Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Aufmerksamkeit in der Software-Wartung: Empirische Untersuchung mit Eye-Tracking

Attention in Software Maintenance:
An Eye-Tracking Study

Masterarbeit

im Studiengang Informatik

von

Maike Ahrens

Prüfer: Prof. Dr. Kurt Schneider

Zweitprüfer: Prof. Dr. Michael Rohs

Betreuer: M.Sc. Melanie Busch und Prof. Dr. Kurt Schneider

Hannover, 2. November 2018

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, 2. November 2018

Maike Ahrens

Zusammenfassung

In Softwareprojekten wird häufig an Programmen weitergearbeitet, die von anderen erstellt wurden. Insbesondere in agilen Projekten ist dies die Regel. Entwickler stehen daher häufig vor der Aufgabe, fremden Code verstehen und sich darin für die Weiterarbeit zurecht finden zu müssen. Diese Arbeit behandelt einen neuartigen Ansatz, um mittels der Weitergabe von Aufmerksamkeit das Finden relevanter Codeabschnitte zu unterstützen. Dazu wurden über Eye-Tracking Blickverteilungen bei der Softwareentwicklung aufgenommen und diese durch eine Heatmap, sowie die Einfärbung der Klassennamen, visualisiert. Um dies zu ermöglichen, wurde eine Umgebung im Browser geschaffen, in der Eye-Tracking bei der Arbeit mit Code eingesetzt werden kann.

In einem Experiment wurde anschließend getestet, inwiefern diese Aufmerksamkeitsdarstellungen Entwicklern während Softwarewartungsarbeiten eine Hilfestellung bieten. Die Ergebnisse zeigen, dass die beiden Darstellungen den Teilnehmern teilweise bei der Orientierung und Suche im Code halfen und sich durch die Heatmap auch die kognitive Last leicht verringerte. Dennoch sahen einige Teilnehmer, insbesondere die erfahreneren Entwickler unter ihnen, die Aufmerksamkeitsrepräsentationen als wenig bis gar nicht hilfreich an. Stattdessen erhöhten die Darstellungen den visuellen Aufwand, verschlechterten die Übersichtlichkeit des Codes und erschwerten das allgemeine Funktionalitätsverständnis, sowie das Finden relevanter Abschnitte.

Attention in Software Maintenance: An Eye-Tracking Study

Abstract

In software projects people often continue working on programs that others created. This is especially the case in agile development. Therefore, developers often need to understand code of others in order to expand, change, fix or review it. This thesis deals with a new approach to transfer attention to support finding relevant pieces of code. For that purpose gaze distributions during software development were recorded via eye-tracking and visualized with heatmaps and the coloring of class names. To make this possible, an environment was built in the browser that enables the recording of eye-tracking data during the work with code.

A following experiment tested whether these attention representations helped developers during software maintenance tasks. Results show that both representations helped some of the participants for orientation and code finding purposes. The heatmaps slightly decreased the cognitive load as well. However, some participants, especially the more experienced developers of them, rated the attention visualizations as barely helpful or not helpful. Instead, the representations increased the visual effort, reduced the code's clarity and made the overall understanding of the program's functionality, as well as finding relevant areas, more difficult.

Inhaltsverzeichnis

1 Einleitung ... 1
1.1 Motivation ... 1
1.2 Lösungsansatz ... 1
1.3 Struktur der Arbeit ... 2
2 Grundlagen zu Eye-Tracking ... 4
3 Verwandte Arbeiten ... 5
3.1 Studien zu Aufgaben in der Software-Wartung ... 6
3.1.1 Eye-Tracking: Code verstehen, editieren und erweitern ... 6
3.1.2 Navigation im Code ... 7
3.1.3 Unerfahrene Entwickler vs. Experten ... 7
3.2 Lenkung von Aufmerksamkeit ... 8
3.2.1 Lenkung von Aufmerksamkeit mittels Eye-Tracking ... 8
3.2.2 Hervorheben einzelner Codeabschnitte ... 9
3.2.3 Filterung taskabhängiger Kontexte ... 10
3.3 Tool zur Aufnahme von Eye-Tracking Daten in Eclipse ... 10
4 Problematik von Eye-Tracking in der Software-Wartung ... 12
4.1 Dynamischer Content ... 12
4.2 Auswirkungen auf die Studie ... 14
5 Asynchrones Teilen von Aufmerksamkeit ... 15
5.1 Arten von Aufmerksamkeitsdaten ... 15
5.2 Mögliche Anwendung in der Software-Wartung ... 16
5.3 Mögliche Aufmerksamkeitsdarstellungen ... 17
5.4 Aufmerksamkeitsdaten- und Darstellungsauswahl für die Studie ... 19
6 Planung der Studie ... 21
6.1 Zielsetzung mittels GQM ... 21
6.2 Forschungsfragen ... 22
6.3 Hypothesen ... 23
6.4 Metriken ... 24
6.5 Experimentobjekte ... 27
6.5.1 Eclipse als Webseite ... 27
6.5.2 Codeauswahl ... 30
6.5.3 Softwarewartungsaufgaben ... 33
6.6 Experimentsubjekte ... 34
6.7 Design des Experiments ... 35

6.8 Aufgabenzuteilung ... 35
7 Aufbereitung der Aufmerksamkeitsdaten ... 37
8 Durchführung der Studie ... 41
8.1 Ablauf der Studie ... 41
8.2 Datenaufnahme ... 42
8.3 Experimentumgebung ... 43
8.4 Wiederholung von Teilnehmerdurchläufen ... 43
8.5 Beobachtungen während der Studie ... 44
9 Analyse und Interpretation ... 45
9.1 Verarbeitung der Rohdaten ... 45
9.2 Formale Hypothesenprüfung ... 45
10 Auswertung und Schlussfolgerung ... 64
10.1 Beantwortung der Forschungsfragen ... 64
10.2 Zusammenfassung der Studienergebnisse ... 66
10.3 Bedrohungen der Validität ... 67
10.3.1 Validität der Konstruktion ... 67
10.3.2 Interne Validität ... 68
10.3.3 Validität der Schlussfolgerung ... 69
10.3.4 Externe Validität ... 69
11 Zusammenfassung und Ausblick ... 71
11.1 Zusammenfassung ... 71
11.2 Ausblick ... 72
A Anhang ... 79
A.1 Erklärung und Fragebögen ... 79
A.2 Aufgabenbeschreibungen und -lösungen ... 85
A.3 Ergebnisdaten ... 89
A.4 Dokumentation der Anwendung zur Heatmapfarbenbestimmung ... 94
A.5 Inhalt der CD ... 96

1 Einleitung

1.1 Motivation

In Softwareprojekten entwickeln Personen häufig an Code und Dokumenten weiter, die Andere erstellt haben. Besonders in agiler Softwareentwicklung ist dies die Regel und wird durch den Ansatz der „collective code ownership“ ausdrücklich gefördert. Dabei darf und soll jeder Entwickler Programmteile editieren, die Andere erstellt haben. Somit stehen Entwickler ständig vor der Aufgabe, unbekannte Software verstehen zu müssen, um diese zu reviewen, zu verändern, zu erweitern oder Fehler darin zu finden. Es wird viel Zeit darauf verwendet, Quellcode, Spezifikationen und andere Softwaredesigndokumente zu lesen und relevante Teile zu finden, um sich darin zur Weiterarbeit zurecht zu finden. Wissen und Expertise kann beim Programmieren jedoch nur schwer verbal transferiert werden. Darüber hinaus findet Arbeit und Kommunikation in der Software-Wartung oft mit zeitlicher oder örtlicher Distanz statt [1]. Nicht immer befinden sich alle Teammitglieder am gleichen Ort oder sogar in der gleichen Zeitzone.

1.2 Lösungsansatz

Um Entwicklern, die an unbekannter Software von Anderen weiterarbeiten, den Einstieg zu erleichtern, wurde in dieser Arbeit ein Ansatz entwickelt, um Aufmerksamkeit zu transferieren. Mithilfe von Eye-Tracking wurde dazu aufgenommen, welche Bereiche häufiger und weniger häufig betrachtet wurden. Eye-Tracking liefert Aufschlüsse über Aufmerksamkeitswechsel und andere kognitive Prozesse während der Lösung von Problemen, die andere traditionelle Metriken nicht liefern können [2]. Diese Eye-Tracking Daten wurden anschließend verarbeitet und visualisiert, damit weitere Entwickler sie als Hilfestellung nutzen können. Die Hypothese ist, dass diese Daten aufzeigen, welche Abschnitte für das Verständnis mehr oder weniger relevant sind. Mithilfe dieser Hinweise würden Entwickler weniger Zeit darauf verwenden, irrelevante Teile zu lesen und sie könnten sich schneller auf die wesentlichen Aspekte konzentrieren.

Dazu wurden verschiedene Möglichkeiten des Aufnahmekontexts, der Darstellung und des Nutzens von Aufmerksamkeit in der Software-Wartung exploriert und diskutiert. Zur weiteren Untersuchung wurde die Aufgabe des Codeverständnisses ausgewählt, Eye-Tracking Daten bei der Wartung dieses Codes aufgenommen und diese mittels Heatmaps visuell repräsentiert. Dazu wurde eine

Umgebung geschaffen, in der Eye-Tracking Daten bei der Arbeit mit Code im Webbrowser aufgenommen werden können. Diese umgeht die Problematik Eye-Tracking in Entwicklungsumgebungen mit scrollbarem und dynamischem Inhalt anzuwenden.

In dieser Arbeit wird evaluiert, wie geeignet und nützlich die Darstellung von Aufmerksamkeitsdaten durch Eye-Tracking ist. Es wird geprüft, ob die visuellen Hinweise auf den Aufmerksamkeitsfokus vorheriger Entwickler die Effizienz und Qualität von Softwarewartungsarbeiten an einem Javaprogramm verbessern. Zusätzlich wird untersucht, inwiefern diese Darstellung die Aufmerksamkeit überhaupt leitet und eine Ablenkung oder einen Störfaktor darstellt. In diesem Zuge wird ebenfalls getestet, ob es tatsächlich eine Korrelation zwischen Blickhäufigkeit auf Abschnitte und deren Relevanz für das Verständnis von Code gibt.

Die Vision dieses Ansatzes ist es, verschiedene Aufmerksamkeitsdaten ohne oder nur durch wenig Zutun der Entwickler aufzunehmen und diese für andere Teammitglieder visuell aufbereitet darzustellen. Mit diesem „By-Product Approach“ [3] soll Entwicklern der Einstieg in unbekannten Code bzw. das generelle Finden relevanter oder komplexer Bereiche erleichtert werden, um sie bei der Software-Wartung zu unterstützen.

1.3 Struktur der Arbeit

Diese Arbeit beginnt damit, einige Grundlagen zu Eye-Tracking zu erklären. Im Anschluss daran werden einige verwandte Arbeiten vorgestellt. Darauf folgend wurden zunächst die Probleme beschrieben, die der Einsatz von Eye-Tracking in der Software-Wartung mit sich bringt. Anschließend wird der Ansatz Aufmerksamkeit asynchron zu teilen, näher erläutert, die verschiedenen Möglichkeiten exploriert und beschrieben, welche Daten und Darstellungen zur weiteren Untersuchung in der Studie ausgewählt wurden. Das Kapitel danach geht dann ausführlich auf die Planung der Studie ein. Dazu gehören die Zielsetzung, Forschungsfragen, aufgestellte Hypothesen, benötigte Metriken, Experimentobjekte und –subjekte, sowie das Design des Experiments und die Zuteilung der definierten Aufgaben auf die Subjekte. Daraufhin wird erläutert, wie die aufgenommenen Aufmerksamkeitsdaten für die ausgewählten Darstellungen aufbereitet wurden. Danach folgt die Beschreibung der Experimentdurchführung, an welche die Analyse und Interpretation anschließen. Darin wird die Verarbeitung der Rohdaten erläutert und die zuvor aufgestellten Hypothesen formal geprüft. Das vorletzte Kapitel beinhaltet die Auswertung und Schlussfolgerung der Studie, welche die

betrachteten Forschungsfragen beantworten, die gewonnenen Ergebnisse zusammenfassen und deren Validität diskutieren. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick ab.

2 Grundlagen zu Eye-Tracking

Eye-Tracking stellt eine Methode dar, um Aussagen über die kognitiven Prozesse beim Lösen von Aufgaben treffen zu können. Dazu zeichnen Eye Tracker die visuelle Aufmerksamkeit von Personen auf, indem sie Blickbewegungen verfolgen und aufnehmen [4] [5]. In der menschlichen Wahrnehmung spielen Blickbewegungen eine große Rolle bei kognitiven Prozessen, da sie die visuelle Aufmerksamkeit auf bestimmte Bereiche eines Stimulus lenken. Bei einem Stimulus handelt es sich um ein Objekt (beispielsweise ein Diagramm oder Text), das zur Lösung einer Aufgabe benötigt wird. Bestimmte Bereiche eines Stimulus, die für eine Studie zur Betrachtung relevant sind, nennen sich Areas of Interest (AOI) [6].

Eye-Tracking Daten stützen sich dabei auf die folgenden zwei Indikatoren des Blickverhaltens [6]:

- Eine *Fixation* ist die Fokussierung des Auges auf einen Punkt. Während dieser steht das Auge zwischen 200 und 300ms still. Fixationen werden von Eye Trackern mit einem Zeitstempel, sowie X- und Y-Koordinaten aufgezeichnet.
- Eine *Sakkade* ist der schnelle Blicksprung zwischen zwei Fixationen und dauert nur zwischen 40 und 50ms. Während dieser kurzen Zeit findet keine Aufnahme von neuen visuellen Informationen statt.

Eine Reihe von aneinander hängenden Fixationen und Sakkaden bezeichnet man als Scan-Path. Zusätzlich zu Fixationen und Sakkaden liefern viele Eye Tracker auch den Pupillendurchmesser der Augen. Dieser gibt Aufschluss über die kognitive Last während der Bearbeitung von Aufgaben, da komplexe Aufgaben zu einer Erweiterung der Pupillen führen [5]. Ebenfalls zum Messen der kognitiven Last lassen sich die Häufigkeit des Blinzelns, sowie die Fixationsdauer verwenden. Längere Fixationen zeigen, dass Probanden mehr Zeit zum Analysieren und Interpretieren der gesehenen Inhalte benötigen. Demnach kostet es sie mehr mentalen Aufwand [6].

In der im Rahmen dieser Arbeit durchgeführten Studie wurde der Eye Tracker X3-120 der Firma Tobii verwendet. Zur Aufnahme, sowie zur Analyse der Eye-Tracking Daten diente das Tobii Pro Studio.

3 Verwandte Arbeiten

Zu Beginn der Arbeit wurde zunächst eine Suche nach verwandten Arbeiten durchgeführt. Dazu wurde vorwiegend Google Scholar verwendet und nach verschiedenen Kombinationen von Stichwörtern wie „eye tracking“, „attention transfer“, „focused attention“, „code comprehension“, „software maintenance“, „debugging“ und „change tasks“ gesucht. Ausgehend von relevanten Quellen wurde anschließend die Methode des Snowballing verwendet, um weitere für das Thema interessante Arbeiten zu finden. Zusätzlich wurden die in diesen Quellen angegebenen Keywords für weitere Suchen genutzt.

Eye-Tracking wurde bereits in sehr vielen Studien im Bereich des Software Engineering verwendet. In den meisten Fällen wird es jedoch ausschließlich als Metrik genutzt, um Aufschluss über Blickdauer und –reihenfolge zu geben und daraus Erkenntnisse über die Arbeitsweise bei verschiedenen Aufgaben in Softwareprojekten zu erlangen, wie beispielsweise dem Lesen, Verstehen und Bearbeiten von Code und Modellen [6]. Nur selten ist in verwandten Arbeiten der Ansatz, diese Daten nicht nur zu interpretieren, sondern für den Nutzer visuell aufbereitet darzustellen, wie es in dieser Arbeit geschieht.

Im ersten Unterkapitel dieses Abschnitts werden verschiedene Arbeiten vorgestellt, die Softwarewartungsarbeiten untersuchen. Dabei habe ich mich jedoch auf Studien beschränkt, die starken Bezug zu dem Thema dieser Arbeit haben und im besten Fall ebenfalls Eye-Tracking verwenden. Es werden Erkenntnisse über die Navigation von Entwicklern im Code präsentiert, um verstehen zu können, wie mit dem Code besonders beim Einstieg in unbekannte Programme gearbeitet wird und wie diese Aufgabe mit dem Ansatz des Aufmerksamkeitstransfers erleichtert werden könnte. Darüber hinaus wurden Studien gesucht, die den Einfluss des Erfahrungsstands auf Softwarewartungsarbeiten untersuchen, um beurteilen zu können, inwiefern der Aufmerksamkeitsfokus und die Ergebnisse für Studenten und Experten variieren.

Im zweiten Unterkapitel werden Arbeiten vorgestellt, die bereits basierend auf Interaktionsdaten der Entwicklungsumgebung oder mittels Eye-Tracking die Aufmerksamkeit gezielt lenken, um entweder Entwicklern die Arbeit zu vereinfachen oder für bessere Ergebnisse bei der Bearbeitung von Aufgaben sorgen. Interaktionsdaten geben jedoch nur sehr begrenzt Auskunft über Aufmerksamkeitsfokus und -wechsel. Die Eye-Tracking Ansätze hingegen beschränken sich aktuell noch auf die Wiederverwendung der Daten bei exakt derselben Aufgabe.

In keiner der gefundenen Studien wurde getestet, inwiefern sich das Blickverhalten für neue Aufgabenkontexte nutzen lässt. Abschließend wird in diesem Kapitel noch kurz ein Tool vorgestellt, das es ermöglichen soll, Eye-Tracking Daten direkt innerhalb der Entwicklungsumgebung Eclipse aufzunehmen.

3.1 Studien zu Aufgaben in der Software-Wartung

Eye-Tracking wurde bereits sehr oft dazu eingesetzt, verschiedene Tätigkeiten an Programmcode näher zu untersuchen. Beispielsweise haben Rodeghero et al. Blickdaten genutzt, um festzustellen, welche Teile einer Methode vorwiegend von Entwicklern angeschaut wurden, um daraus automatisiert Codezusammenfassungen zu erstellen [7]. In diesem Abschnitt werden verschiedene Arbeiten vorgestellt, die sich damit beschäftigen, wie Entwickler Code lesen, editieren, darin navigieren und wie sich das Blickverhalten dabei zwischen Experten und unerfahreneren Entwicklern unterscheidet.

3.1.1 Eye-Tracking: Code verstehen, editieren und erweitern

Bednarik et al. haben Eye-Tracking genutzt, um Codeverstehen mithilfe eines Program Visualization Tools zu untersuchen. Durch Nutzen von Metriken, wie Ort der Fixationen, Fixationsdauer und Aufmerksamkeitswechseln zwischen Areas of Interest (AOIs), konnte gezeigt werden, dass Visualisierungstechniken vor allem zu Beginn des Verstehens von Code helfen [8]. In einem anderen Ansatz wurden Eye-Tracking Metriken genutzt, um mittels eines Naive Bayes Classifiers die Schwierigkeit von Aufgaben in der Softwareentwicklung vorherzusagen. Es konnten knapp 65% Precision und Recall für neue Entwickler und knapp 85% Precision und knapp 70% Recall für neue Tasks erzielt werden. Die Motivation dabei war es, Entwickler bei besonders komplizierten Aufgaben warnen zu können, um Bugs vorzubeugen [9].

Ebenfalls in einer Eye-Tracking Studie haben Kevic et al. [10] Change Tasks an Open Source Systemen untersucht. Es wurde unter anderem die Aussagekraft von Interaktionsdaten der Entwicklungsumgebung und Eye-Tracking Daten miteinander verglichen. Die Ergebnisse zeigen, dass Eye-Tracking Daten deutlich umfangreicher und detaillierter sind. Der Arbeit mit dem Code konnte außerdem entnommen werden, dass Entwickler häufig nur einige Zeilen von Methoden lesen, die meist durch Datenfluss miteinander verbunden sind. Es wird kaum linear Zeile für Zeile gelesen und oft zu Methoden in näherer Umgebung gesprungen. Insgesamt wird sich nur auf wenige Methoden fokussiert und die Entwickler scheinen entweder einer Skimming oder einer Seeking Strategie zu folgen. Das heißt, es wird entweder der Code nur oberflächlich überflogen oder gezielt nach bestimmten Teilen gesucht [10].

3.1.2 Navigation im Code

Entwickler verbringen einen großen Teil ihrer Zeit damit, durch Quellcode zu navigieren, darin zu lesen, zu suchen und ihn an passenden Stellen zu bearbeiten [10]. Laut Ko et al. fallen 35% der Entwicklertätigkeit allein auf Codenavigation und das Suchen relevanter Codeteile. Teils müssen aufgrund des Umfangs und der Komplexität des Codes Abschnitte auch mehrfach gesucht werden [11]. Es gibt einige Untersuchungen darüber wie Entwickler sich im Code bewegen. Ergebnisse zeigen, dass zunächst nach initialen Anhaltspunkten gesucht wird und von dort weitere Beziehungen im Code untersucht werden [12]. Fritz et al. haben dazu betrachtet, inwiefern Navigationsmuster abhängig vom Taskkontext sind. Trotz Abweichungen konnten teilweise Übereinstimmungen in den als besonders relevant erachteten Klassen gefunden werden [13].

Um Entwickler bei dieser komplexen Aufgabe zu unterstützen, hat Robillard einen Ansatz entwickelt, um dem Nutzer potenziell interessante Programmelemente vorzuschlagen, während er sich im Code bewegt. Mittels einer Analyse der strukturellen Abhängigkeiten im Programm wird eine Menge an Elementen angezeigt, um weniger interessante Abhängigkeiten außer Acht lassen zu können. In einer anschließenden Evaluation hat sich diese Technik als schnelle Hilfe erwiesen [14]. Piorkowski et al. haben Information Foraging genutzt, um Entwicklern Empfehlungen zu geben, wo basierend auf Wörtern und Codestruktur relevante Informationen stehen könnten. Bei Information Foraging handelt es sich um eine Theorie, die sich basierend auf den natürlichen Futtersuchmechanismen damit beschäftigt, wie Menschen Informationen suchen. In empirischen Untersuchungen hat sich gezeigt, dass Studienteilnehmer diese Vorschläge nutzten, um entweder unbekannte Codebereiche zu entdecken oder als Shortcuts, um effizient in bereits bekannte Stellen zu navigieren [15].

3.1.3 Unerfahrene Entwickler vs. Experten

Viele Studien beschäftigen sich damit, wie sich der Umgang mit Code zwischen unerfahrenen Entwicklern und Experten unterscheidet. Obwohl die Ergebnisse diesbezüglich variieren und teils nur minimale Unterschiede gefunden wurden, haben sich auch teils signifikante Abweichungen herausgestellt. So lesen Experten beispielsweise in der Anzahl weniger Methoden, vor allem weniger für die Aufgabe irrelevante Methoden [16]. Zudem lesen sie Code auch weniger linear als unerfahrene Entwickler [17]. Experten haben kürzere Fixationen auf für den Task relevanten Bereichen, machen mehr Fixationen insgesamt und längere Sakkaden, also weitere Blicksprünge. Auch gelangen sie schneller zu relevanten Bereichen im Programm [18]. Bednarik konnte in einer Eye-Tracking Studie

ebenfalls unterschiedliche Debuggingstrategien beim Umgang mit verschiedenen Coderepräsentationen feststellen, abhängig von der Expertise der Probanden [2]. Crosby et al. fand in einer Eye-Tracking Studie außerdem heraus, dass unerfahrene Entwickler eher auf Kommentare achten, während sich Experten mehr auf bedeutsame und komplexe Bereiche im Code fokussieren [19]. In weiteren Untersuchungen (auch teils mit Eye-Tracking) wurde geprüft, wie der Erfahrungsstand Einfluss auf den Umgang mit Beacons im Code hat, also Bereichen, die relevant für das Verständnis vom Code und dessen Struktur sind. Unerfahrene Entwickler unterschieden danach kaum zwischen verschiedenen Programmbereichen und machten daher kaum von Beacons Gebrauch. Die Beacons für Experten unterschieden sich zudem von denen unerfahrener Entwickler. Experten konzentrierten sich eher auf wichtige Programmteile und verließen sich daher auch eher auf Beacons [20].

3.2 Lenkung von Aufmerksamkeit

Es gibt bereits verschiedene Studien dazu, inwiefern sich die Aufmerksamkeit von Personen gezielt lenken lässt, um Aufgaben schneller oder besser ausführen zu können. In den Fällen, in denen Eye-Tracking verwendet wurde, handelt es sich aber entweder um ungetestete Empfehlungen oder die Erkenntnisse wurden nicht auf neue Aufgabenkontexte angewendet. Ein Bereich beschäftigt sich zudem mit der Nutzung von Interaktionsdaten, die von Entwicklungsumgebungen detektiert werden. Das bereits in der Praxis eingesetzte Eclipse Plugin Mylyn, das diese Daten nutzt, um Aufgabenkontexte zu extrahieren, wird in diesem Unterabschnitt vorgestellt. Die Initiative DFlow verfolgt ebenfalls das Ziel, die Events, die in der IDE stattfinden und bislang größtenteils ignoriert werden, zu nutzen, um „self-adaptive IDEs“ zu schaffen [21] [22]. Zu diesen Events gehören beispielsweise das Auswählen oder Editieren einer Zeile, das Öffnen eines Menüs oder das Ausführen von Refactoring-Befehlen. Die Vision ist, dass Entwicklungsumgebungen die Interaktionsdaten nutzen, um sich speziell an die Bedürfnisse des Entwicklers und seine aktuelle Aufgabe anzupassen.

3.2.1 Lenkung von Aufmerksamkeit mittels Eye-Tracking

Einige Ansätze beschäftigen sich damit, wie Eye-Tracking Daten von Personen auf Andere bei der Bearbeitung derselben Aufgabe performance- oder erfolgssteigernd wirken können. Stein et al. haben dazu in einem Experiment Teilnehmern die Aufgabe gegeben, Bugs in einem Programm zu finden. Dazu wurden die Blickdaten professioneller Entwickler beim Finden von Bugs in kleinen Java Programmen aufgenommen. Einer zweiten Gruppe von Entwicklern wurden anschließend für die Hälfte der Bugs zuvor aufgenommene Blickdaten in Form

eines Videos abgespielt. In dem Video war der Blickpunkt einer der anfänglichen, professionellen Entwickler als Cursor auf dem Code dargestellt. Für die zweite Hälfte der Bugs gab es diese Hilfestellung nicht. Es hat sich gezeigt, dass die zweite Gruppe die Bugs schneller finden konnte, nachdem sie den Blickverlauf des ersten Entwicklers gesehen hatten [1].

Auch außerhalb des Software Engineering gibt es bereits Studien dazu, wie man Menschen bei Problemlösungen unterstützen kann, indem man ihnen das Blickverhalten anderer Personen zeigt, die die gleiche Aufgabe erfolgreich bewältigt haben. Umfassende Untersuchungen gibt es dabei über das Duncker Bestrahlungsproblem (1945). Dabei handelt es sich um ein Diagramm eines Tumors, der mittels Laserbestrahlung zerstört werden soll, ohne das darum liegende gesunde Gewebe zu beschädigen. Studien darüber haben gezeigt, dass Studenten, die das Problem erfolgreich lösen konnten sich im Blickverhalten von denen unterschieden, die in der vorgegebenen Zeit keine Lösung finden konnten. Erfolgreiche Experimentteilnehmer achteten vermehrt auf die Hautbereiche um den Tumor herum, bevor sie auf die Lösung kamen. Um diese Erkenntnis neuen Studenten als Hilfestellung zu geben, wurden in einem Experiment diese Bereiche hervorgehoben, indem sie vergrößert dargestellt wurden [23]. In einer anderen Studie wurden den Teilnehmern zunächst verschiedene Blickpfade von erfolgreichen Problemlösern abgespielt [24]. In beiden Fällen hat sich der Anteil an Teilnehmern, die auf die richtige Lösung kamen durch die visuelle Aufmerksamkeitsdarstellung signifikant erhöht.

3.2.2 Hervorheben einzelner Codeabschnitte

Einige Studien setzten sich damit auseinander, wie sich Beacons im Code finden lassen. Diese besonders relevanten Codeabschnitte sollen Entwicklern helfen schneller einen Überblick und Verständnis für unbekannten Code zu erhalten und effizient zu wichtigen Programmteilen zu navigieren. Eye-Tracking kann eine große Hilfe dabei sein, Beacons im Code zu finden. Aschwanden et al. schlagen vor, diese dort zu lokalisieren, wo Fixationen mindestens 1000 Millisekunden lang sind [25]. Ebenfalls einen Ansatz Beacons zur Erleichterung des Codeverständnisses zu nutzen, haben Hegarty-Kelly et al. entwickelt, in dem sie mittels Eye-Tracking von Experten Beacons im Code extrahieren und diese hervorheben. Anschließend sollte in drei Gruppen der Effekt dieser Hervorhebung auf unerfahrene Entwickler getestet werden. Eine Gruppe sollte dazu den Code mit Beacons bekommen, eine ohne und eine mit bewusst falsch platzierten Beacons [26]. Bisher wurden jedoch weder die theoretische Vorgehensweise, noch die nähere Untersuchung solcher Darstellungen von relevanten oder intensiv betrachteten Codeabschnitten realisiert.

3.2.3 Filterung taskabhängiger Kontexte

Das Eclipse Plugin Mylyn ist ein Tool, das die Aufmerksamkeit von Entwicklern während Softwarewartungsarbeiten in Form der Auswahl oder des Editierens von Codeelementen aufnimmt und daraus ein Task Context Model erstellt [27]. In diesem Modell wird taskabhängig jedem Element (Methoden, Klassen, etc.) ein Degree-of-Interest (DOI) zugewiesen. Das Auswählen oder Editieren erhöht dabei das DOI dieses Elements und je länger eine Interaktion mit dem Element zurückliegt, desto mehr wird das DOI wieder verringert [28]. Außer direktem Agieren wie dem Auswählen oder Editieren gibt es aber auch indirekte Einflüsse auf das DOI von Elementen, wenn zum Beispiel andere Elemente, die dazu in Beziehung stehen oder darauf referenzieren, verwendet werden. Wenn ein Entwickler beispielsweise eine Methode in einer Klasse bearbeitet, erhöht sich dadurch nicht nur das DOI der Methode, sondern auch der Klasse. Das Mylyn Tool repräsentiert diese Task Context Models innerhalb von gesonderten Eclipse Views. In einer Übersicht über alle Tasks, an denen der Entwickler zur Zeit arbeitet, gearbeitet hat oder plant zu arbeiten, lassen sich Tasks auswählen und so deren Kontext anzeigen. Wenn ein Task ausgewählt ist, werden im Package Explorer nur noch Elemente angezeigt, die für den Task relevant sind. Zusätzlich hat der Nutzer jedoch die Möglichkeit auch manuell das Task Context Model anzupassen, indem er selbst aktiv Elemente als relevant markiert, sodass sie ebenfalls in der Übersicht erscheinen [29].

3.3 Tool zur Aufnahme von Eye-Tracking Daten in Eclipse

Eye-Tracking Daten bei der Arbeit mit Code aufzunehmen, bringt mit der aktuellen Technik noch eine große Problematik mit sich, auf die in Abschnitt 4 noch näher eingegangen wird. Die Herausforderung beim Eye-Tracking von Code ist es, mit den dynamischen Inhalten in Form von gescrollten Bereichen und wechselnden Tabs umzugehen. Das von Shaffer et al. vorgestellte Eclipse Plugin iTrace [30] [31] adressiert genau diese Problematik. Das Tool unterstützt die Aufnahme von Eye-Tracking Daten innerhalb der Entwicklungsumgebung Eclipse. Von der Kalibrierung, über die Aufnahme der Daten bis hin zum Export in XML- oder JSON-Dateien lässt sich das Eye-Tracking direkt in der IDE steuern. Die Handhabung des Plugins verdeutlicht Abbildung 3.1. Über diese Oberfläche, die als kleine View innerhalb von Eclipse dargestellt wird, erfolgt das Einrichten der Eye-Tracking Aufnahmen. Es wird somit nach erfolgreicher Einrichtung des Plugins keine weitere Software mehr zur Durchführung des Eye-Trackings in Eclipse benötigt.

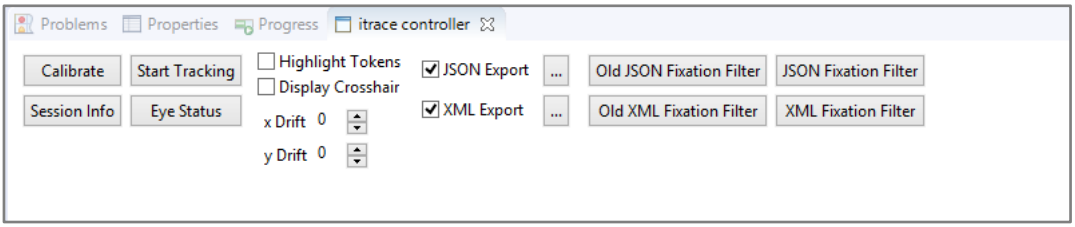


Abbildung 3.1: iTrace Bedienoberfläche in Eclipse

Die erstellten Daten enthalten nicht nur Informationen zu den Fixationen wie etwa Dauer und Pupillenweite, sondern auch das geöffnete Codedokument und das angeschaute Codeelement (If-Statement, For-Schleife, etc.). Damit ist es dem Entwickler möglich, beinahe ungehindert Softwarewartungsarbeiten wie Debuggen, Codeverständnis und –erweiterung nachzugehen. Ebenfalls problematisch ist dabei lediglich das Öffnen von Fenstern innerhalb von Eclipse, wie Suchmasken oder dem Dateiexplorer zum Öffnen oder Speichern von Dateien, ebenso wie das Ein- und Ausklappen von Codebereichen. Während diese Elemente geöffnet sind, nimmt das Tool keine Eye-Tracking Daten auf.

Das Plugin lässt sich mit allen Tobii Eye Trackern verbinden, die von dem Tobii Analytics SDK und dem Tobii Eye X SDK unterstützt werden. Leider sind beide Softwaremodule mittlerweile nicht mehr erhältlich, sondern wurden durch das Tobii Pro SDK abgelöst [32]. Selbst nach Anfrage der alten Softwareversion war es damit jedoch nicht möglich, die aktuell (22.06.2018) verfügbare Version des iTrace Plugins erfolgreich mit dem verwendeten Tobii X3-120 Eye Tracker zu verbinden.

4 Problematik von Eye-Tracking in der Software-Wartung

In diesem Kapitel wird die Problematik vorgestellt, Eye-Tracking im Kontext der Software-Wartung einzusetzen. Infolgedessen wird beschrieben, wie sich dies auf die im Rahmen dieser Arbeit durchgeführte Studie auswirkte.

4.1 Dynamischer Content

Aktuelle Eye Tracker sind bei der Nutzung im Bereich der Software-Wartung noch sehr limitiert, vor allem bei der Arbeit mit Code. Entwicklungsumgebungen stellen Code oder auch andere Dateien üblicherweise scrollbar in einzelnen Tabs und verschiedenen Views dar. Auch das Ein- und Ausklappen von Unterabschnitten ist üblich. Bei dem derzeitigen Eye Tracker Standard stellt dynamischer Content, wie scrollbare Bereiche beispielsweise, jedoch ein großes Problem dar. Sowohl bei den Eye Trackern des Herstellers SMI mit der dazugehörigen Experimentsoftware „Experiment Center“, also auch dem in dieser Arbeit verwendeten Tobii Eye Tracker und dem dazugehörigen Tobii Pro Studio, werden die Blickdaten statisch auf dem Bild des Monitors aufgenommen. Da sich das Bild im Hintergrund jedoch durch Scrolling innerhalb der Tabs verändert, lassen sich die Blickdaten keinen Inhalten mehr zuordnen. Abbildung 4.1 veranschaulicht diese Problematik im Kontext einer Entwicklungsumgebung. Im Vordergrund sieht man die statischen, vom Eye Tracker aufgenommenen Blickpunkte, jeweils durch verschieden große Kreise dargestellt. Jeder Kreis stellt eine Fixation dar. Die Größe des Kreises ist dabei proportional zu deren Länge. Im Hintergrund ist der zu dem Zeitpunkt gerade sichtbare Code zu erkennen. Durch Springen auf der Zeitleiste oder Abspielen des Gaze Plot Videos, verändert sich entsprechend das Hintergrundbild, wenn der Proband in der Zwischenzeit gescrollt oder die Datei gewechselt hat und es kommen weitere Blickpunkte hinzu. Diese bleiben jedoch statisch. Das heißt, Informationen über das tatsächlich Angeschaute sind nur für den aktuellen Blickpunkt zum ausgewählten Zeitpunkt verlässlich. Wenn der Proband dann scrollt, verschiebt sich zwar der Code im Hintergrund, die Blickpunkte wandern jedoch nicht mit. Alle vorherigen Blickpunkte werden damit letztlich unbrauchbar, da sie nicht mehr dem tatsächlich Betrachteten zugeordnet sind.

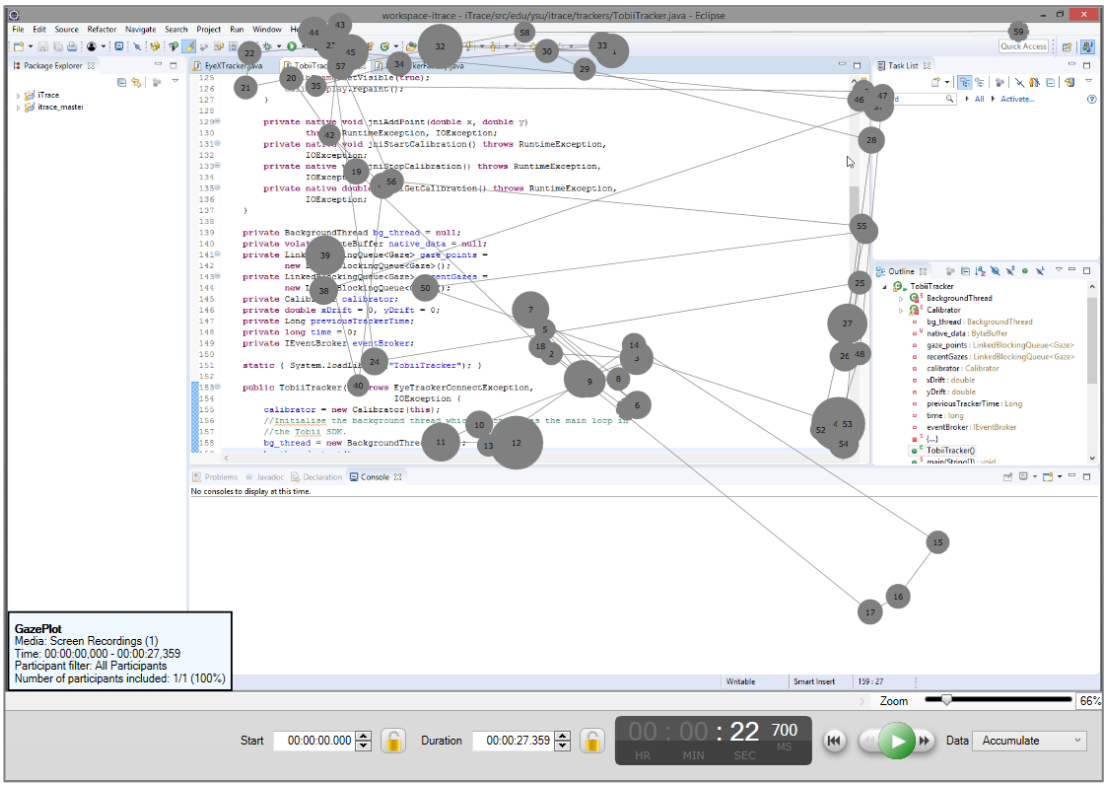


Abbildung 4.1: Gazeplot auf Eclipse im Tobii Pro Studio

Die einzige Umgebung in der Scrolling aktuell unterstützt wird, sind Webseiten im Browser. Auch dabei ist jedoch darauf zu achten, dass es sich um Webseiten ohne dynamischen Inhalt handelt. Das heißt, es dürfen beispielsweise keine wechselnden Bilder angezeigt werden. Scrolling wird ebenfalls nur unterstützt, wenn die gesamte Webseite gescrollt wird und nicht nur einzelne Unterelemente. In der Datenanalyse wird in dem Fall dann nicht einfach das aktuelle Bild des Monitors zu dem ausgewählten Zeitpunkt oder Zeitbereich angezeigt, sondern die Webseite wird als Gesamtbild in voller Länge dargestellt, sodass die Blickpunkte an der jeweils richtigen Stelle aufgetragen sind, unabhängig vom Scrolling. Eine Übersicht darüber gibt Abbildung 4.2. Darin sieht man, dass die Abbildung der Auswertung, auf der der Blickpfad dargestellt wird, nicht mehr das Format des Monitors hat, sondern die gesamte Webseite, inklusive zu scrollendem Inhalt zeigt. Auch Webseiten dürfen jedoch nur eine bestimmte maximale Länge haben und selbst das Wechseln zwischen Tabs im Browser wird aktuell bei dem verwendeten Tobii Eye Tracker nur im Firefox Browser unterstützt [33].

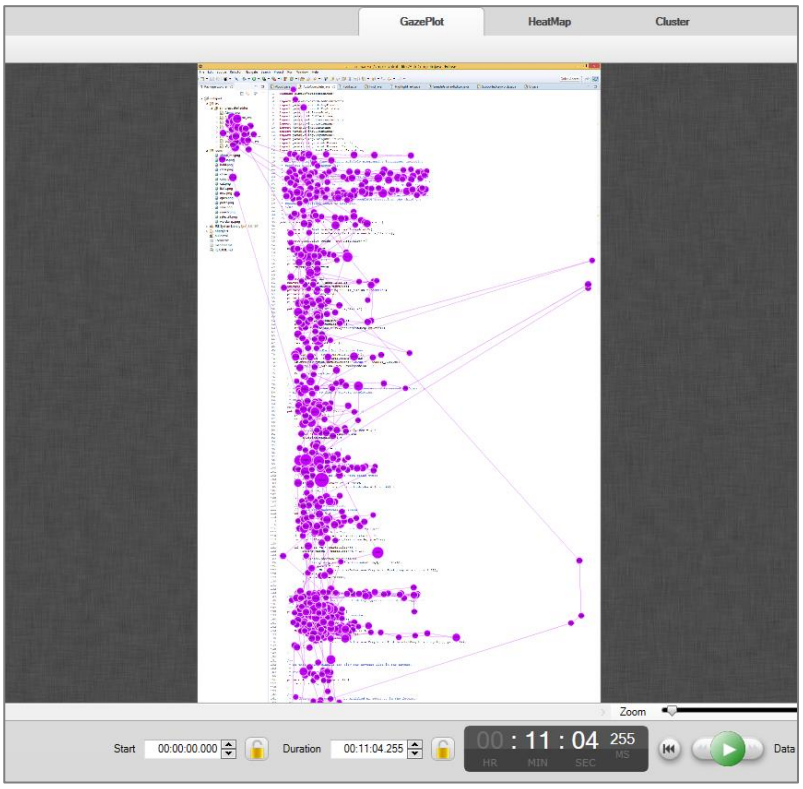


Abbildung 4.2: Gaze Plot auf scrollbarer Webseite im Tobii Pro Studio

4.2 Auswirkungen auf die Studie

Um Eye-Tracking auch für Code nutzen zu können, der die Größe einer Bildschirmseite übersteigt, ist man daher gezwungen, diesen in einer Webseite darzustellen, um von Scrolling Gebrauch machen zu können. Die Codedateien dürfen damit zwar eine Bildschirmseite überschreiten, sind jedoch dennoch in der Länge beschränkt. Bei Überschreiten dieser Grenze wird der Inhalt des Screenshots der Webseite in der Analyse abgeschnitten. Auch das Editieren von Code ist problematisch, da durch das Hinzufügen oder Entfernen von Zeilen ebenfalls der gesamte Code nach unten oder oben rückt und der Bildschirminhalt damit wieder dynamisch ist. Der Eye Tracker würde in diesem Fall wieder nur zum Aufruf der Webseite einen Screenshot des Seiteninhalts machen und darauf die Blickdaten abtragen. Entsprechende Änderungen durch Editieren sind darin somit nicht enthalten. Änderungen im Code dürfen daher entweder gar nicht oder nur in Abschnitten erfolgen, die in der Auswertung vom Eye-Tracking ausgeschlossen werden. Lediglich bei kleinen, seltenen Änderungen ließe sich die Aufnahme manuell in Segmente aufteilen und diese könnten dann getrennt ausgewertet werden. Sobald die Änderungen dagegen häufiger und stetiger werden, wird diese Form der segmentalen Auswertung jedoch schnell impraktikabel. Durch die Anzeige als Webseite lässt sich der Code außerdem weder Ausführen, Testen noch Debuggen. Damit fehlen viele essentielle Funktionalitäten gängiger IDEs.

5 Asynchrones Teilen von Aufmerksamkeit

Die Vision und der Ansatz dieser Arbeit sind die Aufnahme, Verarbeitung und Darstellung von Aufmerksamkeitsdaten in der Software-Wartung. Es gibt jedoch viele verschiedene Arten von Aufmerksamkeit, die während Softwarewartungsarbeiten aufgenommen werden können. Genauso gibt es auch viele verschiedene Möglichkeiten, diese Daten darzustellen und sie lassen sich in unterschiedlichen Kontexten nutzen. Zudem sind höchstwahrscheinlich nicht alle Aufmerksamkeitsdaten in jedem Kontext gleich hilfreich. In diesem Kapitel wird daher vorerst theoretisch exploriert, welche Arten von Aufmerksamkeitsdaten sich überhaupt aufnehmen lassen, wie man diese darstellen könnte und in welchem Aufgabenkontext diese vermutlich den größten Nutzen bringen. Zusätzlich wurde überlegt, mit welchen Metriken sich der Nutzen dieser Darstellungen am besten messen ließe.

5.1 Arten von Aufmerksamkeitsdaten

In diesem Abschnitt werden zunächst mögliche Arten, Aufmerksamkeitsdaten aufzunehmen erkundet und diskutiert. Es wurde sich dabei stark auf Metriken fokussiert, die durch Eye-Tracking aufgenommen werden können, da diese Technik ebenfalls in der im Rahmen dieser Arbeit durchgeführten Studie verwendet wurde. Zudem gibt es bereits einige Ansätze, wie beispielsweise DFLOW oder Mylyn (siehe Abschnitt 3.2), die sich mit den Aufmerksamkeitsdaten beschäftigen, die sich Entwicklungsumgebungen entnehmen lassen. Der Vollständigkeit halber sind diese in diesem Kapitel jedoch dennoch kurz mit aufgeführt.

Im Folgenden sind die Arten von Aufmerksamkeitsdaten aufgelistet, die sich durch Eye-Tracking oder das Interaktionsprotokoll von Entwicklungsumgebungen bei der Software-Wartung automatisch aufnehmen lassen:

Arten von Aufmerksamkeitsdaten:

- Effizienz der Informationssuche: Fixationsanzahl
- Visueller Aufwand: Sakkadenanzahl, Sakkadendauer, Sakkadenamplitude
- Interaktion: Editierfrequenz, Selektionsfrequenz von Abschnitten
- Blickpfade
- Selektionspfade
- Blickdauer (dwell time)
- Kognitive Last: Pupillendurchmesser, Fixationsdauer, Blinzeln

Wenn man nun eine oder mehrere dieser Datenarten auswählt und darstellt, ist es nötig auf irgendeine Weise den Effekt dieser Darstellung zu messen. Schließlich ist der Ansatz bisher nur mit Hypothesen und Annahmen begründet. Um bestimmen zu können, inwiefern die Darstellung dieser Daten tatsächlich hilfreich ist, wurden Metriken herausgearbeitet, mit denen man den Nutzen der Repräsentation von Aufmerksamkeitsdaten messen kann.

Metriken zur Messung des Nutzens von Aufmerksamkeitsdaten:

- Effizienz der Informationssuche: Fixationsanzahl
- Kognitive Last: Pupillendurchmesser, Fixationsdauer, Blinzeln
- Visueller Aufwand: Sakkadenanzahl, Sakkadendauer, Sakkadenamplitude
- Scan-Path Precision
- Scan-Path Recall
- Scan-Path F-Measure
- Zeit zur Bearbeitung von Softwarewartungsaufgaben
- Fragen zum Verständnis der Software
- Qualität/ Korrektheit/ Erfolg von Softwarewartungsaufgaben
- Anzahl der Wechsel/ Sprünge zwischen Bereichen

Bei der Scan-Path Precision wird berechnet, wie viele der angeschauten Abschnitte tatsächlich für die Aufgabe relevant waren. Bei dem Scan-Path Recall wiederum handelt es sich um den Anteil an insgesamt relevanten Abschnitten, die angeschaut wurden. Das heißt, diese Metrik stellt eine Art „Trefferquote“ der relevanten Abschnitte dar. Der Scan-Path F-Measure hingegen ist eine Kombination von Precision und Recall. Die gleichen Metriken ließen sich auch auf selektierte oder editierte, statt angeschauter Bereiche anwenden.

5.2 Mögliche Anwendung in der Software-Wartung

Der Nutzen der Darstellung all dieser Aufmerksamkeitsdaten kann jedoch auch vom Kontext abhängen. Nicht jede Aufgabe im Bereich der Software-Wartung bietet gleich viel Potenzial für die Nutzung von Aufmerksamkeitstransfer. Dazu wurde exploriert und in der folgenden Tabelle dargestellt, welche Aufgaben es in der Software-Wartung überhaupt gibt und wie der geschätzte Nutzen von Aufmerksamkeitsdarstellungen in diesem Bereich ausfällt. Es handelt sich dabei jedoch nur um begründete Hypothesen, nicht um empirisch ermittelte Erkenntnisse. In der ersten Spalte der Tabelle 5.1 sind alle Softwarewartungsarbeiten aufgelistet. In der mittleren Spalte ist der geschätzte Nutzen von Aufmerksamkeitstransfer auf einer Skala von Niedrig, über Mittel bis Hoch angegeben. In der dritten Spalte findet sich die entsprechende Begründung für diese Bewertung.

Arten von Softwarewartungsarbeiten	Geschätzter Nutzen von Aufmerksamkeitstransfer	Begründung
Verstehen	Hoch	Kernkonzepte und besonders relevante Abschnitte schneller erkennen
Erweitern	Mittel je nach Änderung	Verständnis als Voraussetzung
Ändern	Mittel je nach Änderung	Verständnis als Voraussetzung
Refactoring	Mittel	Aufmerksamkeit als Aufteilungskriterium
Fehlersuche/ Debuggen	Hoch	Komplizierte und oft editierte Abschnitte erkennen
Review	Mittel	Komplizierte Abschnitte besonders beachten

Tabelle 5.1: Arten von Softwarewartungsarbeiten und deren Potenzial für die Nutzung von Aufmerksamkeitsübertragung

Bei dieser Auflistung wurde sich stark auf Softwarewartungsarbeiten im Bereich der Softwareentwicklung, also dem direkten Umgang mit Code fokussiert. Es gibt aber natürlich auch noch andere Artefakte neben Quellcode, wie Spezifikationen oder Software Design Dokumente, bei denen die Visualisierung von Aufmerksamkeitsdaten hilfreich sein könnte. All diese Einschätzungen lassen sich auch auf solche Dokumente übertragen, da auch diese natürlich verändert, verstanden und Reviews unterzogen werden müssen. Lediglich die Ergebnisse der Evaluation des Nutzens in einem Bereich lassen sich vermutlich nicht einfach auf andere Kontexte übertragen.

5.3 Mögliche Aufmerksamkeitsdarstellungen

Die oben aufgeführten Aufmerksamkeitsdaten lassen sich auf unterschiedliche Arten darstellen. Insbesondere für Eye-Tracking Daten gibt es sehr viele verschiedene Visualisierungsmöglichkeiten. Die verbreitetsten sind Heatmap, Gaze Plot und Gaze Opacity. Heatmaps visualisieren Eye-Tracking Metriken wie Fixationsanzahl oder –dauer mittels einer Farbskala. Besonders häufig oder lange betrachtete Bereiche werden rot gefärbt, weniger intensiv betrachtete Bereiche gelb und wenig oder selten angeschaute Bereiche grün. Der Übergang zwischen diesen Farben ist jedoch fließend. Abbildung 5.1 zeigt ein Beispiel einer Heatmap auf Code aus dem Tobii Pro Studio.

Dagegen stellen Gaze Plots Blickpfade dar. Fixationen werden dabei als Kreise repräsentiert, wobei die Größe des Kreises die Länge der Fixation angibt. Die

Fixationen sind jeweils durch Linien miteinander verbunden, welche die Sakkaden, also die Blicksprünge dazwischen darstellen. Ein Beispiel dieser Darstellung findet sich in Abbildung 5.2. Bei der Visualisierung mittels Gaze Opacity, auch Schlüssellochdarstellung genannt, werden nur intensiv betrachtete Bereiche des Bildes sichtbar, der Rest bleibt schwarz. Dadurch wird der Fokus bei der Analyse sehr stark auf die Punkte gelenkt, die hauptsächlich betrachtet wurden und der Rest wird ausgeblendet. Für die Darstellung von Aufmerksamkeitsdaten in dieser Arbeit ist diese Visualisierung jedoch ungeeignet, da sich das Gesamtbild so kaum noch erkennen lässt und der Text oder Code damit nicht mehr lesbar wäre.

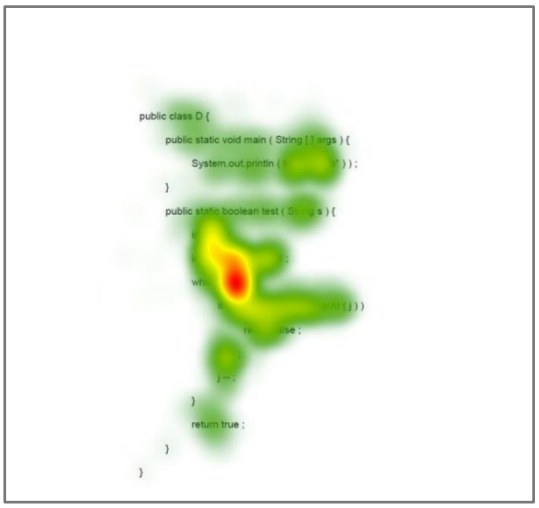


Abbildung 5.1: Heatmap auf Code (aus dem Tobii Studio) [34]

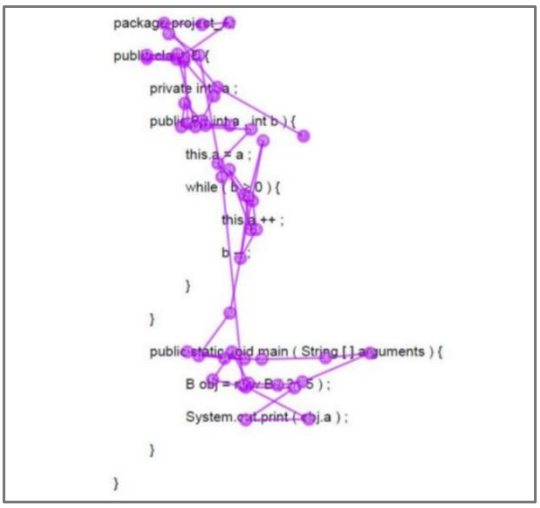


Abbildung 5.2: Gaze plot auf Code (aus dem Tobii Studio) [34]

In einigen Arbeiten wurden noch andere, speziellere Visualisierungsmöglichkeiten entwickelt. Das Tool TAUPE [35] beispielsweise bietet neben der Korrektur von Eye-Tracking Daten auch einige Darstellungsoptionen. Eine davon ist die konvexe Hülle von Fixationen. Dabei wird der Bereich, in dem die längsten 50% der Fixationen liegen, durch ein farbiges Polygon markiert. Busjahn et al. [34] haben außerdem eine Visualisierungsmöglichkeit speziell für Code entwickelt, in der die einzelnen Tokens im Code auf einer Farbskala von grün bis rot eingefärbt werden, abhängig davon wie viele Fixationen auf dem Bereich des Worts oder Zeichens liegen. Abbildung 5.3 zeigt eine solche „Color coded attention allocation map“. Für das in Abschnitt 3.3 bereits beschriebene Eye-Tracking Tool iTrace gibt es zudem eine Erweiterung namens iTraceVis, das die Eye-Tracking Daten aus Dateien im Exportformat von iTrace visualisiert. Aktuell lassen sich damit jedoch nicht mehrere Teilnehmerdaten kombinieren, sondern immer nur Eye-Tracking Daten einer Aufnahme auf einmal anzeigen. Diese Erweiterung bietet zusätzlich zur Heatmap noch die Visualisierungsoptionen Gaze Skyline, Static Gaze Map und Dynamic Gaze Map [36]. Das Besondere an der Static und Dynamic Gaze Map ist, dass sie zusätzlich Zeitinformationen abbilden. Das heißt,

man sieht zu welchem Zeitpunkt welcher Bereich angeschaut wurde. Diese Information geht in den anderen genannten, statischen Darstellungen verloren. Für die Aufmerksamkeitsdarstellung sind aber auch diese weniger geeignet, da es meist sehr aufwendig ist, diese zu interpretieren und hilfreiche Informationen aus ihnen zu ziehen. Das widerspricht dem Ansatz dieser Arbeit, der darauf abzielt, subtile, schnell zu begreifende Hinweise auf den Aufmerksamkeitsfokus anderer Personen geben zu können. Wenn man bereits zum Verstehen der Aufmerksamkeitsdarstellung sehr viel Zeit aufwenden muss, verfehlt dies das eigentliche Ziel der Zeitersparnis.

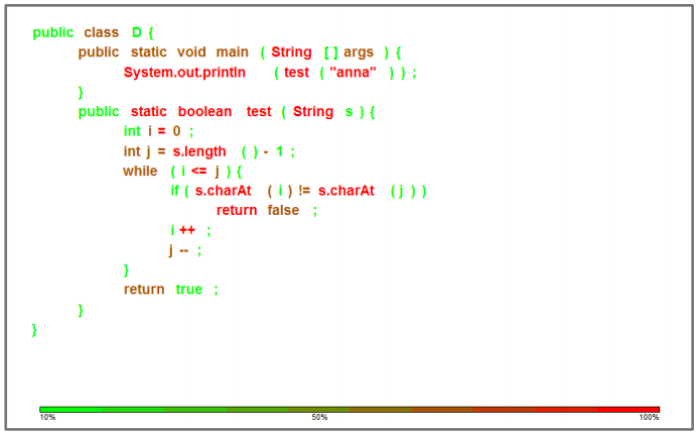


Abbildung 5.3: Color coded attention allocation map [34]

Im Folgenden sind nun die Darstellungsmöglichkeiten aufgelistet, die für die Darstellung von Aufmerksamkeitsdaten sinnvoll wären. Diese Liste ließe sich noch durch weitere, selbst definierte Visualisierungen ergänzen. Ebenso wäre es möglich mit einer Darstellungsform nicht nur eine Art von Aufmerksamkeitsdaten zu visualisieren, sondern eine Kombination von mehreren.

Darstellungsmöglichkeiten für Aufmerksamkeitsdaten:

- Heatmap (im Hintergrund oder tokenweise)
- Heatmap-Streifen am Rand
- Color coded attention allocation map
- Annotationen
- Sprungvorschläge

5.4 Aufmerksamkeitsdaten- und Darstellungsauswahl für die Studie

Für die im Rahmen dieser Arbeit durchgeführte Studie zum Testen des Ansatzes der Aufmerksamkeitsübertragung habe ich mich für eine Heatmapdarstellung entschieden. Diese ist schnell begreifbar, lässt sich sehr gut subtil im Hintergrund von Text und Code darstellen und verursacht nur einen geringen Aufwand beim

Erstellen, da gängige Eye-Tracking Analysesoftware Heatmaps automatisch generieren kann. Auf diese Weise lassen sich einfach dynamisch Eye-Tracking Daten von weiteren Personen einarbeiten. Ich habe mich bei der Untersuchung auf den Bereich des Codeverständnisses spezialisiert, bei dem die Aufmerksamkeitsdarstellung helfen soll, relevante Abschnitte schneller zu finden. Damit könnte der Code schneller verstanden und daran weitergearbeitet werden. Die Visualisierung soll daher den Fokus auf wichtige Bereiche lenken. Daher wurden Heatmaps auf Basis der Fixationsanzahl und nicht der Fixationsdauer generiert. Die Fixationsdauer hätte stattdessen eher auf kompliziertere Bereiche hingewiesen, was beispielsweise beim Finden von Bugs helfen könnte.

Zusätzlich wurden Sprünge zwischen Klassen mittels einer Heatmapdarstellung visualisiert. Diese sollten subtile Sprungvorschläge geben, basierend darauf, welche Wechsel zwischen Klassen andere Entwickler zuvor getätigt haben. Dazu wurden die Anzahlen an Sprüngen von einer Klasse zu den jeweils anderen Klassen für jeden Entwickler einzeln in eine Sprungmatrix extrahiert. Anschließend wurden diese absoluten Werte normalisiert, damit jeder Entwickler gleichen Einfluss auf die resultierende Heatmapfärbung hat und nicht einzelne Entwickler bevorzugt werden, nur weil sie grundsätzlich sehr häufig zwischen Klassen wechseln und daher höhere Werte verursachen. Dazu wurde für jeden Wert in der Matrix der Anteil am Maximum der Anzahlen an Sprüngen aus der Klasse berechnet. Diese relativen Anteile wurden dann für alle Entwickler summiert und auf Prozentwerte gebracht. Diese Prozentwerte für jede Klasse wurden anschließend auf die gleiche Heatmap Farbskala abgebildet, die für die Heatmaps der Fixationsanzahlen verwendet wurde. Um diesen Schritt zu vereinfachen, wurde eine Java Anwendung geschrieben, die aus den Angaben der summierten relativen Sprunganzahl und der summierten maximalen Sprunganzahl eine Farbe auf der Heatmapskala berechnet, wie sie auch im Tobii Pro Studio verwendet wird. Dafür wurden die Farbwerte des entsprechenden Grüns, Gelbs und Rots interpoliert. Abbildung 5.4 zeigt diese Farbskala von grün, über gelb bis rot, wobei der höchste Rotwert auf das Maximum der Klassenwechsel zu einer Klasse abgebildet wird. Grün entspricht theoretisch der Farbe, bei der gar nicht zu der entsprechenden Klasse gesprungen wurde. Dennoch wurde hier stattdessen weiß verwendet, um entsprechende Klassen gar nicht hervorzuheben.

Abbildung 5.4: Farbskala für Einfärbung der Klassennamen

6 Planung der Studie

Um die gewählte Aufmerksamkeitsdarstellung und Datenauswahl zu testen, wurde im Rahmen der Arbeit eine Studie durchgeführt. Die Planung der Studie wird in diesem Kapitel näher beschrieben. Zu Beginn folgt eine Darstellung der Studienvorbereitung mittels des Goal-Question-Metric Ansatzes [37]. Anschließend werden das konkrete Design des Experiments, die Experimentobjekte, -subjekte und die Aufgaben, die für die Teilnehmer definiert wurden, vorgestellt.

6.1 Zielsetzung mittels GQM

Zu Beginn der Vorbereitung wurde zunächst das Forschungsziel der Studie konkret festgelegt. Bei der Zieldefinition wurde sich an der Empfehlung von Wohlin et al. [38] orientiert, um sicherzugehen, dass die Zielsetzung der Studie klar definiert ist.

Forschungsziel RG_1

Verbesserung des geteilten Verständnisses *innerhalb von* Softwareprojekten durch die Aufnahme, Verarbeitung und Repräsentation von Aufmerksamkeitsdaten *zur* Unterstützung asynchroner Kollaboration *von* Entwicklern *im Hinblick auf* die Effizienz und Qualität von Softwarewartungsarbeiten

Der Ansatz dieser Arbeit nutzt zur Aufnahme der Aufmerksamkeitsdaten ausschließlich Eye-Tracking. Andere Quellen von Aufmerksamkeit, wie das Interaktionsprotokoll der Entwicklungsumgebung (siehe Abschnitt 5.1) wurden nicht betrachtet, da diese bereits in anderen Ansätzen genutzt wurden (siehe Abschnitt 3.2) und so sichergestellt werden konnte, dass eine etwaige Effizienz- oder Qualitätssteigerung der Verwendung von Eye-Tracking zuzuschreiben ist.

Dieses Forschungsziel legt sich nicht auf eine konkrete Art von Softwarewartungsarbeiten fest. Wie bereits in Abschnitt 5.2 gezeigt, bieten mehrere Arten von Softwarewartungsarbeiten möglicherweise Potenzial für die Nutzung von Aufmerksamkeitsdarstellungen. Die meisten dieser Arbeiten beinhalten das Verständnis von Code, sowie das mehr oder weniger umfangreiche Editieren. Daher wurden in dieser Studie kleine Änderungs-, Erweiterungs- und Bugfixingaufgaben gestellt, um eine allgemeine Betrachtung des Verstehens und der Wartung von Code zu ermöglichen. Auf diese Art lässt sich Codeverständnis auf sehr realistische Weise untersuchen, anstatt die Teilnehmer ohne konkretes Ziel oder Motivation zu bitten den Code zu verstehen.

6.2 Forschungsfragen

Daraus ergaben sich die folgenden Forschungsfragen. Hintergrund der Darstellung von Aufmerksamkeits- bzw. Eye-Tracking Daten ist in den betrachteten Anwendungsfällen die Annahme, dass für diese Aufgaben relevantere Abschnitte auch intensiver betrachtet werden. Diese Annahme wurde mit der ersten Forschungsfrage überprüft.

Forschungsfrage RQ_1
Werden für Softwarewartungsarbeiten relevantere Abschnitte auch intensiver betrachtet?

Sollte dies der Fall sein, folgen die Fragen, ob die Darstellung dieser Aufmerksamkeitsdaten tatsächlich die Effizienz und Qualität von Softwarewartungsarbeiten verbessert.

Forschungsfrage RQ_2
Verbessert die Darstellung von Aufmerksamkeitsdaten die Effizienz von Softwarewartungsarbeiten?

Forschungsfrage RQ_3
Verbessert die Darstellung von Aufmerksamkeitsdaten die Qualität von Softwarewartungsarbeiten?

Abschließend bleibt zu klären, ob die gewählte Darstellungsform mittels Heatmaps für die Repräsentation von Aufmerksamkeitsdaten geeignet ist. Dazu gehört zum einen, ob die Heatmap überhaupt die Aufmerksamkeit der Entwickler lenkt und falls ja, ob diese Lenkung bei den Softwarewartungsarbeiten hilft oder eher stört. Sollte es zu keiner Effizienz- oder Qualitätssteigerung durch die Aufmerksamkeitsdarstellung kommen, ließe sich auf diese Weise feststellen, ob der Ansatz bereits einer fehlerhaften Grundannahme folgt und die Heatmap die Aufmerksamkeit gar nicht erst auf die hervorgehobenen Bereiche lenkt. Es könnte außerdem der Fall sein, dass die Ergebnisse der Softwarewartungsarbeiten zwar nicht hochwertiger ausfallen oder schneller erzielt werden, die Heatmap bei ihnen aber dennoch als Hilfestellung dient oder die Teilnehmer zumindest das Gefühl haben, dass sie ihnen hilft. Zudem ist interessant zu erfahren, inwiefern die Darstellung nicht nur hilft, sondern auch eventuell stört oder ablenkt und somit genau den gegenteiligen Effekt auf die Software-Wartung hätte, der erzielt werden sollte.

Forschungsfrage RQ_4

Leiten Heatmaps die Aufmerksamkeit von Entwicklern auf die farblich hervorgehobenen Bereiche?

Forschungsfrage RQ_5

Hilft die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps bei der Bearbeitung von Softwarewartungsarbeiten?

Forschungsfrage RQ_6

Stellt die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps einen Störfaktor bei der Bearbeitung von Softwarewartungsarbeiten dar?

Forschungsfrage RQ_7

Ist der Effekt der Aufmerksamkeitsdarstellung abhängig vom Erfahrungsstand des Nutzers?

6.3 Hypothesen

Während des GQM Verfeinerungsprozesses wurden Hypothesen aufgestellt, die als Quelle für Fragen und Metriken genutzt wurden und als Referenzpunkt bei der Auswertung dienten. Die erste Zahl im Index gibt dabei die Zuordnung zu der jeweiligen Forschungsfrage an. Die zweite Zahl nummeriert die Hypothesen zur eindeutigen Zuordnung in der Auswertung.

H_1: Für Softwarewartungsarbeiten relevantere Codeabschnitte werden intensiver betrachtet als für Softwarewartungsarbeiten weniger relevante Abschnitte.

$H_{2.1}$: Durch die Darstellung von Aufmerksamkeitsdaten können Softwarewartungsaufgaben schneller erledigt werden.

$H_{2.2}$: Die Darstellung von Aufmerksamkeitsdaten verändert den visuellen Aufwand, gemessen an der Sakkadenanzahl und der Sakkadendauer.

$H_{2.3}$: Durch die Darstellung von Aufmerksamkeitsdaten sinkt die kognitive Last, gemessen an dem Pupillendurchmesser und der Fixationsdauer.

$H_{2.4}$: Durch die Darstellung von Aufmerksamkeitsdaten steigt die Effizienz der Informationssuche, gemessen an der Fixationsanzahl.

H_3: Die Darstellung von Aufmerksamkeitsdaten hat einen Effekt auf die Qualität von Softwarewartungsarbeiten, gemessen an ihrer Korrektheit.

H_4: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps leitet die Aufmerksamkeit auf die hervorgehobenen Bereiche.

H_5: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps wird von den Entwicklern bei Softwarewartungsarbeiten subjektiv als Hilfestellung empfunden.

$H_{6.1}$: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps wirkt sich störend auf die Übersichtlichkeit des Codes aus.

$H_{6.2}$: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps beeinflusst wie schnell relevanter Code im Programm gefunden werden kann.

$H_{6.3}$: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps beeinflusst wie einfach die Funktionalität des Programms verstanden werden kann.

$H_{6.4}$: Die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps wirkt sich nicht störend auf die Bearbeitung von Softwarewartungsarbeiten aus.

H_7: Der Effekt der Heatmapdarstellung ist abhängig vom Erfahrungsstand des Nutzers.

6.4 Metriken

Um die oben genannten Hypothesen testen zu können, wurden die folgenden zwölf Metriken bestimmt:

- Fixationsverteilung auf Abschnitten (lokale Fixationsanzahl)
- Befragung nach Abschnittsrelevanz
- Zeit zur Fertigstellung von Softwarewartungsaufgaben in Sekunden
- Sakkadenanzahl
- Sakkadendauer
- Pupillendurchmesser
- Fixationsdauer
- Fixationsanzahl
- Korrektheit der Softwarewartungsaufgaben auf vierstufiger Ordinalskala (inkorrekt, teilweise korrekt, überwiegend korrekt, vollständig korrekt)
- Befragung nach Aufmerksamkeitslenkung, Nutzen und Störungsgrad
- Subjektive Bewertung der Übersichtlichkeit, des Findens relevanten Codes und des Funktionalitätsverständnisses
- Zeit für das Suchen eines Elements im Code

Die kognitive Last kann für diese Arbeit nur anhand der Fixationsdauer und dem Pupillendurchmesser bestimmt werden, da die verwendete Eye-Tracking Analysesoftware Tobii Pro Studio kein Blinzeln erkennt. Außerdem ist zu den Metriken zu sagen, dass bei der Befragung der Probanden nach der Relevanz und der Komplexität der Codeabschnitte alle Probanden zu Beginn die gleiche Fragestellung bekamen. Dies diente dem Vergleich der subjektiven, qualitativen Bewertung mit den objektiven, quantitativen Blickdaten. Es wurde gefragt, welche Codeabschnitte – spezifisch in den einzelnen Klassen – besonders relevant, irrelevant, kompliziert oder verwirrend waren. Währenddessen wurde den Teilnehmern der gleiche Code in der gleichen Umgebung im Browser gezeigt, sodass sie die Möglichkeit hatten, durch die Klassen zu navigieren und darauf explizit

Bezug zu nehmen. Während dieses Teils fand jedoch kein Eye-Tracking mehr statt, da der Code nur als Gedächtnisstütze und nicht zur Bearbeitung von Softwarewartungsaufgaben dienen sollte. Wenn sich einzelne Probanden mit ihrer Antwort zu sehr von der eigentlichen Fragestellung weg bewegten und stattdessen die Codequalität bewerteten oder Refactoring Vorschläge machten, wurde die gesamte Fragestellung oder Teile davon nochmals wiederholt. Es wurde sich bei dieser Metrik bewusst für eine Befragung und gegen Thinking Aloud entschieden, da ein Thinking Aloud während der Bearbeitung der Aufgaben zu sehr abgelenkt und die Teilnehmer womöglich durch die bewusste Verbalisierung zu unnatürlichen Vorgehensweisen verleitet hätte. Auch ein retrospektives Thinking Aloud wurde verworfen, bei dem die Teilnehmer ein Video mit ihrem Vorgehen, sowie ihrem Blickverhalten gezeigt bekommen hätten, um den Zeitrahmen der Studie in einem vertretbaren Rahmen zu halten.

Die Befragung nach Nutzen und Störungsgrad der Heatmapdarstellung fand zum einen mit einigen Fragen in einem Fragebogen statt. Die erste Frage adressierte die Geschwindigkeit, mit der relevante Bereiche im Code gefunden werden konnten. Die zweite Frage bezog sich darauf, wie schwierig es für den Teilnehmer war, sich ein Bild der Funktionalität der einzelnen Teile bzw. des gesamten Programms zu machen. Die dritte Frage bewertete die Übersichtlichkeit der Codedarstellung. Den Teilnehmern, die die Heatmapdarstellung gezeigt bekamen, wurden darüber hinaus Fragen gestellt, inwiefern sie die transparente Heatmap im Hintergrund, sowie die Heatmapeinfärbung der Klassennamen im Package Explorer hilfreich oder störend fanden. All diese Fragen sollten auf einer 10-stufigen Likert-Skala beantwortet werden, bei der der Wert ganz links der negierten Antwort (langsam/ schwer/ unübersichtlich/ nicht hilfreich/ nicht störend) entsprach und der Wert ganz rechts der „positiven“ Antwort. Es wurde bewusst eine gerade Anzahl an Antwortmöglichkeiten gewählt, damit die Probanden eine klare Tendenz abgeben mussten und nicht aus Unentschlossenheit die Mitte ankreuzten. Zusätzlich wurde eine Frage hinzugefügt, bei dem die Teilnehmer mitteilen konnten, ob sie eine Option zum Ausschalten der Heatmapdarstellung genutzt hätten. Außerdem war es optional möglich, einen schriftlichen Kommentar am Ende des Fragebogens abzugeben. Als weitere subjektive Metrik zur Bewertung des Nutzens und des Störungsgrads der Heatmapdarstellung wurde sich bei den Teilnehmern in der abschließenden Befragung ebenfalls erkundigt, inwiefern die Darstellung ihnen geholfen hatte, sie eventuell fehlleitete oder von ihnen ignoriert wurde.

Für die letzte Metrik, bei der ein Element im Code zu suchen war und die Zeit bis zum Finden dieses Elements gemessen wurde, wurde sich an dem First Click

Testing aus dem Usability Engineering orientiert [39]. Dabei wird getestet, auf welches Element eines User Interfaces ein Nutzer als erstes klickt, um die Effektivität des Oberflächendesigns zu evaluieren. Ein ähnliches Prinzip wurde in dieser Metrik angewandt. Der Nutzer soll jedoch ein vorgegebenes Element suchen. Dieses befindet sich nicht auf einer Webseite, sondern im Code. Damit lassen sich verschiedene Codedarstellungen miteinander vergleichen. Je übersichtlicher und weniger ablenkend die Darstellung, desto schneller kann das Element gefunden werden. Bei den Darstellungen handelt es sich hierbei einmal um die Standard Codedarstellung in Eclipse und einmal um die Aufmerksamkeitsdarstellung mittels Heatmaps im Hintergrund. Dazu wurde eine kleine Java-Anwendung geschrieben, die einen Screenshot des in der Studie verwendeten Programmcodes aus der Klasse UI.java anzeigt (Zeilen 317 bis 382). Das konkret ausgewählte Programm der Studie wird in Abschnitt 6.5.2 noch näher beschrieben. Auf diesem Screenshot soll der Proband schnellstmöglich den Aufruf der Methode saveFile finden, der sich in Zeile 359 befindet. Je stärker die halbtransparente Heatmapdarstellung ablenkt und damit die Übersichtlichkeit des Codes negativ beeinflusst, desto länger wird die Zeit bis zum Finden des Aufrufs. Abbildung 6.1 zeigt die Startansicht der „Codeelement Suchzeit“ Anwendung, in der in einem Fenster die Teilnehmer-ID und die Datei zum Speichern des Ergebnisses angegeben, sowie ausgewählt werden kann, ob der Code-Screenshot die Aufmerksamkeitsdarstellung beinhalten soll. Diese Angaben werden von der Experimentleitung von einem zweiten Bildschirm aus ausgefüllt. Sobald dies mit „OK“ bestätigt wird, sieht der Proband die Aufgabenstellung und kann anschließend auf „Test starten“ klicken, um zu der Ansicht des Code-Screenshots zu gelangen, den Abbildung 6.2 zeigt. Er versucht dann schnellstmöglich den entsprechenden Methodenaufruf zu finden und anzuklicken. Die Anwendung misst die Zeit ab der Anzeige des Screenshots bis zum Anklicken des Methodenaufrufs, welcher durch das Klick-Event gegeben ist. Anschließend zeigt sie eine Bestätigungsmeldung an, wenn die Daten bestehend aus der Teilnehmer-ID, der gemessenen Zeit und der Angabe, ob ein Screenshot mit oder ohne Aufmerksamkeitsdarstellung angeschaut wurde, erfolgreich gespeichert wurden.

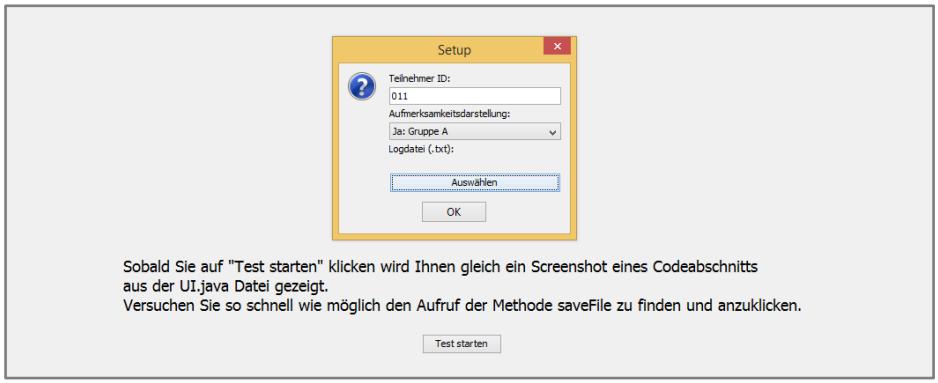


Abbildung 6.1: Startansicht der Codeelement Suchzeit Anwendung

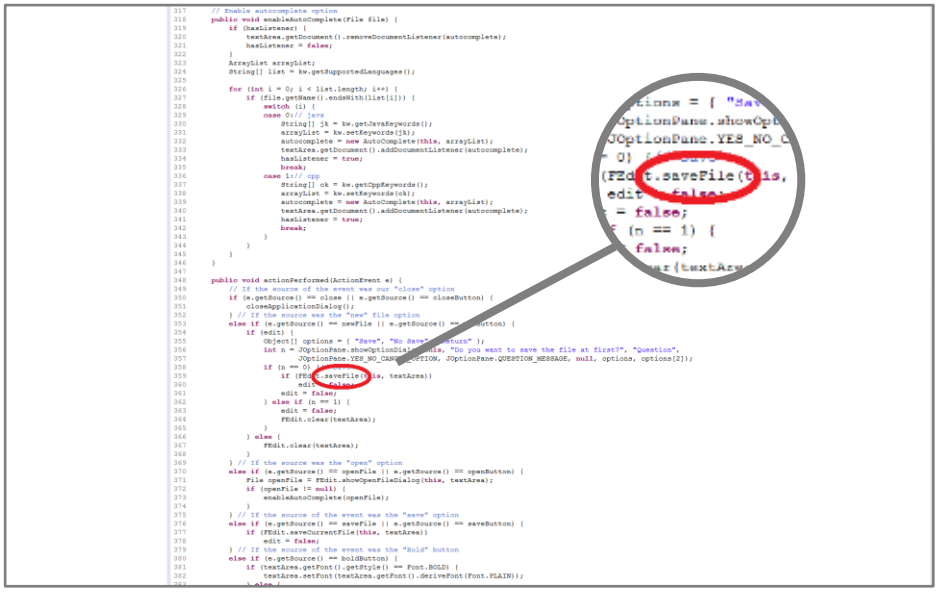


Abbildung 6.2: Codeelement Suchzeit Anwendung mit zu suchendem Element

6.5 Experimentobjekte

6.5.1 Eclipse als Webseite

Aufgrund der bereits in Kapitel 4 beschriebenen Problematik Eye-Tracking Daten von der Arbeit mit Code aufzunehmen, war ich gezwungen eine möglichst realistische Umgebung zur Arbeit mit Code im Browser zu schaffen. Dazu wurde die Eclipse Entwicklungsumgebung als statische HTML-Webseite nachgebaut. Es musste darauf geachtet werden, dass keine Unterelemente, sondern nur die gesamte Webseite gescrollt wird. Abbildung 6.3 zeigt diesen Eclipse Nachbau im Browser. Für die Webseiten wurden Screenshots von Eclipse gemacht, in denen jeweils eine der acht Klassen aus dem für die Studie ausgewählten Programm geöffnet ist. Um trotz der hohen Auflösung und Größe des Monitors, der für die Anzeige der Webseite im Experiment verwendet wurde, präzise Eye-Tracking Daten zu erhalten, wurde die Größe aller Elemente wie Menüpunkte, Beschriftungen, Symbole, etc. in den Systemeinstellungen von „Kleiner“ auf „Mittel“ geändert, während die Screenshots aufgenommen wurden. Durch die geringe

Grundauflösung der Eclipse-Elemente wurden diese durch die Vergrößerung jedoch ein wenig unscharf, sodass die Screenshots anschließend in einem Bildbearbeitungsprogramm zur Schärfung nachbearbeitet wurden.

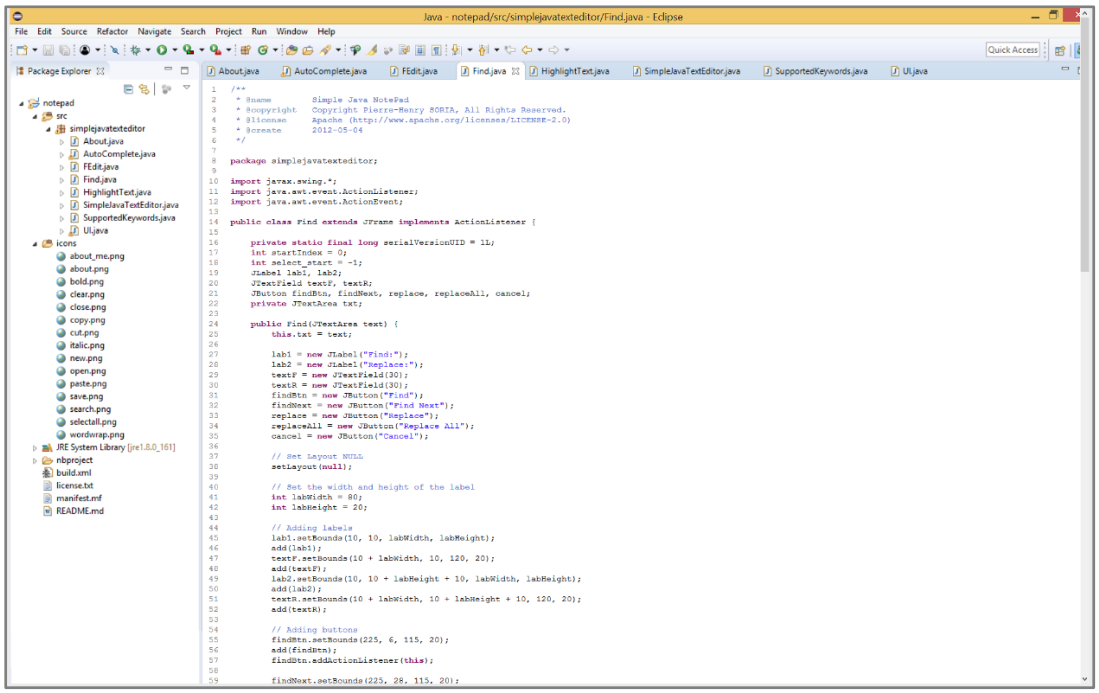


Abbildung 6.3: Scrollbare Eclipse Darstellung im Browser

Aus den Screenshots wurde der in dem jeweils offenen Tab angezeigte Code, herausgeschnitten und stattdessen ein HTML-Codeelement eingefügt. Um die Teilnehmer möglichst wenig zu beeinflussen, wurden die Daten entsprechend der alphabetischen Reihenfolge im Package Explorer geöffnet.

Zum Syntax Highlighten des Codes wurde das Highlight.js Skript verwendet [40]. Da kein Syntax Highlighter gefunden werden konnte, der tatsächlich der Standarddarstellung in Eclipse entspricht, wurde dieses in JavaScript geschriebene Skript verwendet und die Farben der Klassennamen, Kommentare, Strings, etc. entsprechend der Eclipse-Darstellung in der CSS-Datei angepasst. Selbst dann stimmt das Syntax Highlighting jedoch nicht komplett mit Eclipse überein, da der Parser des Skripts nicht zwischen Javadoc und normalen Kommentaren unterscheidet und keine Variablenbezeichner detektiert. Das gleiche gilt jedoch auch für alle anderen gefundenen Syntax Highlighting Skripts und Anwendungen. Globale und lokale Variablenbezeichner werden in Eclipse eigentlich in blau bzw. einem Braunton dargestellt. Diese bleiben durch das verwendete Syntax Highlighting Skript jedoch schwarz und lassen sich auch nicht ohne großen Aufwand anpassen. Da die Kommentare nicht differenziert werden, wurde entschieden, alle Kommentare auf den Blauton der Javadoc-Kommentare zu setzen, da diese im Code dominanter sind als die Zeilenkommentare. Beim Einfügen

von Code in die Webseite muss darauf geachtet werden, dass in HTML Tags befindliche spitze Klammern, wie sie manchmal in Javadoc oder Kommentaren zur Formatierung enthalten sind, zuvor durch < bzw. > ersetzt werden, um nicht als HTML Code interpretiert zu werden.

Zum Hinzufügen der Zeilennummerierung wurde das Highlight.js Zeilennummerierungs-Plugin highlightjs-line-numbers.js verwendet [41]. Dieses führte jedoch zu einem Konflikt mit dem verwendeten Syntax Highlighting Skript. Damit Tobii Eye Tracker eine Webseite erfolgreich aufnehmen können, muss diese über einen korrekt definierten HTML DOCTYPE verfügen. Durch die Verarbeitung des Layouts mit korrektem DOCTYPE wurden jedoch die Zeilenumbrüche und Tabstopps, die mit den HTML-Elementen durch das Zeilennummerierungs-Skript hinzugefügt werden, auch auf der Webseite dargestellt. Das Resultat wäre eine Codedarstellung mit vielen, ungewollten Zwischenräumen und Leerzeilen. Das Skript musste daher entsprechend angepasst und die Zeilenumbrüche und Tabs zwischen den HTML-Tags entfernt werden. Auch der Farbton der Zeilennummern wurde entsprechend der Eclipse Standarddarstellung geändert. Die gesamte Fontgröße des Codes und der Zeilennummern wurde auf 15 Pixel hochgesetzt, um auch auf einem größeren Monitor ausreichend präzise Eye-Tracking Daten zu erhalten und die Lesbarkeit des Codes auch bei für den Eye Tracker ausreichendem Abstand (beim verwendeten Tobii X3-120 50 bis 90cm [42]) zu gewährleisten.

Um den unteren Rahmen der Eclipse-Anwendung von der Unterkante des Bildschirms auf das Ende des scrollbaren Codebereichs zu setzen, wurde dieser von dem jeweiligen Screenshot abgeschnitten und unter das Codeelement gesetzt. Der Rahmen links vom Code wurde entsprechend bis zum Seitenende verlängert. Anschließend wurden Link-Elemente eingefügt, um zwischen den einzelnen Klassen wechseln zu können. Diese wurden auf die Größe der Dateibeschriftung im Package Explorer bzw. die Größe des Tabs gestellt und auf die jeweiligen Positionen der Klassennamen gesetzt. Auf diese Weise lassen sich die im Screenshot dargestellten Tabs und Dateien tatsächlich anklicken und es öffnet sich die ausgewählte Klasse als neue HTML-Webseite.

Die erstellte Eclipse Webanwendung ermöglicht es, Programme scrollbar und über eine Bildschirmseite hinaus darzustellen. Sie ist so nah wie möglich an der echten Entwicklungsumgebung gehalten. Dennoch stellt sie eine Art passive Codeanzeige dar. Es ist möglich, durch das gesamte Programm zu navigieren und auch in der geöffneten Klasse mittels der Tastenkombination STRG+F nach Elementen zu suchen. Es wurde sich dagegen entschieden, den Code editierbar

zu setzen, da wie bereits in Abschnitt 4.1 angemerkt, dadurch die editierten Bereiche für die Auswertung von zeitübergreifenden Eye-Tracking Daten unbrauchbar würden. Der Code lässt sich demnach weder editieren, noch ausführen oder debuggen. Auch die Anzeige von Javadoc durch das Auswählen von Objektnamen ist nicht möglich, ebenso wie das Nachverfolgen von Referenzen. Um den Probanden dennoch Hilfestellung während der Softwarewartungsarbeiten im Experiment geben zu können, wurde die Java 8 API Dokumentation [43] heruntergeladen. Diese wird beim ersten Öffnen der Webdarstellung in einem zweiten Tab im Hintergrund geöffnet. Die Art und Weise wie Tabs geöffnet werden, ist jedoch vom Browser abhängig und nicht durch Webseiten veränderbar. Daher wurden in dem Firefox Browser, der für das Experiment verwendet wurde, die Einstellungen so angepasst, dass das Öffnen und Laden von Tabs im Hintergrund erlaubt wird. Dazu wurde in der Konfigurationsdatei das Element **browser.tabs.loadDivertedInBackground** auf **true** gesetzt. Um zu verhindern, dass während des Experiments ein Fenster erscheint, das über die fehlgeschlagene Suche nach Updates informiert, wurde zudem die automatische Suche nach Updates deaktiviert.

Während der Anzeige der Eclipse-Webseite sollten Eye-Tracking Daten aufgenommen und unter anderem gemessen werden, wie häufig bestimmte Klassen und Codeabschnitte angeschaut wurden. Um zu verhindern, dass die Daten für die als erstes geöffnete Klasse beeinflusst werden, wurde ein zusätzlicher Screenshot als Webseite erstellt. In diesem wird das Programm in Eclipse gezeigt, während alle Klassen noch geschlossen, jedoch dennoch zur Auswahl in den Tabs verfügbar sind. Diese dient als Startseite, aus der die Probanden selbst wählen können, welche Klasse sie als erstes anschauen möchten. Um Ablenkung durch die Menüleiste des Browsers zu verhindern und die „Eclipse-Illusion“ so gut wie möglich aufrecht zu erhalten, wurde Firefox in dem Eye-Tracking Experiment im Vollbildmodus geöffnet.

6.5.2 Codeauswahl

Bei der Auswahl des Codes, der während der Studie in Eclipse angezeigt und gewartet werden sollte, musste auf verschiedene Kriterien Rücksicht genommen werden. Ein für die Studie geeignetes Programm sollte demnach die folgenden fünf Punkte erfüllen:

1. Maximal neun relevante Dateien
2. Unabhängig vom Vorwissen und Hintergrund der Probanden und ohne die Möglichkeit des Testens oder Ausführens gleichermaßen verständlich
3. Nicht zu klein oder trivial

4. Mitarbeit mehrerer Entwickler
5. In Java geschrieben

Der erste Punkt rührt daher, dass alle Klassen des Programms in der Eclipse Webversion parallel in Tabs darstellbar sein mussten. Bei mehr als neun Dateien würde der Package Explorer zu schmal und die Package- und Klassennamen darin nicht mehr lesbar werden oder die Klassen werden in Eclipse klassischerweise in Listen am rechten Rand der Tableiste zusammengefasst. Dies Eclipsegetreu zu ermöglichen, hätte die Webseitenfunktionalität, die im Rahmen dieser Arbeit umsetzbar war, jedoch überschritten. Darüber hinaus musste zusätzlich darauf geachtet werden, dass das verwendete Programm in einem für die Studie angemessenen Zeitrahmen ausreichend verstanden und bearbeitet werden konnte. Programme, die neun relevante Klassen überschreiten, wären daher ohnehin wenig geeignet, insbesondere da es nicht möglich war, in allen Klassen gleichzeitig zu suchen.

Bei Software, die den dritten Punkt nicht erfüllt, würde es zum einen schwierig werden, realistische Softwarewartungsaufhaben zu stellen und zum anderen wäre das Hervorheben relevanter Codeabschnitte in dem Fall redundant. Idealerweise sollte es sich zudem um ein Projekt handeln, an dem mehrere Entwickler mitgearbeitet haben, da kollaborative Projekte die Zielgruppe für die Übertragung und Darstellung von Aufmerksamkeitsdaten darstellen. Abschließend sollte das Programm darüber hinaus in Java geschrieben sein, da dies die zurzeit hauptsächlich gelehrte Programmiersprache an der Leibniz Universität Hannover ist, von der der Großteil der Teilnehmer stammt.

Ich habe mich daher für einen Texteditor mit Syntax Highlighting und Autovervollständigung von Java und C++ Keywords entschieden. Das Projekt „Simple Java Text Editor“ ist unter der Apache License öffentlich auf GitHub verfügbar [44]. Neben dem Autor Pierre-Henry Soria haben fünf weitere Personen an dem Programm mitgearbeitet. Abbildung 6.4 zeigt einen Screenshot des Texteditors mit einer geöffneten Beispieldatei, in der die Java Keywords durch graues Highlighting hervorgehoben sind.

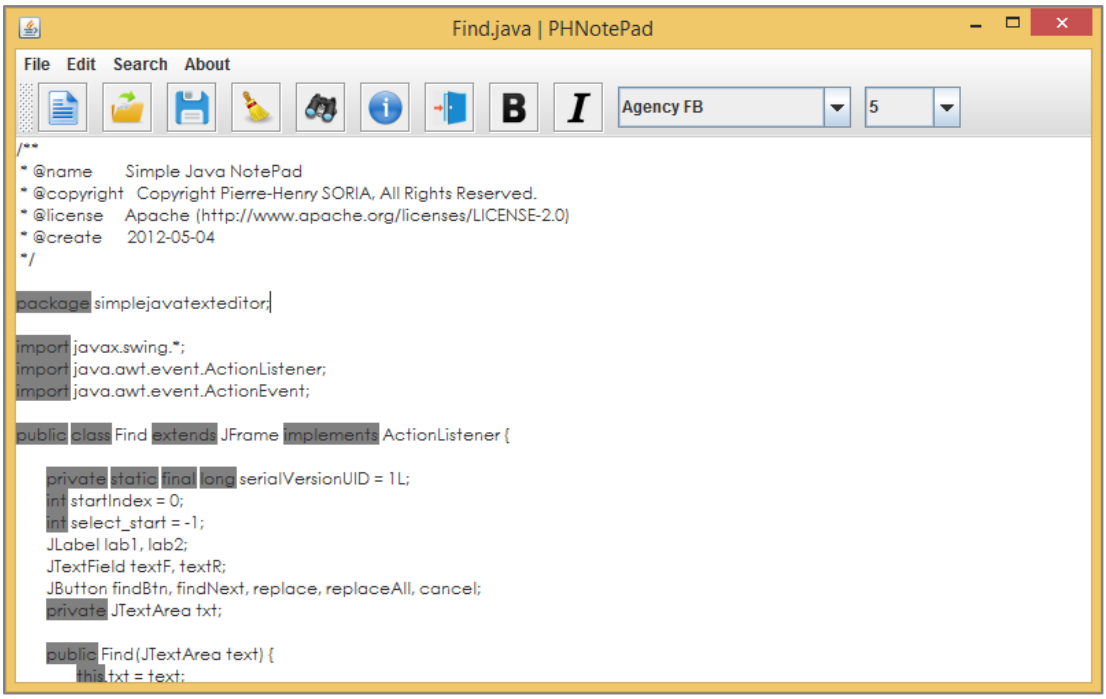


Abbildung 6.4: Screenshot des in der Studie verwendeten Texteditors

Um besser für diese Studie geeignet zu sein, wurden an dem Code kleine Änderungen vorgenommen. Zum einen wurden die teils sehr langen Javadoc Kommentare am Anfang der Klassen gekürzt. Für die Studie irrelevante und zu sehr ablenkende Informationen dazu, wann der Code von wem editiert wurde, wurden in dem Zuge entfernt. Ein Bug, bei dem in einer Fallunterscheidung Java und C++ vertauscht wurden, wurde zudem behoben. Des Weiteren wurden alle Klassen auf die Standard Eclipse Formatierung gesetzt. Abschließend musste die Klasse UI.java, die im Original 587 Zeilen lang ist und nach den soeben genannten Anpassungen noch immer 547 Zeilen umfasste, gekürzt werden. Dadurch dass der Eye Tracker eine Längenbegrenzung für Webseiten vorgibt, durften Klassen in den erstellten Eclipse Webseiten maximal 447 Zeilen lang sein. Bei Webseiten, die diese Grenze überschreiten, würde der Screenshot in der Eye-Tracking Analyse darüber hinaus nur noch weißen Hintergrund anzeigen. Um die UI.java Klasse entsprechend zu kürzen, wurden zwei Methoden in die Klasse FEdit.java ausgelagert und zwei Hilfsmethoden geschrieben, um sich wiederholende Teile im Konstruktor bzw. zwei anderen Methoden zusammenfassen zu können. Zusätzlich wurden die Informationen, die über die Buttons „About“ und „About me“ angezeigt werden, in einem Infodialog und einem Button „About“ zusammengefasst, da detaillierte Hintergrund- und Kontaktinformationen zum Autor des Programms für die Studie nicht von großem Belang waren. Das Programm in der Version wie es letztendlich in dem Experiment verwendet wurde, umfasste auf acht Klassen verteilt 970 LOC (ohne Leerzeilen), davon 176 Zeilen Kommentare.

6.5.3 Softwarewartungsaufgaben

Der Fokus des Experiments lag auf der Untersuchung von Codeverständnis, da das Verstehen von Code zum einen die Grundlage für alle weiteren Softwarewartungsarbeiten darstellt, einen großen Bestandteil der Entwicklertätigkeit ausmacht und des Weiteren am realistischsten mit der passiven Codeanzeige in der erstellten Webversion von Eclipse umzusetzen ist. Zudem wird das Potenzial der Darstellung von Aufmerksamkeitsdaten für das Codeverständnis im Vergleich zu anderen Softwarewartungsaufgaben als besonders hoch eingeschätzt.

Um den Probanden eine realitätsnahe Motivation zu geben, das Programm zu verstehen, wurden insgesamt zwölf Softwarewartungsaufgaben definiert. Vier davon sind Änderungsaufgaben, vier Erweiterungen und vier Fehlerbehebungen. Die Aufgaben sind sowohl während der Verwendung des Texteditors, als auch beim Vergleich mit anderen Texteditoren und der Analyse der Commit History des Projekts auf GitHub entstanden. Es handelt sich somit um Fehler oder Feature-Wünsche, die bei der echten Nutzung des Programms aufgekommen sind, nicht um künstlich erstellte, realitätsferne Aufgaben. Alle Aufgaben setzen das Verständnis des Codes voraus und stellen damit eine gute Möglichkeit dar, den Effekt der Aufmerksamkeitsdarstellung bei Softwarewartungsarbeiten mit Fokus auf Codeverständnis zu untersuchen. Im Zuge des angestrebten „By-Product“ Ansatzes und entgegen verwandter Arbeiten, die Eye-Tracking Daten von Personen bei den gleichen Aufgaben darstellten, wurden die Aufgaben in zwei Gruppen aufgeteilt. Auf diese Weise lassen sich die Aufmerksamkeitsdaten, die während der Bearbeitung der Aufgaben der ersten Gruppe aufgenommen wurden, während der Bearbeitung der Aufgaben der zweiten Gruppe visualisieren – und umgekehrt. Die dargestellten Daten stammen damit von anderen Aufgaben als den von dem Teilnehmer aktuell zu bearbeitenden. Auf diese Weise wurde eine realistischere Umgebung geschaffen, da in der Praxis selten die gleichen Aufgaben mehrfach bearbeitet werden. Das Ziel ist es, somit den Entwicklern durch die Aufmerksamkeitsdarstellung eine Hilfestellung zu geben, obwohl die Daten in einem anderen Aufgabenkontext aufgenommen wurden.

Tabelle 6.1 zeigt eine Auflistung aller Aufgaben mit zugeordneter Gruppe, Nummer und einem kurzen Titel, der beschreibt, worum es grob bei der Aufgabe geht. Zusätzlich ist angegeben, ob es sich bei der Aufgabe um eine Erweiterung, eine Änderung oder eine Fehlerbehebung handelt. Es wurde darauf geachtet, dass selbst die Erweiterungsaufgaben dennoch ein gewisses Verständnis des bestehenden Codes voraussetzen. Eine ausführliche Beschreibung der gestellten Aufgaben, sowie Musterlösungen, befinden sich im Anhang (Abschnitt A.2). Es

handelt sich um kleine, relativ schnell zu lösende Aufgaben, um Aufmerksamkeitsdaten mit einem möglichst weit gestreuten Aufgabenkontext zu erhalten. Dies entspricht eher der Realität, da die Daten dort ebenfalls aus unterschiedlichen Aufgabenkontexten stammen würden. Der Zeitaufwand der Aufgaben wurde in einem Testlauf evaluiert, infolgedessen das Aufgabenpensum für einen Studiendurchlauf von acht auf sechs Aufgaben reduziert wurde.

Aufgabengruppe, –Nr. und –typ	Aufgabentitel	Aufgabengruppe, –Nr. und –typ	Aufgabentitel
A1 - Änderung	Highlightcolor	B1 - Änderung	Keywordliste
A2 -Bugfix	Schriftart- und Schriftgrößen-combobox	B2 - Bugfix	Single Word Highlighting
A3 - Erweiterung	Scrolling	B3 - Erweiterung	Brackets Autocomplete
A4 - Bugfix	Highlight beim Suchen und Ersetzen	B4 - Bugfix	Highlight Pattern
A5 - Änderung	E-Mailadresse des Autors	B5 - Erweiterung	AutoComplete in neuer Datei
A6 - Erweiterung	Fett und Kursiv	B6 - Änderung	Default Schriftart und -größe

Tabelle 6.1: Softwarewartungsaufgaben

6.6 Experimentsubjekte

Es handelt sich bei den Teilnehmern um acht Bachelor oder Master Studenten, acht PhD Studenten der Informatik, sowie sechs Experten mit Java Softwareentwicklungserfahrung aus der Praxis. Der Einfachheit halber werden Bachelor und Master Studenten in dieser Arbeit als Studenten bezeichnet und Doktoranden werden explizit als PhD Studenten genannt. Die Studenten befanden sich entweder im Masterstudiengang Informatik oder technische Informatik oder im sechsten Bachelorsemester Informatik. Die Teilnehmer wurden gefragt, ob sie auf freiwilliger Basis und ohne Entgelt an der Studie teilnehmen würden. Das einzige Auswahlkritium war dabei über ein gewisses Mindestmaß an Javaentwicklungserfahrung zu verfügen.

Alle Teilnehmer hatten zwischen wenig und sehr viel Entwicklungserfahrung, insbesondere mit der Programmiersprache Java und hatten Erfahrung mit der Aufgabe, Code verstehen zu müssen, den sie nicht selbst geschrieben haben. Es wurde außerdem sichergestellt, dass keiner der Probanden von einer Rot-Grün-Sehschwäche betroffen ist, damit die Heatmapdarstellung korrekt wahrgenommen werden konnte. Bei fünf der 22 Teilnehmer handelte es sich jedoch um

Brillenträger. Auf Brillenträger muss beim Eye-Tracking besonderes Augenmerk gelegt werden, da das Tragen einer Brille während der Studie zu Ablenkungen des Infrarotstrahls und daher zu fälschlich oder nicht erkannten Blickdaten kommen kann.

6.7 Design des Experiments

Es handelt sich bei der Studie um ein kontrolliertes Experiment mit Between-Group Design. Dazu wurde das Experiment in zwei Durchläufe aufgeteilt. Im ersten Durchlauf wurden von der Kontrollgruppe an Teilnehmern Eye-Tracking Daten während der Bearbeitung der definierten Softwarewartungsaufgaben aufgenommen. Im zweiten Durchlauf wurden diese Daten dann der Treatmentgruppe an Teilnehmern aufbereitet und visuell dargestellt, während diese die gleichen Softwarewartungsarbeiten bearbeiteten. Das heißt, die Teilnehmer der Treatmentgruppe, die die Aufgabengruppe A bearbeiteten, erhielten die Heatmapdarstellung der Aufmerksamkeitsdaten der Teilnehmer der Kontrollgruppe, die zuvor die Aufgabengruppe B bearbeitet hatten – und umgekehrt. Im Folgenden werden die Probanden, die die Arbeiten an dem Code ohne Aufmerksamkeitsdarstellung ausführten, als Kontrollgruppe bezeichnet. Die Teilnehmer des zweiten Durchlaufs, die den Code mit Heatmapdarstellung sahen, werden als Treatmentgruppe genannt.

Im Anschluss wurde geprüft, ob eine Effizienz- oder Qualitätssteigerung der Wartungsarbeiten der zweiten Gruppe im Vergleich zur ersten Gruppe gemessen werden konnte. Da die Daten der Kontrollgruppe zunächst vollständig aufgenommen worden sein mussten, war es nicht möglich die Reihenfolge der Teilnehmer der Kontrollgruppe und der Treatmentgruppe randomisiert zu mischen. Um einen Informationsaustausch der Teilnehmer untereinander zu verhindern, durch den spätere Teilnehmer einen Vorteil erhalten hätten, wurden alle Teilnehmer in der Einverständniserklärung zu Beginn zur Geheimhaltung der Studieninhalte angehalten.

6.8 Aufgabenzuteilung

Die Aufgaben wurden in randomisierter Reihenfolge den Teilnehmern zugewiesen. Dennoch wurde zum Ausgleich darauf geachtet, dass jede Aufgabe nicht zweimal an der gleichen Position vorkommt. Tabelle 6.2 zeigt die Zuteilung der Aufgaben zu den Teilnehmern. Die Zeilen zeigen die Aufgaben mit Gruppenbezeichnung A oder B und Nummer. Jede Spalte stellt dabei einen Durchlauf dar und die Aufgaben sind von oben nach unten entsprechend ihrer Reihenfolge in

der Studie angegeben. Jeder Spalte sind zwei Teilnehmer mit ihren IDs zugewiesen. Der erste dieser Teilnehmer stammt aus dem ersten Durchlauf und der zweite aus dem zweiten Durchlauf. Die Teilnehmer mit den IDs 001 bis 010 entsprechen den zehn Probanden, die an dem ersten Studiendurchlauf teilnahmen. Von diesen Teilnehmern wurden die Eye-Tracking Daten erhoben, um sie der zweiten Gruppe visuell aufbereitet darstellen zu können. Der Teilnehmer mit der ID 021 wurde bereits nach Abschluss des ersten Durchlaufs hinzugefügt. Seine Daten flossen daher nicht mit in die Heatmapdarstellung. Dennoch diente er als einer der Vergleichswerte in der Analyse. Die Teilnehmer mit den IDs 011 bis 020, sowie 022 bilden die zweite Gruppe. Beide Gruppen bestehen daher aus gleich vielen Subjekten, um Balancing zu gewährleisten. Die Teilnehmer bearbeiteten abwechselnd die Aufgaben der Gruppen A und B, um etwa gleich viele Werte pro Gruppe zu erhalten. Zudem entspricht die Reihenfolge und Zuteilung der Aufgaben während des ersten Durchlaufs exakt der Reihenfolge im zweiten Durchlauf, um identische und damit vergleichbare Umstände zu schaffen. Zusätzlich wurde darauf geachtet, dass Teilnehmer der ersten und zweiten Gruppe, die die gleiche Aufgabengruppe und –reihenfolge bearbeiteten (also eine Spalte in Tabelle 6.2 bilden) den gleichen Erfahrungsstand hatten. Wenn also beispielsweise der Teilnehmer mit ID 004 ein Student war, so wurde sichergestellt, dass auch der Teilnehmer mit ID 014 ein Student war, um sicherzugehen, dass die beiden Datensätze am Ende valide verglichen werden können. Das gleiche wurde für PhD Studenten und Experten vorgenommen.

Teilnehmer IDs	001 011	002 012	003 013	004 014	005 015	006 016	007 017	008 018	009 019	010 020	021 022
Aufgaben	A2	B6	A3	B3	A4	B5	A6	B2	A1	B4	A5
	A1	B3	A2	B1	A5	B4	A4	B6	A3	B2	A6
	A3	B5	A6	B4	A2	B6	A1	B3	A5	B1	A4
	A6	B2	A4	B5	A3	B1	A5	B4	A2	B3	A1
	A4	B1	A5	B2	A1	B3	A3	B5	A6	B6	A2
	A5	B4	A1	B6	A6	B2	A2	B1	A4	B5	A3

Tabelle 6.2: Aufgabenzuteilung der Teilnehmer

7 Aufbereitung der Aufmerksamkeitsdaten

Nach Ablauf des ersten Durchlaufs für die Kontrollgruppe wurden die aufgenommenen Aufmerksamkeitsdaten verarbeitet und für die Treatmentgruppe repräsentiert. Die Vorgehensweise bei diesem Schritt wird in diesem Abschnitt beschrieben.

Zunächst wurden die Heatmaps aller Teilnehmer aus dem ersten Durchgang mit den IDs 001 bis 010 aus dem Tobii Pro Studio exportiert. Anschließend wurden aus dem Aufnahmeprotokoll die Wechsel zwischen den Webseiten protokolliert, welche die Wechsel zwischen den Klassen darstellen, da jede Webseite genau eine Programmdatei beinhaltet. Daraus wurden wie in Abschnitt 5.4 beschrieben, Heatmap Farben berechnet, welche die Klassenwechsel abhängig von der relativen Anzahl farblich darstellen. In diesen Farben wurden in der Eclipse Webversion für den zweiten Durchgang die Klassennamen im Package Explorer eingefärbt. Diese können den Entwicklern Hinweise geben, welche Klassen im Anschluss an die aktuelle Klasse voraussichtlich relevant sind.

Die Heatmaps, sowie die Farben für die Klassennamen wurden in die erstellte Webversion von Eclipse eingepflegt. Das Ergebnis zeigt Abbildung 7.1. Darin ist ein Screenshot einer der Klassen des verwendeten Texteditor-Programms in der Eclipse Webdarstellung zu sehen. Der Screenshot zeigt außerdem aus welchen Elementen sich die Webseite für die Variante mit der Heatmapdarstellung zusammensetzt. Die Nummern an den Elementbeschriftungen geben dabei die Layoutebene in dem HTML-Dokument an. Je größer die Zahl, desto weiter liegt das Element im Vordergrund. Ganz hinten befindet sich die Heatmap für die jeweilige Klasse mit 30% Transparenz. Darüber befindet sich der ausgeschnittene Screenshot der Eclipse Menüleiste und dem Package Explorer. Da sich der untere Rahmen in Eclipse normalerweise an der Unterkante des Bildschirms befindet, musste dieser ebenfalls abgeschnitten und bis zum Webseitenende verlängert werden. In der darüber liegenden Ebene befindet sich der Quellcode der Klasse mit transparentem Hintergrund. Abschließend sind darauf im Vordergrund die transparenten Links in den Tabs, sowie die mit 20% Transparenz farblich markierten Links im Package Explorer.

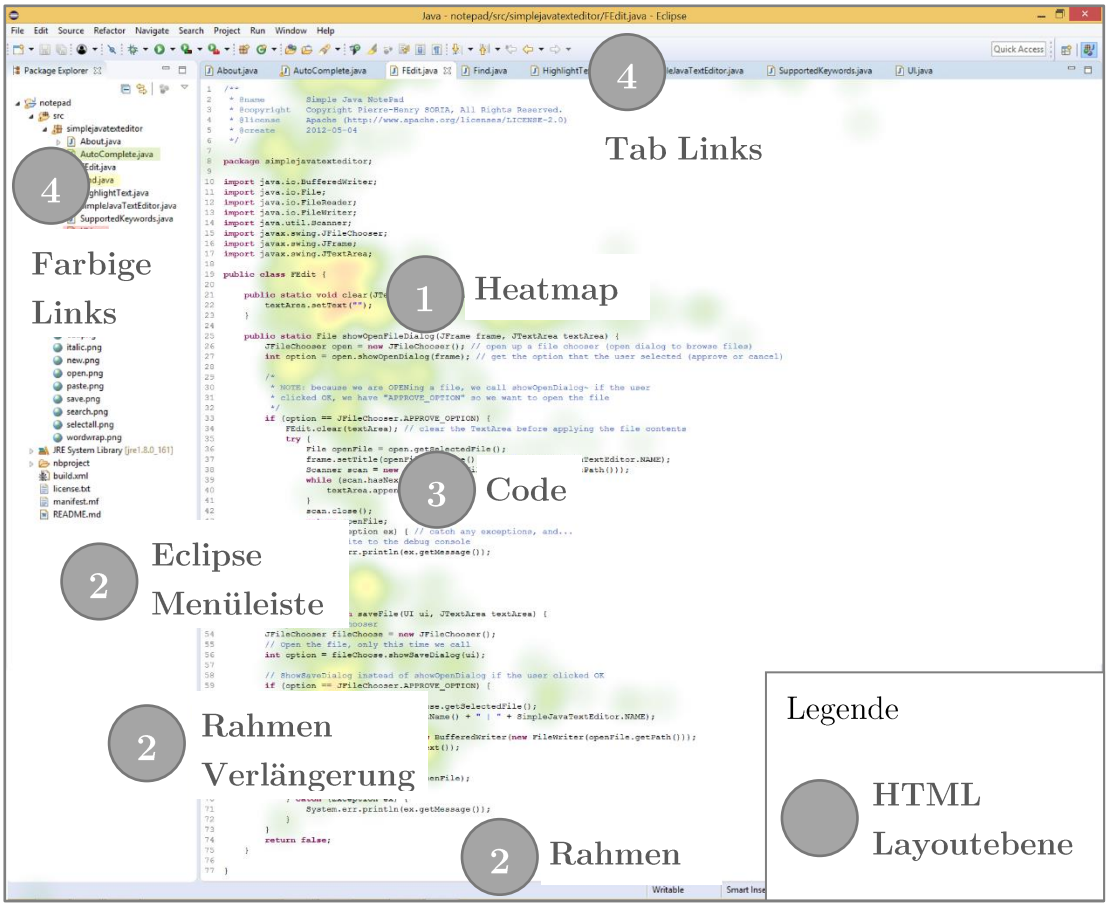


Abbildung 7.1: Elemente der Aufmerksamkeitsdarstellung in der Eclipse Webseite

Zur besseren Veranschaulichung sind in der folgenden Abbildung 7.2 beispielhaft zwei Screenshots einer Klasse in der Eclipse Webversion dargestellt. Es handelt sich um die gesamte Webseite und nicht nur um den sichtbaren Bereich. Die Scrollbar am rechten Rand verdeutlicht wie groß der Bereich ist, zu dem erst gescrollt werden muss, damit er sichtbar wird. Die gestrichelte Linie dient dazu, diese Abgrenzung noch mehr hervorzuheben. Gezeigt ist die Klasse AutoComplete.java einmal links mit der zusammengefassten Heatmap aller Teilnehmer, die im ersten Durchgang die Aufgaben der Gruppe A bearbeiteten (IDs 001, 003, 005, 007, 009) und rechts mit der Heatmap der Teilnehmer, die die Aufgaben der Gruppe B bearbeiteten (IDs 002, 004, 006, 008, 010). Die Bilder zeigen einerseits wie die Teilnehmer der Treatmentgruppe den Code dargestellt bekamen und andererseits verdeutlichen sie exemplarisch die Abhängigkeit der Heatmapdarstellung von dem Aufgabenkontext. Man sieht, dass die Teilnehmer, die die Aufgabengruppe A bearbeiteten, deutlich mehr auf den Anfang der Klasse schauten, inklusive dem Javadoc Kommentar zu Beginn der Klasse. Die Teilnehmer der Aufgabengruppe B hingegen schenkten diesem Kommentar weniger Beachtung und achteten dafür mehr auf die Methode keyTyped in der privaten Unterklasse HandleBracketEvent. Dies ist höchstwahrscheinlich der Tatsache

geschuldet, dass nur die Aufgabengruppe B eine Aufgabe enthielt, bei der die Klasse AutoComplete aktiv bearbeitet werden musste. In Aufgabenkontext A wurde diese Klasse hingegen nicht benötigt und die wenigen Blickdaten darauf entstanden vermutlich, als Teilnehmer sich dennoch ein grobes Bild der Klasse und ihrer Aufgabe machen wollten. Aus diesem Grund ist die Heatmap im linken Screenshot auch leicht gefleckter als im rechten Screenshot, da sie aus weniger Datenpunkten entstand. Der Unterschied zwischen den beiden Kontexten wird auch in der Klasseneinfärbung im Package Explorer deutlich. Im linken Screenshot des Aufgabenkontexts A ist die Klasse SupportedKeywords beispielsweise weiß hinterlegt, was bedeutet, dass diese von keinem der Teilnehmer im Anschluss an die aktuell gezeigt Klasse AutoComplete geöffnet wurde. Im rechten Screenshot ist sie hingegen rot gefärbt und ist damit die am häufigsten im Anschluss angewählte Klasse. Auch dies liegt wieder daran, dass in den Aufgaben der Gruppe B die Klassen AutoComplete und SupportedKeywords in einer Aufgabe beide editiert werden mussten.

Die restlichen Einfärbungen dagegen ähneln sich stark und auch bei Betrachtung der Heatmaps der anderen Klassen finden sich bei allgemeinen Bereichen, wie dem Konstruktor, Javadoc Kommentaren und elementaren Funktionen ähnliche Färbungen. Lediglich bei Abschnitten, auf die für einzelne Aufgaben sehr stark fokussiert wurde, gibt es Abweichungen.

Abbildung 7.2: Heatmapdarstellungen in Eclipse im Browser (links: Aufgabenkontext A, rechts: Aufgabenkontext B)

8 Durchführung der Studie

In diesem Kapitel wird die Durchführung der Studie näher beschrieben. Dazu wird zunächst der konkrete Ablauf und der Vorgang der Datenaufnahme vorgestellt und im Anschluss die Experimentumgebung spezifiziert. Danach wird genannt, aus welchen Gründen einige Teilnehmerdurchläufe wiederholt werden mussten. Abschließend werden noch einige allgemeine Beobachtungen, die während des Experiments gemacht wurden, aufgeführt.

8.1 Ablauf der Studie

Zu Beginn der Studie wurde den Teilnehmern zunächst eine Einverständniserklärung (siehe Abschnitt A.1.1 im Anhang) vorgelegt, in der sie über den Inhalt der Studie und die Art der Datenaufnahme und –verarbeitung aufgeklärt wurden und diesen zustimmten. Anschließend wurden sie gebeten den ersten Fragebogen zu beantworten (siehe Abschnitt A.1.4 im Anhang). Dieser enthält allgemeine Fragen zum Hintergrund und dem Erfahrungsstand, um die Probanden einordnen zu können. Zusätzlich wurde sich darin erkundigt, ob der Teilnehmer eine Rot-Grün-Sehschwäche habe, da in dem Fall die Wirkung der Heatmapdarstellung stark beeinträchtigt wäre.

Darauf folgend wurde mit dem Eye-Tracking Teil begonnen. Dazu wurde darauf geachtet, dass die Teilnehmer in richtigem Abstand zum Bildschirm bzw. dem Eye Tracker sitzen. In wenigen Fällen konnten die Pupillen von der Eye-Tracking Software zunächst nicht richtig erkannt und nur flackernde Daten dargestellt werden. Ein einfacher Neustart des Tobii Pro Studios konnte dieses Problem jedoch jedes Mal lösen. Dann begann das Eye-Tracking Experiment mit einer 5-Punkte Kalibrierung. Nach erfolgreicher Kalibrierung wurde den Teilnehmern eine Einleitung in die Aufgabenstellung gegeben, sowie im Falle der Treatmentgruppe eine Erklärung der Heatmapdarstellung (siehe A.1.2 und A.1.3 im Anhang). In der Einleitung wurde den Teilnehmern ihre Rolle und Aufgabe und die Art der gewünschten Lösung beschrieben. Es wurde explizit kein syntaktisch korrekter Javacode gefordert, da die Programmierleistung und nicht die Javakenntnisse bewertet werden sollten und das Schreiben von korrektem Code ohne die Hilfe der Entwicklungsumgebung darüber hinaus stark erschwert wird. Zudem wurde mitgeteilt, dass es sich um insgesamt sechs Aufgaben handelte, die sie nacheinander innerhalb von maximal 50 Minuten bearbeiten sollten. Das Zeitlimit diente zum einen dazu, den Zeitrahmen der Studie zu begrenzen und zum anderen realistische Verhältnisse zu schaffen, da man in der Praxis ebenfalls häufig unter Zeitdruck steht und die Probanden auf diese Weise dazu angehalten

waren, die Aufgaben möglichst schnell zu lösen. Es war möglich eine Aufgabe abzubrechen, jedoch nicht zu einer abgebrochenen Aufgabe noch einmal zurück zu kehren. Die Zeiteinteilung wurde den Teilnehmern selbst überlassen. Erst wenn sich sehr lange (länger als 15-20 Minuten) an einer Aufgabe aufgehalten wurde und keine Lösung in Sicht schien, wurde dem Probanden ein entsprechender Hinweis gegeben auf die nächste Aufgabe überzugehen. Bei der Beschreibung der Aufmerksamkeitsdarstellung wurde lediglich die Bedeutung der Heatmap erklärt und die Probanden wurden bewusst nicht darauf hingewiesen diese als Hilfestellung zu nutzen, um ihnen dies selbst zu überlassen.

Zu Beginn der 50 Minuten Bearbeitungszeit erhielten die Teilnehmer einen Screenshot der Javaanwendung, um sich ein Bild der Softwareoberfläche machen zu können, auch ohne das Programm ausführen zu können. Die Aufgaben wurden den Probanden nacheinander, jeweils auf Bescheid des Teilnehmers auf Papier ausgeteilt. Aufgrund der passiven, nicht editierbaren Codedarstellung im Browser wurden die Aufgabenlösungen auf Papier abgegeben. Zusätzlich stand den Teilnehmern in einem zweiten Tab die Java 8 API Dokumentation als Hilfestellung zur Verfügung, um zu verhindern, dass Teilnehmer eine Aufgabe nur nicht lösen können, weil ihnen das Wissen über die Umsetzungsmöglichkeiten oder einen bestimmten Funktionsumfang in Java fehlte.

Nach Bearbeitung aller Aufgaben oder Ablauf des Zeitlimits, erhielten die Teilnehmer den zweiten Fragebogen (siehe A.1.5 im Anhang) mit den im Metrikteil (Abschnitt 6.4) beschriebenen Fragen. Anschließend folgte die Befragung nach Relevanz und Komplexität der Codeabschnitte, sowie im Falle der Treatmentgruppe dem Nutzen der Heatmapdarstellung. Dieser Teil wurde mithilfe eines Mikrofons aufgezeichnet. Vor Start der Aufnahme wurden die Teilnehmer erneut gefragt, ob sie mit der Tonaufzeichnung einverstanden waren. Als letzter Teil des Experiments folgte der Codeelement Suchtest. Insgesamt dauerte das Experiment pro Proband etwa eine Stunde.

8.2 Datenaufnahme

Sämtliche Eye-Tracking Metriken wurden mit dem Tobii Pro Studio aufgenommen. Dazu wurde die erstellte Eclipse Startseite als Web Stimulus im Vollbildmodus aus einem Experiment im Tobii Pro Studio geöffnet und während der Bearbeitung der Softwarewartungsaufgaben die Eye-Tracking Daten im Hintergrund aufgenommen, ohne den Teilnehmer in irgendeiner Weise zu stören oder abzulenken. Die Bearbeitungszeiten der Aufgaben wurden mittels einer Stoppuhr gemessen. Dazu wurde jedes Mal, wenn der Proband Bescheid gab, mit der aktuellen Aufgabe fertig zu sein oder diese abbrechen zu wollen, der aktuelle

Zeitpunkt notiert und im Anschluss die Zeitdistanzen berechnet. Die Lösungen der Aufgaben wurden auf Papier abgegeben. Jede Aufgabe wurde auf einem neuen Blatt Papier gelöst, sodass sichergestellt werden konnte, dass sich die Bearbeitungen der Aufgaben nicht überschnitten und nicht noch einmal zu einer beendeten Aufgabe zurückgegangen werden konnte.

Die Audioaufnahmen der Befragung im Anschluss an den Eye-Tracking Teil wurden mit einem externen Tischmikrofon aufgenommen. Die Codeelement Suchzeit wurde zudem mit der bereits in Abschnitt 6.4 beschriebenen Hilfsanwendung aufgenommen und die Zeit mit der jeweiligen Teilnehmer-ID in einer .txt Logdatei gespeichert. Sämtliche Daten wurden nur mit der zugewiesenen Teilnehmernummer und ohne personenbezogene Daten gespeichert. Auf diese Weise wurde sichergestellt, dass die Auswertung komplett anonym erfolgte.

8.3 Experimentumgebung

Das Experiment fand in einem geschlossenen, störungsfreien Raum statt. Durch die Fenster eventuell einfallendes Sonnenlicht wurde durch Jalousien abgehalten, da die im Sonnenlicht enthaltene Infrarotstrahlung zu Störungen beim Eye-Tracking führen könnte und zudem das Arbeiten am Bildschirm erschwert. Die Teilnehmer arbeiteten an einem Monitor mit einer Auflösung von 1920x1200 Pixeln. Darüber hinaus standen ihnen eine Maus mit Scrollrad und eine Tastatur zur Verfügung. Für das Eye-Tracking wurde der Tobii Pro X3-120 Eye Tracker verwendet. Die Eclipse Webversion wurde als Webseite im Firefox Browser in der Version 46.0 im Vollbildmodus dargestellt. Bei dieser Version handelt es sich um die neueste Firefox Version, die aktuell vom Tobii Pro Studio unterstützt wird.

8.4 Wiederholung von Teilnehmerdurchläufen

Die Experimentdurchläufe der Teilnehmer mit den IDs 004 und 016 mussten wiederholt werden, da die Bachelorstudenten in dem Fall deutlich sichtbar mit den gestellten Softwarewartungsaufgaben überfordert waren. Es bestanden große Schwierigkeiten, die relevanten Codestellen für die Aufgaben zu finden bzw. wurde teilweise bereits die Aufgabenstellung grundlegend missverstanden. Die erhobenen Daten wären daher nicht für die Zielgruppe repräsentativ gewesen. Aus diesem Grund wurden die Durchläufe nochmals mit Masterstudenten wiederholt, die die Aufgaben deutlich besser lösen und sich in dem Javacode zurecht finden konnten.

Darüber hinaus musste der Durchlauf des ersten Teilnehmers mit der ID 015 abgebrochen werden, da er während des Eye-Trackings eine Brille trug und der

Eye Tracker in dem Fall nicht in der Lage war, konstante, korrekte Blickdaten zu erfassen. Auch dieser Durchlauf wurde mit einem neuen PhD Studenten wiederholt.

8.5 Beobachtungen während der Studie

In diesem Abschnitt wird kurz beschrieben, wie die Teilnehmer die Arbeit mit dem Webnachbau von Eclipse empfanden und wie sie generell an die Aufgaben herangingen. Der Großteil der Teilnehmer nahm die gegebene Eclipseversion im Browser kommentarlos hin und hatte auch keine Probleme sich damit zurecht zu finden. Einige Teilnehmer versuchten dennoch den „Anwendung ausführen"-Button oder andere UI-Elemente zu betätigen, obwohl anfangs explizit beschrieben wurde, dass dies nicht möglich war. Ein paar Mal wurde auch kommentiert, dass es die Arbeit etwas umständlicher machte, dass der Package Explorer und die Menüleiste jedes Mal aus dem Bild scrollten, wenn man zu den unteren Teilen der Klasse navigierte. Ein paar Probanden hätten auch gerne die Möglichkeit gehabt, zwei Klassen parallel darstellen zu können. Dies ließ sich aufgrund der Beschränkungen des Eye Trackers jedoch nicht anders umsetzen.

Zu der allgemeinen Vorgehensweise lässt sich sagen, dass es zwei unterschiedliche Herangehensweisen zu Beginn des Experiments gab. Die Mehrheit der Teilnehmer las zunächst gründlich die erste gegebene Aufgabenstellung und versuchte dann direkt die dafür passende Klasse zu finden. Dazu gehörten alle Experten, sowie auch zwei Drittel der Studenten und PhD Studenten. Einige Teilnehmer hingegen versuchten sich, unabhängig von der Aufgabe, zunächst ein generelles Bild des gegebenen Programms zu machen, bevor sie begannen die erste Aufgabe zu bearbeiten. Des Weiteren ließ sich bemerken, dass je mehr Entwicklungserfahrung die Teilnehmer hatten, desto häufiger wurde die STRG+F Funktion zum direkten Suchen und Springen zu bestimmten Stellen verwendet.

9 Analyse und Interpretation

In diesem Kapitel wird beschrieben, wie die während der Studie erfassten Daten zunächst verarbeitet und analysiert wurden. Im Anschluss daran werden die anfangs definierten Hypothesen anhand der verarbeiteten Daten sukzessive geprüft und das Ergebnis vorgestellt.

9.1 Verarbeitung der Rohdaten

Um die aufgestellten Hypothesen formal prüfen zu können, mussten die Daten zunächst aus ihrer Rohform weiter verarbeitet werden. Dazu wurden sämtliche Antworten auf die Fragebögen, sowie während des Experiments festgehaltene Kommentare und Auffälligkeiten in eine digitale Form gebracht. Auch die Aufnahmen der Befragung zum Ende eines jeden Studiendurchlaufs wurden transkribiert und die Aussagen den einzelnen Metriken zugeordnet. Um die reinen Bearbeitungszeiten der Aufgaben zu erhalten, wurden dem Tobii Pro Studio die Zeitintervalle entnommen, die der Teilnehmer die Java API betrachtete. Diese Intervalle wurden anhand der protokollierten Zeitpunkte, zu denen eine Aufgabe beendet wurde, den einzelnen Aufgaben zugeordnet und von ihrer errechneten Bearbeitungszeit abgezogen. Der Grund dafür ist, dass der Fokus der Analyse auf der Arbeit mit dem Code und der eigentlichen Entwicklungstätigkeit und nicht auf dem Lesen von Dokumentation lag.

Sämtliche Eye-Tracking Daten wurden in Exceltabellen aus dem Tobii Pro Studio für jeden Teilnehmer einzeln exportiert und damit Metriken wie Fixations- und Sakkadenanzahl, durchschnittliche Fixations- und Sakkadendauer, Pupillendurchmesser, etc. bestimmt. In diesem Schritt wurden die Daten bereits für jeden Teilnehmer in Diagrammen dargestellt, um etwaige Ausreißer zu erkennen und aus der Analyse auszuschließen. Da die meisten Punkte jedoch in der gleichen Größenordnung lagen, mussten nur sehr wenige Werte vor der finalen Auswertung entfernt werden.

9.2 Formale Hypothesenprüfung

Sobald sämtliche Daten verarbeitet und in der für die einzelnen Metriken notwendigen Form vorlagen, konnten die jeweiligen Hypothesen formal geprüft werden. Dazu wurde entsprechend jeweils die Nullhypothese aufgestellt und auf Basis der erhobenen Daten auf Gültigkeit geprüft. Dazu wurde jeweils ein Signifikanzniveau von 5% betrachtet. Das heißt, es wurde getestet, ob die

Wahrscheinlichkeit, dass die Ergebnisse zufällig entstanden sind und die Nullhypothese damit fälschlicherweise abgelehnt würde, unter 5% liegt. Die einzelnen Hypothesen werden im Folgenden nacheinander aufgelistet, die Daten dazu beschrieben und das Ergebnis der Hypothesenprüfung vorgestellt. Eine Zusammenfassung, die Beantwortung der Forschungsfragen, sowie die Diskussion der Ergebnisse folgen im anschließenden Kapitel.

Bei den Hypothesen, bei denen die Metriken Fixations- oder Sakkadenanzahl betrachtet wurden, wurden diese Werte unter anderem mit dem Gaze Samples Wert relativiert. Der Gaze Samples Prozentsatz gibt an, wie viele der Versuche des Eye Trackers, Blickdaten für ein oder beide Augen aufzunehmen, erfolgreich waren. Er gibt also den Anteil nutzbarer Eye-Tracking Samples an, welche Informationen über Fixationen, Sakkaden und Pupillen liefern. Der Wert wird vom Tobii Pro Studio jedoch nur für die gesamte Aufnahme angegeben und lässt sich nicht nur für einen Unterabschnitt bestimmen, wie in diesem Fall die Betrachtung der Webseiten mit den Programmklassen. Die Berechnung erfolgt somit unter der Annahme, dass die Qualität der Aufnahme während des Lesens der Aufgabeneinführung und der Betrachtung der Java API, sowie dem Betrachten des Codes weitestgehend konstant ist, beziehungsweise, dass eventuelle Abweichungen für alle Teilnehmer ähnlich sind. Mit der Berücksichtigung dieses Wertes werden Einflüsse durch etwaige Unterschiede in der Qualität der Aufnahmen, sowie abweichende Betrachtungsdauern der Aufgabenstellungen und der Lösungsblätter, ausgeschlossen. Die detaillierte Berechnungsformel findet sich in den Abschnitten der betroffenen Hypothesen.

Zu den Fragen in den Fragebögen, bei denen eine 10-stufige Likert-Skala verwendet wurde, ist außerdem anzumerken, dass hierbei dennoch der Mittelwert zur Auswertung berechnet wurde. Die Skala wurde demnach wie eine Intervallskala behandelt, da nur der niedrigste und höchste Wert jeweils mit einem Attribut versehen waren. Es handelte sich aufgrund der somit äquidistanten Abstände zwischen den Stufen daher um eine quasimetrische Skala.

H_1: Betrachtungshäufigkeit und Abschnittsrelevanz

Die erste Hypothese H_1 sagt aus, dass für Softwarewartungsarbeiten relevantere Abschnitte häufiger betrachtet werden als für Softwarewartungsarbeiten weniger relevante Abschnitte. Als Metriken wurden dazu die Aussagen zu der Relevanz von Abschnitten aus den Audioaufnahmen, sowie die Fixationszahlen pro Klasse betrachtet. Es war nicht möglich eine allgemeine Betrachtung auf kleinerer Ebene, also beispielsweise Methoden oder Konstruktor, durchzuführen, da die Aussagen vieler Teilnehmer dafür nicht präzise genug waren. Eine konkretere

Bewertung, etwa in einem weiteren Fragebogen, hätte wiederum den Zeitraum der Studie deutlich verlängert, sodass sich hier weitestgehend auf Klassenebene beschränkt wurde.

Zunächst wurden dazu die Fixationsanzahlen der Teilnehmer pro Klasse bestimmt. Anschließend wurden diese pro Teilnehmer normiert, um diese miteinander vergleichen und aussagekräftige Mittelwerte bilden zu können.

Dadurch, dass viele Antworten der Teilnehmer entweder manche Klassen außen vor ließen oder relativ unpräzise waren, erfolgte die Auswertung hier auf einer sehr qualitativen Ebene. Im ersten Schritt wurden dazu die getätigten Aussagen bezüglich der Relevanz einzelner Klassen mit den entsprechenden Fixationsanzahlen dieser Klassen verglichen und so geprüft, ob es Widersprüche gab. Anschließend wurden die wenigen Aussagen, die nicht nur Klassen als Gesamtes betreffen, sondern auch Unterbereiche, mit den Heatmaps dieser Teilnehmer für die entsprechende Klasse verglichen. Im letzten Schritt wurde anhand der Antworten der Teilnehmer eruiert, wie es zu der entsprechenden Relevanzbewertung kam und welche Begründung für die intensive Betrachtung einzelner Bereiche gegeben wurde.

1. Vergleich klassenspezifischer Relevanzbewertungen mit Fixationsanzahlen

Als erstes wurden die Aussagen zu der Relevanz der einzelnen Klassen mit den normierten Fixationsanzahlen für die entsprechende Klasse verglichen. Das Ergebnis ist, dass von 87 Klassen, zu denen klare Relevanzaussagen getroffen wurden, 83 mit der Fixationsanzahl übereinstimmen. Fünf der Aussagen weichen hingegen geringfügig ab. Dabei handelt es sich um die Widersprüche zwischen den folgenden Aussagen und Anteilen der gesamten Fixationsanzahl:

- „einmal kurz rein [geschaut] und gleich wieder raus“ und 4,7% der gesamten Fixationen
- „nur kurz drin“ und 4,6% der gesamten Fixationen
- „länger reingeguckt, dann aber nicht gebraucht“ und nur 1,1% der gesamten Fixationen
- „[Die Klasse] muss man sich ja nicht so genau angucken“ und dennoch 12,1% der gesamten Fixationen
- „[UI.java] nur kurz benutzt“ und trotzdem 26,9% der gesamten Fixationen

2. Vergleich der Relevanzbewertungen von Unterabschnitten mit Heatmaps

Im nächsten Schritt wurden die Aussagen zu konkreten Unterabschnitten einzelner Klassen mit den Heatmaps dieser Klasse und dieses Teilnehmers verglichen.

Es wurde geprüft, ob Aussagen über die Irrelevanz eines Bereichs auch auf die geringe Farbintensität der Heatmap an dieser Stelle schließen lassen und umgekehrt als relevant erachtete Bereiche der Heatmap zufolge auch intensiver betrachtet wurden. Zunächst einmal muss festgestellt werden, dass es nur sehr wenige solcher konkreten Relevanzaussagen gibt. Die getätigten Aussagen stimmen jedoch alle mit der Färbung der Heatmap dieser Bereiche überein. Alle als relevant genannten Teilstücke sind auch mittel bis stark durch die Heatmap hervorgehoben. Als irrelevant erachtete Bereiche hingegen sind auch kaum farblich markiert. Es kann jedoch festgestellt werden, dass es teilweise Abschnitte gibt, die zwar relativ stark eingefärbt waren, jedoch nicht in der Befragung erwähnt wurden. Dies muss aber nicht die Fehlerhaftigkeit der Heatmap bedeuten, sondern kann einfach damit zusammenhängen, dass der Teilnehmer diese übergangen oder vergessen hatte.

3. Gründe für Relevanzbewertungen und intensive Betrachtung

Abschließend wurden die Aussagen der Teilnehmer dahingehend untersucht, welche Gründe zum einen für die jeweilige hohe oder niedrige Relevanzbewertung gegeben wurden und zum anderen, wie intensives Blickverhalten auf Bereichen erklärt wurde. Dies gibt Ausschluss darüber, ob relevante Bereiche auch intensiver betrachtet werden und auf der anderen Seite, ob intensiv betrachtete Bereiche auch immer relevant sind.

Als mit Abstand häufigstes Kriterium für die hohe Relevanz einer Klasse oder eines Unterbereichs einer Klasse, wurde angegeben, dass diese besonders häufig oder lange betrachtet wurde. Auf der anderen Seite galt ein Abschnitt als nicht oder wenig relevant, wenn dieser gar nicht angeschaut wurde. Lediglich ein Teilnehmer gab bei einer Klasse an, diese nicht wirklich betrachtet zu haben und daraus jedoch nicht zu schließen, dass sie zwangsläufig irrelevant sei. Dabei spielt mit rein, dass nicht jeder Proband die gleiche Auffassung von Relevanz hatte. Die meisten haben den Schwerpunkt auf die Relevanz für die gestellten Aufgaben gelegt, wohingegen andere auch die allgemeine Relevanz zum Verständnis des Programms mit berücksichtigten. Ein weiteres häufig genanntes Argument für die Relevanz von Abschnitten war, ob und in welchem Maße diese verändert wurden. Ebenfalls einige Male wurde angegeben, dass ein Bereich relevant war, weil dieser zum Verständnis des Kontexts benötigt wurde oder weil verwendete Methoden oder Variablen dort initialisiert wurden.

Das häufige oder lange Betrachten eines Abschnitts oder einer Datei hingegen wurde oft damit erklärt, dass dieser wichtig für die Bearbeitung der Aufgabe war. Teilweise wurde auch genannt, dass solche Bereiche sehr viel Funktionalität

enthielten. Teile, die sehr einfach zu verstehen waren, wurden dagegen kürzer oder weniger angeschaut. Außerdem wurde als Begründung für das lange Betrachten eines Abschnitts angegeben, dass dieser sehr kompliziert war und nur schwer verstanden werden konnte. In wenigen Fällen wurde ein Teil auch länger angesehen, weil sich die Teilnehmer an der Stelle hilfreiche Informationen erhofften, die sie dann doch nicht dort fanden. Das intensive Betrachten eines Bereichs lässt somit nur bedingt auf dessen Relevanz schließen. Auch eine hohe Komplexität oder eine falsche Erwartungshaltung über den Inhalt, können zu einer längeren Betrachtungszeit führen.

Hohe Betrachtungszeiten für einen Abschnitt lassen somit nicht mit voller Sicherheit auf dessen Relevanz schließen. Dennoch lässt sich sagen, dass obwohl die Komplexität, Länge und Informationsdichte eines Bereichs zwar einen Faktor bei der Betrachtungsintensität darstellen, dieser jedoch nur verstärkend wirkt. Das heißt, wenn ein Teil überhaupt nicht relevant wäre, würde dieser unabhängig von seiner Komplexität auch nicht lange angeschaut werden. Gleiches gilt für relativ einfache und übersichtliche Abschnitte. Auch wenn diese nicht so intensiv betrachtet werden, erhalten sie dennoch mehr Aufmerksamkeit als komplett irrelevante Abschnitte. Abschließend lässt sich somit sagen, dass die Betrachtungsintensität eines Bereichs ein zuverlässiger Indikator für dessen Relevanz ist, der nur in wenigen Fällen eine gewisse Unsicherheit mit sich bringt.

$H_{2.1}$: Bearbeitungszeit

Hypothese $H_{2.1}$ besagt, dass durch die Darstellung von Aufmerksamkeitsdaten Softwarewartungsarbeiten schneller erledigt werden können. Dazu wurden für jede Aufgabe die durchschnittlichen Bearbeitungszeiten zur Fertigstellung der Lösungen berechnet. Von diesen wurden die jeweiligen Betrachtungszeiten der Java API subtrahiert, da der Fokus dieser Forschungsfrage und Hypothese auf Softwarewartungsaufgaben liegt. Das heißt, es geht um tatsächliches Lesen und Entwickeln von Code und nicht um das Lesen und Suchen von Dokumentation. Dazu kommt, dass es sehr stark differiert, wie intensiv die Teilnehmer die API nutzten und wie gut sie sich darin zurecht fanden. Abschließend erhielt man damit zwischen drei und sechs Zeitwerte pro Aufgabe und Gruppe (ohne bzw. mit Aufmerksamkeitsdarstellung), abhängig davon, wie viele Teilnehmer die jeweilige Aufgabe vollständig bearbeitet hatten. Der Datenpunkt von dem Teilnehmer mit der ID 019 wurde für Aufgabe A1 an dieser Stelle entfernt, da er über 20 Minuten zur Fertigstellung brauchte, während die restlichen Teilnehmer im Mittelwert nur etwa zwei Minuten benötigten. Der Grund dafür war, dass der Proband große Probleme damit hatte, sich zunächst in dem Programm

zurecht zu finden und damit abzufinden, dass es in der erstellten Webversion von Eclipse nicht möglich war, direkt die Javadoc Dokumentation zu einzelnen Objektklassen anzuzeigen. Er hat sich daher sehr lange an Programmstellen aufgehalten, die für die Aufgabe gar nicht relevant waren. Der Rest der Zeitwerte ist hingegen relativ gleichmäßig verteilt.

Für die Teilnehmer der Kontrollgruppe, die mit dem Code ohne Heatmap arbeiteten, ergibt sich damit eine mittlere Bearbeitungszeit der Aufgaben von 451,53 Sekunden (etwa 7,53 Minuten). Für die Teilnehmer der Treatmentgruppe, die den Code mit Heatmap gezeigt bekamen, ergibt sich hingegen eine mittlere Bearbeitungszeit von 436 Sekunden (etwa 7,27 Minuten). Die Teilnehmer der Gruppe mit Heatmap waren somit im Durchschnitt 15,5 Sekunden schneller als die Teilnehmer der Gruppe ohne Heatmap. Abbildung 9.1 zeigt die einzelnen Werte in einem Säulendiagramm aufgeschlüsselt. Auf der X-Achse befinden sich die einzelnen Aufgaben aufgelistet und auf der Y-Achse die entsprechende mittlere Bearbeitungszeit für die Teilnehmer mit und ohne Heatmapdarstellung. Bei näherer Analyse der Daten wurde festgestellt, dass bei zwei Drittel der Aufgaben diese ohne Heatmap langsamer bearbeitet wurden und bei dem restlichen Drittel die Teilnehmer mit Heatmap hingegen langsamer waren.

Die entsprechende Nullhypothese, dass die Softwarewartungsarbeiten mit der Aufmerksamkeitsdarstellung nicht schneller bearbeitet werden, wurde mithilfe eines Zweistichproben-t-Tests geprüft. Die vorausgesetzte Normalverteilung wurde zuvor mit dem Kolmogorov-Smirnov Test sichergestellt. Das Ergebnis zeigt einen p-Wert von 0,48. Die Nullhypothese muss daher beibehalten werden.

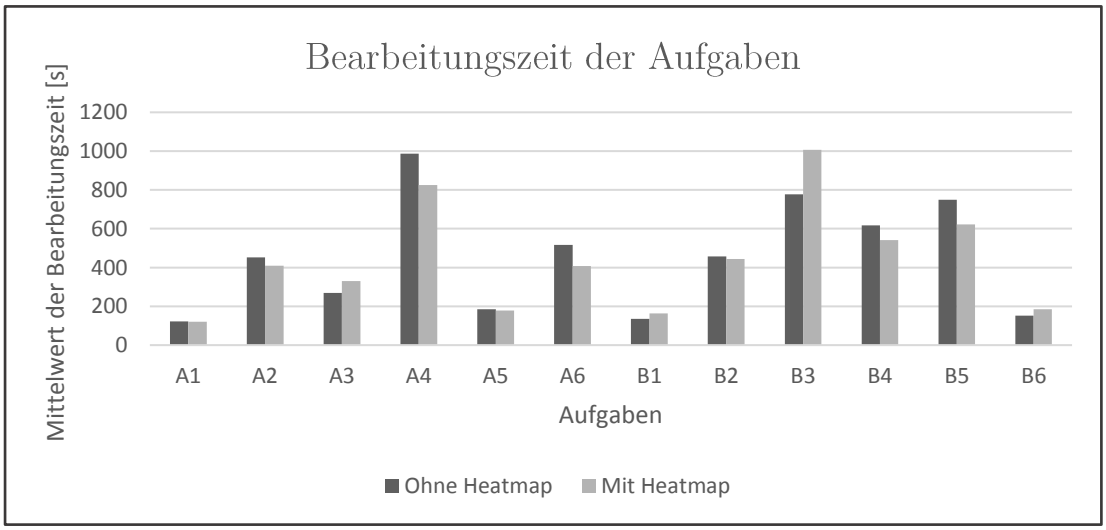


Abbildung 9.1: Bearbeitungszeit der Aufgaben mit und ohne Heatmap

$H_{2.2}$: Visueller Aufwand

Die Hypothese $H_{2.2}$ sagt aus, dass die Darstellung von Aufmerksamkeitsdaten einen Effekt auf den visuellen Aufwand ausübt, gemessen an Sakkadenanzahl und Sakkadendauer. Dabei bedeuten eine kleinere Sakkadenanzahl bzw. –dauer einen geringeren visuellen Aufwand [6]. Da die Sakkadenanzahl insgesamt stark durch die gesamte Bearbeitungsdauer, die Qualität der Eye-Tracking Aufnahme, sowie die Betrachtungsdauer der Aufgabenstellung und Lösungsblätter beeinflusst wird, mussten die Daten mit diesen Einflussgrößen relativiert werden. Dazu wurde nicht nur die gesamte Sakkadenanzahl bestimmt, sondern die Sakkadenanzahl pro Sekunde. Für die Zeit in Sekunden musste die Dauer berechnet werden, die der jeweilige Teilnehmer auf den Bildschirm schaute, während die jeweiligen Webseiten mit den Programmklassen geöffnet waren. Diese wurde aus dem Produkt der summierten Zeitintervalle, in denen die jeweiligen Klassen geöffnet waren, und dem Gaze Samples Prozentsatz berechnet. Auf diese Weise erhält man mit der folgenden Formel die Sakkadenanzahl pro Sekunde Bildschirmbetrachtungszeit für die Arbeit mit dem Code. Der Nenner ergibt dabei die Zeit in Sekunden, die auf Codewebseiten geschaut wurde, während erfolgreich Eye-Tracking Daten erkannt werden konnten. Der Zähler wiederum enthält die Summe der erkannten Sakkaden auf den Codewebseiten.

$$Sakkaden\ pro\ Sekunde = \frac{\sum_{Eclipse\ Webseiten} Sakkadenanzahl}{(\sum_{Eclipse\ Webseiten} Zeitintervall) * Gaze\ samples}$$

Formel 9.1: Sakkadenanzahl pro Sekunde Codebetrachtung

Abbildung 9.2 zeigt die resultierenden Werte für alle Teilnehmer in einem Säulendiagramm. Man erkennt, dass der Wert der Sakkadenanzahl pro Sekunde des Teilnehmers 005 mit etwa 15 Sakkaden pro Sekunde deutlich über den Werten der restlichen Teilnehmer liegt. Der Grund dafür ist vermutlich, dass es sich bei diesem Teilnehmer um einen Brillenträger handelte und dies in manchen Fällen dazu führt, dass die Infrarotstrahlen des Eye Trackers abgelenkt werden und es so zu fälschlich erkannten Blicksprüngen kommt. Der Wert wurde daher aus der Auswertung ausgeschlossen. Für die übrigen Teilnehmer der Kontrollgruppe ergibt sich eine durchschnittliche Sakkadenanzahl von 6,55 pro Sekunde und für die Teilnehmer der Treatmentgruppe von 7,77 pro Sekunde. Im Mittelwert machten die Teilnehmer, die mit dem Code mit Heatmapdarstellung arbeiteten, somit etwa 1,2 Sakkaden mehr pro Sekunde, was für einen höheren visuellen Aufwand spricht. Die Nullhypothese, dass sich die mittlere Sakkadenanzahl der beiden Gruppen nicht unterscheidet, wurde mit dem Zweistichproben-t-Test geprüft. Das Vorliegen einer Normalverteilung wurde zuvor getestet. Anhand des

Ergebnisses mit p=0,0323 kann die Nullhypothese zum Signifikanzniveau von 5% abgelehnt werden.

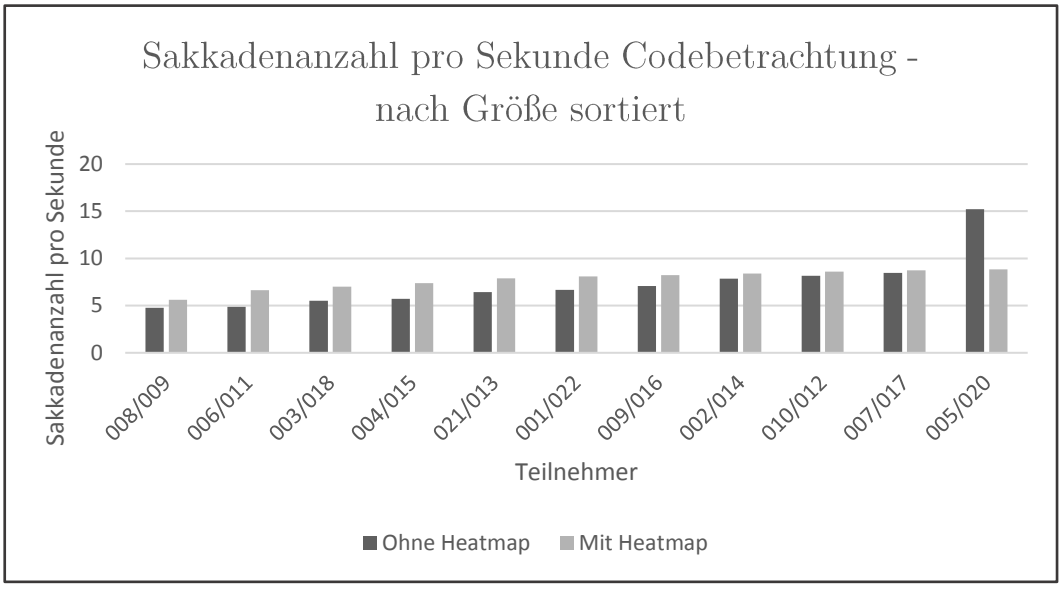


Abbildung 9.2: Sakkadenanzahl pro Sekunde Codebetrachtung

Bei der Sakkadendauer wurde zunächst für jeden Teilnehmer die durchschnittliche Dauer aller Sakkaden in Millisekunden berechnet. Anschließend wurde der Mittelwert dieser Werte einmal für die Teilnehmer der Kontrollgruppe und einmal für die Teilnehmer der Treatmentgruppe ermittelt und einander gegenüber gestellt. Es ergibt sich für die Teilnehmer ohne Heatmapdarstellung eine mittlere Sakkadendauer von 30,64 Millisekunden und für die Teilnehmer mit Heatmapdarstellung eine mittlere Sakkadendauer von 30,45 Millisekunden. Der Unterschied liegt damit bei lediglich 0,18 Millisekunden und ist nicht signifikant. Die Nullhypothese, dass sich die mittlere Sakkadendauer der Kontroll- und Treatmentgruppe nicht unterscheiden, muss somit beibehalten werden.

Abschließend lässt sich somit festhalten, dass der visuelle Aufwand der Treatmentgruppe an der Sakkadenanzahl gemessen signifikant über dem der Kontrollgruppe liegt. Bei der Sakkadendauer hingegen lässt sich kein Unterschied nachweisen.

$H_{2.3}$: Kognitive Last

Die Hypothese $H_{2.3}$ sagt aus, dass die kognitive Last durch die Darstellung von Aufmerksamkeitsdaten sinkt, gemessen an dem Pupillendurchmesser und der Fixationsdauer. Dazu wurde für jeden Teilnehmer der durchschnittliche Pupillendurchmesser des linken und rechten Auges bestimmt und diese miteinander gemittelt. Zusätzlich wurde der Mittelwert der Fixationsdauer für die

Aufnahmeabschnitte bestimmt, während derer die Teilnehmer die Eclipse Webseiten geöffnet hatten. Bei beiden Metriken weisen kleinere Werte auf eine geringere kognitive Last hin [6].

Für den Pupillendurchmesser erhält man für die Kontrollgruppe damit einen Mittelwert von 2,93 Millimetern und für die Treatmentgruppe einen Mittelwert von 2,83 Millimetern. Im Schnitt sind die Pupillen der Teilnehmer mit Heatmap somit um 0,1 Millimeter kleiner, was auf eine geringere kognitive Last hindeutet. In Abbildung 9.3 sind alle Werte der Teilnehmer der Größe nach sortiert in einem Diagramm dargestellt. Bei näherer Betrachtung der Werte wurde festgestellt, dass die Werte der Kontrollgruppe stets über denen der Treatmentgruppe liegen, bis auf eine Ausnahme. Der größte Wert, der zu dem Teilnehmer mit der ID 014 gehört, sticht als einziger hervor und liegt damit deutlich über dem größten Pupillendurchmesser der Kontrollgruppe. Eine Ursache dafür könnte sein, dass dieser Teilnehmer mit 8:30 Uhr den frühesten Studientermin unter allen Probanden hatte und sowohl dunklere Lichtverhältnisse, als auch Kaffeekonsum ebenfalls zu einer Erweiterung der Pupillen führen. Schließt man den Teilnehmer mit der ID 014 aus der Betrachtung aus, erhält man eine Differenz des durchschnittlichen Pupillendurchmessers von etwa 0,2 Millimetern. Die entsprechende Nullhypothese, dass der Pupillendurchmesser der Kontrollgruppe nicht über dem der Treatmentgruppe liegt, wurde erneut mit dem Zweistichproben-t-Test unter vorangehender Prüfung einer Normalverteilung, getestet. Das Ergebnis ist mit p=0,1893 nicht signifikant. Trotz sichtbarem Trend, muss die Nullhypothese daher beibehalten werden.

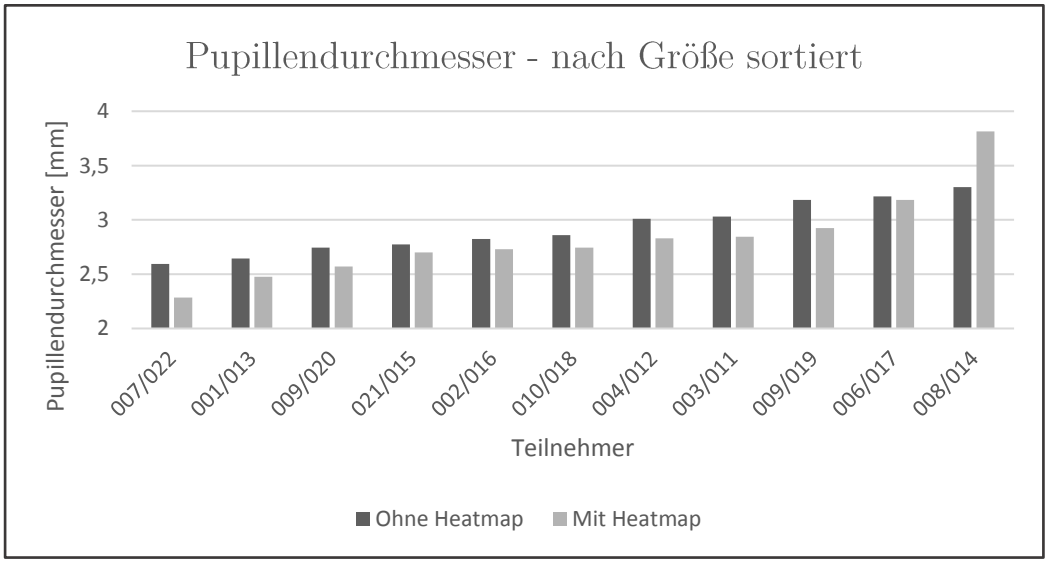


Abbildung 9.3: Pupillendurchmesser

Für die Fixationsdauer wurde ebenfalls für jeden Teilnehmer der Mittelwert bestimmt und diese anschließend für die Teilnehmer mit und ohne

Heatmapdarstellung miteinander verglichen. Die resultierenden Werte zeigt Abbildung 9.4 in einem Säulendiagramm dargestellt. Für die Kontrollgruppe ergibt sich dabei eine durchschnittliche Fixationsdauer von 209,73 Millisekunden und für die Treatmentgruppe von 201,45 Millisekunden. Das heißt, die Teilnehmer, die mit dem Code mit Aufmerksamkeitsdarstellung arbeiteten, fixierten Punkte im Durchschnitt 8,27 Millisekunden kürzer, was dafür spricht, dass die Aufgaben für sie tendenziell leichter zu bewerkstelligen waren. Beim Testen der Nullhypothese, dass die Fixationsdauer der Kontrollgruppe nicht über der Fixationsdauer der Treatmentgruppe liegt, ergibt sich ein p-Wert von 0,3245. Das Ergebnis ist somit nicht signifikant und die Nullhypothese muss beibehalten werden. Als Test wurde dafür der Zweistichproben-t-Test verwendet.

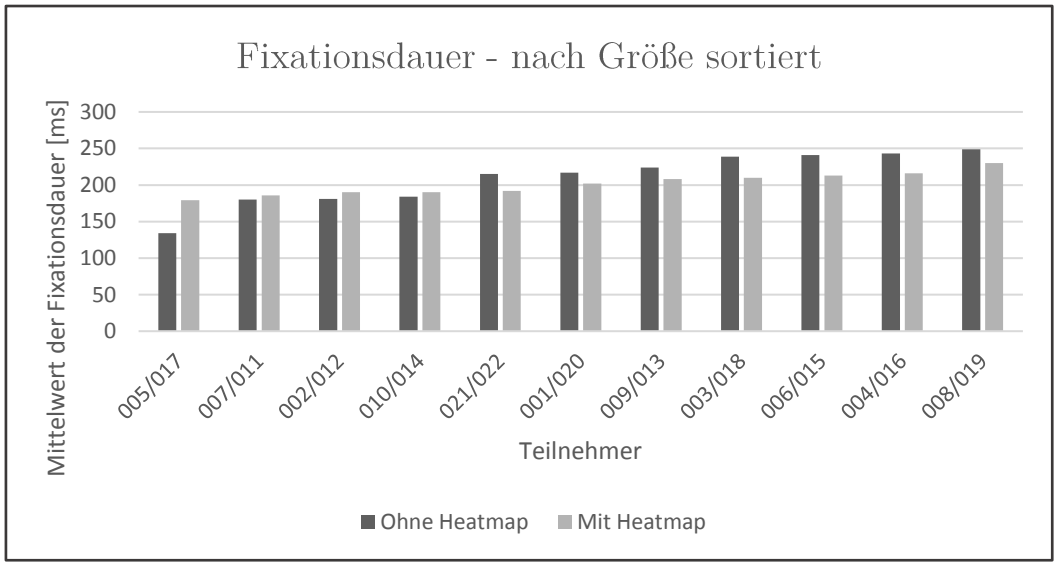


Abbildung 9.4: Fixationsdauer

Des Weiteren auffällig an dieser Stelle ist jedoch zum einen die Differenz in der Standardabweichung von 20,48 Millisekunden. Die Mittelwerte der Fixationsdauern der Treatmentgruppe liegen näher beieinander als die der Kontrollgruppe. Zum anderen liegen die einzigen vier Werte der Teilnehmer mit Heatmapdarstellung, die die Werte der Fixationsdauer der Teilnehmer ohne Heatmapdarstellung übersteigen, alle im unteren Bereich. Tendenziell deuten diese Daten also darauf hin, dass die Heatmap einen „ausgleichenden Effekt“ auf die Fixationsdauer hat und die kognitive Last damit weniger abhängig vom Teilnehmer variiert.

Abschließend lässt sich über die kognitive Last somit aussagen, dass man in den Daten, sowohl bei dem Pupillendurchmesser, als auch bei der Fixationsdauer, eine Tendenz erkennt, dass die kognitive Last durch die Darstellung der Aufmerksamkeitsdaten sinkt. Dieser Effekt konnte jedoch nicht als statistisch signifikant unter einem Signifikanzniveau von 5% nachgewiesen werden.

$H_{2.4}$: Effizienz der Informationssuche

Die Hypothese $H_{2.4}$ besagt, dass die Effizienz der Informationssuche, gemessen an der Fixationsanzahl, durch die Darstellung von Aufmerksamkeitsdaten steigt. Genauso wie die Sakkadenanzahl wird auch die Fixationsanzahl stark durch die tatsächliche Betrachtungszeit des Codes beeinflusst. Die Werte für die Fixationsanzahl wurden somit entsprechend mit der folgenden Formel relativiert, um diese Einflüsse auszuschließen.

$$Fixationen\ pro\ Sekunde = \frac{\sum_{Eclipse\ Webseiten} Fixationsanzahl}{(\sum_{Eclipse\ Webseiten} Zeitintervall) * Gaze\ samples}$$

Formel 9.2: Fixationsanzahl pro Sekunde Codebetrachtung

Für die Teilnehmer der Kontrollgruppe ergibt sich damit eine mittlere Fixationsanzahl pro Sekunde Codebetrachtung von 3,48. Für die Treatmentgruppe ergibt sich ein durchschnittlicher Wert von 3,55 Fixationen pro Sekunde Codebetrachtung. Der Unterschied liegt somit lediglich bei 0,07 Fixationen und ist nicht signifikant. Auch bei näherer Betrachtung der einzelnen Werte lässt sich kein Trend erkennen.

H_3: Qualität von Softwarewartungsarbeiten

Die Hypothese $H_{3.1}$ sagt aus, dass sich die Qualität von Softwarewartungsarbeiten durch die Darstellung von Aufmerksamkeitsdaten verändert. Dazu wurden die Bewertungen der Korrektheit der Aufgabenlösungen der Teilnehmer mit und ohne Heatmap miteinander verglichen. Dabei wurden nur die Lösungen betrachtet, die fertig gestellt und nicht aufgrund des Erreichens des Zeitlimits oder einer fehlenden Lösungsidee abgebrochen wurden. Die Lösungen wurden auf einer vierstufigen Ordinalskala von 1 bis 4 bewertetet, wobei 1 für falsch steht, 2 für ansatzweise korrekt, 3 für weitestgehend korrekt und 4 für vollständig korrekt. Der Fokus bei der Bewertung wurde dabei auf den Ansatz und die Beachtung der für die Umsetzung der Aufgabe relevanten Details gelegt. Fehler, die bei der Verwendung einer voll funktionalen Entwicklungsumgebung ohne große Expertise leicht aufgefallen und zu beheben gewesen wären, führten zu keinem Abzug. Dazu gehörten beispielsweise die Verwendung der falschen Anführungszeichen, eine fehlende Datentypenkonvertierung oder das Vergessen des Datentyps beim Erstellen eines Objekts. Der Grund dafür ist, dass der Fokus auf der Entwicklertätigkeit bei Softwarewartungsarbeiten lag und nicht auf der Bemessung der Javakenntnisse.

Um die Reliabilität der Korrektheitsbewertungen zu prüfen, wurden sämtliche Aufgabenlösungen der Teilnehmer zusätzlich von einer zweiten Person, unabhängig von der ersten Bewertung, noch einmal auf der gleichen Skala unter den gleichen Kriterien evaluiert. Dabei zeigte sich, dass die Werte der zweiten Bewertung alle entweder mit denen der ersten Bewertung übereinstimmten oder leicht darunter lagen. Insgesamt ergibt sich eine mittlere Abweichung von 0,26. Dafür wurde der Cohen's Kappa Wert bestimmt, der die Stärke der Übereinstimmung angibt. Dieser liegt bei 0,639 für alle Bewertungen. Trotz geringer Abweichungen, besteht somit dennoch eine moderate Übereinstimmung, die für die Zuverlässigkeit der Daten spricht. Des Weiteren lässt sich feststellen, dass der Kappa-Wert der Treatmentgruppe mit 0,692 über dem der Kontrollgruppe mit 0,585 liegt. Dadurch, dass keine der zweiten Bewertungen über denen der ersten Bewertung liegt, ist das Risiko einer fälschlichen Ablehnung der Nullhypothese somit noch weiter verringert.

Es ergibt sich für die Teilnehmergruppe ohne Aufmerksamkeitsdarstellung eine mittlere Korrektheit von 3,24 und für die Gruppe mit Aufmerksamkeitsdarstellung ein durchschnittlicher Korrektheitswert von 3,32. Der Unterschied liegt damit bei etwa 0,07 und ist nicht signifikant. Die Nullhypothese wurde an dieser Stelle mit dem Wilcoxon-Mann-Whitney-Test geprüft und musste aufgrund eines p-Werts von 0,3742 beibehalten werden. Auch bei näherer Betrachtung der einzelnen Werte und Differenzen zwischen den Gruppen pro Aufgabe erkennt man keinen Effekt.

H_4: Leiten der Aufmerksamkeit

Die Hypothese $H_{4.1}$ drückt aus, dass die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps die Aufmerksamkeit auf die hervorgehobenen Bereiche lenkt. Dazu wurden die Audioaufnahmen der Befragung der Teilnehmer mit dem Fokus untersucht, inwiefern die Heatmap und die Klassennameneinfärbung ihre Aufmerksamkeit leitete – unabhängig davon, ob zum Vorteil oder zum Nachteil. Dabei lässt sich feststellen, dass nahezu alle Teilnehmer zu einem gewissen Grad auf die Heatmap im Code, sowie die Färbung im Package Explorer achteten. Lediglich ein Teilnehmer gab an, die Heatmap nicht wirklich wahrgenommen zu haben. Sie hat ihn demnach auch weder gestört, noch hat sie ihm geholfen. Ein weiterer Teilnehmer sagte zudem aus, dass ihm zwar die Heatmap im Hintergrund des Codes aufgefallen war, die Färbung im Package Explorer jedoch nicht. Aus der Aussage wurde jedoch auch deutlich, dass er die anfängliche Erklärung der beiden Darstellungen entweder nicht aufmerksam gelesen oder nicht verstanden hatte, da er nicht einmal deren Bedeutung kannte. Bei den restlichen neun

Teilnehmern wiederum leitete die Heatmap bzw. die Klassennamenhervorhebung in allen Fällen die Aufmerksamkeit zu einem gewissen Grad. Dabei wurde sie teils als hilfreich und teils als ablenkend empfunden, wenn die Hervorhebungen nicht mit dem eigenen Relevanzempfinden übereinstimmten. Beides deutet darauf hin, dass die Heatmap den Aufmerksamkeitsfokus auf die farblich markierten Bereiche lenkt. Gleiches gilt für die Hervorhebung im Package Explorer. Manche Teilnehmer gaben auch an, durch die Markierungen in die richtige Richtung geleitet worden zu sein oder aber auf eine in ihrem Kontext eigentlich eher irrelevante Klasse. Viele Teilnehmer beschrieben außerdem, dass die Färbung auch ohne Erklärung sehr intuitiv verständlich sei. Des Weiteren sagte etwa die Hälfte der Probanden aus, dass sie die Heatmap zumindest in Teilen der Studie bewusst ausgeblendet hatten, da sie für sie zu den Zeitpunkten nicht relevant war. Zusammenfassend lässt sich aus den Aussagen der Probanden somit schließen, dass die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps die Aufmerksamkeit auf die hervorgehobenen Bereiche lenkt. Dennoch sticht sie nicht so sehr hervor, dass diese Lenkung nicht ignoriert werden könnte.

H_5: Nutzen der Aufmerksamkeitsdarstellung

Die Hypothese $H_{5.1}$ besagt, dass die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps eine Hilfestellung bei der Bearbeitung von Softwarewartungsarbeiten bietet. Dazu wurde zum einen die im Fragebogen abgegebene subjektive Beurteilung des Nutzens der Heatmap im Hintergrund des Codes, sowie der Heatmapeinfärbung der Klassennamen, betrachtet. Zusätzlich wurden die Aussagen, die zu dem Nutzen der Aufmerksamkeitsdarstellung in der abschließenden Befragung im Experiment getroffen wurden, analysiert.

Bei der Frage danach, wie hilfreich die Heatmapdarstellung im Code auf einer Skala von 1 bis 10 war, wobei 1 für „nicht hilfreich“ und 10 für „hilfreich“ stand, ergibt sich ein Mittelwert von 3,4. Die Klassennameneinfärbung wurde im Durchschnitt mit 4,3 bewertet. Das bedeutet, dass beide Darstellungen im Schnitt nur als mäßig hilfreich erachtet wurden und die Einfärbung der Klassennamen im Package Explorer tendenziell hilfreicher zu sein scheint. Abbildung 9.5 zeigt die einzelnen Bewertungen der Teilnehmer in einem Säulendiagramm dargestellt. Die Daten zeigen deutlich, dass es wenig mittlere Werte gibt. Das heißt, der Großteil der Teilnehmer findet die Aufmerksamkeitsdarstellungen entweder fast bis gar nicht hilfreich oder hilfreich bis sehr hilfreich. Es gibt kaum neutrale Antworten.

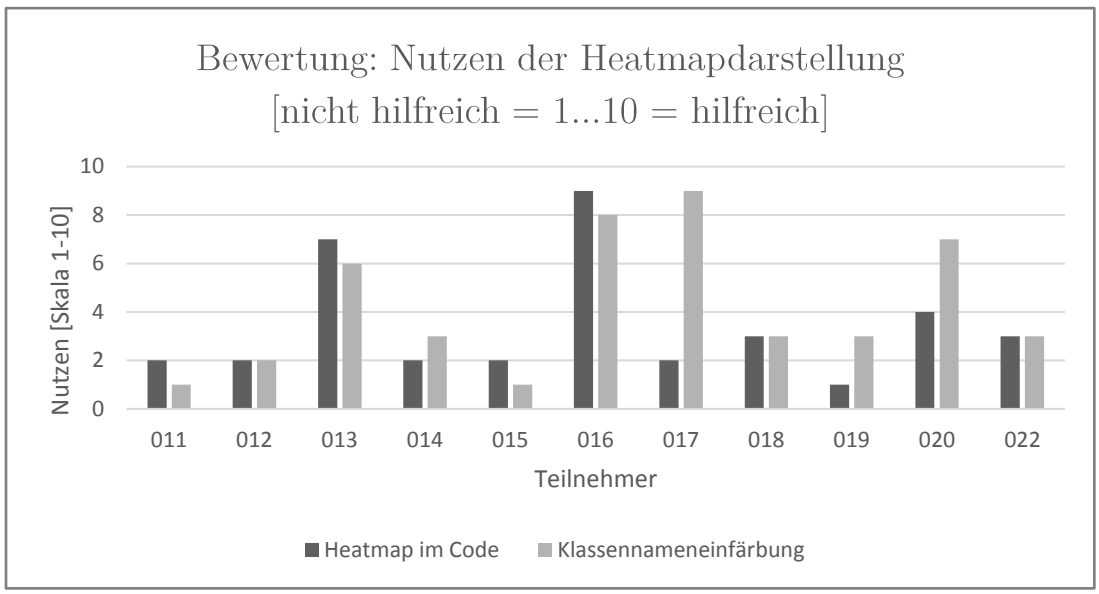


Abbildung 9.5: Bewertung des Nutzens der Heatmapdarstellung

Im nächsten Schritt wurden die Aussagen der Befragung zu den beiden Darstellungen analysiert. Dabei gaben sieben der elf Teilnehmer an, dass die Heatmap ihnen in irgendeiner Weise in unterschiedlichem Maße geholfen habe. Drei Teilnehmer gaben an, dass die Heatmapdarstellung gut als Anhaltspunkt beim Suchen und bei Orientierungslosigkeit diene. Ebenfalls drei der Probanden führten auf, dass die Heatmapverteilung vermutlich sehr kontextabhängig und der Nutzungsgrad daher abhängig von der Aufgabe sei. Dazu gaben zwei weitere Teilnehmer an, dass ihnen die Heatmap vor allem bei ähnlichen Aufgaben geholfen habe, relevante und irrelevante Informationen lokalisieren zu können. Darüber hinaus nannten fünf der elf Teilnehmer, dass die Codehervorhebung durch die Heatmap nicht mit dem eigenen Relevanzempfinden übereinstimmte und sie daher teilweise etwas verwirrt oder irritiert habe. Zu diesen Anzahlen der Aussagen sei angemerkt, dass dies nicht bedeutet, dass der jeweilige Rest der Teilnehmer dem nicht zustimmt, sondern lediglich dass sie dies nicht in ihren Ausführungen erwähnten.

Insgesamt lässt sich somit feststellen, dass die beiden Aufmerksamkeitsdarstellungen dem Großteil der Teilnehmer nur mäßig halfen. Dennoch konnten einige Teilnehmer einen positiven Nutzen aus den Darstellungen ziehen.

$H_{6.1}$: Einfluss auf die Übersichtlichkeit

Die Hypothese $H_{6.1}$ sagt aus, dass sich die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps störend auf die Übersichtlichkeit des Codes auswirkt. Dazu wurde zum einen die subjektive Bewertung der Übersichtlichkeit der Teilnehmer mit und ohne Aufmerksamkeitsdarstellung miteinander verglichen, sowie die aufgenommene Codeelement Suchzeit.

Bei der Bewertung der Übersichtlichkeit ergibt sich für die Teilnehmer ohne Heatmapdarstellung auf einer Skala von 1 bis 10, wobei 1 für unübersichtlich steht und 10 für übersichtlich, eine mittlere Bewertung von 7,7. Die Teilnehmer mit Heatmapdarstellung wiederum haben die Übersichtlichkeit im Schnitt nur mit 6,4 bewertet. Die Abbildung 9.6 zeigt die Bewertungen der Teilnehmer der Größe nach sortiert. Man erkennt deutlich, dass die Werte der Probanden ohne Aufmerksamkeitsdarstellung entweder über oder auf der gleichen Höhe wie die Bewertungen der Teilnehmer mit Heatmap liegen. Die Nullhypothese, dass die Bewertungen der Kontrollgruppe nicht über denen der Treatmentgruppe liegen, wurde mit dem Wilcoxon-Mann-Whitney-Test geprüft. Das Ergebnis zeigt mit p=0,03142, dass die Nullhypothese mit einem Signifikanzniveau von 5% abgelehnt werden kann.

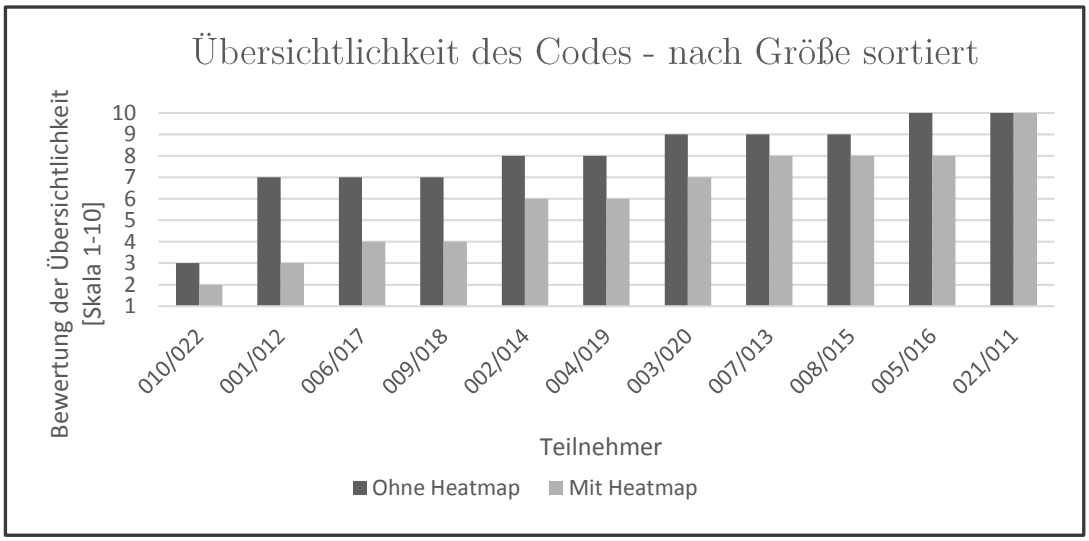


Abbildung 9.6: Bewertung der Übersichtlichkeit

Die Codeelement Suchzeit zeigt einen ähnlichen Trend. Dabei ergibt sich für die Teilnehmer der Kontrollgruppe eine mittlere Zeit zum Suchen des vorgegebenen Elements im Code von 22,98 Sekunden. Die Teilnehmer der Treatmentgruppe hingegen kommen auf eine mittlere Suchzeit von 24,49 Sekunden. Insgesamt sind die Teilnehmer mit der Heatmap im Hintergrund des Codes somit etwa 1,51 Sekunden langsamer. Alle einzelnen, genauen Werte finden sich in einer Tabelle im Anhang in Abschnitt A.3. Die Nullhypothese, dass die Codeelement Suchzeit der Treatmentgruppe nicht über der der Kontrollgruppe liegt, wurde erneut mit dem Wilcoxon-Mann-Whitney-Test geprüft. Mit p=0,1658 muss die Nullhypothese jedoch beibehalten werden. Dennoch lässt sich bei näherer Betrachtung der Werte erkennen, dass zwar die größten drei Werte der gemessenen Zeiten zu Teilnehmern der Kontrollgruppe gehören, der Rest der Werte dieser Gruppe jedoch unter denen der Treatmentgruppe liegt.

Abschließend lässt sich somit feststellen, dass die Übersichtlichkeit des Codes mit Heatmap signifikant schlechter bewertet wurde als der gleiche Code ohne Heatmap. Auch die Suche nach Elementen im Code dauert mit Aufmerksamkeitsdarstellung tendenziell länger. Dieser Effekt konnte jedoch nicht als signifikant nachgewiesen werden.

$H_{6.2}$: Finden relevanten Codes

Die Hypothese $H_{6.2}$ besagt, dass die Darstellung der Aufmerksamkeitsdaten mittels Heatmaps einen Einfluss auf das Finden relevanten Codes im Programm hat. Dazu sollten alle Teilnehmer gegen Ende der Studie im Fragebogen bewerten, wie schnell sie den für die jeweiligen Aufgaben relevanten Code im Programm finden konnten. Die Frage wurde auf einer zehnstufigen Likert-Skala von langsam (1) bis schnell (10) beantwortet. Für die Kontrollgruppe ergibt sich dabei ein Mittelwert von 6,1 und für die Treatmentgruppe von 4,8. Der Unterschied liegt somit bei 1,3. Die exakten einzelnen Werte finden sich im Anhang im Abschnitt A.3. Die Nullhypothese, dass die Aufmerksamkeitsdaten keinen Einfluss auf das Finden relevanten Codes haben, wurde mit dem Wilcoxon-Mann-Whitney-Test mit einem Signifikanzniveau von 5% geprüft. Das Ergebnis zeigt mit p=0,1448, dass der Effekt nicht stark genug ist, um die Nullhypothese ablehnen zu können. Dennoch erkennt man beim Betrachten der nach Größe sortierten Daten deutlich, dass die Bewertungen der Kontrollgruppe stets über den Bewertungen der Treatmentgruppe liegen.

$H_{6.3}$: Verstehen der allgemeinen Programmfunktionalität

Die Hypothese $H_{6.3}$ sagt aus, dass die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps einen Einfluss darauf hat, wie leicht sich ein Bild der generellen Funktionalität des Programms gemacht werden kann. Dazu sollten die Teilnehmer ebenfalls auf einer zehnstufigen Likert-Skala von schwer (1) bis leicht (10) angeben, wie schnell sie verstehen konnten, was die Aufgabe der einzelnen Komponenten und des gesamten Programms ist. Es ergibt sich ebenfalls eine Differenz der durchschnittlichen Bewertung von 1,3, wobei die Kontrollgruppe im Mittel mit 7,3 bewertet hat und die Treatmentgruppe mit 6. Der Abschnitt A.3 im Anhang zeigt alle Daten noch einmal nach Teilnehmer aufgeschlüsselt. Zum Testen der Nullhypothese, dass die Aufmerksamkeitsdarstellung keinen Einfluss hat, wurde wieder der Wilcoxon-Mann-Whitney-Test verwendet. Mit p=0,1088 kann die Nullhypothese jedoch nicht mit einem Signifikanzniveau von 5% abgelehnt werden. Dennoch zeigen die Daten, dass die Bewertungen der Teilnehmer ohne Heatmapdarstellung hier ebenfalls bis auf zwei Ausnahmen über denen der Probanden mit Heatmapdarstellung liegen.

$H_{6.4}$: Störungsgrad bei Softwarewartungsarbeiten

Die Hypothese $H_{6.4}$ lautet, dass sich die Darstellung von Aufmerksamkeitsdaten mittels Heatmaps nicht störend auf die Bearbeitung von Softwarewartungsarbeiten auswirkt. Als Metriken wurden dabei die zugehörigen Fragen im Fragebogen betrachtet, sowie die Aussagen, die diesbezüglich in der abschließenden Befragung gemacht wurden.

Im Fragebogen sollten die Teilnehmer der Treatmentgruppe auf einer zehnstufigen Likert-Skala angeben, als wie störend oder ablenkend sie die Heatmap im Hintergrund des Codes und die Klassennameneinfärbung im Package Explorer empfanden. 1 stand dabei für nicht störend und 10 für störend. Für die Heatmap ergibt sich dabei ein Mittelwert von 4,5, wobei die Werte zwischen 1 und 9 streuen. Für die Klassennameneinfärbung hingegen ergibt sich eine mittlere Bewertung von 2,2, wobei die Werte nur noch zwischen 1 und 5 liegen. Die Heatmap stört demnach eher die Lesbarkeit des Codes als die Färbung im Package Explorer. In Abbildung 9.7 sieht man alle Bewertungen noch einmal einzeln in einem Säulendiagramm dargestellt. Auffällig an dieser Stelle ist, dass der Teilnehmer mit der ID 013 die Heatmap im Code, sowohl als relativ hilfreich (7) (siehe Abbildung 9.5), als auch als relativ störend (8) bewertet hat. Bei den anderen Teilnehmern wiederum wurden die Aufmerksamkeitsdarstellungen hingegen entweder als hilfreich oder als störend erachtet.

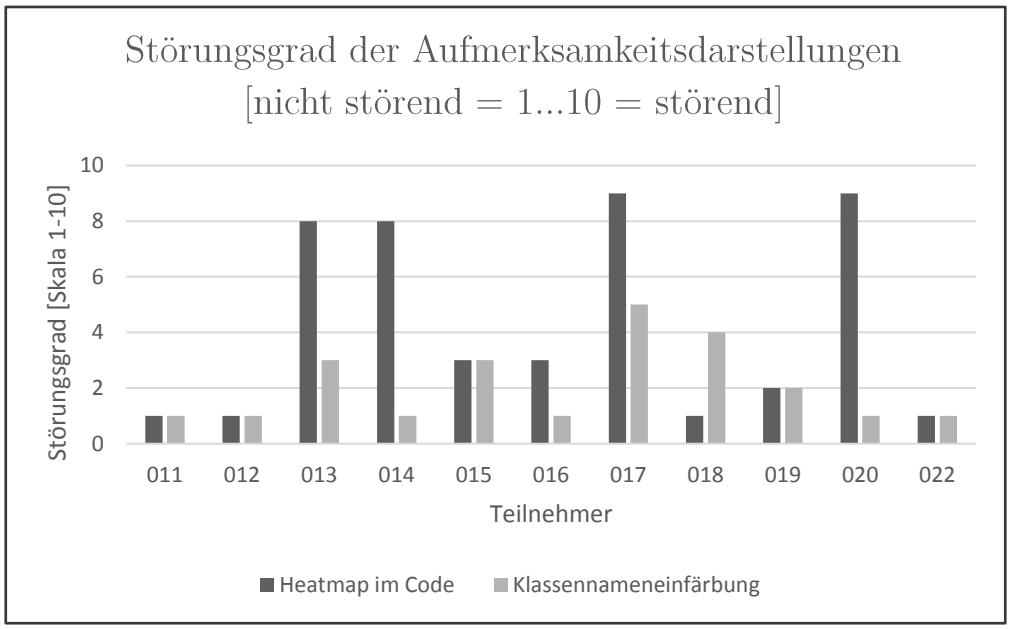


Abbildung 9.7: Bewertung des Störungsgrad der Aufmerksamkeitsdarstellung

Darüber hinaus wurden die Probanden im Fragebogen ebenfalls gefragt, ob sie eine Funktion zum Ausschalten der Heatmap (beispielsweise mittels Toggle-Button) genutzt hätten. Dabei gaben sechs der elf Teilnehmer an, dass sie diese

Möglichkeit zumindest temporär verwendet hätten. Zusätzlich wurde sich bei den Teilnehmern auch in der Befragung danach erkundigt, als wie ablenkend, fehlleitend oder störend sie die beiden Aufmerksamkeitsrepräsentationen empfanden. Dazu wurden alle negativen Äußerungen aus den transkribierten Audioaufnahmen extrahiert und analysiert. Drei der Teilnehmer gaben demnach an, dass die Heatmap im Hintergrund des Codes die Lesbarkeit erschwere. Ein Teilnehmer teilte mit, dass die Heatmap vom eigentlichen Code ablenke. Dagegen äußerten sieben Teilnehmer explizit oder implizit, dass sie die Heatmap nicht als störend aufgefasst hätten. Speziell zu der Klassennameneinfärbung im Package Explorer nannten zwei Teilnehmer, dass die Farben sie teils fehlgeleitet hätten und sie so mehrfach in für sie eigentlich wenig relevanten Klassen gelandet wären. Wiederum zwei Teilnehmer machten auch deutlich, dass sie die Färbung nicht wirklich verstanden hätten, nicht wussten was die einzelnen Farben bedeuteten und nach welchem System sich diese abhängig von der aktuell geöffneten Klasse änderten.

Insgesamt lässt sich somit feststellen, dass die Heatmap zumindest in Teilen die Bearbeitung von Softwarewartungsarbeiten stört. Die Klassennameneinfärbung im Package Explorer scheint dagegen keinen großen Störfaktor darzustellen.

H_7: Abhängigkeit von der Entwicklungserfahrung

Die Hypothese H_7 sagt aus, dass der Effekt der Heatmap von der Erfahrung des Entwicklers abhängt. Um dies zu untersuchen wurden sämtliche quantitativen Metriken, die den Effekt der Heatmap maßen, noch einmal in Abhängigkeit mit der Softwareentwicklungs-, Javaentwicklungs- und Berufserfahrung betrachtet. Zunächst einmal konnte dabei festgestellt werden, dass die Experten unter den Teilnehmern sowohl den Nutzen, als auch den Störungsgrad der Heatmap, sowie der Klassennameneinfärbung alle nur zwischen 1 und 3 bewerteten. Damit liegen sie in allen vier Fällen unter dem Durchschnitt der restlichen Probanden. Bei den Experten handelte es sich hier um die Teilnehmer mit mehr als vier Jahren Berufserfahrung bzw. sehr viel Softwareentwicklungserfahrung. Die beiden Aufmerksamkeitsdarstellungen hatten daher nach der subjektiven Einschätzung einen geringeren Einfluss auf die Teilnehmer mit mehr Berufs- bzw. Entwicklungserfahrung. Das heißt, die Darstellungen haben ihnen weniger genützt, sie aber auch weniger gestört. Des Weiteren lässt sich bei dem Pupillendurchmesser feststellen, dass sich der Effekt stärker bei den Teilnehmern mit weniger Berufserfahrung zeigt. Die Differenz des Pupillendurchmessers überwiegt demnach bei den Teilnehmern, die zwischen keiner und zwei Jahren einschlägiger Berufserfahrung haben. Ein ähnlicher Trend zeigt sich bei der Fixationsdauer. Dort liegen

die Werte der Kontrollgruppenteilnehmer mit weniger Javaentwicklungserfahrung eher über denen der Treatmentgruppe. Bei den Teilnehmern mit mehr Javaentwicklungserfahrung hingegen kehrt sich dieser Effekt eher um und die durchschnittlichen Fixationsdauern der Treatmentgruppe liegen über den Werten der Kontrollgruppe. Auf der anderen Seite scheint die Entwicklungs- bzw. Berufserfahrung der Teilnehmer bei den restlichen Metriken Fixationsanzahl, Sakkadenanzahl, Sakkadendauer, Bearbeitungszeit und Korrektheit keinen Effekt zu haben. Bei der Fixationsanzahl, der Fixationsdauer, der Sakkadendauer, der Bearbeitungszeit, sowie der Korrektheit lässt sich dafür feststellen, dass die Werte aller Teilnehmer, aber auch der Kontroll- und Treatmentgruppe für sich, bei den Teilnehmern mit mehr Berufserfahrung näher zusammen liegen als bei den Teilnehmern mit weniger Berufserfahrung. Insgesamt lässt sich demnach feststellen, dass der Effekt der Aufmerksamkeitsdarstellungen bei den erfahreneren Teilnehmern geringer ist als bei den unerfahreneren Teilnehmern, oder teilweise sogar gegensätzlich zu ihnen.

10 Auswertung und Schlussfolgerung

In diesem Kapitel werden zunächst die anfangs definierten Forschungsfragen anhand der mit den Studiendaten getesteten Hypothesen beantwortet. Darauf folgend werden die gesamten Ergebnisse des Experiments zusammengefasst und eine abschließende Schlussfolgerung daraus gezogen. Zuletzt wird noch die Validität der Ergebnisse diskutiert, wie diese sichergestellt wurde und an welchen Stellen sie möglicherweise eingeschränkt ist.

10.1 Beantwortung der Forschungsfragen

Die zuvor in Kapitel 6.2 aufgestellten Forschungsfragen werden in diesem Abschnitt nacheinander anhand der formal geprüften Hypothesen beantwortet und die Ergebnisse anschließend zusammengefasst. Die erste Forschungsfrage RQ_1 bezog sich darauf, ob die Betrachtungshäufigkeit von Abschnitten einen guten Indikator für dessen Relevanz darstellt. Die Ergebnisse zeigen, dass die objektiv über die Betrachtungshäufigkeit gemessenen Bereiche weitestgehend mit den subjektiv als relevant empfundenen Bereichen übereinstimmen. Lediglich in wenigen Fällen gibt es Abweichungen aufgrund von besonders hoher oder niedriger Komplexität, der Länge einer Klasse oder hoher Informationsdichte.

Die zweite Forschungsfrage RQ_2 beschäftigte sich damit, ob die Darstellung von Aufmerksamkeitsdaten die Effizienz von Softwarewartungsarbeiten verbessert. Dazu wurden die Bearbeitungszeit, der visuelle Aufwand, die kognitive Last, sowie die Effizienz der Informationssuche für die beiden Teilnehmergruppen mit und ohne Aufmerksamkeitsrepräsentation jeweils miteinander verglichen. Beim visuellen Aufwand stellte sich heraus, dass die Heatmapdarstellung einen negativen Effekt auf die Sakkadenanzahl hat, der aber nur bei der Sakkadenanzahl als signifikant nachgewiesen werden konnte. Das heißt, mit Heatmap springen die Teilnehmer häufiger mit ihrem Blick hin und her. Bei der Bearbeitungszeit, sowie der kognitiven Last hingegen, wirkt sich die Aufmerksamkeitsdarstellung mittels Heatmap positiv aus. Bei der Effizienz der Informationssuche, die anhand der Informationssuche gemessen wurde, konnte kein Effekt festgestellt werden.

Die dritte Forschungsfrage RQ_3 betraf den Einfluss der Darstellung von Aufmerksamkeitsdaten auf die Qualität von Softwarewartungsarbeiten. Dazu wurde die Korrektheit der bearbeiteten Aufgaben betrachtet. Es konnte jedoch kein signifikanter Unterschied zwischen Kontroll- und Treatmentgruppe festgestellt werden. Das heißt, die beiden Aufmerksamkeitsdarstellungen hatten in dieser

Studie keinen nachweisbaren positiven oder negativen Effekt auf die Qualität der Softwarewartungsarbeiten.

Die vierte Forschungsfrage RQ_4 untersuchte zunächst, inwiefern die Aufmerksamkeit der Teilnehmer überhaupt auf die durch die Heatmap hervorgehobenen Bereiche bzw. auf die durch die Klassennamenfärbung im Package Explorer markierten Klassen, gelenkt wurde. In einschlägigen Studien konnte bereits herausgefunden werden, dass andere Arten, Eye-Tracking Daten für den Nutzer zu visualisieren, den Fokus auf diese Bereiche lenken. In der Studie dieser Arbeit wurde dies für Heatmaps auf Code geprüft. Dabei konnte festgestellt werden, dass bis auf jeweils eine Ausnahme alle Teilnehmer durch die Heatmap bzw. Klassennameneinfärbung auf die hervorgehobenen Bereiche gelenkt wurden und diese Darstellung in den meisten Fällen auch intuitiv verstanden werden konnte. Dennoch war die Hervorhebung nicht zu präsent, sodass sie bei Bedarf trotzdem ignoriert werden konnte.

Die fünfte Forschungsfrage RQ_5 schloss daran an und prüfte, inwiefern diese Aufmerksamkeitslenkung den Teilnehmern einen Nutzen bei der Bearbeitung von Softwarewartungsarbeiten brachte. Dazu wurden zum einen die subjektiven, quantitativen Antworten im Fragebogen, sowie die Aussagen in der abschließenden Befragung der Studie betrachtet. Die Ergebnisse variieren diesbezüglich. Etwa zwei Drittel der Teilnehmer konnte in Teilen einen positiven Nutzen aus den Heatmapdarstellungen ziehen. Dennoch führte die Aufmerksamkeitsrepräsentation teilweise zu Irritation oder Verwirrung, wenn diese nicht mit dem eigenen Relevanzempfinden im aktuellen Aufgabenkontext übereinstimmte. Auf einige Teilnehmer hatten die Heatmapdarstellungen auch überhaupt keinen Einfluss, weder positiv noch negativ. Die Antworten der Teilnehmer im Fragebogen zeigen wenig mittlere Bewertungen des Nutzens. Sie gehen entweder stark in die Richtung „hilfreich“ oder „nicht hiflreich“. Nur etwa ein Drittel der Teilnehmer bewertete mindestens eine der Darstellungen als hilfreich. Der Rest der Teilnehmer empfand sie als wenig bis gar nicht hilfreich, wobei die Klassennameneinfärbung im Package Explorer leicht besser abschnitt als die Heatmap im Hintergrund des Codes. Die Forschungsfrage RQ_5 kann damit nicht in eine eindeutige Richtung beantwortet werden. Obwohl die Aufmerksamkeitsdarstellungen einem Teil der Probanden nützten, so scheint der Nutzungsgrad noch von weiteren, subjektiven Faktoren abzuhängen.

Die sechste Forschungsfrage untersucht, inwiefern die Heatmapdarstellungen einen Störfaktor bei der Bearbeitung von Softwarewartungsarbeiten darstellen. Dazu wurde zunächst der Effekt auf die Übersichtlichkeit des Codes gemessen.

Das Ergebnis zeigt, dass die Heatmap sich negativ sowohl auf die Übersichtlichkeit der Codedarstellung, als auch auf das Finden von relevantem Code, sowie die Schwierigkeit, die generelle Funktionalität des Programms zu verstehen, auswirkt. Gut die Hälfte der Teilnehmer gab außerdem an, dass sie die Heatmap bei Bedarf abgeschaltet hätten, wenn diese Funktion verfügbar gewesen wäre. In der Bewertung des Störungsgrads im Fragebogen gaben etwa ein Drittel der Teilnehmer an, die Heatmap als störend empfunden zu haben. Die Klassennameneinfärbung im Package Explorer wurde dagegen als deutlich weniger störend bewertet. In der abschließenden Befragung wurde teilweise genannt, dass die Heatmap an einigen Stellen die Lesbarkeit des Codes erschwere oder vom eigentlichen Inhalt des Codes ablenke.

Die Forschungsfrage RQ_7 beinhaltete die Abhängigkeit des Effekts der Aufmerksamkeitsdarstellungen von der Erfahrung der Entwickler. Die Ergebnisse zeigen, dass die Heatmap, sowie die Klassennameneinfärbung weniger Einfluss auf die Teilnehmer mit sehr viel Softwareentwicklungs- oder Praxiserfahrung hatten. Die nachgewiesenen Effekte zeigten sich daher deutlicher bei den unerfahreneren Entwicklern.

10.2 Zusammenfassung der Studienergebnisse

Die Studie dieser Arbeit beinhaltete im Wesentlichen die Untersuchung dreier Aspekte. Durch die Auswahl konkreter Aufmerksamkeitsdaten, einer Darstellung dieser und eines Nutzungskontexts konnte jeweils getestet werden, inwiefern diese geeignet und hilfreich sind. Die Ergebnisse zeigen, dass sowohl die Darstellung mittels Heatmaps, als auch die Einfärbung der Klassennamen im Package Explorer in Heatmap Farben dazu geeignet sind die Aufmerksamkeit auf die entsprechend hervorgehobenen Bereiche zu lenken. Dennoch erschwert die Heatmap teilweise die Lesbarkeit des Codes und lenkt von diesem ab. Es konnte nachgewiesen werden, dass sich durch die beiden Darstellungen die Sakkadenanzahl signifikant erhöhte, was für einen höheren visuellen Aufwand spricht. Auch die Übersichtlichkeit des Codes, sowie der Schwierigkeitsgrad des Funktionalitätsverständnisses verschlechterten sich signifikant. Relevante Codeteile konnten ebenfalls nicht schneller gefunden werden. Darüber hinaus wurde geprüft, inwiefern es sinnvoll ist, als Aufmerksamkeitsdaten die Blickhäufigkeit in Form von der lokalen Fixationsanzahl zu wählen. Dabei kam heraus, dass diese Daten zwar einigen Teilnehmern zur ersten Orientierung oder bei der Suche im Code halfen, diese jedoch auch sehr kontextabhängig sind. Das heißt, das Blickverhalten variiert an manchen Stellen stark abhängig von der aktuellen Aufgabe des

Entwicklers. Aus diesem Grund wirkte die Darstellung dieser Daten daher teilweise auch irritierend. Bezüglich des Nutzungskontextes lässt sich feststellen, dass die Darstellung der Blickverteilung durch Heatmaps im Bereich der Codeerweiterung, -editierung und Fehlerbehebung nur eine begrenzte Hilfestellung darstellt. Insbesondere für erfahrenere Entwickler, die es gewohnt sind, sich anhand von Package-, Klassen- und Methodennamen und mittels Suche im Programm zurecht zu finden, bot die gewählte Aufmerksamkeitsdarstellung wenig Mehrwert. Dennoch stellte sich eine leichte, wenn auch nicht signifikante Verbesserung der kognitiven Last heraus.

Eine Problematik bei der Umsetzung dieser Darstellung mittels Eye-Tracking als „By-Product Approach“ besteht zudem darin, dass sich durch die Heatmap ein selbstverstärkender Effekt einstellen könnte. Durch die Hervorhebung bestimmter Bereiche, wird die Aufmerksamkeit des Entwicklers vorwiegend auf diese Abschnitte gelenkt. Dies führte dazu, dass wenn die Aufmerksamkeitsdaten dieses Entwicklers ebenfalls mit in die aktualisierte Heatmap fließen, dass die hervorgehobenen Bereiche unter Umständen stärker hervorgehoben werden, als es dem Relevanzempfinden des Entwicklers entspricht, da er lediglich durch die bestehende Darstellung darauf gelenkt wurde. Auch Fehlleitungen auf eher unwichtige Bereiche könnten sich so aufschaukeln. Bei der konkreten Umsetzung ist daher darauf zu achten, eine geeignete Nachverarbeitung der Daten vorzunehmen.

Insgesamt lässt sich somit festhalten, dass die Darstellung der Blickverteilung mittels Heatmaps und die Darstellung von Klassenwechseln mittels Klassennameneinfärbung in einigen Fällen zwar den Teilnehmern geholfen haben, der Mehrwert dennoch sehr gering war. Zusätzlich hat die Heatmapdarstellung auch teils gestört und könnte daher noch optimiert werden.

10.3 Bedrohungen der Validität

In diesem Abschnitt wird beleuchtet, an welchen Stellen die Validität der Ergebnisse gefährdet oder eingeschränkt sein könnte. Ebenfalls wird beschrieben, auf welche Weise die Validität der Resultate sichergestellt und so Bedrohungen der Validität entgegen gewirkt wurde.

10.3.1 Validität der Konstruktion

Die Validität der Konstruktion beschreibt, wie gut die Abbildung von der Korrelation zwischen der zu untersuchenden Ursache und Wirkung auf die tatsächlich in der Studie umgesetzte Korrelation zwischen Eingabe und Ergebnis.

Problematisch an dieser Stelle ist die durch das Eye-Tracking und die Aufmerksamkeitsdarstellungen erzwungene, künstliche Entwicklungsumgebung im Browser, sowie das „Entwickeln auf Papier“, also das Aufschreiben der Programmierlösungen statt dem tatsächlichen Editierens des Codes. Auch das Syntax Highlighting stimmte wie bereits in Abschnitt 6.5.1 beschrieben nicht exakt mit dem gängigen Eclipse Standard überein. Die Unterschiede waren jedoch minimal und wurden von keinem der Teilnehmer kommentiert oder als störend empfunden. Um sicherzustellen, dass die Ergebnisse auch in der Umgebung einer tatsächlichen IDE bei natürlichem Entwickeln gelten, müsste dies dennoch noch einmal getestet werden.

Darüber hinaus konnten die Teilnehmer der Treatmentgruppe möglicherweise schlussfolgern, dass in der Studie der Effekt der Heatmap gemessen wurde. Dennoch wurde ihnen nicht mitgeteilt, welche Metriken konkret untersucht wurden. Zusätzlich wurden sie auch nicht dazu aufgefordert die Aufmerksamkeitsdarstellungen zu nutzen, sondern bekamen lediglich eine Erklärung, was diese zu bedeuten hatten. Der Rückmeldung der Teilnehmer zufolge, konzentrierten sich diese auch schnell ausschließlich auf ihre gegebenen Aufgaben, statt sich auf die Wirkung ihrer Arbeit auf die Studienergebnisse zu fokussieren.

Außerdem wurden Teilnehmern realistische Softwarewartungsaufgaben gegeben. So wurde der Arbeit mit dem Code ein Sinn verliehen, statt die Probanden in fiktive Szenarien hinein zu versetzen und sie ohne konkretes Ziel das Programm verstehen zu lassen.

10.3.2 Interne Validität

Die Interne Validität bezieht sich darauf, dass die gemessenen Effekte tatsächlich durch die beobachteten Faktoren verursacht wurden und nicht durch unkontrollierte Zufallsgrößen. Diese wurde zum einen durch die Verwendung eines Between-Group Designs sichergestellt, da eventuelle Lerneffekte der Teilnehmer so ausgeschlossen wurden. Zusätzlich wurden die Teilnehmer zur Verschwiegenheit über die Studieninhalte verpflichtet. Eventuelle Einflüsse durch eine Rot-Grün-Sehschwäche wurden ebenfalls verhindert.

Darüber hinaus wurde dafür gesorgt, dass die Kontroll- und Treatmentgruppe jeweils gleich viele Aufgaben des Typs A und B bearbeiteten. So wurden Einflüsse durch unterschiedlichen Umfang oder abweichende Komplexität ausgeblendet. Auch bestanden beide Gruppen aus jeweils gleich vielen Studenten, PhD Studenten und Experten. Dennoch konnte der Erfahrungsstand bzgl. Praxis- und Entwicklungserfahrung allgemein und speziell in Java nicht so weit

kontrolliert werden, dass dieser in beiden Gruppen exakt gleich war. In der Treatmentgruppe gab es eine Person mehr, die statt wenig, sehr viel Javaentwicklungserfahrung hatte, sowie eine Person mehr, die statt wenig, viel Softwareentwicklungserfahrung hatte. Zusätzlich liegt der Median der Praxiserfahrung der Treatmentgruppe bei „1-2 Jahren“ und bei der Kontrollgruppe bei „maximal einem Jahr“. Die Unterschiede sind jeweils sehr gering, müssen aber dennoch bei der Bewertung der Ergebnisse berücksichtigt werden, insbesondere, da die Abweichung zugunsten der Treatmentgruppe fällt.

Eventuelle Einflüsse durch Abweichungen in der gesamten Bearbeitungszeit, der Qualität der Eye-Tracking Aufnahmen und des Lesens der Java API Dokumentation wurden außerdem ausgeschlossen, indem diese kontrolliert und bei den betroffenen Metriken herausgerechnet wurden. Des Weiteren wurde bei allen Teilnehmern für ein gleiches Studiensetup und ähnliche Lichtverhältnisse gesorgt. Zusätzlich bekamen alle Probanden die exakt gleiche Experimenteinführung und Aufgabeneinleitung, da diese schriftlich vorgelegt wurden.

Als weitere Maßnahme zur Gewährleistung der internen Validität wurden eventuelle Annahmen, beispielsweise das Leiten der Aufmerksamkeit durch die Heatmap, direkt mit in der Studie geprüft. So wurden etwaige fehlerhaft geschlossene Erkenntnisse ebenfalls unterbunden.

10.3.3 Validität der Schlussfolgerung

Die Validität der Schlussfolgerung wird auch statistische Validität genannt und befasst sich damit, dass die gewonnenen Erkenntnisse auf statistisch korrekte Weise gewonnen wurden. Dies wurde sichergestellt, indem für alle verwendeten Signifikanztests zunächst die Voraussetzungen geprüft und auf diese Weise eingehalten wurden. Auch wurde stets ein Signifikanzniveau von 5% betrachtet.

10.3.4 Externe Validität

Die externe Validität bezieht sich darauf, inwiefern die gewonnen Erkenntnisse sich generalisieren lassen. Dabei wurde zum einen darauf geachtet, die Versuchungssubjekte, -objekte und die Umgebung so zu wählen, dass diese möglichst repräsentativ sind. Es wurde zusätzlich sichergestellt, dass die Teilnehmer nicht nur aus Studenten ohne Praxiserfahrung, sondern auch aus Experten und PhD Studenten mit mehr, auch praktischer Entwicklungserfahrung bestanden. Dennoch müsste für eine bessere Verallgemeinerbarkeit eine größere Stichprobe gewählt werden. Da die Ergebnisse zeigen, dass der Effekt der beiden Aufmerksamkeitsdarstellungen auf die erfahreneren Entwickler geringer war, müsste die

Ursache dieses Zusammenhangs noch genauer geklärt werden, um aussagen zu können, inwiefern sich diese Ergebnisse auf die Praxis übertragen ließen.

Es handelte sich außerdem um ein vergleichsweise kleines Programm in der Sprache Java. Da die Programmiersprache an dieser Stelle keinen großen Einfluss hat, sollten die Erkenntnisse auch für andere Sprache gelten. Darüber hinaus lässt sich für größere Programme feststellen, dass der Einfluss und das Nutzungspotenzial in größeren Projekten mit mehr Dateien vermutlich verstärkt würden. Für zuverlässige Aussagen müsste aber auch dies überprüft werden.

Aufgrund der möglichst praxisnahen Umgebung und der realistischen Aufgabenstellungen gelten die Ergebnisse demnach für kleinere Projekte, an denen Änderungsaufgaben in überschaubarem Umfang vorgenommen werden. Ebenfalls nicht getestet wurde der Effekt über einen längeren Zeitraum hinweg. Auch dabei könnte es sich um einen verstärkenden Faktor des Nutzungspotenzials der Heatmapdarstellungen handeln.

11 Zusammenfassung und Ausblick

In diesem Kapitel werden nun die Vorgehensweise, Erkenntnisse und Ergebnisse dieser Arbeit zusammengefasst. Abschließend wird ein ausführlicher Ausblick gegeben, wie der in dieser Arbeit vorgestellte Ansatz noch weiter verbessert und darauf aufbauend weiter vorgegangen werden könnte.

11.1 Zusammenfassung

Das Ziel dieser Arbeit war die Evaluation des Ansatzes das Teilen von Aufmerksamkeit mittels Eye-Tracking in der Software-Wartung anzuwenden. Dazu wurde zunächst die bei dieser Arbeit noch bestehende Problematik beschrieben, Eye-Tracking bei Studien im Bereich der Softwareentwicklung durchzuführen. Die Schwierigkeiten von scrollbaren und dynamischen Inhalten wurden für die Studie der Arbeit umgangen, indem die Entwicklungsumgebung Eclipse als Webseite mit wenigen Basisfunktionalitäten nachgebaut wurde.

Im Anschluss daran wurde der Ansatz des Aufmerksamkeitstransfers näher untersucht, indem verschiedene Aufmerksamkeitsdaten, mögliche Darstellungen, sowie Nutzungsszenarien im Kontext der Software-Wartung exploriert wurden. Daraus wurde eine Auswahl getroffen und diese in einem Experiment getestet. Dabei wurde untersucht, inwiefern die Darstellung von Blickverteilungen durch Heatmaps und die Darstellung von Klassensprüngen durch Klassennameneinfärbung bei der Bearbeitung von Softwarewartungsarbeiten als Hilfestellung dienen. Die 22 Teilnehmer, darunter Studenten, PhD Studenten und Experten, bekamen dazu kleine Erweiterungs-, Änderungs-, sowie Fehlerbehebungsaufgaben an einem Javaprogramm. Es stellte sich heraus, dass die Heatmap und Klassennameneinfärbungen einigen Teilnehmern zur Orientierung und Suche im Code halfen und sich durch die beiden Aufmerksamkeitsdarstellungen die kognitive Last leicht verringerte. Dennoch boten sie überwiegend keinen großen Nutzen, wobei die Einfärbung der Klassennamen als hilfreicher empfunden wurde als die Heatmap im Hintergrund des Codes. Stattdessen wirkte sich die Aufmerksamkeitsdarstellung negativ auf den visuellen Aufwand, die Codeübersicht, das Funktionalitätsverständnis und das Finden relevanten Codes aus. Insbesondere die Teilnehmer mit sehr viel Entwicklungserfahrung konnten nur wenig bis keinen Nutzen aus den Darstellungen ziehen.

11.2 Ausblick

Diese Arbeit diente als Pilotstudie für den allgemeinen Ansatz Aufmerksamkeit in der Software-Wartung zu transferieren. Auf Basis der gewonnenen Erkenntnisse und Ergebnisse ließe sich zur weiteren Forschung an mehreren Punkten anknüpfen. Diese beziehen sich zum Teil auf die Optimierung der verwendeten Darstellungen und Bedingungen und zum anderen auf die Weiterführung des Ansatzes zu anderen Zwecken oder in anderen Kontexten.

Zum einen ließen sich die Versuchsbedingungen für die Aufnahme und Darstellung von Aufmerksamkeitsdaten mittels Eye-Tracking erheblich verbessern, wenn es möglich wäre, Eye-Tracking direkt bei der Arbeit mit Entwicklungsumgebungen im Hintergrund nutzen zu können. Dazu wäre es ebenfalls nötig, die entsprechende Darstellung der Aufmerksamkeitsdaten in Eclipse vorzunehmen. Wenn es auf diese Weise möglich wäre, direkt in der IDE zu arbeiten, könnte dies die Blickverteilung auch nochmals präzisieren. Darüber hinaus ließen sich der Störungsgrad der Heatmap und die resultierende erschwerte Lesbarkeit des Codes verbessern, indem ein Farbschema gewählt würde, dass sich weniger stark mit dem Syntax Highlighting des Codes überlagert. Beispielsweise könnten Graustufen eine bessere Wahl sein, um so auch Probleme durch Rot-Grün-Schwäche zu umgehen. Den Kommentaren der Teilnehmer konnte außerdem entnommen werden, dass es teilweise irritierte, wie nah die Heatmap an den Eye-Tracking Daten lag. Es gab dementsprechend teils auch schwach hervorgehobene Flecken auf weißer Fläche oder Bereiche, die sich nicht exakt mit den Zeilen im Code deckten. Man könnte dementsprechend die Darstellung eventuell optimieren, indem man die Blickdaten noch weiter verarbeitet und so ein besseres Mapping von Aufmerksamkeitsdaten mittels Eye-Tracking und Code erhält. Auch komplett andere Visualisierungstechniken könnten sinnvoll sein.

Des Weiteren wäre es ratsam die in dieser Arbeit getesteten Aufmerksamkeitsdarstellungen in Projekten größeren und somit realistischeren Umfangs zu testen. Obwohl die beiden gewählten Darstellungen sich in der durchgeführten Studie nicht als große Hilfestellung erwiesen, könnten sie in einem Programm mit mehr als einem Package, mehr als acht Klassen und mehr Zeilen Code sich eines größeren Nutzens erweisen. Gleiches gilt für die Arbeit über einen längeren Zeitraum. Bei der 50 minütigen Arbeit mit Code ist es weniger und seltener nötig, sich erneut im Programm zurecht zu finden und sich ein Bild von einer neuen Klasse zu machen, da man sich an ihren Inhalt unter Umständen nicht mehr genau erinnern kann. In dem Zuge könnte es auch sinnvoll sein, nicht die

Aufmerksamkeitsdaten anderer Personen, sondern die eigenen Eye-Tracking Daten aufzuzeichnen und darzustellen. Auf diese Weise könnte es als eigene Gedächtnisstütze dienen, um sich schneller und einfacher zu erinnern, wo der eigene Fokus bei der letzten Arbeit mit dem Programm lag. Es wäre denkbar, einen ähnlichen Ansatz wie das Eclipse Plugin Mylyn nur mit Eye-Tracking Daten zu verfolgen, indem man während der Bearbeitung einer definierten Aufgabe, das Blickverhalten aufzeichnet und diese Blickverteilung über die Klassen der aktuellen Aufgabe zuweist. Anschließend könnte zu einer anderen Aufgabe übergegangen werden. Sobald eine Aufgabe erneut ausgewählt wird, würden die entsprechend während der Bearbeitung dieser einen Aufgabe aufgenommenen Daten durch Hervorhebungen dargestellt.

Darüber hinaus ließe sich der Ansatz auch noch auf andere Kontexte übertragen. Es könnte zum Beispiel erfolgsversprechend sein, die Darstellung von Blickdauern bzw. Fixationsdauern bei Bedarf beim Finden von Bugs in Programmen zu nutzen. Es müsste getestet werden, ob häufiger und stärker fokussierte Bereiche mehr Bugs enthalten als weniger häufig betrachtete Abschnitte. Wenn dies der Fall ist, könnte die Darstellung helfen, diese Fehler schneller zu finden. So würde gleichzeitig auch auf kompliziertere Bereiche hingewiesen werden. Diese könnten auch für das Durchführen von Reviews interessant sein.

Außerdem könnte man untersuchen inwiefern der Transfer von Aufmerksamkeit bei anderen Artefakten, wie Spezifikationen oder anderen Softwaredesigndokumenten, nützlich sein könnte. Laut Binkley et al. sind das Lesen und Verstehen von Quellcode grundsätzlich verschieden zu dem Lesen und Verstehen von natursprachlichem Text [45]. Die Ergebnisse lassen sich daher vermutlich nicht einfach auf andere Artefakte als Programmcode übertragen.

Literaturverzeichnis

[1] R. Stein und S. E. Brennan, „Another Person's Eye Gaze as a Cue in Solving Programming Problems," in *Proceedings of the 6th International Conference on Multimodal Interfaces*, State College, Pennsylvania, USA, 2004.

[2] R. Bednarik, „Expertise-dependent visual attention strategies develop over time during debugging with mutliple code representations," in *International Journal of Human-Computer Studies*, Februar 2012.

[3] K. Schneider, „Rationale as a By-Product," in *Rationale Management in Software Engineering*, Springer, Berlin, Heidelberg, 2006, pp. 91-109.

[4] A. Duchowski, Eye Tracking Methodology: Theory and Practice, Springer-Verlag New York Inc., 2007.

[5] K. Rayner, „Eye Movements in Reading and Information Processing: 20 Years of Research," *Psychological Bulletin,* Bd. 124, Nr. 3, pp. 372-422, 1998.

[6] Z. Sharafi, Z. Soh und Y.-G. Huéhéneuc, „A systematic literature review on the usage of eye-tracking in software engineering," *Information and Software Technology,* Bd. 67, pp. 79-107, November 2015.

[7] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch und S. D'Mello, „Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers," in *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014.

[8] R. Bednarik und M. Tukiainen, „An Eye-tracking Methodology for Characterizing Program Comprehension Processes," in *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, San Diego, California, USA, März 2006.

[9] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott und M. Züger, „Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development," in *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014.

[10] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd und T. Fritz, „Eye gaze and interaction contexts for change tasks - Observations and potential," *The Journal of Systems and Software 128,* pp. 252-266, 31. März 2016.

[11] 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,* Bd. 32, Nr. 12, pp. 971-987, Dezember 2006.

[12] J. Sillito, K. De Volder, B. Fisher und G. Murphy, „Managing Software Change Tasks: An exploratory study," in *Proceedings of the International Symposium on Empirical Software Engineering*, 2005.

[13] T. Fritz, D. C. Shepherd, K. Kevic, W. Snipes und C. Bräunlich, „Developers' Code Context Models for Change Tasks," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, China, 2014.

[14] M. P. Robillard, „Automatic Generation of suggestions for program investigation," in *SIGSOFT Software Engineering Notes*, Lisbon, Portugal, 2005.

[15] D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy und C. Swart, „Reactive Information Foraging: An Empirical Investigation of Theory-based Recommender Systems for Programmers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Austin, Texas, USA, 2012.

[16] T. D. LaToza, D. Garlan, J. D. Herbsleb und B. A. Myers, „Program Comprehension as Fact Finding," in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, Dubrovnik, Croatia, September 2007.

[17] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif und S. Tamm, „Eye Movements in Code Reading: Relaxing the Linear Order," in *2015 IEEE 23rd International Conference on Program Comprehension*, Florence, Italy, 2015.

[18] P. A. Orlov, „Experts vs Novices in programming: "Who knows where to look?",“ in *Eye Movements in Programming: Models to Data. Proceedings of the Third International Workshop*, Joensuu, Finland, 2016.

[19] M. E. Crosby und J. Stelovsky, „How do we read algorithms? A case study,“ *IEEE Computer,* Bd. 23, pp. 24-35, Januar 1990.

[20] M. E. Crosby, J. Scholtz und S. Wiedenbeck, „The Roles Beacons Play in Comprehension,“ in *14th Annual Workshop of the Psychology of Programming Interest Group*, 2002.

[21] R. Minelli, „Towards Self-Adaptive IDEs,“ in *2014 IEEE International Conference on Software Maintenance and Evolution*, Victoria, BC, Canada, 2014.

[22] R. Minelli, „DFlow: Towards Self-Adaptive IDEs,“ [Online]. Available: http://dflow.inf.usi.ch/. [Zugriff am 27. 07. 2018].

[23] E. R. Grant und M. J. Spivey, „Eye movements and problem solving: Guiding attention guides thought,“ *Psychological Science,* Nr. 14, pp. 462-466, September 2003.

[24] D. Litchfield und L. J. Ball, „Rapid communication: Using another's gaze as an explicit aif to insight problem solving,“ *Quaerterly Journal of Experimental Psychology,* Bd. 64, Nr. 4, pp. 649-656, 1. April 2011.

[25] C. Aschwanden und M. Crosby, „Code Scanning Patterns in Program Comprehension,“ in *Proceedings of the 39th Hawaii International Conference on System Sciences*, Kauai, Hawaii, 2006.

[26] E. Hegarty-Kelly, S. Bergin und A. Mooney, „Using focused attention to improve programming comprehension for novice programmers,“ in *Eye Movements in Programming: Models to Data. Proceedings of the Third International Workshop*, Joensuu, Finland, 2016.

[27] „Mylyn Official Website,“ The Eclipse Foundation, [Online]. Available: http://www.eclipse.org/mylyn/. [Zugriff am 09. 05. 2018].

[28] M. Kersten und G. C. Murphy, „Mylar: a degree-of-interest model for IDEs,“ in *Proceedings of the 4th International Conference on Aspect-oriented Software Development*, Chicago, Illinois, 2005.

[29] M. Kersten und G. C. Murphy, „Using Task Context to Improve Programmer Productivity," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Portland, Oregon, USA, 2006.

[30] T. R. Shaffer, J. Wise, B. M. Walters, S. Müller, M. Falcone und B. Sharif, „iTrace: Enabling eye tracking on software artifacts within the IDE to support software engineering tasks," in *Proceedings of the 1015 10th Joint Meeting on Foundations of Software Engineering*, Bergamo, Italy, 2015.

[31] „iTrace Eclipse Plugin Download on GitHub," [Online]. Available: https://github.com/trshaffer/iTrace. [Zugriff am 09. 05. 2018].

[32] „Tobii Pro SDK," [Online]. Available: https://www.tobiipro.com/product-listing/tobii-pro-sdk/. [Zugriff am 22. 06. 2018].

[33] „Tobii Pro Studio User Manual," Tobii AB, [Online]. Available: https://www.tobiipro.com/siteassets/tobii-pro/user-manuals/tobii-pro-studio-user-manual.pdf. [Zugriff am 16. 08. 2018].

[34] T. Busjahn, C. Schulte und A. Busjahn, „Analysis of Code Reading to gain more Insight in Program Comprehension," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, Koli, Finland, ACM, 2011, pp. 1-9.

[35] B. De Smet, L. Lempereur, Z. Sharafi, Y.-G. Guéhéneuc, G. Antoniol und N. Habra, „Taupe: Visualizing and analyzing eye-tracking data," *Science of Computer Programming, Volume 79,* pp. 260-278, 1. Januar 2014.

[36] B. Clark und B. Sharif, „iTraceVis: Visualizing Eye Movement Data Within Eclipse," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, Shanghai, China, 2017.

[37] V. R. Basili, C. Caldiera und H. D. Rombach, „Goal Question Metric Paradigm," in *Encyclopedia of Software Engineering, Volume 1*, John Wiley & Sons, 1994, pp. 528-532.

[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell und A. Wesslén, Experimentation in Software Engineering, Springer, 2012.

[39] B. Bailey, „FirstClick Usability Testing,“ Web Usability, 08. 10. 2013. [Online]. Available: http://webusability.com/firstclick-usability-testing/. [Zugriff am 30. 10. 2018].

[40] „Highlight.js: Syntax highlighting for the Web,“ [Online]. Available: https://highlightjs.org/. [Zugriff am 02. 07. 2018].

[41] Y. Pakala, „highlightjs-line-numbers.js,“ GitHub, [Online]. Available: https://github.com/wcoder/highlightjs-line-numbers.js/. [Zugriff am 02. 07. 2018].

[42] „Tobii Pro X3-120 User Manual,“ Tobii AB, [Online]. Available: https://www.tobiipro.com/siteassets/tobii-pro/user-manuals/tobii-pro-x3-120-user-manual.pdf/?v=1.0.7. [Zugriff am 02. 07. 2018].

[43] „Java SE Development Kit 8 Documentation,“ Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html. [Zugriff am 02. 07. 2018].

[44] P.-H. Soria, „GitHub - Simple Java Text Editor,“ [Online]. Available: https://github.com/pH-7/Simple-Java-Text-Editor. [Zugriff am 05. 06. 2018].

[45] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell und B. Sharif, „The impact of identifier style on effort and comprehension,“ *Empirical Software Engineering,* Bd. 18, Nr. 2, pp. 219-276, 03. Mai 2012.

[46] M. Kersten und G. C. Murphy, „Using Task Context to Improve Programmer Productivity,“ in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Portland, Oregon, USA, November 2006.

[47] R. Müller, J. R. Helmert, S. Pannasch und B. M. Velichkovsky, „Gaze transfer in remote cooperation: Is it always helpful to see what your partner is attending to?,“ *The Quarterly Journal of Experimental Psychology,* Bd. 66, Nr. 7, pp. 1302-1316, 2013.

A Anhang

A.1 Erklärung und Fragebögen

A.1.1 Einverständniserklärung

Studie: Eye-Tracking in der Software-Wartung

Einverständniserklärung

Bitte lesen Sie die Einverständniserklärung sorgfältig durch, bevor Sie sich entscheiden an der Studie teilzunehmen.

Bezeichnung der Studie/des Experiments: Eye-Tracking in der Software-Wartung

Beschreibung: In dem Experiment wird mittels Eye-Tracking das Ausführen von Softwarewartungsarbeiten untersucht. Es werden ungefähr 20 Studenten und Experten teilnehmen. In dem Experiment werden Ihnen verschiedene Aufgaben gestellt, die von einem stationären Eye-Tracker überwacht werden. Die Dauer des Experiments beschränkt sich auf ca. 60 Minuten.

Risiken und Vorteile: Die Teilnahme am Laborexperiment ist mit keinen Risiken oder direkten Vorteilen verbunden. Ebenso entstehen dem Teilnehmer durch das Experiment keine Nachteile.

Kosten und Entgelt: Außer der eigenen Zeitinvestition entstehen keine weiteren Kosten. Weiterhin wird kein Entgelt für die Teilnahme am Experiment ausgezahlt.

Vertraulichkeit: Alle während des Experiments erhobenen Daten werden anonymisiert und nur für das Fachgebiet Software Engineering der Leibniz Universität Hannover zugänglich sein. Die Ergebnisse werden bei der Auswertung zufällig verteilten IDs zugeordnet und sind damit keiner echten Person zuweisbar. Der Teilnehmer verpflichtet sich zur Geheimhaltung der Studieninhalte.

Datenaufnahme und -verarbeitung: Zum Ende der Studie werden Antworten des Teilnehmers nach nochmaliger Ankündigung mit einem Mikrofon aufgezeichnet. Ich stimme der Aufnahme, Speicherung und Verarbeitung dieser Tonaufnahmen, sowie der Antworten auf die Fragebögen zu. Auch diese Daten werden anonym behandelt und es wird in der Auswertung keine Personenzuordnung möglich sein.

Abbruch des Experiments: Dem Teilnehmer ist es zu jeder Zeit möglich das Experiment zu beenden oder abzubrechen. Diese Entscheidung wird keinen Nachteil oder Vorteil für den Teilnehmer mit sich bringen.

Freiwilliges Einverständnis: Die oben aufgeführten Punkte wurden mir erklärt und meine Fragen dazu beantwortet. Vor, während und nach dem Experiment werden weitere Fragen vom Experimentator geklärt. Mit meiner Unterschrift bestätige ich, dass ich an dem beschriebenen Experiment teilnehmen möchte.

_____ Hannover, _____

(Name und Unterschrift des Teilnehmers)

A.1.2 Aufgabeneinleitung

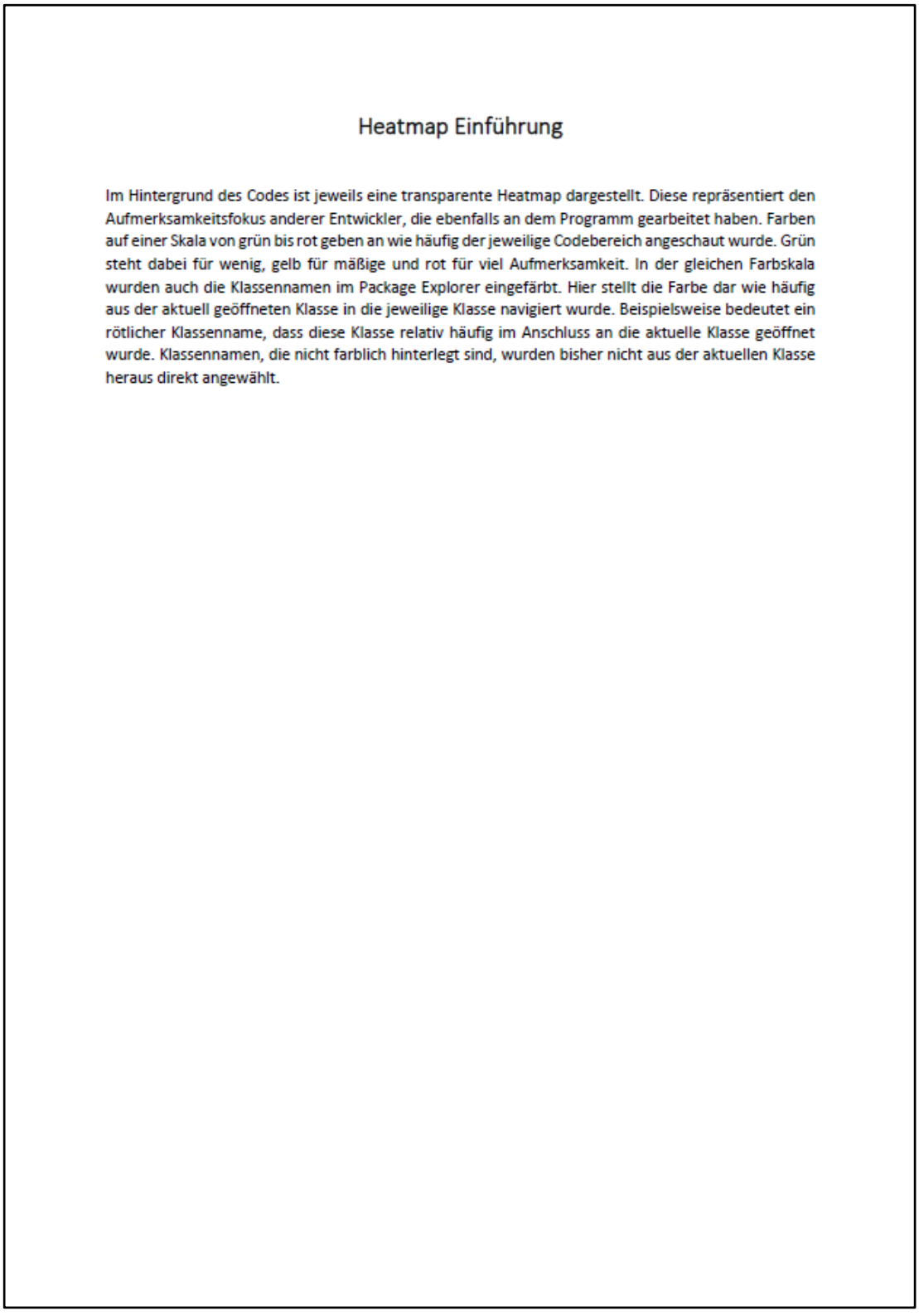
Aufgabeneinleitung

Ihnen wird die Weiterarbeit an einem Javaprogramm zugeteilt. Bei dem Programm handelt es sich um einen einfachen Texteditor mit Syntaxhighlighting für Java und C++. Einen Screenshot der Anwendungsoberfläche erhalten Sie gleich auf Papier. Ihre Aufgabe ist es nun kleine Änderungen an dem Programm vorzunehmen. Dazu erhalten Sie im Anschluss die Beschreibungen, die jeweils eine Änderung, Erweiterung oder Fehlerbehebung des Programms vorsehen. Das Programm ist aktuell noch nicht zur Änderung freigegeben, daher beschreiben Sie Ihre Lösungsvorschläge für die einzelnen Tasks bitte in Schriftform. Geben Sie an in welcher Datei an welcher Stelle bzw. welchen Stellen (Zeilennummern) Änderungen oder Ergänzungen nötig sind. Es muss sich bei den Beschreibungen nicht unbedingt um syntaktisch korrekten Java-Code handeln, auch Pseudocode ist ausreichend. Die Anweisungen sollten ausreichend sein, um Sie anschließend an einen Java-Entwickler weiterzugeben, der diese dann einfach umsetzen kann ohne sich selbst mit dem Problem auseinandersetzen zu müssen.

Die Aufgaben werden Ihnen nacheinander ausgeteilt. Sobald Sie mit einer Aufgabe fertig sind oder die Bearbeitung abbrechen möchten, geben Sie dies bitte Bescheid. Dann erhalten Sie die nächste Aufgabenbeschreibung. Es ist nicht möglich zu einer abgebrochenen Aufgabe noch einmal zurück zu kehren. Es handelt sich um insgesamt sechs Aufgaben unterschiedlichen Umfangs. Für die Bearbeitung haben Sie 50 Minuten Zeit. Versuchen Sie innerhalb dieser Zeit so viele Aufgaben wie möglich so korrekt wie möglich zu bearbeiten.

Das Programm wird Ihnen in einer Darstellung der Eclipse-Entwicklungsumgebung im Browser angezeigt. Darin ist es Ihnen möglich zwischen den einzelnen Dateien sowohl im Package Explorer als auch in den Tabs zu wechseln und mit STRG+F zu suchen. Andere Funktionalitäten wie das Ausführen oder Debuggen des Programms, Referenzen nachverfolgen etc. werden nicht unterstützt. Als Hilfestellung steht Ihnen in einem zweiten Tab die Java API zur Verfügung.

A.1.3 Heatmap Einführung



Heatmap Einführung

Im Hintergrund des Codes ist jeweils eine transparente Heatmap dargestellt. Diese repräsentiert den Aufmerksamkeitsfokus anderer Entwickler, die ebenfalls an dem Programm gearbeitet haben. Farben auf einer Skala von grün bis rot geben an wie häufig der jeweilige Codebereich angeschaut wurde. Grün steht dabei für wenig, gelb für mäßige und rot für viel Aufmerksamkeit. In der gleichen Farbskala wurden auch die Klassennamen im Package Explorer eingefärbt. Hier stellt die Farbe dar wie häufig aus der aktuell geöffneten Klasse in die jeweilige Klasse navigiert wurde. Beispielsweise bedeutet ein rötlicher Klassenname, dass diese Klasse relativ häufig im Anschluss an die aktuelle Klasse geöffnet wurde. Klassennamen, die nicht farblich hinterlegt sind, wurden bisher nicht aus der aktuellen Klasse heraus direkt angewählt.

A.1.4 Fragebogen zum Hintergrund der Teilnehmer

Teilnehmer ID: _____ 1

Herzlich Willkommen zu der Studie
„Eye-Tracking in der Software-Wartung“!

Vor Beginn der eigentlichen Studie werden Sie noch darum gebeten, einige Fragen zu Ihrem Hintergrund zu beantworten. Dies dient lediglich dazu die Teilnehmer der Studie charakterisieren zu können. Es geht nicht darum Ihre Fähigkeiten zu bewerten. Alle Antworten werden anonym behandelt. Es wird nicht möglich sein, Ihre Angaben auf Ihre Person zurückzuführen. Zu jeder Frage werden Ihnen einige Antwortmöglichkeiten vorgegeben. Bitte kreuzen Sie die für Sie zutreffende Antwort an.

Vielen Dank für Ihre Teilnahme!

Frage	**Antwortmöglichkeiten**		
1.1 Was ist Ihr Studiengang?	O Informatik O Technische Informatik O Informatik als Nebenfach O Ich bin kein Student (mehr).		
1.2 Was ist Ihr aktuelles Studienziel?	O Bachelor O Master O Ich bin kein Student (mehr).		
1.3 In welchem Fachsemester sind Sie?	O 1 O 2 O 3	O 4 O 5 O 6	O 7 O ≥ 8 O kein Student
1.4 Wie viel Erfahrung in der Softwareentwicklung haben Sie?	O Keine O Wenig O Viel O Sehr viel		
1.5 Wie viel Erfahrung in der Java-Entwicklung haben Sie?	O Keine O Wenig O Viel O Sehr viel		
1.6 Wie lange haben Sie bereits in der Softwareentwicklung gearbeitet?	O Gar nicht O Maximal 1 Jahr O 1-2 Jahre O 3-4 Jahre O Mehr als 4 Jahre		

Teilnehmer ID: _____ 2

Frage	Antwortmöglichkeiten
1.7 Haben Sie Erfahrung mit Pair Programming?	O Nein O Ja, habe ich schon mal gemacht O Ja, gelegentlich O Ja, mache ich ständig
1.8 Haben Sie Erfahrung damit Code zu verstehen, den Sie nicht selbst geschrieben haben?	O Nein O Ja, wenig O Ja, viel O Ja, sehr viel
1.9 Haben Sie eine Rot-Grün-Sehschwäche?	O Nein O Ja, aber nur schwach O Ja

A.1.5 Fragebogen am Ende (Treatmentgruppe)

Teilnehmer ID: _____

Eye-Tracking in der Software-Wartung

Fast geschafft! Zum Ende des Experiments werden Sie noch kurz gebeten ein paar Fragen zu Ihren Erfahrungen mit den soeben bearbeiteten Aufgaben zu beantworten.

Frage	Antwortmöglichkeiten
2.1 Wie schnell konnten Sie die jeweils für die Aufgabe relevanten Codeabschnitte im Programm finden?	langsam schnell O O O O O O O O O O
2.2 Wie einfach konnten Sie sich ein Bild der generellen Funktionalität des Programms machen? Wie schnell konnten Sie verstehen was die Aufgabe einzelner Komponenten und des gesamten Programms ist?	schwer leicht O O O O O O O O O O
2.3 Wie übersichtlich fanden Sie die Codedarstellung (unabhängig vom konkreten Inhalt des Codes)?	unübersichtlich übersichtlich O O O O O O O O O O
2.4 Wie hilfreich war die Darstellung der Heatmap im Hintergrund des Codes für Sie?	nicht hilfreich hilfreich O O O O O O O O O O
2.5 Wie ablenkend oder störend war die Darstellung der Heatmap im Hintergrund des Codes für Sie?	nicht störend störend O O O O O O O O O O
2.6 Wie hilfreich war die Heatmap Einfärbung der Klassen im Package Explorer für Sie?	nicht hilfreich hilfreich O O O O O O O O O O
2.7 Wie ablenkend oder störend war die Heatmap Einfärbung der Klassen im Package Explorer für Sie?	nicht störend störend O O O O O O O O O O
2.8 Wenn es die Möglichkeit zum Ausschalten (Togglen) der Heatmap-Anzeige gegeben hätte, hätten Sie diese genutzt?	O Nein O Ja
2.9 Haben Sie Anmerkungen oder Kommentare?	

A.2 Aufgabenbeschreibungen und -lösungen

A.2.1 Aufgaben Set A

Aufgaben während des Experiments: Gruppe A

Task A1 – Änderung (Highlightcolor): Nutzer haben sich beklagt, dass die hervorgehobenen Keywords schlecht lesbar sind. Passen Sie die Highlightfarbe an, indem Sie eine hellere Farbe wählen.

Erforderliche Änderung: 1 Zeile, Zeile 55 in UI.java. Gesetzte Farbe bei der Instanziierung von HighlightText anpassen, beispielweise zu LIGHT_GRAY statt GRAY.

Task A2 – Bugfix (Defaultwert von Schriftart- und Schriftgrößencombobox): Wenn der Texteditor gestartet wird, sind die Comboboxen für Schriftart und –größe beide auf das erste Element der Liste gesetzt, obwohl der voreingestellte Font „Century Gothic" und Schriftgröße 12 ist. Passen Sie die voreingestellten Werte der Comboboxen entsprechend an.

Erforderliche Änderung: Jeweils eine Zeile für die Schriftart und –größe. Voreingestellten Wert der Combobox für die Schriftart anpassen in Zeile 262 in UI.java: `fontType.setSelectedItem("Century Gothic");` Voreingestellten Wert der Combobox für die Schriftgröße anpassen in Zeile 283 in UI.java: `fontSize.setSelectedItem(12);`

Task A3 – Erweiterung (Scrolling): In der aktuellen Version des Texteditors wird zu langer Text an der Unterkante des Textfeldes abgeschnitten. Ermöglichen Sie die Darstellung von längeren Textdateien, indem Sie bei Bedarf vertikales Scrolling hinzufügen.

Erforderliche Änderung: Hinzufügen von zwei Zeilen und Ändern einer zusätzlichen Zeile in Zeile 91 in UI.java.

```
JScrollPane scroll = new JScrollPane(textArea);
scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS
_NEEDED);
getContentPane().add(textArea);
```

Task A4 – Bugfix (Highlight beim Suchen und Ersetzen): Bei der Suche werden gefundene Strings nicht sichtbar markiert, wenn diese innerhalb eines durch das Highlighting hervorgehobenen Keywords liegen. Schalten Sie das Highlighting während das Suchfenster geöffnet ist aus, um dieses Problem zu umgehen und highlighten Sie den Text anschließend erneut, wenn es wieder geschlossen wird.

Erforderliche Änderung: Eine Möglichkeit ist es der Find.java Klasse bei der Instanziierung in UI.java das HighlightText-Objekt zu übergeben und bei Aufruf der Find und Replace Buttons jeweils die Methode removeHighlights aufzurufen. Anschließend, wenn das Suchfenster geschlossen wird (cancel, Z.152), wird der Text erneut gehighlightet, indem die Methode highlight aufgerufen wird. Zusätzlich muss auch die close Operation angepasst werden, wenn statt dem Cancel-Button das X genutzt wird:

```
@Override
      protected void processWindowEvent(WindowEvent e) {
            if (e.getID() == WindowEvent.WINDOW_CLOSING) {
                  SupportedKeywords kw = new SupportedKeywords();
                  this.languageHighlighter.highlight(txt,
                        kw.getCppKeywords());
                  this.languageHighlighter.highlight(txt,
                        kw.getJavaKeywords());
            }
      }
```

Für die Lösung der Aufgabe ist eine der beiden Varianten ausreichend.

Task A5 – Änderung (E-Mailadresse des Autors): Der Autor des Programms möchte seine Kontakt-E-Mailadresse zu pierrehs@hotmail.com ändern. Passen Sie die Anzeige, wenn man auf den Infobutton „About Notepad PH“ klickt entsprechend an.

Erforderliche Änderung: In Zeile 15 in SimpleJavaTextEditor.java:

```
public final static String AUTHOR_EMAIL = "pierrehs@hotmail.com";
```

Task A6 – Erweiterung (Fett und Kursiv): In der aktuellen Version des Texteditors ist es nur möglich den Text entweder auf kursiv oder auf fett zu setzen. Ermöglichen Sie das Setzen von kursiv (italic) und fett (bold) in Kombination.

Erforderliche Änderung: Die Fallabfrage, ob der Font fett bzw. kursiv ist bei Aufruf des Bold- bzw. Italic-Buttons durch eine switch-Anweisung ersetzen, die die verschiedenen möglichen Kombinationen von fett und kursiv absetzt und den Font entsprechend anpasst.

Für den bold-Button in Zeile 381 in UI.java:

```
switch (textArea.getFont().getStyle()) {
      case Font.PLAIN:
            textArea.setFont(textArea.getFont().deriveFont(Font.BOLD));
            break;
      case Font.BOLD:
            textArea.setFont(textArea.getFont().deriveFont(Font.PLAIN));
            break;
      case Font.ITALIC:
            textArea.setFont(textArea.getFont().deriveFont(Font.ITALIC +
                  Font.BOLD));
            break;
      case Font.BOLD + Font.ITALIC:
            textArea.setFont(textArea.getFont().deriveFont(Font.ITALIC));
}
```

Für den Italic-Button in Zeile 388 in UI.java:

```
switch (textArea.getFont().getStyle()) {
      case Font.PLAIN:
            textArea.setFont(textArea.getFont().deriveFont(Font.ITALIC));
            break;
      case Font.BOLD:
            textArea.setFont(textArea.getFont().deriveFont(Font.ITALIC +
                  Font.BOLD));
            break;
      case Font.ITALIC:
            textArea.setFont(textArea.getFont().deriveFont(Font.PLAIN));
            break;
      case Font.BOLD + Font.ITALIC:
            textArea.setFont(textArea.getFont().deriveFont(Font.BOLD));
}
```

A.2.2 Aufgaben Set B

Aufgaben während des Experiments: Gruppe B

Task B1 – Änderung (Keywordliste): Mit dem Update von Java auf Java 5.0 wurde „enum" zu den Keywords hinzugefügt. Passen Sie die Liste unterstützter Keywords des Texteditors ebenfalls an. Darüber hinaus handelt es sich bei „String", „print()" und „println" um keine offiziellen Keywords. Entfernen Sie diese daher aus der Liste.

Erforderliche Änderung: SupportedKeywords.java, in Zeile 19 einen String zu dem Array hinzufügen und dafür zwei in Zeile 22 und einen in Zeile 24 entfernen.

Task B2 – Bugfix (Single Word Highlighting): Keywords werden auch hervorgehoben, wenn sie Teil eines Wortes, beispielweise eines Variablennamens sind. Korrigieren Sie dies und sorgen Sie dafür, dass Keywords nur hervorgehoben werden, wenn sie als einzelnes Wort vorkommen.

Erforderliche Änderung: Hinzufügen einer längeren If-Abfrage in HighlightText.java, in Zeile 23 von etwa 7 Zeilen, testet ob das Zeichen vor und nach dem Keyword kein Buchstabe ist

```
// highlight only if token is not contained in a longer word
if (!(pos > 0 && (Character.getType(text.charAt(pos - 1)) ==
Character.UPPERCASE_LETTER || Character.getType(text.charAt(pos - 1)) ==
Character.LOWERCASE_LETTER)|| (pos + pattern[i].length() < text.length() &&
(Character.getType(text.charAt(pos + pattern[i].length())) ==
Character.UPPERCASE_LETTER || Character.getType(text.charAt(pos +
pattern[i].length())) == Character.LOWERCASE_LETTER))))
```

Aufgrund der undeutlichen Aufgabenstellung „einzelnes Wort" ist es auch ausreichend auf Leerzeichen zu prüfen. Es muss jedoch die Ausnahme am Textanfang und –ende berücksichtigt werden.

Task B3 – Erweiterung (Bracket AutoComplete): Die AutoComplete Funktion wird aktuell auf Keywords, runde und geschweifte Klammern angewendet. Ermöglichen Sie auch die Autovervollständigung von eckigen Klammern [ ], sowie einfachen Anführungszeichen ' '. Beachten Sie dabei, dass in dem Fall kein doppelter Zeilenumbruch eingefügt werden darf.

Erforderliche Änderung: In SupportedKeywords.java in Zeile 38 und 39 jeweils die Zeichen hinzufügen.
Zusätzlich in Zeile 52 in AutoComplete.java `private boolean isSimpleBracket;` einfügen.
Außerdem in Zeile 146:

```
if (s.equals("[") || s.equals("'")) {
      isSimpleBracket = true;
} else
      isSimpleBracket = false;
}
```

In Zeile 242:

```
if (isSimpleBracket) {
      textArea.setCaretPosition(pos);
      mode = Mode.INSERT;
      textArea.removeKeyListener(this);
      break;
}
```

Task B4 – Bugfix (Highlighting Pattern): Das Keyword Highlighting funktioniert aktuell nur für Java Keywords, nicht für C++ Keywords. Finden Sie den Fehler und beheben Sie ihn, sodass sowohl Java als auch C++ Keywords im Text hervorgehoben werden.

Erforderliche Änderung: In HighlightText.java werden bei Aufruf der highlight-Methode alle zuvor gesetzten Highlights gelöscht, sodass der erste Aufruf der highlight-Methode wirkungslos bleibt. Um

dies zu beheben, muss die Zeile 13 in HighlightText.java entfernt werden und stattdessen in Zeile 88 in UI.java der Aufruf der removeHighlights-Methode hinzugefügt werden:

```
languageHighlighter.removeHighlights(textArea);
```

Alternativ könnte man die beiden Keyword String-Arrays vorher zusammenführen. Änderung dafür in UI.java, Zeile 88:

```
languageHighlighter.highlight(textArea,Stream.concat(Arrays.stream(kw.getCp
pKeywords()), Arrays.stream(kw.getJavaKeywords())).toArray(String[]::new));
```

Task B5 – Erweiterung (AutoComplete): Die AutoComplete Funktion wird aktuell nur bei über den Dateiexplorer geöffneten Dokumenten angewendet, jedoch nicht bei neu erstelltem Text. Ermöglichen Sie es die Autovervollständigung auch in neu erstellten Dokumenten zu benutzen. Da in einem neu erstellten Dokument noch nicht bekannt ist, ob es sich um ein Java oder C++ Dokument handelt, sollen Keywords aus beiden Sprachen autovervollständigt werden.

Erforderliche Änderung: Einfachste Möglichkeit ist hier das Erstellen einer neuen enableAutoComplete() Funktion in UI.java, die keine Datei als Eingabe bekommt, sondern sowohl die Java Keywords, als auch die C++ Keywords als Liste für die AutoComplete Komponente setzt.

```
// Enable autocomplete option
public void enableAutoComplete() {
	if (hasListener){
		textArea.getDocument().removeDocumentListener(autocomplete);
		hasListener = false;
	}
	ArrayList<String> arrayList;
	String[] ck = kw.getCppKeywords();
	String[] jk = kw.getJavaKeywords();
	arrayList = kw.setKeywords(ck);
	arrayList.addAll(kw.setKeywords(jk));
	autocomplete = new AutoComplete(this, arrayList);
	textArea.getDocument().addDocumentListener(autocomplete);
	hasListener = true;
}
```

Task B6 – Änderung (Default Schriftart und -größe): Nutzer haben sich gewünscht, dass der voreingestellte Font mit dem von Microsoft Word übereinstimmt. Setzen Sie daher die default Schriftart auf „Calibri“ und die voreingestellte Schriftgröße auf 11.

Erforderliche Änderung: In UI.java in Zeile 79 die Schriftart auf „Calibri“ und die Schriftgröße auf 11 setzen.

A.3 Ergebnisdaten

A.3.1 Fixationsanzahl

Teilnehmer ohne Heatmap	Fixationsanzahl pro Sekunde Codebetrachtung	Teilnehmer mit Heatmap	Fixationsanzahl pro Sekunde Codebetrachtung
001	3,71563296	011	3,31567767
002	3,80187659	012	3,60058827
003	4,07888472	013	3,29103532
004	3,13021266	014	3,7820724
005	2,8325423	015	3,29149798
006	3,52566828	016	3,19955568
007	3,25868499	017	3,56386445
008	3,34426202	018	3,69453519
009	3,4162449	019	3,39886482
010	3,79753322	020	4,31048987
021	3,39192044	022	3,63648622
Mittelwert gesamt	3,48122392		3,55315162
Standardabweichung gesamt	0,35234462		0,3158577

A.3.2 Fixationsdauer

Teilnehmer ohne Heatmap	Durchschn. Fixationsdauer [ms]	Teilnehmer mit Heatmap	Durchschn. Fixationsdauer [ms]
001	217	011	186
002	181	012	190
003	239	013	208
004	243	014	190
005	134	015	213
006	241	016	216
007	180	017	179
008	249	018	210
009	224	019	230
010	184	020	202
021	215	022	192
Mittelwert gesamt	209,7272727		201,4545455
Standardabweichung gesamt	35,88061011		15,39716621

A.3.3 Sakkadenanzahl

Teilnehmer ohne Heatmap	Sakkadenanzahl pro Sekunde Codebetrachtung	Teilnehmer mit Heatmap	Sakkadenanzahl pro Sekunde Codebetrachtung
001	6,654928884	011	6,63135534
002	7,850395576	012	8,599119516
003	5,511848515	013	7,904552837
004	5,711564611	014	8,415697648
005	15,20499482	015	7,377192982
006	4,871111111	016	8,232879359
007	8,48111157	017	8,741157675
008	4,769964741	018	7,00595766
009	7,061651233	019	5,617829554
010	8,165955079	020	8,835824759
021	6,439606336	022	8,086324409
Mittelwert gesamt	7,338466589		7,767990158
Standardabweichung gesamt	2,900946418		1,009072846

A.3.4 Sakkadendauer

Teilnehmer ohne Heatmap	Durchschn. Sakkadendauer [ms]	Teilnehmer mit Heatmap	Durchschn. Sakkadendauer [ms]
001	30	011	31
002	29	012	30
003	29	013	32
004	29	014	31
005	36	015	30
006	29	016	32
007	32	017	31
008	33	018	28
009	32	019	30
010	29	020	28
021	29	022	32
Mittelwert gesamt	30,63636364		30,45454545
Standardabweichung gesamt	2,335496832		1,439696938

A.3.5 Pupillendurchmesser

Teilnehmer ohne Heatmap	Pupillendurchmesser	Teilnehmer mit Heatmap	Pupillendurchmesser
001	2,645	011	2,845
002	2,825	012	2,83
003	3,03	013	2,475
004	3,01	014	3,815
005	3,185	015	2,7
006	3,215	016	2,73
007	2,595	017	3,185
008	3,3	018	2,745
009	2,745	019	2,925
010	2,86	020	2,57
021	2,775	022	2,285
Mittelwert gesamt	2,925909091		2,827727273
Standardabweichung gesamt	0,238115709		0,403759604

A.3.6 Bearbeitungszeit der Softwarewartungsaufgaben

Aufgaben	Aufgabentitel	Zeit ohne Heatmap [s] Mittelwert	Zeit mit Heatmap [s] Mittelwert
A1	Highlightcolor	121,8	119,86
A2	Defaultwert von Schriftart- und Schriftgrößencombobox	451,967	409,383
A3	Scrolling	268,3	330,3
A4	Highlight beim Suchen und Ersetzen	986,483	824,55
A5	E-Mailadresse des Autors	184,183	177,67
A6	Fett und Kursiv	515,88	407,5
B1	Keywordliste	136	163
B2	Single Word Highlighting	457,275	444,62
B3	Brackets Autocomplete	777,75	1006,4
B4	Highlight Pattern	617,75	542
B5	AutoComplete	749,4	622,267
B6	Default Schriftart und -größe	151,6	184,5
Mittelwert gesamt		451,5323611	436,0038889
Standardabweichung gesamt		288,6348483	276,2554624

A.3.7 Korrektheit der Softwarewartungsaufgaben

Aufgaben	Aufgabentitel	Korrektheit ohne Heatmap Mittelwert	Korrektheit mit Heatmap Mittelwert
A1	Highlightcolor	4	4
A2	Defaultwert von Schriftart- und Schriftgrößencombobox	3,5	3,83
A3	Scrolling	2,8	3
A4	Highlight beim Suchen und Ersetzen	3,17	2,5
A5	E-Mailadresse des Autors	3,67	4
A6	Fett und Kursiv	2,8	3
B1	Keywordliste	4	4
B2	Single Word Highlighting	2,5	3
B3	Brackets Autocomplete	2,75	2,6
B4	Highlight Pattern	2,25	2,8
B5	AutoComplete	3,5	3,33
B6	Default Schriftart und -größe	4	3,75
Mittelwert gesamt		3,244444444	3,318055556
Standardabweichung gesamt		0,618663162	0,572406683

A.3.8 Codeelement Suchzeiten

Teilnehmer ohne Heatmap	Codeelement Suchzeit [ms]	Teilnehmer mit Heatmap	Codeelement Suchzeit [ms]
001	40980	011	23048
002	24735	012	26673
003	9380	013	20844
004	23300	014	24890
005	17761	015	9139
006	8232	016	26326
007	45534	017	21078
008	17588	018	35067
009	20770	019	26996
010	29798	020	25983
021	14699	022	29345
Mittelwert gesamt	22979,72727		24489,90909
Standardabweichung gesamt	11893,6313		6441,405506

A.3.9 Antworten auf Fragebögen

Die folgende Tabelle zeigt die Antworten der Teilnehmer auf die Fragebögen. Anhand der Fragen IDs lässt sich in den Fragebögen nachsehen, um welche Frage es sich handelt. Die Antworten sind entsprechend ihrer Reihenfolge aufsteigend mit Zahlen ab 1 codiert. Die Antworten auf die zehnstufigen Likert-Skalen wurden entsprechend mit Zahlenwerten zwischen 1 und 10 angegeben. Die obere Hälfte der Tabelle mit den Teilnehmer IDs von 001 bis 010, sowie 021 beinhaltet die Teilnehmer der Kontrollgruppe und die untere Hälfte ab Teilnehmer ID 011 zeigt die Antworten der Treatmentgruppe.

Teilnehmer ID	Fragen ID																
	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
001	1	2	5	2	2	1	2	3	-	3	6	7	-	-	-	-	-
002	4	3	9	4	4	5	3	4	-	9	8	8	-	-	-	-	-
003	1	3	9	3	3	3	1	2	-	7	9	9	-	-	-	-	-
004	1	2	6	2	3	1	2	2	-	7	6	8	-	-	-	-	-
005	1	3	4	4	4	2	2	4	-	5	10	10	-	-	-	-	-
006	1	1	6	3	3	4	1	3	-	6	8	7	-	-	-	-	-
007	4	3	9	2	2	1	2	2	-	9	8	9	-	-	-	-	-
008	1	2	2	2	2	1	2	3	-	4	6	9	-	-	-	-	-
009	4	3	9	4	3	5	3	4	-	8	8	7	-	-	-	-	-
010	1	3	9	3	3	1	2	2	-	3	4	3	-	-	-	-	-
021	4	3	9	3	2	5	4	4	-	6	7	10	-	-	-	-	-
011	1	2	4	3	3	3	2	3	1	9	6	10	2	1	1	1	1
012	4	3	9	4	4	5	2	3	1	7	7	3	2	1	2	1	1
013	4	3	9	3	4	4	2	3	1	6	8	8	7	8	6	3	2
014	2	2	6	3	3	2	2	2	1	3	4	6	2	8	3	1	2
015	1	3	8	2	2	1	1	3	1	2	4	8	2	3	1	3	2
016	1	2	5	2	2	2	2	2	1	7	6	8	9	3	8	1	1
017	1	3	9	3	3	4	2	3	1	5	10	4	2	9	9	5	2
018	2	1	6	2	2	1	1	2	1	3	2	4	3	1	3	4	1
019	4	3	9	4	4	5	2	3	1	1	5	6	1	2	3	2	2
020	1	3	4	3	3	1	2	3	1	5	8	7	4	9	7	1	1
022	4	3	9	4	4	5	3	4	1	2	3	2	3	1	3	1	2
Mittelwert	2	3	7	3	3	3	2	3	1	5	7	7	3	5	4	2	2

A.4 Dokumentation der Anwendung zur Heatmapfarbenbestimmung

Zur Bestimmung der Heatmapfarben für die Einfärbung der Klassennamen im Package Explorer wurde eine Hilfsanwendung in Java geschrieben, die als Eingabe den Anteil an Klassensprüngen, sowie den maximalen Anteil aus der aktuell betrachteten Klasse, bekommt. Als Ergebnis werden die RGB-Werte der resultierenden Farbe ausgegeben. Zusätzlich wird die Ergebnisfarbe auf der verwendeten Heatmap-Farbskala markiert.

Zur Berechnung des Farbwertes auf der verwendeten Heatmap-Farbskala wurden die folgenden vier Funktionen genutzt:

1. **Color calculateColor(value, maxValue)**
   Diese Methode bekommt als Input den eingegebenen Anteil an Klassensprüngen als value, sowie den maximalen Anteil an Klassensprüngen als maxValue. In der Methode wird zunächst geprüft, ob maxValue kleiner als value ist. In dem Fall wird die Farbe Weiß zurückgegeben. Andernfalls wird der Faktor x als Anteil von value an maxValue berechnet:
   x = value / maxValue
   Mit diesem Wert wird die folgende Funktion gebBlendedColor aufgerufen, dessen Rückgabewert als Ergebnisfarbe zurückgegeben wird.
2. **Color getBlendedColor(fraction)**
   Diese Methode bekommt als Eingabe den Anteil am Maximum als Wert zwischen 0 und 1 übergeben. Anschließend teilt sie die Farbbestimmung in zwei Bereiche auf. Für Werte größer 0 und kleiner 0,5, welches dem Farbbereich von Grün nach Gelb entspricht, wird die Methode interpolate (siehe 3.) mit den Farben Grün, Gelb und dem verdoppelten fraction Wert. Der Wert wird an dieser Stelle verdoppelt, um dennoch Werte zwischen 0 und 1 für den fraction-Bereich zwischen 0 und 0,5 zu erhalten.
   Für Werte ab 0,5 wird die interpolate Funktion (siehe 3.) mit den Farben Gelb und Rot aufgerufen, sowie dem zunächst mit 0,5 subtrahierten und anschließend verdoppelten fraction-Wert. Dieser Aufruf behandelt den Heatmap-Farbskalenbereich von Gelb nach Rot. Die Umrechnung des fraction-Werts dient an dieser Stelle wieder dazu, Werte zwischen 0 und 1 für den fraction-Bereich von 0,5 und 1 zu erhalten.
   Die Heatmap-Farbskala wurde versucht so nah wie möglich an den Farben der Heatmap zu halten, die im Tobii Pro Studio verwendet wurde. Auf diese Weise wird der Codehintergrund und die Klassennamen mit dem gleichen Farbschema eingefärbt. Daher wurden als Grünton nicht

reines, kräftiges Grün (RGB-Werte = (0,255,0)), sondern leicht abgedämpftes Grün (RGB-Werte = (0,128,0)) verwendet. Bei den Rot- und Gelbtönen handelt es sich hingegen um die reinen, kräftigen Farben mit Rot- bzw. Grünwerten von 255.

3. **Color interpolate(color1, color2, fraction)**
   Diese Methode bekommt als Eingabe zwei Farben, sowie eine Gleitkommazahl als Anteil. Als Rückgabe liefert sie eine aus drei RGB-Werten zusammengesetzte Farbe. Der Rot-Wert wird dabei aus der Interpolation der Rotanteile der beiden Inputfarben unter Berücksichtigung der fraction berechnet. Der Grün-Wert entsprechend aus den Grünanteilen und der Blau-Wert entsprechend mit dem Blauanteil. Da die verwendete Heatmapskala jedoch von Grün, über Gelb bis Rot geht, ist der Blauanteil in diesem Fall immer 0. Da es sich bei den Interpolationswerten um Gleitkommazahlen handelt, werden diese zur Bestimmung der RGB-Farbe auf ganze Zahlen gerundet.
4. **Double interpolate(d1, d2, fraction)**
   Diese Methode bekommt als Eingabe drei Gleitkommazahlen. Die ersten beiden geben dabei Farbanteile der linken und rechten Farbe der Farbskala an und die dritte gibt den relativen Anteil der zu bestimmenden Farbe dazwischen an. Das Ergebnis wird aus der Summe des ersten Farbwerts und dem Produkt aus der Differenz der beiden Farbwerte mit dem Anteil (fraction) gebildet:

   result = d1 + (d2 – d1) * fraction

   Diese Formel berechnet den dem Anteil entsprechenden Farbwert zwischen d1 und d2, also den eingegeben Farbwerten für die linke und rechte Grenze.

A.5 Inhalt der CD

Die beiliegende CD beinhaltet die folgenden Materialien:

- Masterarbeit in digitaler Form (PDF-Datei)
- Experimentunterlagen bestehend aus:
    - Verwendetes Texteditorprogramm als Eclipse Projekt
    - Aufgabenlösungen der Teilnehmer in digitaler Form (PDF-Datei)
    - Studienergebnisse als Excel-Datei
    - Eye-Tracking Studienergebnisse als separate Excel-Datei
- Codeelement Suchzeit Anwendung als Eclipse Projekt und als ausführbare Jar-Datei
- Anwendung zur Bestimmung der Heatmapfarben als Eclipse Projekt und als ausführbare Jar-Datei
- Screenshots einiger Webseiten der erstellten Eclipse Webversion mit Heatmaps
- Kurzes Gaze Plot Video, das die Problematik von dynamischem Content bei Verwendung von Eye-Tracking während der Arbeit mit Code zeigt
- Eclipse Webseiten (Offline Java API Dokumentation wurde nur im ersten Fall mit beigefügt)
    - Ohne Aufmerksamkeitsdarstellungen
    - Mit Aufmerksamkeitsdarstellungen der Aufgabengruppe A
    - Mit Aufmerksamkeitsdarstellungen der Aufgabengruppe B

Danksagung

Hiermit möchte ich mich bei allen Personen bedanken, die mich bei der Verfassung dieser Arbeit unterstützten. Dazu gehören vor allem meine beiden Betreuer, die mir stets bei Fragen behilflich waren. Ebenfalls einen großen Dank aussprechen möchte ich an alle Teilnehmer der Studie, insbesondere die Mitarbeiter von tecRacer.