Einträge des Endformates entspricht der Anzahl der Navigationen. Eine Navigation stellt dabei das in den Vordergrund bringen des Eclipse-IDE-Editors einer Datei dar.

Zusätzlich werden die Zeitpunkte des Aufrufens des Debuggers und des Quellcodes aus dem "recordingsEvents" in einer weiteren Tabelle ermittelt.

Das Auswertungsverfahren unterscheidet sich zu dem in der DFG-Mimesis-Studie, da dort ein Visualisierungssystem für entwickelt worden ist, welches für diese Auswertung nicht genutzt wird. Eine statistisch fokussierte Datenanalyse scheint für die Untersuchung des Navigationsverhaltens viel angebrachter, als mit Visualisierungen zu arbeiten, deren Zweck die Untersuchung der Arbeitsmuster ist. In der Studie von Moritz Weinig, gab es keine Analyse seiner Datensammlung, da aufgrund von Software-Fehlern die Aufzeichnungsdaten beschädigt worden sind[7]. Daher ist ein Vergleich zu dieser Datensammlung nicht möglich.

Autobookmarks-XML Die Autobookmarks-XML ist für die Forschungsfrage dieser Arbeit nicht von großer Relevanz. Es war dennoch notwendig

dieses Feature zu realisieren und die Daten zu sammeln um:

- Den Auto-Bookmark-Algorithmus und die Gewichtung der Eventtypen systematisch zu kalibrieren, wie in Kap 2.2.1 bereits beschrieben wurde.
- Funktion des Autobookmarksplugins während der Studien-Teilnahme zu validieren.
- Aufzuzeichnen, ob Autobookmarks sinnvoll gesetzt worden sind.

Die Autobookmarks-XML hat das Ziel festzuhalten, welche Bookmarks mit welchen Score dem Nutzer sichtbar waren. Da die Autobookmarks zur Laufzeit sich stetig ändern und 3 stark miteinander gekoppelte Datenstrukturen hierbei beteiligt sind, war es nicht ganz offensichtlich wie eine sinnvolle Persistierung der Autobookmarks zu erreichen ist.

Als sinnvollste Lösung erschien:

- Verschieben und Zentralisieren der Anwendungslogik sodass:
  - Alle Autobookmarks zentral in einer übergeordneten Datenstruktur "Repository" festgehalten werden
  - Anlegen, verändern, un-/sichtbar stellen bzw. löschen von Autobookmarks darf nur durch Methoden dieser "Repository" geschehen. Hierbei ist es erwähnenswert, dass es nicht ohne noch weitere Umstrukturierungen möglich war einen Bookmark zu löschen, dieser daher hier lediglich als unsichtbar markiert wurde.
  - Andere Klassen die Referenzen auf Autobookmarks speichern, wie z. B. in der Visualisierungsschicht, dürfen keine Anwendungslogik zur Selektion oder Verarbeitung von Autobookmarks haben
- Protokollierung aller relevanten Methodenaufrufe der Repository

Folgende Daten werden protokolliert:

timestamp	filename	filelocation	line	score	repository-event

Dabei sind:

- timestamp: Der Zeitpunkt des Methodenaufrufs in der Bookmarkrepository.
- filename: Der Name der Datei die dem Bookmark zugeordnet ist.
- filelocation: Das relatives Verzeichnis von der Datei.
- line: Die Zeile in welcher das Autobookmark gesetzt wurde.

- score: Der Degree of Interest der zu dem Zeitpunkt dem Bookmark zugeordnet war.
- repository-events: Stehen für Methodenaufrufe in der Bookmarkrepository, die alle Autobookmarks verwaltet. Folgende Repository-Events gibt es:

Repository-Event	**Beschreibung**
create	Erstellung eines Bookmarks
setVisable	sichtbar machen eines Bookmarks
setInvisable	unsichtbar machen eines Bookmarks
update	verändern des Scores eines Bookmarks
delete	löschen eines Bookmarks

Tabelle 2: Repository-Protokollierung

Mit diesen Daten ist es möglich festzustellen, welche Bookmarks dem Nutzer sichtbar waren und welchen Score die Bookmarks zu diesem Zeitpunkt hatten. Hierfür benötigt es entsprechender Aggregations- und Filteroperationen. Für das Tunen der Gewichtung einzelner Events war von besonderer Relevanz, welche Bookmarks mit welchem Score dem Nutzer zum Zeitpunkt einer Navigation angezeigt wurden. Dies kann man wie folgt ermitteln:

- Gruppiere mit 'fileName','filelocation','line' in der Tabelle A
- Gruppiere die Einträge mithilfe des 'create' Events in einer weiteren Tabelle B
- Aus der Tabelle B kann man nun mithilfe der Spalte 'timestamp' alle Autobookmarks eines gewissen Zeitraumes filtern
- Ob ein Autobookmark dem Nutzer sichtbar war, kann man anhand der Spalte Score erkennen. Wenn der Score den Nullpunkt darstellt bedeutet dies, das Bookmark zu dem Zeitpunkt für den Nutzer unsichtbar gewesen war.

3.3 Auswertungsmethode

Aus den gesammelten Daten lassen sich ein Vielfaches an Metriken ermitteln. Dennoch war Ziel dieser Ausarbeitung lediglich zu prüfen, ob man durch die automatische Bookmark-Setzung, die Anzahl der Navigationen durch den Code verringern kann.

Für die Untersuchung dieser These ist es, zwingend erforderlich nur Datensamples zu berücksichtigen, bei denen die Programmieraufgabe erfolgreich

gelöst worden ist. Nur so ist die Vergleichbarkeit der Datensamples zueinander gegeben. Daher wurden nach diesem Kriterium die Daten aus dem Datensatz der Mimesis und Autobookmark Studie selektiert. Aus diesen Daten wurde anschließend die Navigationsanzahl, welche notwendig zum Lösen der Aufgabe war, je Teilnehmer ermittelt. Ebenso wurde der Durchschnitt aller Teilnehmer einzeln für die Studie Autobookmarks und Mimesis berechnet. Für individuelle Auswertungen interessanter Datensamples wurde die Darstellung der Navigationspfade als Graph und die Darstellung der Sequenz der Navigationen als Scatterplot unterstützend eingesetzt. Zeitangaben hierbei dienen nur zu Zwecken der Plausibilität. Denn es wurde nicht berücksichtigt, ob und wie viel ein Teilnehmer Idle war. Die Angaben von den erfolgreichen Teilnahmen aus der End-Umfrage wurden bezüglich folgender 3 Punkte ausgewertet:

1. Haben Entwickler die Autobookmarks aktiv genutzt?
2. Finden Entwickler, dass mit Autobookmarks sich unnötige Navigationen vermeiden lassen?
3. Finden Entwickler das Konzept an sich als nützlich?

4 Ergebnisse

Insgesamt konnten 35 Entwickler für die Studie gewonnen werden. 19 Teilnehmer versuchten folgten allen Anweisungen, aber nur 7 hiervon lösten die Programmieraufgabe erfolgreich. 16 Teilnehmer brachen die Teilnahme nach Lesen der Aufgabenstellung ab.

35	Abgaben der demografischen Umfrage	Gruppe A
29	Teilnehmer starteten die Desktop-Sitzung	Gruppe B
19	lasen die Aufgabestellung der Programmieraufgabe	Gruppe C
18	gaben die End-Umfrage ab	Gruppe C1
12	Teilnehmer folgten allen Anweisungen, aber lösten die Aufgabe nicht	Gruppe C2
7	Teilnehmer haben die Aufgabe gelöst	Gruppe C3

Tabelle 3: Datensammlung Autobookmarks. Gruppe B ist eine Untergruppe von der Gruppe A. Gruppe C wiederum ist eine Untergruppe der Gruppe B. Gruppe C1, C2 und C3 sind Untergruppen von der Gruppe C. Die beiden Gruppen C2 und C3 sind disjunkt.

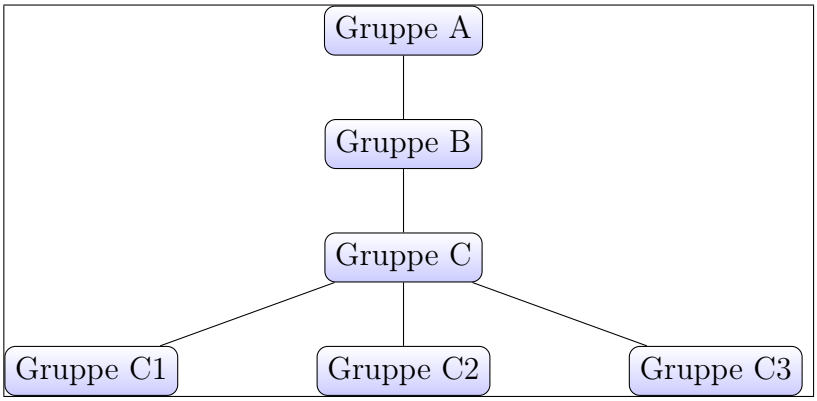

Abbildung 6: Hierarchie der Gruppierungen von der Datensammlung Autobookmarks

```
key_group_c2  = {
  k:key_group_c[k] for k in key_group_c
    if k not in key_group_c3
}
```

Listing 5: Selection der Sitzung-IDs der Gruppe C2 aus der Gruppe C

Die Teilnehmer der Gruppe C hatten mindestens mit der Eclipse-IDE interagiert und die Aufgabenstellung gelesen. Gruppe A, B, C und C1 werden für diese Arbeit nicht weiter betrachtet. Die Gruppe C2 wird nur stichprobenartig untersucht, um etwas Verständnis über die geringe Erfolgsquote der Teilnehmer zu bekommen.

Gruppe C2 Obwohl von ausreichend Zeit und Mühe in die Bearbeitung investiert worden ist, gelang es vielen Teilnehmern nicht die Aufgabe erfolgreich zu lösen. Siehe hierzu die Tabelle 10. Durchschnittlich betrug die Arbeitssitzung der Gruppe C2 1 Stunde 36 Minuten und 32 Navigationen. Zur Erinnerung eine Navigation stellt den Wechsel der im Vordergrund geöffneten Datei des Eclipse-IDE-Editors dar. Aus der Auswertung von den Daten der Gruppe C2 scheint erkennbar zu sein:

- dass offenbar entweder die Teilnehmer keinen sinnvollen Anhaltspunkt im Quellcode fanden
- oder trotz des Findens der richtigen Anhaltspunkte und der Sichtung von relevanten Code-Fragmenten, die fehlerhafte Objektinitialisierung allem Anschein nach übersehen worden ist. Als Folge hiervon sei vergeblich in noch tieferen Schichten der Fehler gesucht worden.

Beide Annahmen kann man exemplarisch anhand der folgenden 2 Datensätzen beobachten:

Gruppe C2, user-E1ML An dem Datensample mit der Sitzungs-ID 'user-E1ML' aus der Gruppe C2 ist zu erkennen, dass der Teilnehmer die richtige Datei 'AboutDialogView' früh gefunden hat, den Fehler aber, trotz erkennbar sehr intensiver Arbeit mit dieser Datei, nicht als solches entlarven schien. Siehe Abbildung 7 und 8 für die Visualisierung des Navigationsverhaltens.

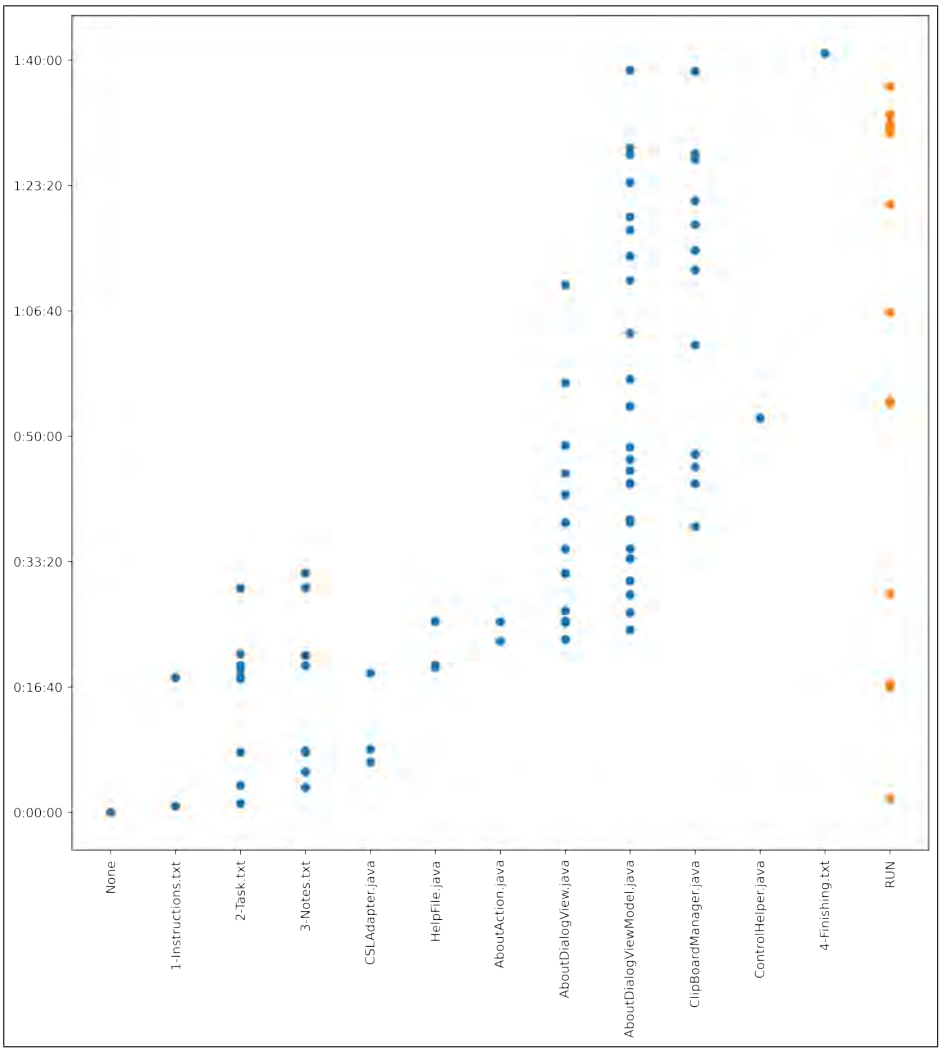

Abbildung 7: user-E1ML

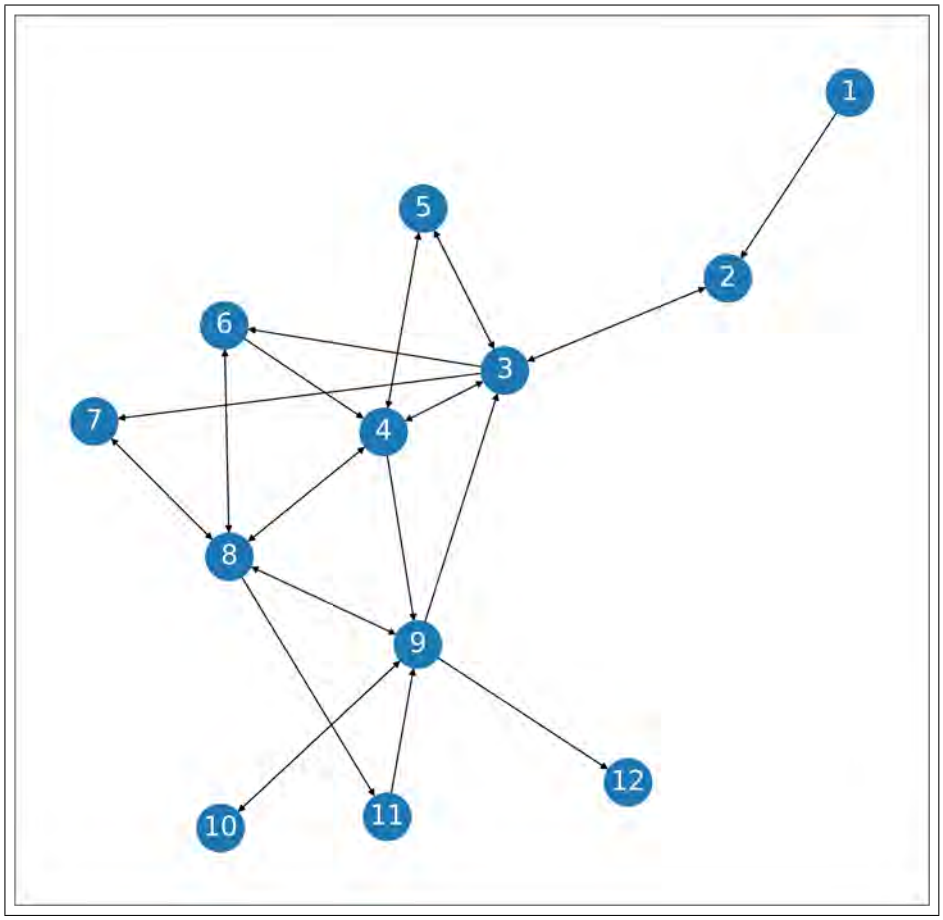

Abbildung 8:

	user-E1ML		
1	None	2	1-Instructions.txt
3	2-Task.txt	4	3-Notes.txt
5	CSLAdapter.java	6	HelpFile.java
7	AboutAction.java	8	AboutDialogView.java
9	AboutDialogViewModel.java	10	ClipBoardManager.java
11	ControlHelper.java	12	4-Finishing.txt

Es sticht hervor, dass es sehr viele Navigationen gab, diese aber innerhalb weniger Dateien navigiert wurde. Dies ist der Abbildung 8 ablesbar.

Zusätzlich kann gedeutet werden, dass die Navigationspfade scheinbar zyklisch sind. Der Teilnehmer versuchte offenbar durch hin und her springen zwischen den Dateien Verständnis zu erlangen. Besonderes interessant sind

die Knoten 11 ‘ControlHelper’ und 10 ‘ClipBoardManager’. Anhand der Abbildung 7 sieht man, dass ab circa Minute 33 der Teilnehmer sich hierhin stetig verläuft und von dort aus in immer weiter tiefer gelegene Abhängigkeiten navigiert. Dennoch kehrt der Teilnehmer immer wieder zu der Datei ‘AboutDialogView’ zurück. Weiterhin ist zu beobachten, dass ab Minute 16 der Entwickler sehr intensiv und ununterbrochen an der Aufgabe arbeitet. Insgesamt dauerte die Sitzung 1:40:55 und hatte 80 Navigationen. In der Endumfrage gab der Nutzer an Autobookmarks wenig genutzt zu haben.

Gruppe C2, user-GFDL Nun zu einem Beispiel für einen Teilnehmer der Gruppe C2 mit der ID ‘user-GFDL’ offensichtlich keinen sinnvollen Anhaltspunkt finden konnte. Die Visualisierungen des Navigationsverhaltens vom Teilnehmer sind auf der Abbildung 10 und 9 dargestellt.

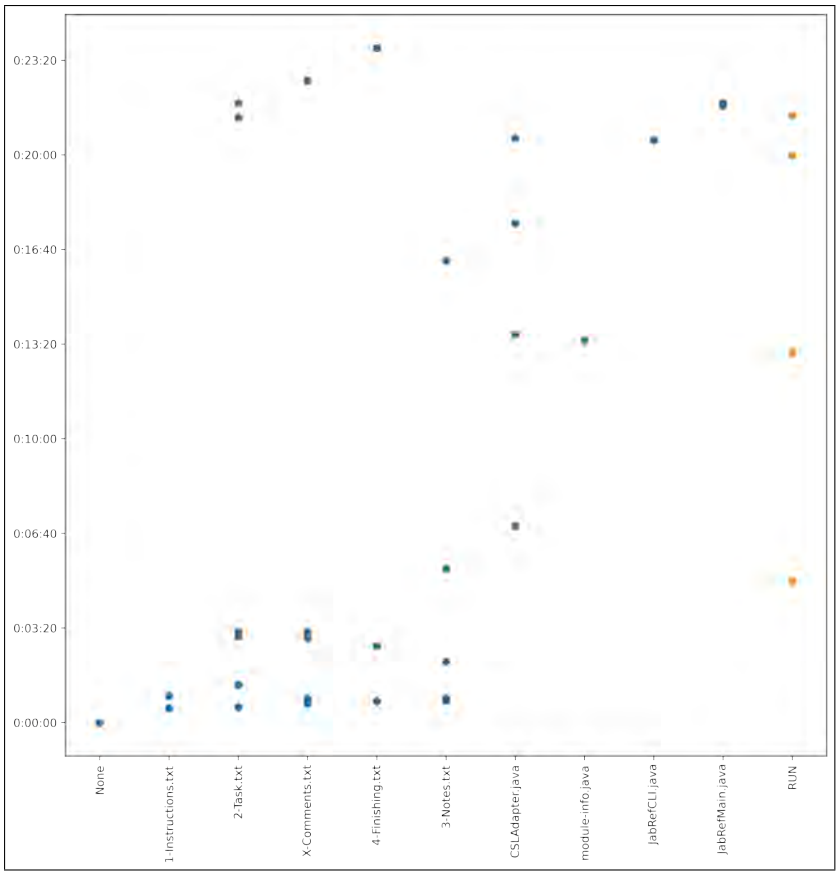

Abbildung 9: user-GFDL

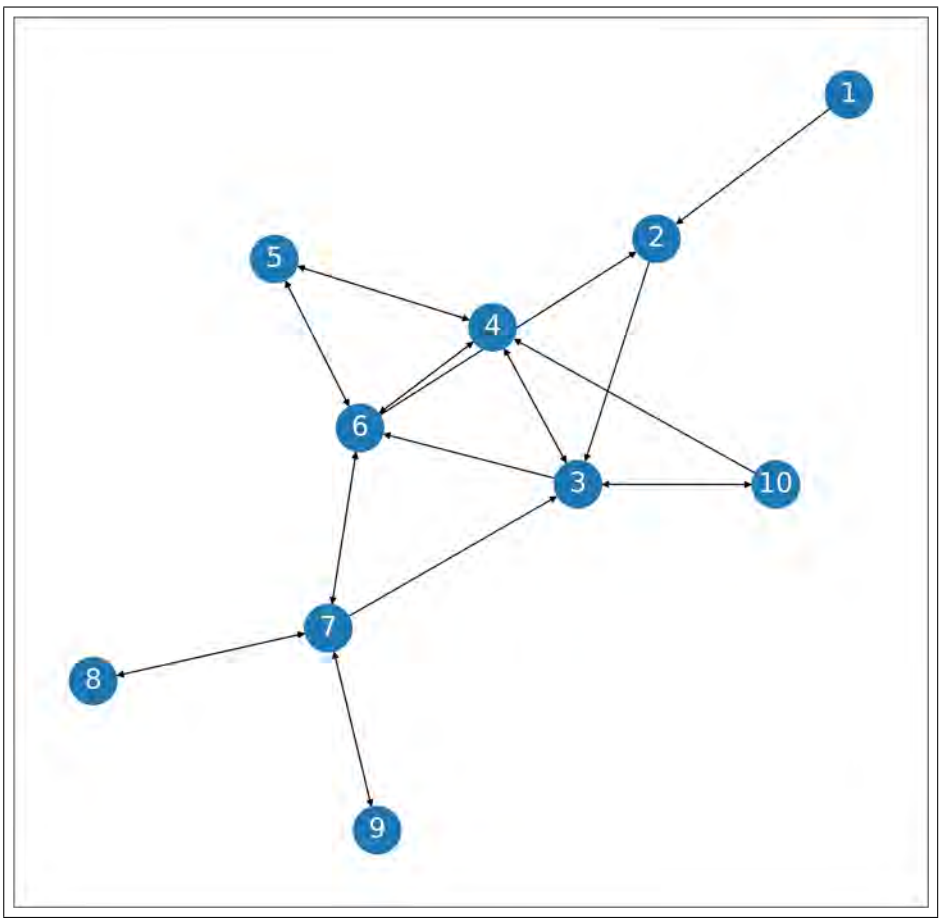

Abbildung 10:

	user-GFDL		
1	None: 1	2	1-Instructions.txt
3	2-Task.txt	4	X-Comments.txt
5	4-Finishing.txt	6	3-Notes.txt
7	CSLAdapter.java	8	module-info.java
9	JabRefCLI.java	10	JabRefMain.java

Der Teilnehmer hinterließ als Kommentar “I’m pretty rusty with Eclipse and Java. No idea where to start. Sorry”. Auf den Darstellungen ist zu sehen, dass der sich zwar über 20 Minuten Zeit genommen hat, aber offensichtlich keinen Einstiegspunkt gefunden hatte, um an der Aufgabe zu arbeiten. Keine der besuchten Quell-Codedateien erscheinen für die Aufgabenstellung von Relevanz. Nach den demografischen Angaben handelt es sich hier um einen Studenten zwischen 26-35 Jahre alt und weniger als 3 Jahren Programmiererfahrung.

Weiterhin lässt sich davon ausgehen, dass es sich bei der Gruppe C2 um überwiegend unerfahrene Entwickler handelt, siehe hierzu Tabelle 11.

Gruppe C3 Zwar liegt der Fokus dieser Arbeit nicht auf der Auswertung des individuellen Navigationsverhaltens, dennoch waren bei der Datensichtung Navigationsmuster erkennbar, welche sich bei anderen Teilnehmern genauso wiederfanden. Einige Teilnehmer hatten scheinbar gezielt Dateien gesichtet, die mit 'About' begannen. Ebenfalls scheint es als, wenn diese Teilnehmer nicht dazu neigten den Abhängigkeiten der gesichteten Dateien zu folgen. Weitergehend wird die Gruppe C3 Kapitel 5 im Vergleich zu der Gruppe D ausgewertet. Ein solches Navigationsmuster kann Anhand der folgenden zwei Datensamples abgelesen werden.

Gruppe C3, user-P2CQ Der Studienteilnehmer brauchte 44 Navigationen um die Aufgabe zu lösen und dauerte 25 Minuten. Dies entspricht bezüglich der Bearbeitungszeit etwa dem Durchschnitt, die Navigationsanzahl weichte dennoch von dem Durchschnittswert ab. Nach Sichtung der Erläuterungen und Aufgabenstellung sowie den Anweisungen sichtete der Nutzer die Datei 'UpdateTimeStampListenerTest.java' in welche, er nicht mehr zurückkehrte. Hiernach wurden innerhalb etwa einer Minute vier Dateien geöffnet, welche 'About' im Namen tragen und im Ordner 'src\main\java\org\jabref\gui\help' lagen. Darauf wurden mit diesen vier Dateien abwechselnd mehrfach gesichtet, bis der Fehler gefunden und behoben wurde. Zum Abschluss hinterließ der Teilnehmer das Kommentar "Since I have almost never worked in Eclipse before, the navigation felt more demanding than the actual task". Besonders interessant ist die Tatsache, dass der Entwickler sehr fokussiert an den 4 Dateien gearbeitet hat in denen, er den Fehler vermutet hatte. Der Teilnehmer gab an Autobookmarks wenig benutzt zu haben und gab sich bezüglich des Konzeptes der Autobookmarks unentschlossen.

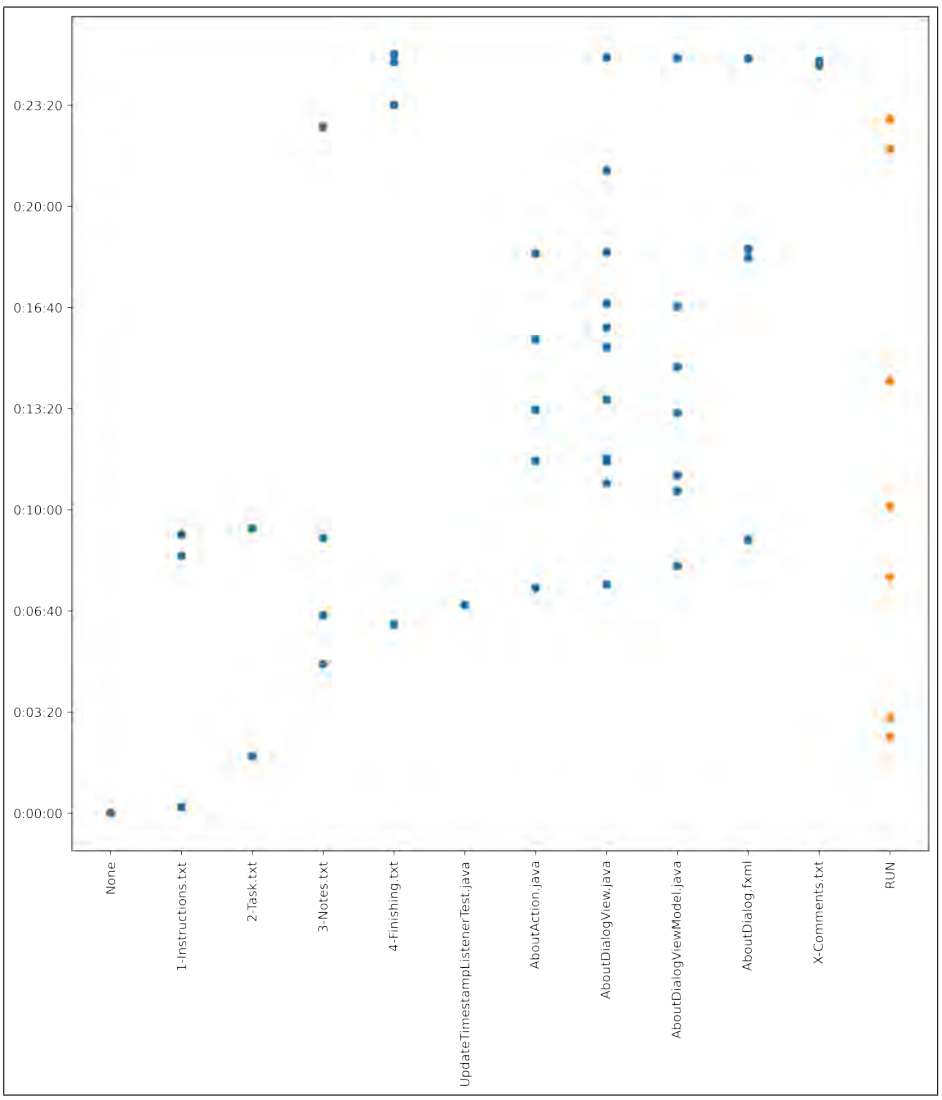

Abbildung 11: user-P2CQ

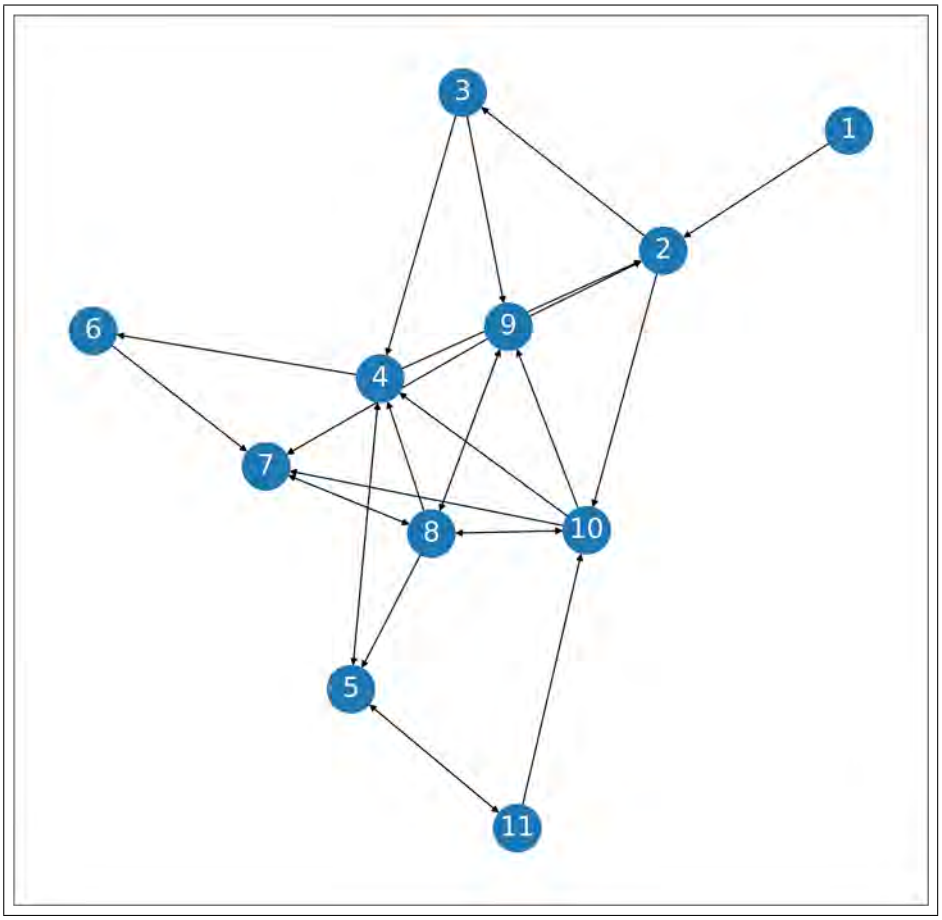

Abbildung 12:

	user-P2CQ		
1	None: 1	2	1-Instructions.txt
3	2-Task.txt	4	3-Notes.txt
5	4-Finishing.txt	6	UpdateTimestampListenerTest.java
7	AboutAction.java	8	AboutDialogView.java
9	AboutDialogViewModel.java	10	AboutDialog.fxml
11	X-Comments.txt		

Gruppe C3, user-CUAX In dieser Teilnahme wurden für die Lösung der Aufgabe 25 Navigationen gebraucht. Die Sitzung dauerte etwa 28 Minuten. Anhand der geringen Anzahl an Knoten im Navigationsgrafen ist ersichtlich, dass der Teilnehmer nur sehr wenige verschiedene Dateien benötigte. Nach Kenntnisnahme der Aufgabenbeschreibung und Anweisungen verschaffte sich der Teilnehmer vermutlich einen Überblick über die drei Dateien 'AboutAction.java' 'AboutDialogView.java' 'AboutDialogViewModel.java'. In die Datei

‘AboutAction.java’ kehrte der Nutzer nicht mehr zurück und fokussierte seine Aufmerksamkeit auf die übrigen zwei, bis der Programmierfehler gefunden wurde. Der Entwickler gab in der Endumfrage an keine Autobookmarks genutzt zu haben und fand das Konzept nicht überzeugend.

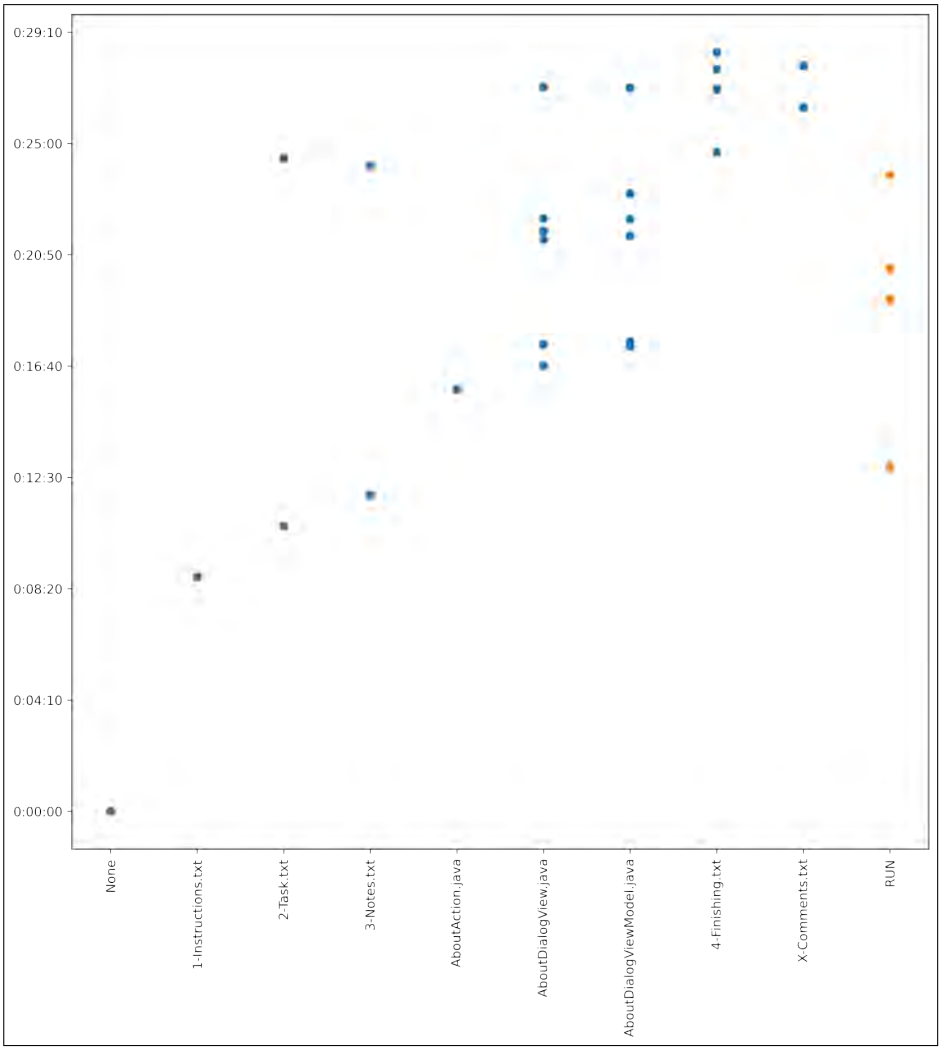

Abbildung 13: user-CUAX

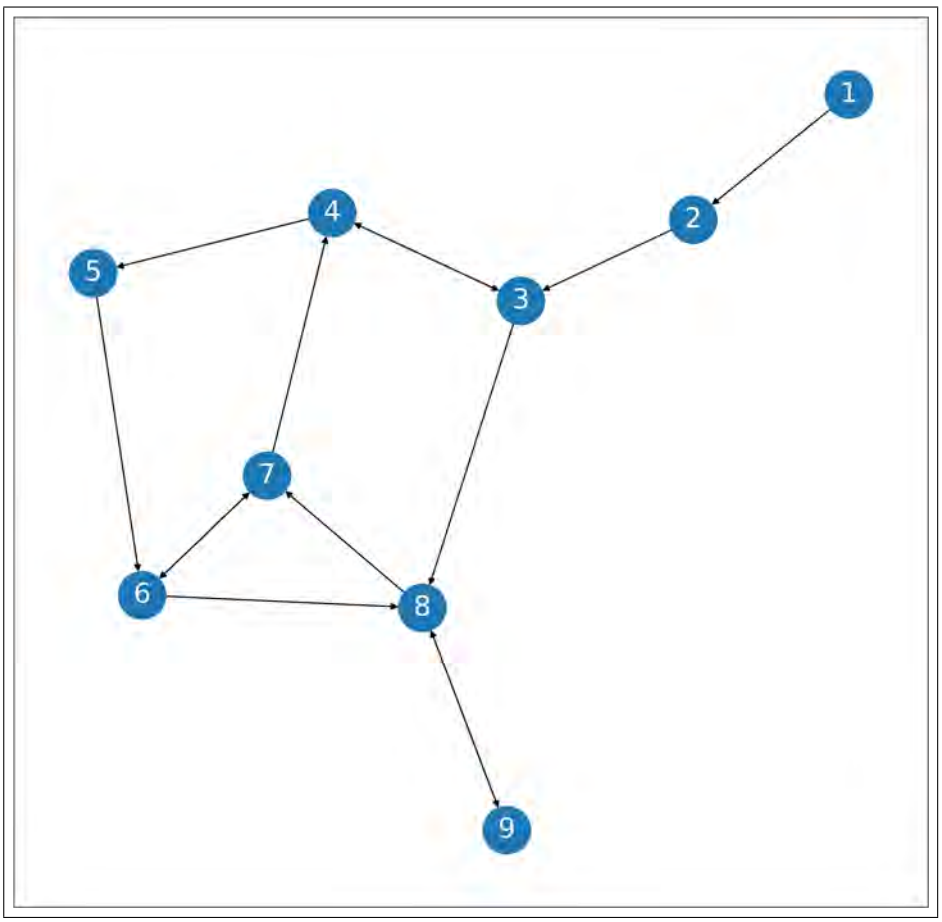

Abbildung 14:

	user-CUAX		
1	None: 1	2	1-Instructions.txt
3	2-Task.txt	4	3-Notes.txt
5	AboutAction.java	6	AboutDialogView.java
7	AboutDialogViewModel.java	8	4-Finishing.txt
9	X-Comments.txt		

4.1 Erfolgsquote

Die augenscheinlich geringe Erfolgsquote ist vermutlich auf die Komplexität der Aufgabenstellung zurückzuführen. Es war nicht offensichtlich, wie die Aufgabe zu bearbeiten ist, daher mussten Studien-Teilnehmer gewisse Vorerfahrungen mitbringen. Die Aufgabe wurde bewusst für die Studie zu Mimesis so gewählt, dass diese ohne ausreichende Fähigkeiten beim Umgang mit großen Mengen an unbekannten Quell-Code nicht bewältigt werden konnte.

Ein weiterer Grund für die geringe Erfolgsquote kann die allgemeine Unbeliebtheit von Eclipse als IDE bei den Studierenden der Universität-Bremen sein.

4.2 Untersuchung der Fragestellung

In dem Datensatz Autobookmarks der Gruppe C3 wurden durchschnittlich 30 Navigationen gebraucht (Standardabweichung von 22 Navigationen) um die Aufgabe zu lösen. In der Mimesis Studie Gruppe D, waren es durchschnittlich 30 (Standardabweichung von 13 Navigationen). Für genaue Aufschlüsselung der Kennzahlen siehe Tabelle 4 und 5. Aus der Gruppe C3 Sample 'user-8NK9' scheint auf den ersten Blick ein Ausreißer zu sein, da die Anzahl an getätigten Navigationen höher ist, als bei allen anderen Samples der Gruppen C3 und D, später gehen wir auf dieses Datensample ein.

Sitzungs-ID	**Navigationen**	**Dauer**
09bee503-f95a-4830-959f-fb3731688444	54	0:36:46
22bfb949-2e0a-4bdf-8057-548e41af3f92	56	1:32:21
59620d75-e225-48db-bd9a-e60c7a156ccb	31	1:14:05
67d55b11-5162-4aef-bf39-89eb1aedbe30	25	0:13:45
749cdc6f-5239-4902-8026-7ab3e1d4362b	25	0:33:36
a4e3dfe5-27d7-491c-8e8a-bf36bdc397d1	21	0:25:15
c77a0437-91a7-4fbf-b616-509afcb855c9	23	0:19:25
14c6b31b-69e6-4410-b926-819104a8fdfd	26	0:48:29
42be987d-7481-48a2-b262-6b7b44dc0742	13	0:18:50
525120c1-043b-41ba-937e-7a5ac142aafd	22	0:29:47
5d25db63-210c-4d1d-b3c1-38563daf638e	36	0:23:43
7b3b5005-aa91-4f80-afb8-660be2e5a917	41	0:33:35
fb1b1ab7-93b7-4b5b-a4c1-9b541e01bbb9	16	0:27:57

Tabelle 4: Gruppe D, Kennzahlen der jeweiligen Arbeitssitzung

Sitzungs-ID	**Navigationen**	**Dauer**
user-Q139	9	0:33:12
user-Q265	15	0:14:42
user-8NK9	78	0:54:21
user-CUAX	25	0:28:25
user-EIV0	24	0:24:44
user-L193	18	0:13:32
user-P2CQ	44	0:25:01

Tabelle 5: Gruppe C3, Kennzahlen der Arbeitssitzung

Die ungünstige Kombination aus weniger erfolgreichen Teilnehmern und Teilnehmer mit geringer Erfahrung gegenüber den augenscheinlich im Mittel deutlich erfahreneren Teilnehmern bei der Anwendungsentwicklung der MIMESIS Studie lässt keine trennscharfe Unterscheidung zwischen den Gruppen mit und ohne Autobookmark zu. Die hohe Standardabweichung und Variation zwischen den einzelnen Samples zeigen, dass keine Aussage bezüglich der notwendigen Navigationsanzahl anhand dieser Daten möglich ist. Dennoch kann man ausschließen, dass das Autobookmark Plugin die allgemeine Navigationsanzahl beim Bearbeiten einer Programmieraufgabe verringern könnte. Dies hätte sich sonst in den Daten zumindest tendenziell abzeichnen müssen. Betrachtet man den Scatterplot Abbildung 15, auf dem Gruppe C3, D gemeinsam dargestellt sind, so sind keinerlei Cluster zu erkennen, an denen man Gruppe C3 und D unterscheiden könnte, was ebenfalls gegen den Einfluss auf die Navigationsanzahl der Autobookmarks spricht.

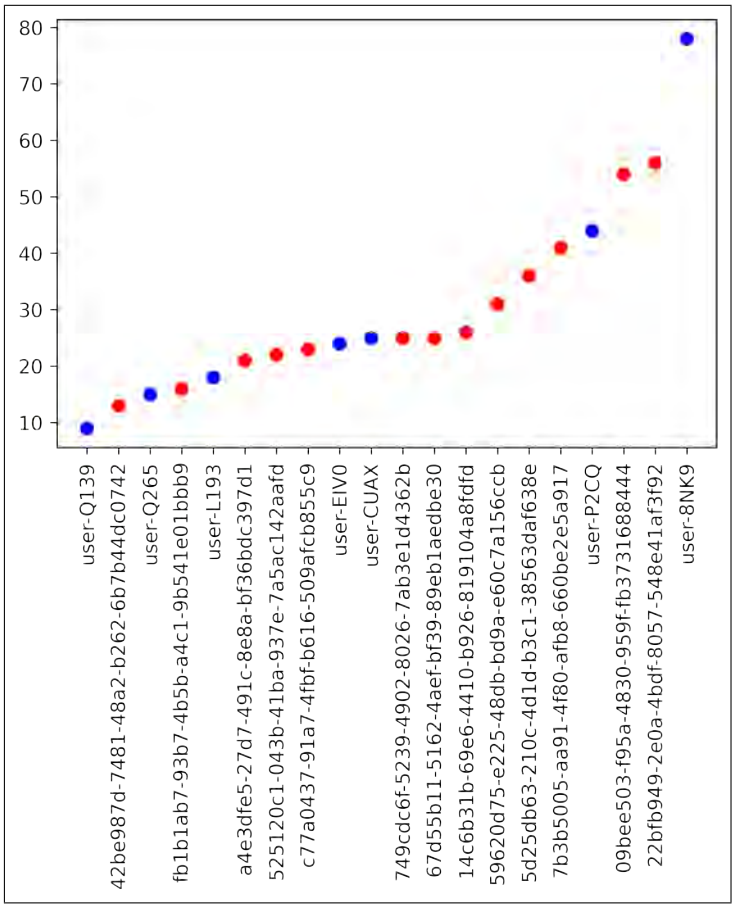

Abbildung 15: Scatterplot der insgesamt benötigten Navigationen für die Bearbeitung der Aufgabe von der Gruppe C3 und D

4.3 End-Umfrage-Auswertung

Betrachtet man die Tabelle 6 so sieht man, dass Entwickler, welche die Programmieraufgabe lösten, die Autobookmarks gar nicht bis wenig nutzten. Die geringe Nutzung der Autobookmarks durch die Entwickler könnte ein weiteres Indiz dafür sein, dass das Konzept der automatischen Bookmarksetzung in dieser Form nicht geeignet ist, um Entwickler beim Navigieren zu unterstützen. Obendrauf gaben nur zwei davon an, dass sich hiermit unnötige Navigationen vermeiden lassen.

user-Q139	nein	vielleicht	vielleicht
user-P2CQ	wenig	vielleicht	vielleicht
user-Q265	nein	vielleicht	vielleicht
user-EIV0	ja	ja	ja
user-CUAX	wenig	nein	vielleicht
user-8NK9	nein	vielleicht	vielleicht
user-L193	nein	ja	ja

Tabelle 6: Angaben bei der End-Umfrage von der Gruppe C3 zu den Fragen: 1. "Haben sie Autobookmarks benutzt?" 2. "Finden sie Autobookmarks können dabei helfen unnötige Navigationen zu vermeiden?" 3. "ist das Konzept der Autobookmarks hilfreich beim wiederauffinden von Codestellen?" Folgende 3 Angaben waren möglich: Nein, vielleicht und ja

Die Angaben der Teilnehmer in dieser Studie sind dennoch eine Steigerung im Vergleich zu der Studie von Moritz Weinig. Hier gaben fast alle Entwickler an, dass die Autobookmarks nicht hilfreich waren und nicht genutzt wurden. Siehe Tabelle 7 und 8 zum Vergleich.

Angabe	Anzahl
1	3
2	5
3	0
4	0
5	0

Tabelle 7: Angaben bei der End-Umfrage aus der Studie von Moritz Weinig "How intensivly did you use the automatic bookmarks? (1: little, 5: very much)?"[7]

Angabe	Anzahl
1	1
2	4
3	3
4	0
5	0

Tabelle 8: Angaben bei der End-umfrage aus der Studie von Moritz Weinig "To what extent did the automatic bookmarks support you in fulfilling the assignment? (1: not at all, 5: very much)"[7]

Auch wenn diese Studie nicht, mit der von Moritz Weinig vergleichbar ist, kann man dennoch zumindest ausschließen, dass die Rationalisierungen und Verbesserungen im Rahmen dieser Arbeit, das Autobookmark Tool weniger nützlich für die Entwickler gemacht haben.

5 Interpretation

5.1 Vergleich der Gruppe C3 und D

Der Tabelle 3 nach stehen 7 Samples der Gruppe C3, also die Menge erfolgreich Teilgenommenen an der Autobookmarks-Studie, für die Evaluierung des Autobookmarks Konzeptes zur Verfügung. Dem gegenüber stehen aus Datensammlung der Mimesis-Studie 13 Aufzeichnungsdateien von Teilnehmern mit erfolgreich bearbeiteter Programmieraufgabe und den entsprechenden demographischen Umfrageergebnissen vor, referenziert als Gruppe D.

Die Selbsteinschätzung bei den demographischen Angaben der Gruppe C2, C3 und D zeigten einige Unterschiede:

Frage	Gruppe	ø
Avg: How profound is your knowledge of the Eclipse IDE?	C3	2
	C2	2.37
	D	2.69
Avg: How profound is your knowledge of Application Development?	C3	1.71
	C2	2.64
	D	3.92
Avg: How would you describe your programming experience compared to experts with 20 years of practical experience?	C3	3.43
	C2	3.27
	D	3.23
Avg: How would you describe your programming experience overall?	C3	2.43
	C2	2.55
	D	3.92
Avg: How profound is your knowledge of GUI Development?	C3	2.43
	C2	2.55
	D	3.92

Tabelle 9: Unterschiede zwischen Gruppe C3, C2 und D anhand von selektierten demographischen Angaben dargestellt. Zur Errinnerung C3 für erfolgreiche, C2 für erfolgslose und D für die Teilnahmen an der Mimesis-Studie. Bei den demographischen Fragebogen konnten befragte ihre Kenntisse mit der Scala 1: für No experience und 5: für Very experienced mitteilen

Bei den Teilnehmern der Gruppe C3 handelt es sich nicht um klassische Anwendungsentwickler. Diese Teilnehmer schätzten Fähigkeiten bei der Anwendungsentwicklung, die Kenntnisse der Eclipse-IDE und die allgemeine Programmier-Erfahrung als gering ein. Über den Erfahrungswert der Gruppe C3 lässt sich keine genaue Aussage treffen. Die Vermutung liegt nahe, dass es sich um Entwickler verschiedenerer Erfahrungslevel handelt, wie die folgende Tabelle 11 bestätigt. Diese aber in anderen Anwendungsgebieten als die Anwendungsentwicklung tätig sind.

Sitzungs-ID	Navigationen	Dauer
user-3J3G	30	0:53:35
user-3VWB	9	0:17:21
user-7A08	10	0:17:51
user-AO2R	19	0:08:13
user-C4QD	28	6:04:06
user-D75G	36	0:19:58
user-E1ML	80	1:40:55
user-FCL0	18	0:20:16
user-GFDL	34	0:23:46
user-IIMQ	58	6:54:21
user-UD5B	31	0:23:40

Tabelle 10: Gruppe C2, Kennzahlen der jeweiligen Arbeitssitzung

Gruppe	unter 3 Jahre	>3
C2	10	1
C3	4	3
D	6	7

Tabelle 11: Unterschiede zwischen Gruppe C3, C2 und D bezüglich der angegebenen Programmiererfahrung in größeren Projekten

5.2 Navigationsverhalten

Bei der Auswertung der Daten war aufgefallen, dass die Anzahl an geöffneten Dateien bei den beiden Gruppen im Vergleich unterschiedlich war. So wurden in der Mimesis Teilnehmer Gruppe D durchschnittlich 13 (mit einer Standartabweichung von 6) unterschiedliche Dateien geöffnet und in der Gruppe C3 10 (mit einer Standartabweichung von 3).

Da die Navigationsanzahl der beiden Gruppen im Durchschnitt gleich sind, aber bei der Gruppe C3 weniger verschiedene Dateien gesichtet werden mussten, um zu einer Lösung zukommen, könnte dies darauf hinweisen, dass Autobookmark-Nutzer schneller irreführende Navigationspfade abbrechen. Anhand des Scatterplots 16 ist erkennbar, dass Teilnehmer der Autobookmarks Gruppe C3 weniger verschiedene Dateien öffneten.

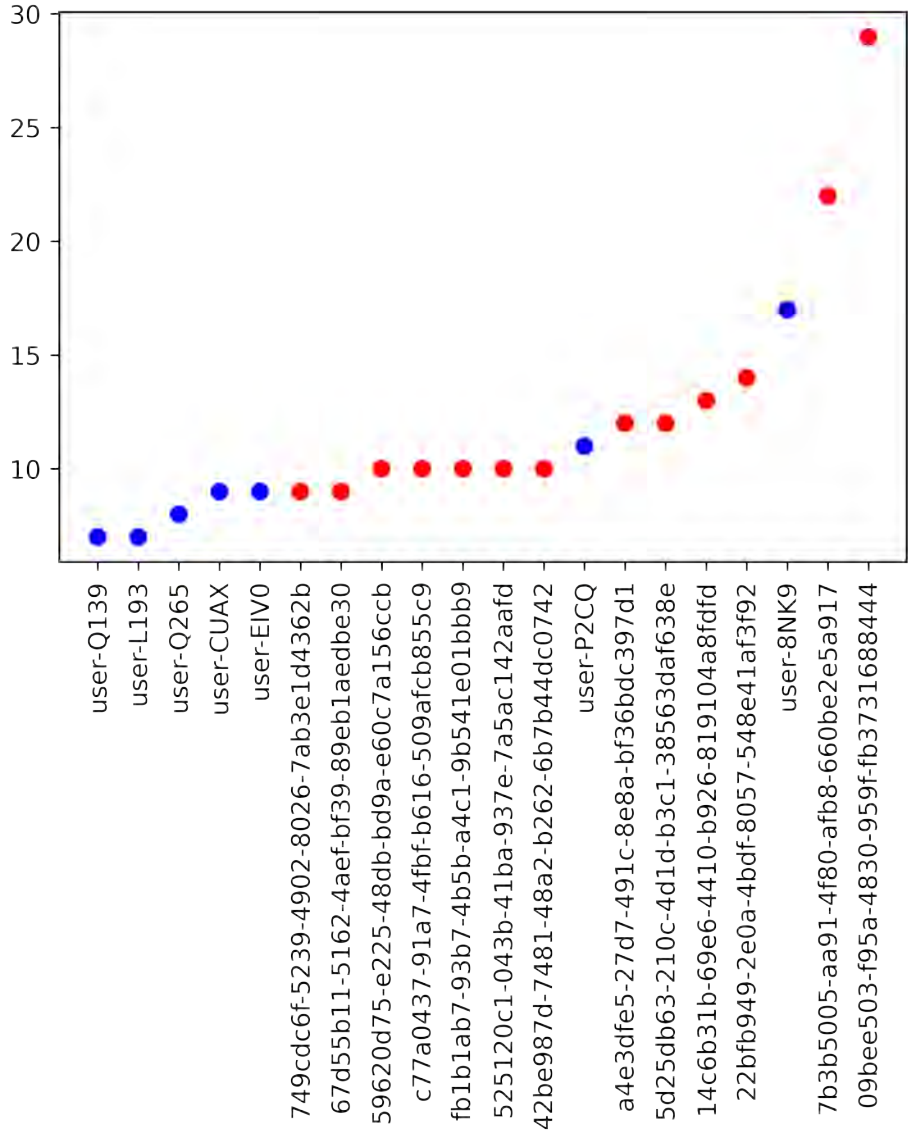

Abbildung 16: Die Anzahl an geöffneten unique Dateien der Gruppe C3 und D

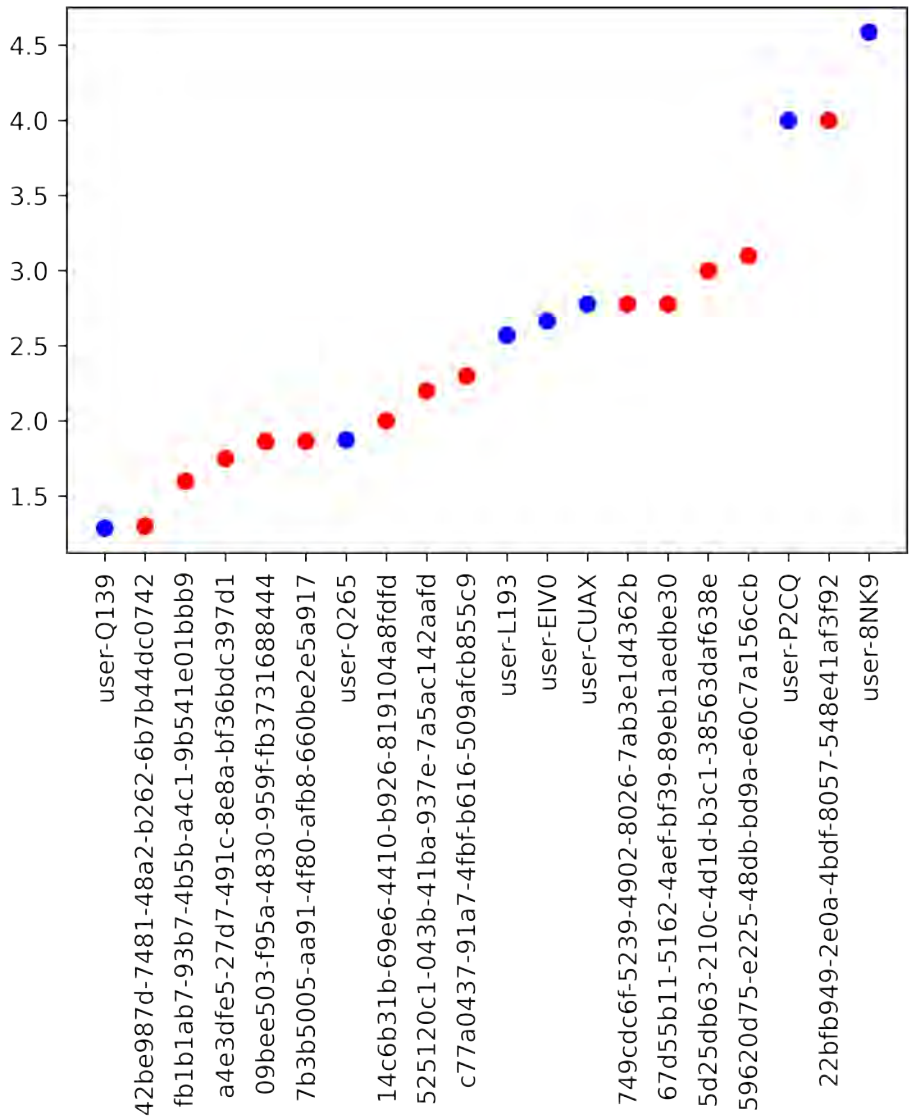

Abbildung 17: Die Anzahl wie oft zu bereits gesichteten Dateien zurückgekehrt wurde der Gruppe C3 und D

Besonders auffällig ist hier, dass der Teilnehmer mit der Sitzungs-ID ‘user-8NK9’, welcher mit insgesamt 78 Navigationen am meisten Navigationen benötigte (siehe Tabelle 5) nur 17 Dateien geöffnet hatte und in diese durchschnittlich 4 mal zurückgekehrt ist. Die durchschnittliche Anzahl des Zurückkehrens in bereits gesichtete Dateien. Dies lässt vermuten, dass Teilnehmer der Gruppe C3 öfters in bereits gesichtete Dateien zurückgekehrten sind, als die aus der Gruppe D. Betrachtet man den Scatterplot 17 lässt sich hier kein

so eindeutiges Cluster zwischen den beiden Gruppen erkennen.

Die Anzahl an Dateien, die gesichtet werden muss, um eine Programmieraufgabe erfolgreich zu bearbeiten, kann als Faktor für die Navigations-Performance gesehen werden. Denn hierdurch, wird weniger Information zur Kenntnis genommen. Dies wird aber nicht aus der Anzahl der Navigationen ersichtlich.

Die ursprüngliche Annahme, dass die Navigationsanzahl eine geeignete Metrik ist, um die Navigations-Performance zu messen ist daher so nicht haltbar.

Vielmehr muss die Anzahl an gesichteten Dateien in Relation zu der Navigationsanzahl für die Bewertung der Navigations-Performance betrachtet werden. Den eine geringere Anzahl an Dateien, die ein Entwickler zur Kenntnis nimmt, zeigt meiner Meinung nach, dass der Entwickler sehr fokussiert arbeitet und sich nicht zu Navigationen in irrelevante Dateien verleiten lässt.

Ohne weitere Datensamples, ist es allerdings nicht möglich diese Beobachtung zu erklären. Daher wird die Auswertung der Daten an dieser Stelle auch beendet.

6 Threads of Validity

An dieser Stelle muss darauf hingewiesen werden,

1. dass es sich bei den beiden Gruppen C3 und D um unterschiedliche Teilnehmer-Kreise handelt.
2. es zu wenige Datensamples gibt, um über mögliche Einflüsse des Autobookmarksplugins endgültige Schlüsse zu ziehen und daher es denkbar wäre, dass sich das Ergebnis durch weitere Teilnehmer verschiebt.
3. Auch ist nicht ausgeschlossen, dass die technischen Unterschiede der beiden Studien das Ergebnis verzehrten.
4. Ebenso ist es möglich, dass die Einladungstexte zur Teilnahme an der Studie zum Autobookmarktool, geringfügig mehr über die Aufgabenstellung verriet, als den der Mimesis Studie.
5. Weiterhin ist nicht ausgeschlossen, dass die Aufgabenstellung nicht geeignet war, um das Navigationsverhalten zu untersuchen.
6. Die Nutzer mit der Sitzung-ID ‘user-L193’ und ‘user-P2CQ’ trafen zu beginn der Teilnahme auf technische Probleme, daher wurde Eclipse-

IDE neu gestartet. Hierbei ist nicht ausgeschlossen, dass es zu Datenverlust in den Autobookmarks-Aufzeichnungsdaten kommen konnte.

7. Den Studien-Teilnehmern der Autobookmarkstudie wurde das Autobookmarkplugin von vornherein als Plugin vorgestellt, welches helfen soll schneller zwischen oft besuchten Code-Stellen zu navigieren. Siehe Abbildung 2. Dadurch konnten die Teilnehmer schlussfolgern, dass die getätigten Navigationen durch den Quellcode für diese Arbeit von Relevanz sein werden. Den Teilnehmern der Mimesis Studie wurde aber nicht bewusst gemacht, dass deren Navigationsverhalten ausgewertet werden wird. Wie in der Einleitung bereits erläutert wurde, könnte es beim fehlenden Bewusstsein zuverlässig über Nützlichkeit von Navigationen zu entscheiden, sich um einen "blind spot" bei dem heuristischen Entscheidungsprozess[6] der Entwickler handeln. Wenn dies so wäre, erscheint es möglich, dass die in dieser Arbeit festgestellten Unterschiede, bezüglich der Navigationsverhaltens alleine darauf zurückzuführen sind, dass die Mimesis-Teilnehmer keine Hinweise auf eine mögliche Auswertung des Navigationsverhaltens bekommen hatten.

7 Fazit

Mit der Navigationsanzahl als Metrik für Navigations-Performance konnte kein Einfluss des Autobookmarks-Konzeptes auf der Navigationsanzahl nachgewiesen werden. Für weitergehende Analysen gab es trotz hoher Teilnehmerzahlen zu wenig erfolgreiche Abgaben der Programmieraufgabe. Von Entwicklern, die zu einer Lösung gekommen sind, wurden Autobookmarks selten gebraucht und das Konzept war in dieser Form nicht überzeugend genug, um Mehrheit der Teilnehmer von dessen Nutzen zu überzeugen.

Auch wurde argumentiert, dass die Navigationsanzahl nicht ausreicht, um den Aufwand der Entwickler, sich ein Verständnis von unbekanntem Code zu machen, zumessen. Denn es war aufgefallen, dass die Autobookmarks-Nutzer dazu tendierten innerhalb eines kleineren Kreises zu navigieren, und zwar als Nutzer ohne das Plugin. Als Folge wurde vorgeschlagen, dass die Anzahl der gesichteten Dateien mitberücksichtigt werden muss, um das Navigationsverhalten bewerten zu können.

Wertet man die Daten der Anzahl der gesichteten Dateien aus, so wurde die Tendenz deutlich, dass Nutzer des Autobookmarksplugins innerhalb einer geringen Anzahl an Dateien navigierten. Auch wenn die für diese Arbeit gesammelten Daten keinen endgültigen Schluss zulassen, deutet dies auf

einen Positiven Einfluss Autobookmarksplugins hin. Hier müsste allerdings eine weitere Iteration der Datensammlung oder eine weitere größer angelegte Studie erfolgen.

Mit dieser Arbeit wurde gezeigt, dass sich Remote-Studien mit komplexen Programmieraufgaben mithilfe des Mimesis Frameworks durchführen lassen. Diese Erkenntnisse können als Grundlage für weitere Untersuchungen herangezogen werden. Eine solche Studie ist im Vergleich zu einer Laborstudie hoch-skalierbar und kann nahezu unbeaufsichtigt durchgeführt werden. Die Wiederaufnahme einer Arbeitssitzung ist ebenfalls durch das Mimesis Framework möglich, sodass auch das Arbeiten an einer deutlich komplexeren und zeitaufwendigeren Programmieraufgabe erfasst werden könnte. Es wurde gezeigt, dass mithilfe einer zusätzlichen Webanwendung eines Setup-Skriptes und eines RDP-Gateways, die Studienteilnahme mittels Webbrowser erfolgen kann. Dies erspart jeglichen Aufwand beim Einrichten der Programmier-Umgebung für die Teilnehmer.

Die vorgestellte Vorgehensweise ermöglicht ebenfalls die systematische und voll automatisierte Auswertung des Navigationsverhaltens. Mit der Darstellung der Navigationssequenzen und Navigationspfade durch die Navigationsgrafen, lässt sich das Navigationsverhalten auch von langen Arbeitssitzungen sehr übersichtlich veranschaulichen und analysieren. Ein solcher Studienaufbau hat sich zusätzlich als geeignet erwiesen, den Einfluss von Tools und Plugins auf die Arbeitsweise der Entwickler beim Umgang mit unbekanntem Code zu untersuchen, und zu messen.

Ergänzend wäre es relevant, die Aufzeichnungen der erfolglosen Teilnehmer zu untersuchen. Hier könnte eine Analyse der Navigationspfade und Sequenzen dieser Nutzer erfolgen, um deren Vorgehensweise mit den erfolgreichen Teilnehmern gegenüberzustellen. Auf diese Weise könnten Probleme ermittelt werden, um Anhaltspunkte für das Scheitern der Teilnehmer zu finden.

Abschließend lässt sich feststellen, dass die Integration der einzelnen Komponenten für den Studienaufbau zur Ermöglichung des reibungslosen Betriebes, die zeitaufwendigste und komplexeste Aufgabe dieser Arbeit war. Daher ist zu empfehlen, weitere Remote-Studien mittels Eclipse-Aufzeichnung-Frameworks auf den Abläufen und dem Aufbau dieser Studie aufzubauen.

Literatur

[1] R. Fjelstad und W. Hamlen, „Applications program maintenance study: Report to our respondents," *Proceedings Guide*, Jg. 48, S. 13–27, 1979.

[2] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan und S. Li, „Measuring Program Comprehension: A Large-Scale Field Study with Professionals," en, *IEEE Transactions on Software Engineering*, Jg. 44, Nr. 10, S. 951–976, Okt. 2018, ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2017.2734091. Adresse: https://ieeexplore.ieee.org/document/7997917/ (besucht am 14. 04. 2021).

[3] D. Piorkowski, A. Z. Henley, T. Nabi, S. D. Fleming, C. Scaffidi und M. Burnett, „Foraging and navigations, fundamentally: developers' predictions of value and cost," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Ser. FSE 2016, New York, NY, USA: Association for Computing Machinery, Nov. 2016, S. 97–108, ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950302. Adresse: https://doi.org/10.1145/2950290.2950302 (besucht am 14. 04. 2021).

[4] A. J. Ko, B. A. Myers, M. J. Coblenz und H. H. Aung, „An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks," *IEEE Transactions on Software Engineering*, Jg. 32, Nr. 12, S. 971–987, Dez. 2006, Conference Name: IEEE Transactions on Software Engineering, ISSN: 1939-3520. DOI: 10.1109/TSE.2006.116.

[5] A. Guzzi, L. Hattori, M. Lanza, M. Pinzger und A. v Deursen, „Collective Code Bookmarks for Program Comprehension," in *2011 IEEE 19th International Conference on Program Comprehension*, ISSN: 1092-8138, Juni 2011, S. 101–110. DOI: 10.1109/ICPC.2011.19.

[6] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos und Y. Zhuang, „It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots," in *Proceedings of the 30th Annual Computer Security Applications Conference*, Ser. ACSAC '14, New York, NY, USA: Association for Computing Machinery, Dez. 2014, S. 296–305, ISBN: 978-1-4503-3005-3. DOI: 10.1145/2664243.2664254. Adresse: https://doi.org/10.1145/2664243.2664254 (besucht am 30. 11. 2021).

[7] M. Weinig, „Automatische Lesezeichen zur Unterstützung des Programmverstehens," en, Magisterarb., University of Bremen, Dez. 2020. Adresse: https://www.szi.uni-bremen.de/wp-content/uploads/2021/02/thesis_weinig_online.pdf.

[8] M. Schröer und R. Koschke, „Recording, Visualising and Understanding Developer Programming Behaviour," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, ISSN: 1534-5351, März 2021, S. 561–566. DOI: 10.1109/SANER50967.2021.00066.

[9] M. Kersten und G. Murphy, „Mylar: A degree-of-interest model for IDEs," Jan. 2005, S. 159–168. DOI: 10.1145/1052898.1052912.

8 Appendix

Source-Code: Autobookmarkplugin Der Quellcode ist auf dem Gitlab-server der Universität Bremen hinterlegt. `https://gitlab.informatik.uni-bremen.de/dfgpc/automatic-bookmarks` Zum Abgabezeitpunkt war der letzte Commithash:
0fd1c3c5cd8a60613d29e333e6bff2fe640c48a4

Source-Code: Autobookmark-Survey-App Für die Funktion des Umfrage-Systems muss der Code vom folgenden Repository ausgeführt werden:

Source-Code: Infrastruktur Ebenfalls ist das dort enthaltende Git-Submodul notwendig. Auch abrufbar unter: `https://gitlab.informatik.uni-bremen.de/dfgpc/automaticbookmarks-infrastructure`. Der Commithash von der auf dem Server installierten Version:
6f8c7595a167d58e60f42b1f67c70826ae9273e3

Source-Code: Autobookmark-Survey-Landingpage `https://gitlab.informatik.uni-bremen.de/dfgpc/automaticbookmarks-landingpage`. Die auf dem Studien-Server benutze Version hatte den Commithash:
83c227bb8f0695720f6add08cc19e092b60b927f

Source-Code: Evaluation-Notebook Für die Auswertung der Datensätze benötigt man ein Jupyter-Notebook. Dies ist zu finden unter: `https://gitlab.informatik.uni-bremen.de/dfgpc/automaticbookmarks-evaluation` mit dem Commithash:
08c7a2840f8886edf1448fc1885f3d83e1493d8f Dort sind bereits alle für diese Arbeit verwendeten Daten eingelesen und ausgegeben. Alternativ kann man das Notebook als PDF-Datei herunterladen von:
`https://seafile.zfn.uni-bremen.de/f/92a6b95f74594419bc92/`

Source-Code: Programmieraufgabe Der als Programmieraufgabe verwendete Code mit der Aufgabenstellung ist zu finden unter: `https://seafile.zfn.uni-bremen.de/f/7ca0c0a1fbac4f2cbbe6/`. Checksumme (sha1sum):
3f741729feded62b4fa1066516f4e1ec6a4547ec

Binary: Autobookmarkplugin Die für die Studie verwendete-Version von Autobookmarks-Plugin kann von: `https://seafile.zfn.uni-bremen.de/f/1489d2b6ead247448a7d/` Heruntergeladenen werden. Checksumme: cca041692fab6c09633643d990586fe598568f84 Das Plugin lässt sich einfach in Eclipse als Dropin integrieren. Siehe hierzu die Anleitungen vom Autobookmarkplugin Repository:

`https://gitlab.informatik.uni-bremen.de/dfgpc/automatic-bookmarks`

Datensatz Autobookmarks Die Datensammlung dieser Studie ist auf: `https://seafile.zfn.uni-bremen.de/f/eabdcc0a84544d8ba330/` als Archiv-Datei abrufbar. Checksumme:
74f80a37aaad627c650de78f5c9d61295cf38ced

Datensatz Mimesis Der verwendete Datensatz kann unter `https://seafile.zfn.uni-bremen.de/f/d59c7882275f42159e6f/` gefunden werden. Checksumme:
445470ece78df9674952867803687bf583589f80