Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Identity Management

Benutzerfreundliches Werkzeug zur Codeanalyse zur Erkennung von Sicherheitslücken

Sandra Kostic
Matrikelnummer: 4521679
sandra.kostic@fu-berlin.de

Erstgutachter: Prof. Dr. Marian Margraf
Zweitgutachter: Prof. Dr. Jörn Eichler

Berlin, 11. Mai 2018

Sandra Kostic

Zusammenfassung

Es gibt dutzende von Analysewerkzeugen zur Erkennung von Sicherheitslücken in bereits verfasstem Code, sei es in Form von Plug-ins oder selbstständigen Werkzeugen. Trotz dessen existiert weiterhin das Problem, dass noch immer unnötige Sicherheitslücken in veröffentlichten Codes vorhanden sind, welche unter anderem Endnutzer erreichen, in Form einer App oder Desktop Anwendung und so Angreifern Zugang zu den privaten Daten des Endnutzers ermöglichen.

Das Konzept dieser Arbeit ist es, das Problem am Ursprung anzugehen, nämlich bei dem Programmierer, welcher den Code verfasst. Hierzu wurde ein Werkzeug zur Codeanalyse entwickelt, das die Programmierer für Sicherheitslücken sensibilisiert, vielfältige Hilfestellungen anbietet und dabei benutzerfreundlich ist.

Basierend auf einer im Rahmen dieser Abschlussarbeit selbst erstellten Umfrage, gerichtet an Programmierer, zur Erfassung derer Erfahrungen und Anforderungen an Werkzeugen zur Codeanalyse, wurde ein Prototyp entwickelt. Dieser Prototyp wurde anschließend von Studienteilnehmern getestet und von mir ausgewertet.

Das Ergebnis ist der Prototyp eines Werkzeugs, welches anhand der Ergebnisse der Auswertungen für potenzielle Sicherheitslücken sensibilisiert und dabei hilft, sicherer zu programmieren.

Sandra Kostic

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 11.05.2018

Sandra Kostic

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel . . . 2
- 1.3 Aufbau der Arbeit . . . 2

2 Einführung **2**
- 2.1 Werkzeuge zur Codeanalyse . . . 2
- 2.2 Grund der Studie . . . 4
  - 2.2.1 Vergangene Ereignisse . . . 4
  - 2.2.2 Nutzerverhalten . . . 6
  - 2.2.3 Das Ziel des Werkzeugs . . . 11
- 2.3 Ablauf der Studie . . . 11
- 2.4 Bestandteile der Studie . . . 12

3 Grundlagen **12**
- 3.1 Beliebte Werkzeuge . . . 12
  - 3.1.1 OWASP LAPSE+ . . . 12
  - 3.1.2 Coverity . . . 14
  - 3.1.3 RIPS . . . 14
- 3.2 Beliebte Editoren . . . 15
  - 3.2.1 Sublime . . . 15
  - 3.2.2 Visual Studio Code . . . 16
  - 3.2.3 Atom . . . 16
- 3.3 Was ist Security . . . 17
- 3.4 Was ist der Unterschied zwischen Security und Safety . . . 18
- 3.5 Was ist Usable Security . . . 19
- 3.6 User Centered Design . . . 21
- 3.7 Was ist Benutzbarkeit . . . 22
- 3.8 Auswertung der Benutzerfreundlichkeit . . . 24
- 3.9 Usability Test . . . 25
- 3.10 Was sind die Trust Guidelines . . . 27
- 3.11 Welche Sicherheitskriterien gibt es . . . 28
- 3.12 Wie wird ein Fragebogen aufgebaut . . . 29
- 3.13 Grundsätze der Dialoggestaltung . . . 31
  - 3.13.1 Aufgabenangemessenheit: . . . 32
  - 3.13.2 Selbstbeschreibungsfähigkeit: . . . 32
  - 3.13.3 Die Erwartungskonformität: . . . 32
  - 3.13.4 Die Lernförderlichkeit: . . . 32
  - 3.13.5 Steuerbarkeit: . . . 32
  - 3.13.6 Fehlertoleranz: . . . 32
  - 3.13.7 Individualisierbarkeit: . . . 33

4 Umfrage **33**
4.1 Das Ziel . . . 33
4.2 Vorstellung . . . 33
4.3 Aufbau . . . 34
4.4 Zielgruppe . . . 35
4.5 Kontrollfragen . . . 36
4.6 Erreichen der Testpersonen . . . 36
4.6.1 Forschungsplattformen . . . 36
4.6.2 Onlineforen . . . 38
4.6.3 Mailverteiler der Universitäten . . . 39
4.7 Ergebnis der Studie . . . 39
4.7.1 Die Zahlen . . . 39
4.7.2 Die Anforderungen . . . 45
4.8 Nebenerkenntnisse . . . 45

5 Das Werkzeug **48**
5.1 Überlegungen und Konzept . . . 48
5.2 Ansatz des verbesserten Werkzeugs . . . 49
5.3 Vorstellung des Prototyps . . . 50
5.4 Abschlussbemerkung . . . 56
5.5 Entwicklung drei verschiedener Werkzeuge . . . 57

6 Benutzbarkeitsstudie **58**
6.1 Einführung . . . 58
6.2 Bestandteile . . . 59
6.2.1 Beginn . . . 59
6.2.2 Fragebogen . . . 59
6.2.3 Das Werkzeug . . . 59
6.2.4 Das Nachgespräch . . . 60
6.2.5 Beantwortung des ISO Fragebogen 9241 /110 . . . 61
6.3 Anzahl Testpersonen . . . 61
6.4 Ergebnisse der Studie . . . 61
6.4.1 Ergebnis der ersten Umfrage . . . 61
6.4.2 Aussagen zum Werkzeug . . . 62
6.4.3 Eindruck am Ende des Testes: . . . 66
6.4.4 Ergebnis des Nachgesprächs . . . 67
6.4.5 Ergebnis der ISO Studie . . . 68
6.5 Verbesserungsvorschläge: . . . 70

7 Fazit **73**

8 Ausblick **74**

A Inhalt der CD-Rom **75**
A.1 Masterarbeit 75
A.2 Werkzeuge 75
A.3 Umfrage 75
A.4 Online_Umfrage 75

Literaturverzeichnis **76**

1 Einleitung

1.1 Motivation

Durch die nach wie vor immer stärker voranschreitende Digitalisierung der Gesellschaft [Inf15] in den relevanten Gebieten Business, Government und Privacy, steigt der Bedarf an funktionalem Code stetig an. Es entsteht beobachtbar eine immer weiter übergreifende Informationsverarbeitung zwischen den genannten Gebieten. Gesetzliche Regularien, ethische Teilaspekte und reine Bedarfsorientierung treffen hierbei teilweise frontal aufeinander und müssen datentechnisch aufeinander abgestimmt werden. Die Wichtigkeit und Komplexität dieser Problemstellungen betrachtend, nimmt die IT Security naturgemäß eine überaus wichtige Position ein. Sie ist der einzige Garant, dass sämtlichen Bedürfnissen der genannten Teilgebiete Rechnung getragen wird.
Doch trotz der hohen Investitionen in die IT-Security insgesamt [C.e17], sind Geschehnisse, wie zuletzt der Angriff auf das deutsche Regierungsnetz [Tag18a] leider keine Seltenheit [Stab]. Hacker[1] lassen keine Chance ungenutzt auch in zunächst einmal unwichtig erscheinende Systeme einzudringen. Programmierer, welche das Abdecken von Sicherheitslücken unsachgemäß ausführen und gar nicht erst bedenken, erleichtern dem besagten Personenkreis so das Eindringen in ein Netzwerk. Werden ehedem unwichtig erscheinende Systeme später zu komplexeren Systemen zusammengeschlossen, entstehen so teils unüberschaubar viele Sicherheitslücken.
Um diesem Problem etwas entgegen zu stellen, werden Werkzeuge zur Codeanalyse verwendet, welche Sicherheitslücken in bereits verfassten Code aufzudecken vermögen. Diese Tools sind sowohl als isolierte Anwendungen sowie als Plug-ins vorhanden und sollen dem Programmierer dabei helfen, spezielle Sicherheitslücken zu erkennen und im besten Fall zu beheben, um den Zugriff von z.B. Hackern auf geschützte Daten zu verhindern. Gleichwohl diese Werkzeuge bereits existieren, bleibt das Problem beobachtbar vorhanden, dass die Programmierer die Security beim Schreiben von Code zweitrangig beachten.
Die Idee ist nun ein Analysewerkzeug zu entwickeln, welches benutzerfreundlich ist, dem Programmierer dabei hilft die Sicherheitslücken im Code zu entdecken und ihn auch bzgl. der Sicherheit selbst zusätzlich sensibilisiert, um diesem Problem von Beginn an aktiv entgegenzuwirken.

[1] Hier wird sich auf sog. Black-Hats bezogen. Programmierer, welche in klar krimineller Motivation handeln, gerne auch als Auftragnehmer für die Privatwirtschaft (Industriespionage) u./o. Regierungen (Spionage) und beispielsweise versuchen ein System zu beschädigen oder Daten zu beschaffen.

1.2 Ziel

Das Ziel dieser Abschlussarbeit ist es nicht nur aufzuzeigen, dass die Nachlässigkeit von Programmierern bzgl. Sicherheitskriterien heute noch immer vorhanden ist, sondern auch eine Lösung anzubieten, welche auch wirklich angewendet werden wird. Diese benutzerfreundliche Lösung stellt ein Codeanalyse Werkzeug dar, wird anhand eines selbst entwickelten Prototypen getestet und mittels einer Usability Studie ausgewertet. Mithilfe der Ergebnisse der Usability Studie wird ein Katalog an Verbesserungsvorschlägen für die Ausarbeitung der finalen Anwendung sowie Empfehlungen zur Gestaltung des Werkzeuges erarbeitet. Dieses Codeanalyse Werkzeug kann dann implementiert und eine weitere Studie erstellt werden, inwiefern sich die Einstellung zur Beachtung und Abdeckung von Sicherheitslücken der Programmierer durch das Tool substanziell hat ändern lassen.

1.3 Aufbau der Arbeit

Nach dieser Einleitung wird in Kapitel zwei damit begonnen zu erklären, was Werkzeuge zur Codeanalyse sind und in welcher Art und Ausprägung sie heute genutzt werden. Anschließend wird die Notwendigkeit der Entwicklung eines neuen benutzerfreundlichen Tools aufgezeigt mit den nachweislich heute auftretenden Problemen der Programmierer zur Beachtung und Abdeckung von Sicherheitslücken. Im dritten Kapitel werden die erforderlichen Grundlagen und Begrifflichkeit für diese Abschlussarbeit erarbeitet. In Kapitel vier wird die Umfrage vorgestellt, dessen Ergebnisse den Prototypen des Werkzeugs zur Codeanalyse geformt haben. Schlussendlich folgt mit Kapitel fünf die Vorstellung des Werkzeugs sowie in Kapitel sechs die dazu gehörende Benutzbarkeitsstudie. Die Abschlussarbeit schließt mit der Auswertung des Werkzeugs und den Verbesserungsvorschlägen der Testpersonen.

Hinweis:
Weil Security und Safety im Deutschen die selbe Übersetzung mit dem Wort Sicherheit haben, diese aber semantische different sind (siehe Kapitel 3.4) werde ich in dieser Arbeit das englische Wort Security statt dem Wort Sicherheit verwenden, um diese eindeutig zu unterscheiden .

2 Einführung

2.1 Werkzeuge zur Codeanalyse

Werkzeuge zur Codeanalyse werden verwendet, damit der Quellcode auf Schwachstellen untersuchenwerden kann, welche zu Sicherheitslücken führen könnten [DB13]. Es existieren Werkzeuge, welche nicht nur alleinstehend funktionieren, sondern auch als Plug-in in eine Entwicklungsumgebung integriert werden können. Sie sind unterschiedlich mächtig und können funktio-

nale sowie technische Fehler im Quellcode erkennen. Andere hingegen sind zusätzlich in der Lage qualitative Schwachstellen im Quellcode zu entdecken, wie zum Beispiel das Vorhandensein von Duplikaten im Code. Das Prinzip der Überprüfung des Code durch solche Werkzeuge folgt hierbei grundsätzlich immer, zumindest grob, dem gleichen Schema (siehe Abb. 1).

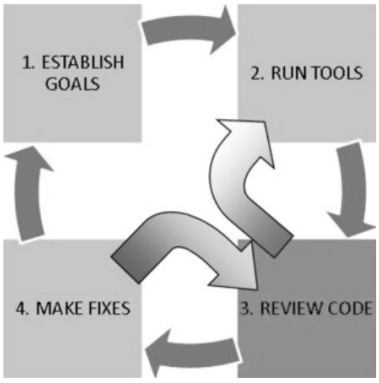

Abbildung 1: Der Prozess des Code Review [DB13]

In Phase eins wird der Code verfasst. Anschließend wird in Phase zwei das Werkzeug ausgeführt. In Phase drei wird der Quellcode dahingehend überprüft, ob Sicherheitslücken vorhanden sind. Letztlich werden in Phase vier die erkannten Fehler behoben. Weil nach jedem Hinzufügen bzw. Variieren von weiterem Code auch weitere Fehler auftreten könnten, werden die genannten Zyklen kontinuierlich wiederholt.
Die Überprüfung des Quellcodes kann hierbei sowohl statisch als auch dynamisch erfolgen. In der statischen Analyse [Owag] wird direkt der Quellcode betrachtet und nach logischen Fehlern untersucht, ohne dass der Code ausgeführt wird.
Bei der dynamischen Analyse [Tecc] hingegen wird nicht der Quellcode betrachtet, sondern das umgesetzte Programm im Ganzen und dessen verursachten Fehler. Es sind also Fehler, welche während der Ausführung einer Applikation oder eines Programms auftreten. Es lässt sich auch zwischen einem so genannten White Box und Black Box ([Lig02], S. 40) Test unterscheiden. Die statische Analyse entspricht einem White Box Test [tecd], denn die Technik nutzt die Struktur des Quellcodes. Das meint, dass der Quellcode sichtbar und zugänglich ist. Die dynamische Analyse entspricht dem Black Box Test [tecd]. Der Quellcode selbst gehört nicht zur Auswertung. Man betrachtet ausschließlich das gesamte Programm und sein Verhalten.

Es existierenden Dutzende von Werkzeugen zur statischen sowie auch dynamischen Codeanalyse, wie aus den entsprechenden Listen zu entnehmen ist: [Con] [Cur] [OWAe] [Pee]
Der nachfragebasierten Bedarfsorientierung folgend (es wird nichts produ-

ziert, was nicht nachgefragt wird) ist die schiere Menge an schon vorhandenen Werkzeugen ein klares Indiz für den hohen Bedarf an solchen Werkzeugen.

Der primäre Vorteil solcher Werkzeuge liegt klar auf der Hand, denn im Gegensatz zu einer manuellen Untersuchung des Codes, sind sie deutlich schneller. Für den Audit selbst braucht der Benutzer nicht das Expertenwissen eines Auditors([IGM], Kapitel 2.2). Darüber hinaus werden die erkannten Fehler präzise angezeigt [Owag].
Gleichwohl die Erkennung von Fehlern sowie die Reparatur des Programms sehr aufwendig und komplex sein kann, helfen diese Werkzeuge dabei ein besseres und sichereres Programm und Endprodukt zu erstellen [Seab].

2.2 Grund der Studie

2.2.1 Vergangene Ereignisse

Die heutige Gesellschaft ist schon jetzt stark digital durchsetzt und oft auch untereinander vernetzt. Die Nachfrage nach digitalen Lösungen, in Business, Government und privat steigt stetig [Onl] [Ser16]. Wer Lösungen sucht, will sich jedoch üblicherweise nicht mit dem Weg zur Lösung selbst, und schon garnicht mit dem Problem selbst auseinander setzen. Wie beim Autokauf wünschen die Auftraggeber schöne Produkte nach ihren jeweiligen Bedürfnissen. Die Zahl der Anbieter, welche immer auch ihr individuelle Lösung mitbringen, von Hard- und Software steigt, dem Nachfrageprinzip folgend, also auch stetig. Es ist ersichtlich, dass sich dadurch die IT Security immer komplexeren Herausforderungen stellen und Lösungsansätze anbieten muss. Oft sogar für Problemstellungen, die durch die Märkte noch nicht einmal zur Gänze formuliert wurden. Grundlegende Probleme sind hier nach wie vor alle Formen und Ausprägungen des Identitätsdiebstahles, das stehlen u./o. die Gefangennahme von privaten oder unternehmenswichtiger Daten oder gezielte Angriffe auf IT Systeme aller Art, um diese lahm zu legen [Bun16].

Laut einer Studie des Bundeskriminalamts im Jahr 2016 gab es deutschlandweit ca. 82.000 offizielle Fälle von Cybercrime [Stab]. Darunter fallen Straftaten, die sich auf die illegale Nutzung von Internetzugängen beziehen, wie z.B. der elektronische Diebstahl von Zugangsberechtigungen. Ebenso wie Straftaten welche zum Ziel haben Teile der Netzstruktur entweder zu hijacken, oder einfach nur lahmzulegen [Bun16]. Letztlich nehmen natürlich auch die Angriffe auf informationstechnische Systeme selbst zu. Ca. 70 % der Fälle fallen dabei unter die sog. Betrugsfälle wie z.B. Kreditbetrug, Überweisungsbetrug oder auch der Abrrechnungsbetrug beim Gesundheitswesen([Bun16], Seite 6). 12% der Fälle zielten auf das direkte Stehlen von

digitalen Identitäten ab, z.B. durch das Phishen [2] von Kontodaten([Bun16], Seite 6). Nahezu sämtliche E-Commerce Dienste sind davon betroffen. Ca 4% der Fälle sind Vorfälle mit der Absicht zur Beschädigung von Systemen([Bun16], Seite 6). Rechtlich betrachtet handelt es sich dabei im Prinzip um eine Art der digitalen Sachbeschädigung und das mit Hilfe von z. B sog. DDos Attacken[3] oder der Nutzung spezieller Schadsoftware. Weltweit nahmen sogar DDos Attacken vom Jahr 2016 zu 2017 um 14% zu [Aka17] von denen Akamai in ihrem Sicherheitsbericht selber noch zusätzlich erwähnen, dass dies „auf einen langfristigen Anstieg hindeutet“ [Aka17, S. 3].

Laut Feststellung des Fachverbandes bitkom[4], in ihrem Bericht zur „Spionage, Sabotage und Diebstahl“, waren in den Jahren 2013 und 2014 rund 51% der deutschen Unternehmen von elektronischer Wirtschaftsspionage u./o. Datendiebstahl betroffen [e.V16]. Ganze 19% der Unternehmen wurden laut eigenen Aussagen sogar mindestens einmal im Monat geschädigt oder ausspioniert [Staa].
Die aktuelle Studie der bitkom, welche die Aussagen ihrer Mitglieder auswertet, gibt sogar an das 67% der deutschen Unternehmen im letzten Jahr Opfer von IT-Angriffen waren [e.V18].

Immer wieder kommt es auch zu derart großen Angriffen, dass deren Auswirkungen auch der Presse nicht verborgen bleiben.
Ein derart bekannter Fall war der eines 29 jährigen Briten, welcher im Jahr 2016 Router der Telekom erfolgreich angriff und lahmlegte [DW18] [fSidI17]. Dabei soll es am Sonntag den 27. November 2016 zu rund 900.000 Störungen deutscher Telefonanschlüsse gekommen sein [fSidI17] .

Oder der Fall im Jahr 2015, als das Netz des Bundestages selbst angegriffen wurde [Onl17]. Die vermuteten Angreifer waren die sog. Sofacy Group aka. Fange Bear aka. Advanced Persistent Threat 28 (APT28), eine Gruppe welcher unterstellt wird, dass sie Verbindungen zur Russischen Regierung hätte [Heu18]. Zu den Zielen diesen Angriffes soll die Bundeskanzlerin selbst gehört haben [Onl17]. Es soll dazu geführt haben, dass das gesamte Netzt des Bundestages ausgetauscht werden musste [Zei15].

Aktuell wurde die Bundesregierung erneut Opfer eines Angriffes. Am 28.02.18 wurde bekannt, dass es Angreifern schon im Dezember 2017 gelungen war

[2] Es setzt sich aus dem Wort „Password “und „fishing“zusammen und entspricht dem „Angeln nach Passwörtern“. Phishing-Betrüger haben über E-Mails und Internetseiten einen Weg gefunden, um an vertrauliche Daten wie Passwörter, Zugangsdaten oder Kreditkartennummern heran zu kommen [fSidIa]

[3] steht für Distributed Denial of Service und bezeichnet einen koordinierten und großflächigen Angriff mit einer Vielzahl von unterschiedlichen Systemen [fSidIc]

[4] https://www.bitkom.org

Daten abzugreifen [Tag18b]. Das Auswärtige Amt war dieses mal nachweislich betroffen. Ob weitere Ministerien betroffen waren oder sind, ist bisher noch unklar [Tag18c]. Auch ist noch nicht eindeutig wer diesen Angriff ausgeübt hat. Einerseits wird er erneutet der Gruppe ATP28 zugeschrieben [Tag18b], andererseits werden auch öffentlich Annahmen gemacht, dass sich hinter dem Angriff die sog. Gruppe „Snake“verbergen könnte [Tag18c]. Was aber bekannt ist, ist, dass die Angreifer sich den Zugang über die Hochschule des Bundes ermöglicht haben und von dort aus weiter ins Auswärtige Amt vorgedrungen sind [Tag18b].
Aktuell wird das Netzwerk noch nach Hinterlassenen Hintertüren durchsucht, mit denen ein erneuter Angriff möglich wäre. Dennoch soll es gelungen sein, die Angreifer innerhalb der Bundesverwaltung isoliert zu haben [Tag18c].

Sowohl die Menge der Werkzeuge zur Codeanalyse, als auch die schwere der Angriffe in der Vergangenheit zeigt nicht nur den substanziellen Bedarf, sondern auch die zwingende Notwendigkeit zur Erkennung von Sicherheitslücken auf.

Neben der statischen und dynamischen Analyse von Quellcode gibt es natürlich auch Werkzeuge, welche sich speziell auf das Erkennen von Securitylücken spezialisiert haben. Fehler dieser Art zu erkennen, sind besonders schwierig [IGM, Kapitel 2.2]. Das NIST[5] (National Institut of Standards and Technology) führt eine Liste dieser Art Werkzeuge [NIS]. Sie führen über 50 auf, welche unter anderem SQL Injektions[6], Cross Site Cripting[7] oder auch hart codierte Passwörter aufdecken und verhindern sollen.
Security Tools sollen dabei helfen die Sicherheitslücken zu erkennen und diese aufzuzeigen. In Zeiten von Zeitdruck und dem regelrecht obsessiven Fokus auf „effizientes“Handeln, bleibt die Security im Allgemeinen dann doch auf der Strecke.
Der Fokus dieser Abschlussarbeit liegt daher auf der Analyse der Werkzeuge, welche speziell für das Aufdecken von Security Problemen konstruiert wurden.

2.2.2 Nutzerverhalten

Obwohl, wie bereits festgestellt, eine Vielzahl an Werkzeugen vorhanden ist, sowie erfolgreiche Angriffe hinreichend dokumentiert und öffentlich kommu-

[5] https://www.nist.gov

[6] „Eine SQL-Injection beschreibt die Ausführung von Datenbank-Code auf Basis einer Schwachstelle. Hierdurch ist es dem Angreifer möglich die komplette Datenbank auszulesen und gegebenenfalls zu entwenden“ [fSidIb].

[7] sind eine Art von Injektion, bei der schädliche Skripts in ansonsten gutartige und vertrauenswürdige Websites eingefügt werden [OWAb].

niziert wurden, sind noch immer die gleichen Verhaltensweisen der Nutzer zu beobachten. So sind sich z.B. die Mitarbeiter eines Unternehmens der Gefahr einer sog. Phishing Mail durchaus bewusst, dennoch klicken 76% den unbekannten Link einer Email an, um zu sehen was sich dahinter verbirgt [hBN].
Dieses willkürlich herausgegriffene Beispiel ist zwar nicht repräsentativ, aber es vermag dennoch aufzuzeigen, welche Priorität der Gedanke der Security im Allgemeinen beim Endnutzer einnimmt. Gilt dann der Programmierer, in der Form der Programmcode schaffenden, hier dann noch als besagter Endnutzer und zeigt also die gleiche Einstellung auf, muss diesem Verhalten, aus Sicht der Security, definitiv entschieden entgegen gewirkt werden.
Die Frage, die sich also stellt, ist:
„Wieso sind solche Angriffe noch möglich, wenn doch so viele Security Tools existieren?“

Paper - Why Dont Software Developers Use Static Analysis Tools to Find Bugs

Das Paper „Why Dont Software Developers Use Static Analysis Tools to Find Bugs“ [JSMHB13], welches auf der ICSE[8] 2013 veröffentlicht wurde, sah die Antwort in der fehlenden Benutzbarkeit von Analyse Werkzeugen, weshalb Programmierer diese Werkzeuge nicht oder nur ungern nutzen.
Sie hatten 20 Personen interviewt und ließen sich das Feedback in Form von Thinking Aloud Test (siehe Kapitel 3.9) zu kommen. Das Ergebnis war, dass sie zwar die Zeitersparnis sehr positiv empfanden, weil kein eigenständiges Testen mehr notwendig war, aber das Ergebnis der Analyse wurde schlecht dargestellt.

Sie haben sich ein benutzerfreundliches und intuitiveres Design gewünscht, weil sie sich von der Menge der Warnungen erschlagen fühlten([JSMHB13], Seite 5). Die Fehlermeldungen waren nicht konkret und haben nicht präzise angezeigt, wo der Fehler liegt.
Aufgrund dieser riesigen Darstellung der Fehlermeldungen äußerte sich ein Studienteilnehmer mit „a bunch of junk to shift through“ [JSMHB13, S. 5].
Ein weiteres Problem stellte das grundsätzliche Verständnis dar. So hieß es:

> „A developer not being able to understand what the tool is telling her, according to our participants, is a definite barrier to use“ [JSMHB13, S. 6]

Die Fehlermeldungen sollten demnach verständlich genug sein.
Was sich am meisten an einem Tool gewünscht wurde, war die Funktion eines Quickfixes, ein schneller Verbesserungsvorschlag des Tools, zur Behebung der

[8] International Conference on Software Engineering http://www.icse-conferences.org

Sicherheitslücke.

> „Most of our participants expressed interest in having their tool provide code suggestions or quick fixes [...]“ [JSMHB13, S. 6]

Zusammenfassend lässt sich sagen, dass sich die Studienteilnehmer dieses Paper ein ansprechendes Design wünschen, welches benutzerfreundlich und intuitiv bedient werden kann, sowie eine nachvollziehbare und verständliche Darstellung der Fehler. Zuletzt sollte noch eine Quickfix Lösung existieren, welche eine vorgeschlagene Lösung des Problems präsentiert.

Paper - Maybe Poor Johnny Really Cannot Encrypt

Ein anderes Paper, welches ein anderes Problem aufzeigt, ist im Jahr 2015 auf der NSPW[9] mit dem Titel „Maybe Poor Johnny Really Cannot Encrypt“ [BLO+15] veröffentlicht worden. Es thematisiert die Theorie, dass die menschliche Kapazität Grund für das Nichterfassen eines Themas sein kann. In diesem Fall wurde behauptet, dass ein Mensch aufgrund der kognitiven Fähigkeiten konkret nicht in der Lage sein könnte, ein Verschlüsselung vorzunehmen und das dafür nicht dem Benutzer die Schuld gegeben werden kann. Sie belegte die Erkennung von kognitiven Fähigkeiten einer Person anhand von Literatur aus der Psychologie, welche aufzeigt, dass gewisse Grenzen vorhanden sind. Sie behaupten, dass eine Messung der Grenzen mit Hilfe eines Elektro Kardiogramms möglich ist. Es nimmt die elektrische Aktivität des Herzens auf und kann durch einen Arzt oder eine Software interpretiert werden. Sie sagen:

> „Several studies identify a relationship between heart rate variability (HRV) in ECG recordings and stress caused by strain (cognitive or physical). [ ]Stress is a feeling of strain and pressure and occurs when people feel they are unable to manage the demands placed on them, which can be cognitive or physical tasks, deadlines, major life events etc.“ [BLO+15, S. 8]

Wenn sie demnach zu viel Stress bei der Bewältigung einer Aufgabe machen, geht die Herzrate hoch, was ist ein Hinweis darauf ist, dass die Aufgabe kognitiv von der Person nicht, oder nicht mehr, erfasst werden kann. Wie genau diese Messungen stattfinden und deren Interpretation ist in diesen Paper zwar nicht weiter ausgeführt. Was sie jedoch aufzeigen, sind die Prioritäten einer Person, wenn sie eine Aufgabe bewältigen möchte.

> „[...] the primary goal of sending an encrypted email is communication, and the secondary goal is confidentiality “ [BLO+15, S. 3]

[9] New Security Paradigms Workshop , https://www.nspw.org

Es wird demnach erst auf die eigentliche Aufgabe der Fokus gesetzt, hier nämlich das Versenden einer Nachricht, und dann erst die Sicherheit der Sache selbst berücksichtigt.

Wenn nun, wie in diesem Paper behauptet, die eigenen kognitiven Fähigkeiten die Grenzen setzen, inwiefern eine Person überhaupt Security verstehen kann, dann kann dieses grundsätzliche Defizit nur über das Design und die Benutzerfreundlichkeit des Werkzeugs ausgeglichen werden [BLO+15, S.2]. Werkzeuge müsste also die Defizite des Nutzers kompensieren und sich an ihn anpassen und nicht anders herum, indem der Nutzer sein Verhalten an das Werkzeug anpasst. Nur in dieser Art ist es zielführend möglich Securitylücken mit einem Werkzeug verständlich für den Benutzer aufzuzeigen.

Paper - Stack Overflow Considered Harmful?

Nach diesen zwei Bespielen könnte man annehmen, dass allein das Erstellen eines wirklich guten Designs die eigentliche Herausforderung für ein gutes Werkzeug zur Codeanalyse für Securitylücken darstellt. Das dem nicht so ist zeigt jedoch folgendes Paper auf, indem es ein noch viel schwerwiegenderes Problem aufzeigt. Es nennt sich „Stack Overflow Considered Harmful? The Impact of Copy and Paste on Android Application Security“ [FBX+17] und wurde 2017 auf dem IEEE Symposium on Security and Privacy[10] veröffentlicht.
Stack Overflow[11] ist eine online Plattform, welche Programmieren die Möglichkeit bietet Fragen sowie Antworten zur Softwareentwicklung zu veröffentlichen [Ove]. Die Plattform versammelt eine gewaltige Community aus Programmieren aller Bildungsstufen. Vom Profiprogrammierer bis zum Hobbyenthusiasten ist alles vertreten.
Es können Fragen an diese Community gestellt werden und auch Antworten nach deren wahrgenommener Qualität bewertet werden. Durch die sog. „Votes“ [Ove] kann die Reputation eines Mitglieds erhöht werden, welches für andere Nutzer aufzeigt, wie nützlich die Antworten eines Nutzers offensichtlich für andere waren. Je mehr Votes eine Person bekommt, desto kompetenter erscheint also die Person.
Mit 8.8 Millionen Nutzern [Exc] stellt die Plattform eine der, wenn nicht gar die, größte Community in diesem Bereich dar. Es ist ein riesiges Nachschlagewerk für Programmierer zur Softwareentwicklung, durch die Veröffentlichung von kompletten Codeabschnitten.

Im besagten Paper wird eine Studie zur Untersuchung dieser Platform dokumentiert. Es wurde hierbei untersucht inwiefern Codeschnipsel aus der

[10] https://www.ieee-security.org/TC/SP2018/
[11] https://stackoverflow.com

Plattform wieder in Apps, welche im Google Play Store veröffentlicht wurden, zu finden sind und in wiefern diese noch Securitylücken aufweisen. Dafür wurden alle Veröffentlichungen, welche Android[12] betreffen untersucht und geprüft, welche davon sicherheitsrelevante Codeschnipseln beinhalten. Anschließend wurde untersucht, ob sich diese Codeschnipsel wieder in den rund 1,3 Millionen veröffentlichen Android Applikationen wiederfinden.

Das erstaunliche Ergebnis [FBX+17] war, dass 15.4% der rund 1,3 Millionen Android Applikationen Security relevante Codeschnipsel aus Stack Overflow enthielten und von diesen hatten fast 98% wenigstens eine Security Lücke.

Paper - You Get Where Youre Looking For

Eine ganz ähnliche Tendenz wurde in dem Paper „You Get Where Youre Looking For“ [ABF+16] behandelt, welches 2016 auf dem IEEE Symposium on Security and Privacy[13] veröffentlicht wurde.
Inhaltlich ging es darum, dass 54 Entwickler, sowohl Anfänger als auch Profis, Android Anwendungen programmieren sollten. Die Verfasser dieses Papers wollten aufzeigen, welchen Einfluss das Material oder die Informationen auf die Entwickler haben, bzgl. der Sicherheit des Endproduktes. Je nachdem, welche Quelle die Programmierer nutzen durften, sind entweder sehr sichere oder sehr unsichere Ergebnisse entstanden. Es wurden vier Gruppen gebildet, sortiert nach dem ausschließlichen Nachschlagematerial, welches sie verwenden durften.
Die erste Gruppe hatte eine freie Materialwahl. Die zweite Gruppe durfte nur die Plattform Stack Overflow nutzen. Die dritte Gruppe durfte nur die Android Dokumentation nutzen und der letzten Gruppe war es ausschließlich gestattet Bücher zu verwenden.

Diejenige Gruppe welche nur die Plattform Stack Overflow als Nachschlagewerk nutzen durften, haben im Vergleich zu den Gruppen mit der Android Dokumentation und der ausschließlichen Nutzung von Büchern deutliche mehr funktionalen Code verfasst, aber er zeigte auch deutlich mehr Securitylücken auf. Die Programmierer begründeten dies damit, dass viele der Security Fehler obhin der „chaotischen“ Organisation der Plattform zuzuschreiben wäre.

Das Ergebnis dieser Studie zeigte auf, dass die Nutzung der API Dokumentionen zu sehr sicherem Code führen, jedoch schwer zu lesen und zu nutzen, wohingegen Plattformen wie Stack Overflow sehr leicht erreichbar sind, aber schnell zu einem unsicheren Code führen. Weil jedoch der öko-

[12] https://www.android.com
[13] https://www.ieee-security.org/TC/SP2018/

nomische Druck so groß ist und die Android Dokumentation nicht wirklich gut verständlich und nutzerfreundlich ist, wird angenommen, dass die Entwickler weiterhin die Plattform Stack Overflow nutzen werden, weil es der schnellste und vor allem einfachste Weg ist seine Lösungen zu bekommen.

2.2.3 Das Ziel des Werkzeugs

All das o.g. berücksichtigend wird ersichtlich, das es nicht genügt lediglich ein benutzerfreundliches Werkzeug zur Codeanalyse zu gestalten. Solange die Nutzer nicht bereit sind solche Werkzeuge auch aktiv nutzen und solange Programmierer nicht die Notwendigkeit der Überprüfung von Codeschnipseln vor der Veröffentlichung realisieren, werden auch weiterhin die beschriebenen Securitylücken vorhanden sein. Wie schon Bruce Schneier [Sch], ein international renommierter Kryptograf und Securityspezialist der ersten Stunde, sagte, ist der Mensch die eigentliche Schwachstelle.

> „People often represent the weakest link in the security chain and are chronically responsible for the failure of security systems.“ [Sch00, Kapitel 17]

Wird dieser nicht für die Problemstellungen sensibilisiert, wird sich auch nichts mit einem besseren Werkzeug ändern, weil es ja nie vollumfänglich genutzt wird. Denn obwohl sich schon sehr lange, teilweise sehr ausgereifte Werkzeuge (siehe Kapitel 3.1) im Umlauf befinden, ist das Problem noch bis heute vorhanden.

Ziel dieser Arbeit ist es daher ein benutzerfreundliches Werkzeug zur Codeanalyse zu gestalten, welches die sichtbare Komplexität der Problemstellung deutlich reduziert, Securitylücken verständlich darstellt, nutzerorientierte Lösungsansätze bietet und darüber hinaus den Nutzer bzgl. der Beachtung der Security sensibilisiert.

2.3 Ablauf der Studie

Diese Abschlussarbeit besteht aus einer Umfrage, mit deren Hilfe ich die Anforderung für das Werkzeug ermittelt habe. Zusätzliche wurde dieses Werkzeug vorgestellt und von Testpersonen getestet. Anschließend wurde es mittels einer Usabilitystudie ausgewertet, um festzustellen wie benutzerfreundlich es ist.
Das Ziel war es herauszufinden, wie dieses Werkzeug von den Testpersonen angenommen wird, ob sie es verstehen und welche Verbesserungen sie ggf. vornehmen würden. Dabei sollte nicht nur herausgefunden werden, ob die Testpersonen mit der Funktionalität des Werkzeuges zufrieden sind, sondern inwiefern sie im Faktor Security beeinflusst wurden.

Sämtliche Verbesserungsvorschläge wurden notiert und zum Schluss ausgewertet. Anhand dieses Ergebnis wird die Verbesserung des Werkzeugs vorgestellt.

2.4 Bestandteile der Studie

Die Umfrage selbst besteht aus einem Fragenkatalog, welcher in Form einer Onlineumfrage erhoben wurde. Sie diente dazu, um im Rahmen dieser Abschlussarbeit selber festzustellen, ob Programmierer Werkzeuge zur Codeanalyse, zur Erkennung von Securitylücken verwenden. Wenn sie diese verwenden, soll herausgefunden werden, welche Gründe dazu vorliegen, ebenso wenn sie solche nicht nutzen, warum sie diese nicht verwenden.
Die Benutzerfreundlichkeit des entworfenen Werkzeuges wurde mit einem Fragebogen nach ISO Norm 9241/110 [PA] ermittelt.

Das entwickelte Werkzeug gleicht einem Editor, welcher zusätzlich auch Sicherheitskriterien überprüft, Verbesserungsvorschläge vorschlägt und diese, auf einfache Art und Weise, erläutert. Obgleich das Werkzeug ein Prototyp ist, ist es soweit mit Dialogfenster sowie Interaktionen ausgestattet, dass er den Testpersonen einen vollständigen Eindruck vermitteln soll und die Funktionen der Überprüfung eines Quellcodes ermöglicht.

3 Grundlagen

3.1 Beliebte Werkzeuge

Wie im Kapitel 2.1 erwähnt, gibt es einige Werkzeuge, welche nicht nur für die statische, sondern auch für die dynamische Analyse verwendet werden können. Um mir einen besseren Eindruck von der Qualität und Darstellungsform dieser Werkzeuge zu verschaffen, habe ich mir die Werkzeuge genauer angesehen, welche entweder als sehr gut [Che] oder beliebt [Hel] zur Codeanalyse u./o. zur Erkennung von Securitylücken beworben wurden.

3.1.1 OWASP LAPSE+

OWASP LASPE+[14] ist ein Opensource[15] Werkzeug und Security Scanner für Java EE[16]. Es wird demnach als Plug In verwendet [Owaf]. Es wird die Logik des Codes untersucht, ohne dass der Code ausgeführt werden muss. Es deckt Schwachstellen in Webanwendungen wie SQL Injections (siehe Kapitel 3.11), Cross site Scripting (XSS) (siehe Kapitel 3.11) oder auch Cookie

[14] https://www.owasp.org/index.php/OWASP_LAPSE_Project
[15] Software, deren Quellcode offen von Dritten eingesehen werden kann
[16] ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von, in Java programmierten, Anwendungen und insbesondere Web-Anwendungen

Poising[17] auf [Owaf].

Für die Erkennung der Fehler sind drei Schritte notwendig [Owaf].
Der erste Schritt ist die Erkennung der Schwachstelle im Quellcode (Vulnerability Source), welche zur Dateninjektion genutzt werden kann.
Der zweite Schritt ist die Feststellung, wie die Manipulation erfolgen kann (Vulnerability Sink).
Als dritter Schritt wird zuletzt festgestellt, ob die Manipulation die Schwachstelle des Quellcodes erreichen kann (Provenance Tracker). Wenn das zutrifft, weiß man dass eine Sicherheitslücke im eigenem Quellcode vorhanden ist.

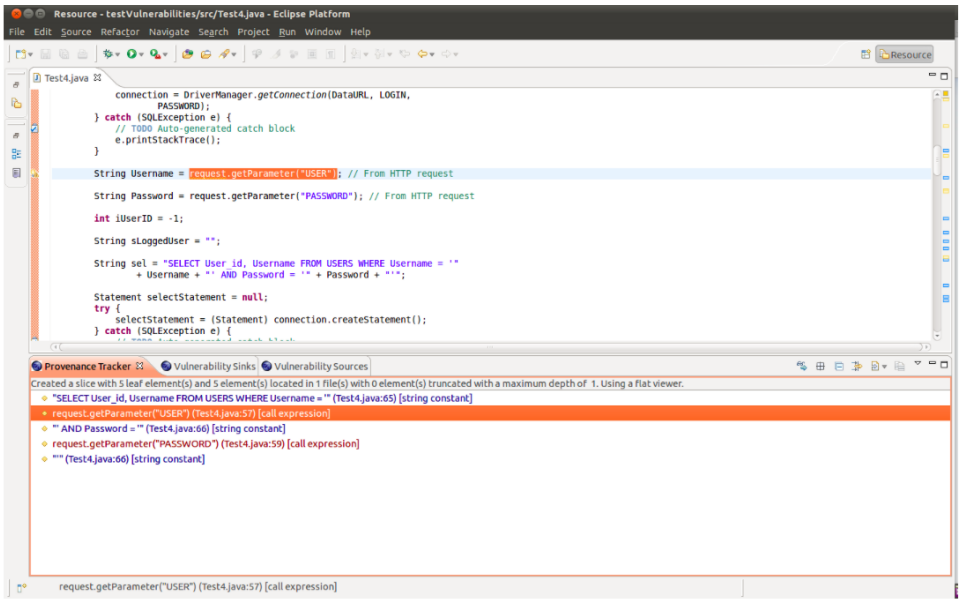

Abbildung 2: OWASP LAPSE Plus Screenshot [Lap]

Abbildung 2 stellt das Plug-in LAPSE+ in der Entwicklungsumgebung Eclipse[18] dar.
Hier ist zu sehen, dass deutlich angezeigt wird, wo sich die Schwachstellen im Code befinden. Das genaue Problem, wird jedoch nicht angezeigt. Auch wird keine konkrete Hilfestellung angeboten, wie das gezeigte Problem behoben werden könnte oder sollte.
Und es fehlt eine Erklärung zum Problem selbst, was es für den Programmierer schwieriger macht die Sicherheitslücken zu schließen.

[17] Modifikation eines Cookies von einem Angreifer, um an unautorisierte Inhalte zu geraten [Seaa]
[18] https://www.eclipse.org

3.1.2 Coverity

Coverity[19] ist auch ein Open Source Werkzeug jedoch kein Plug-in, sondern eine eigenständige Anwendung und Unterstützt unter anderem Sprachen wie C, C++, Java, Javascript, PHP sowie Python [Syn]. Dem Werkzeug wurde eine sehr gute Benutzerfreundlichkeit zugesprochen [Cro].

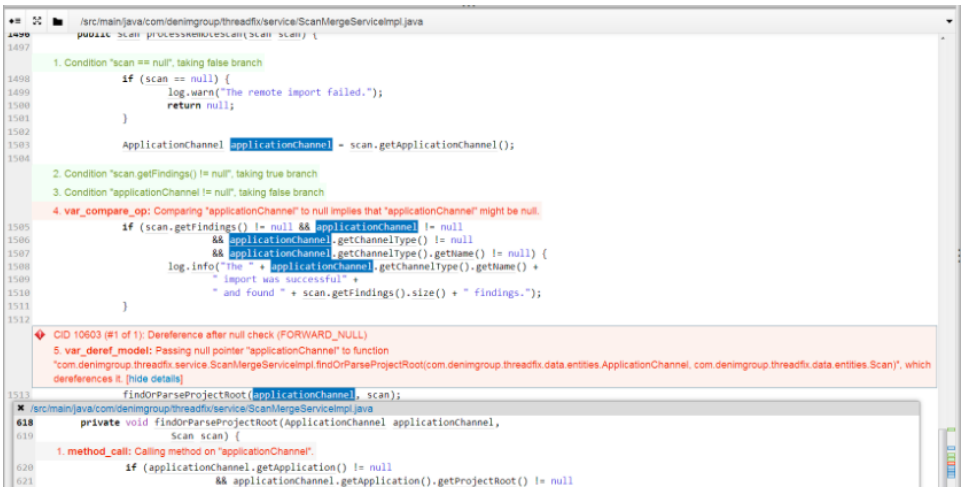

Abbildung 3: Coverity Screenshot [Cov]

Anhand der Abbildung 3 lässt sich erkennen, dass Coverity nicht nur deutlich aufzeigt, wo genau im Quellcode das Problem liegt, sondern diese auch präzise bezeichnet.
Erklärungen zu dem/den Securityproblem(en) sind jedoch nicht gegeben, damit der Programmierer diese verstehen kann.

3.1.3 RIPS

Das neueste Werkzeug dieser drei Beispiele ist die 2016 veröffentlichte Software RIPS[20]. Eine Software zur Erkennung von Sicherheitslücken ausschließlich für PHP [Teca]. Es entdeckt sehr komplexe Sicherheitslücken [Tecb] und kann entweder als Desktopsoftware oder als Cloud Service genutzt werden.

Dieses Werkzeug zeigt nicht nur an, welche Sicherheitslücken vorhanden sind, sondern auch genau wo im Quellcode. Es zeigt einem an, wie hoch die Gefahr der jeweiligen Sicherheitslücke ist und damit auch eine Priorität, welches Problem man als erstes beheben sollte.
In Form der Aufklärung bietet es nicht nur eine Gesamtübersicht über alle Sicherheitslücken im Quellcode, sondern auch eine Erklärung zum Problem

[19] https://scan.coverity.com
[20] https://www.ripstech.com

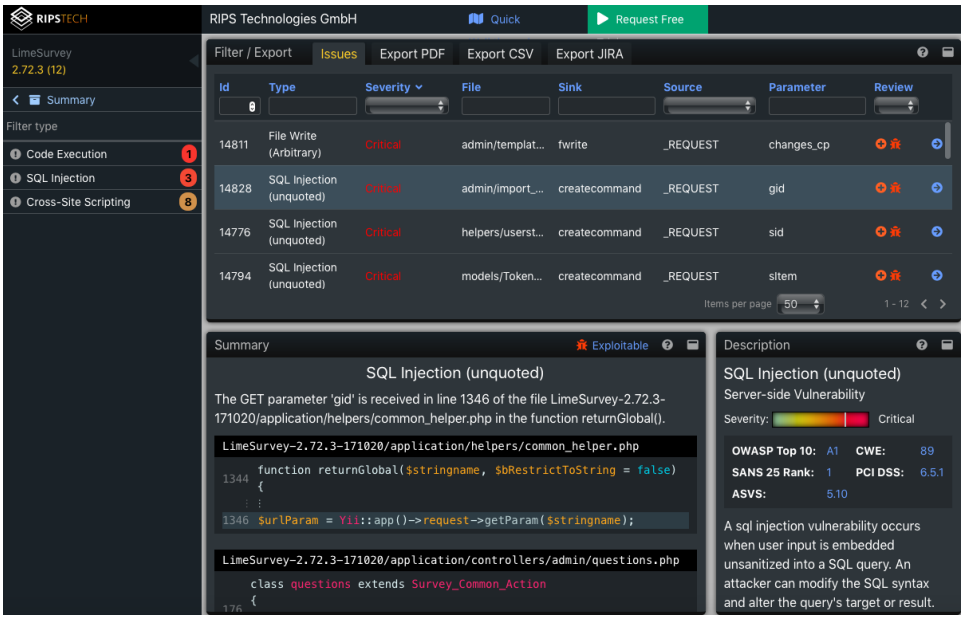

Abbildung 4: RISP Screenshot [RiS]

selbst (siehe Abb. 4).

Allerdings fehlt ein Quickfix, welches schnelle übersichtliche Lösungen anbietet und eine positive Bestätigung dafür, wenn der Code bereits gewisse Sicherheitslücken abgedeckt hat.

3.2 Beliebte Editoren

Weil für das Werkzeug ein Editor benötigt wird, damit darin der Quellcode nach Sicherheitslücken untersucht werden kann, habe ich mir auch hier empfohlene sowie beliebte Editoren genauer angesehen [t3n] [Liv] [Jac].

3.2.1 Sublime

Sublime[21] (siehe Abb. 5) ist ein Codeeditor, welcher, aufgrund seiner Möglichkeit dutzende von Plug-ins zu installieren und seinem sehr gutem User Interface, eine gute Alternative zu mächtigen IDEs[22] darstellt. Sublime hilft einem dabei seinen Quellcode sowohl akkurat als auch effizient zu verfassen, zum Beispiel durch den Einsatz vom gleichzeitigen editieren mehrerer Zeilen [Sub]. Er ist schnell und kann gleich nach der Installation genutzt werden [Sub].

[21] https://www.sublimetext.com
[22] Integrated Development Environment

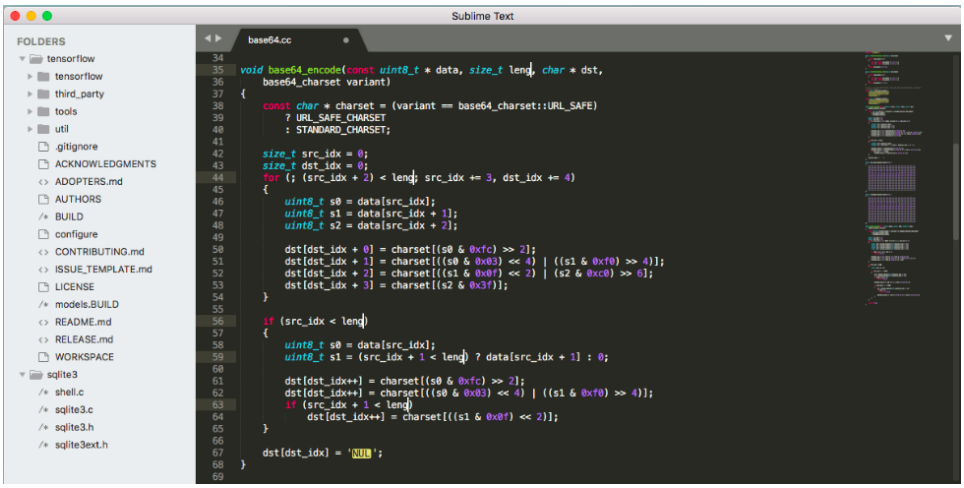

Abbildung 5: Sublime Screenshot [Sub]

3.2.2 Visual Studio Code

Visual Studio Code[23] ist ein Codeeditor und ermöglicht es einem Quellcode zu verfassen ohne das gesamte Visual Studio[24] laden zu müssen, welches immerhin üppige 3GB umfasst. Anders als Sublime kann dieser Editor kostenlos genutzt werden [Coda]. Es unterstützt sehr viele Programmiersprachen [Codb], enthält ein Autocomplete des Quellcode [Coda] sowie eine Git Versionskontrolle [Codc]

3.2.3 Atom

Atom[25](siehe Abb. 6) bezeichnet sich sich selbst als einen „Hackable“ Codeeditoren, welcher ebenso kostenlos sowie Open Source ist und vom Team von GitHub[26] entwickelt wurde [Atoa]. Damit ist es nicht nur selbstverständlich, dass sie eine Versionskontrolle mit Git Hub durchführen, sondern fordern einen auch auf über Git Hub den Editor weiter gemeinsam zu verbessern [Atoc]. Es setzt mit seiner umfänglichen Sammlung an Themes auch einen großen Fokus auf das Design an sich, sowie eine einfache Anbindung zur Nutzung als IDE mit einer Auto-Completion, das Anzeigen von Definition, als auch eine Diagnose zur Erkennung von Fehlern und Warnungen [Atob]. Aufgrund ihrer jeweiligen Beliebtheit, habe ich mich beim Design meines Editors an diesen drei Editoren orientiert, um nicht nur den Testpersonen ein gewohntes Werkzeug zu präsentieren, sondern auch eines welches optisch offensichtlich in breiter Masse ansprechend ist.

[23] https://code.visualstudio.com
[24] https://www.visualstudio.com/de/
[25] https://atom.io
[26] https://github.com

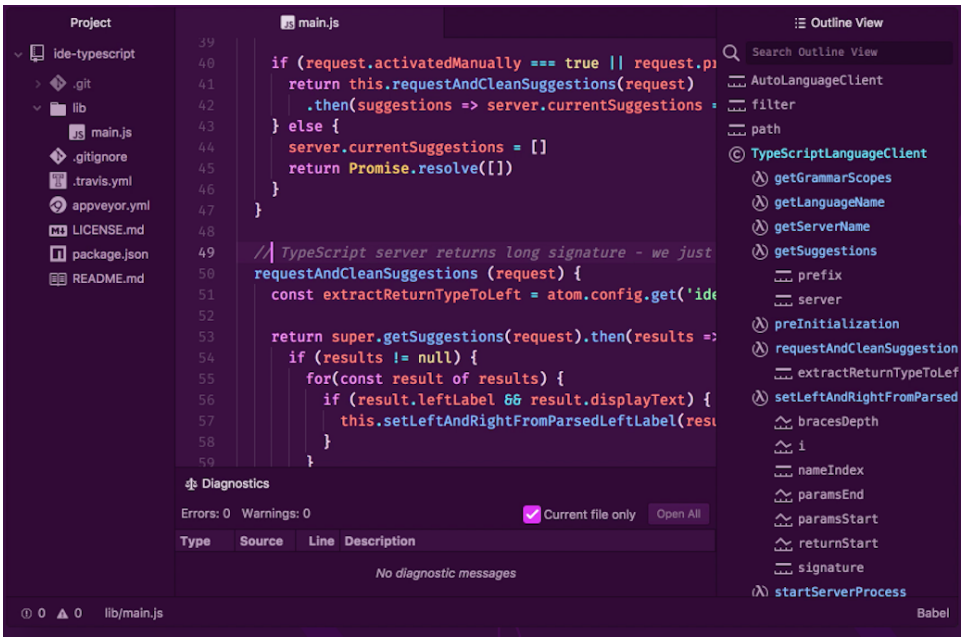

Abbildung 6: Atom Screenshot [Atob]

3.3 Was ist Security

Es existieren verschiedene Definitionen von „Security“. Laut ISO Norm steht die Security von Informationen für die

> „Preservation of confidentiality, integrity and availability of informatio“ [ISO]

Übersetzt meint das sinngemäß, „die Wahrung der Vertraulichkeit, Integrität und Verfügbarkeit von Informationen “.

Eine ausführlichere Definition bietet das CNSS[27]. Sie definiert es mit den Worten

> „protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.“ [Sec]

Übersetzt meint das sinngemäß: „Der Schutz von Informations- und Informationssystemen vor unberechtigtem Zugriff, Verwendung, Offenlegung, Unterbrechung, Änderung oder Zerstörung, um Vertraulichkeit, Integrität und Verfügbarkeit zu gewährleisten“.

[27] Committee on National Security System, https://www.cnss.gov

Eine allgemeinere Definition zum Begriff Security [GL14, S. 8] bezeichnet sie als Prozess, der Sicherheitsanforderungen identifiziert, um eine Sicherheitsstrategie zu entwickeln, welche diese Anforderungen und Mechanismen zu gewährleisten im Stande ist, sowie diese Strategie dann zu implementieren .

Diese Definitionen beziehen sich jedoch nur auf die Security von Informationen oder schließen die Zuverlässigkeit von Maschinen ein.

Eine präzisere Definition zum Wort „Security“ im Kontext dieser Abschlussarbeit lautet nach ISO Norm 2382-8, dass Computer Security

> „the protection of data from accidental or malicious acts, usually by taking appropiat actions. [...] these acts my be modification, destruction access, disclosure, or acquisition if not authorized.“ [GL14, S. 9]

ist. Übersetz steht dies sinngemäß für
„den Schutz von Daten vor zufälligen oder böswilligen Handlungen, normalerweise durch geeignete Maßnahmen. Diese Handlungen sind Modifikationen, Zugriff zur Zerstörung, Offenlegung oder Erwerb von Daten, wenn sie nicht autorisiert sind.“

Diese letzte Definition zeigt deutlicher auf, was Usable Security (siehe Abschnitt 3.5) so schwer macht und welches Hindernis überbrückt werden muss.

3.4 Was ist der Unterschied zwischen Security und Safety

Im Deutschen lassen sich beide Worte mit dem Wort Sicherheit übersetzten. Semantisch haben jedoch die Worte „Security“ und „Safety“ eine differente Bedeutung, welche durch die deutsche Übersetzung verloren geht.

Das Oxford Dictionary definiert „Safefty“ mit:

> „The condition of being protected from or unlikely to cause danger, risk or injury. Denoting something designed to prevent injury or damage, e.g. safety barrier“ [Oxfa]

„Security“ hingegen definiert das Oxford Dictionary als

> „the state of being free from danger or threat, e.g. the system is designed to provide maximum security against toxic spills“ [Oxfc]

Oder auch das Wort „Secure“:

> „Certain to remain safe and unthreatened. Protected against attack or other criminal activity. Feeling confident and free from fear or anxiety “ [Oxfb]

Anhand der Definition ist der Unterschied nicht außerordentlich groß. In beiden Fällen geht es um Schutz.

Im Fall von „Safety“ geht es jedoch um eine Form von geschützt sein.
Im Fall von „Security“ geht es um dem Schutz vor Gefahren.

Anders als bei „Safety“ können die Gefahren bei „Security“ nicht leicht gesehen werden [Alb03].
„Safety“ ist also die Prävention vor zufälligen Vorfällen. „Security“ ist der Schutz vor beabsichtigten, meist geplanten Vorfällen [Alb03].

Weil ich mich semantisch in dieser Abschlussarbeit auf den Sinn der beabsichtigten Attacken mit dem Wort „Sicherheit“ beziehe, werde ich daher das englische Wort „Security“ verwenden.

3.5 Was ist Usable Security

Usable Security [GL14, S. 4] ist das interdisziplinäre Vermischen von Methoden der Human-Computer Interaktion, mit den Methoden der Information Security und Privacy. Es ist oft einfacher in einzelnen Bereichen als Spezialist tätig zu sein, als zu versuchen diese beiden Bereichen zu miteinander zu vereinen.
Denn es kann dazu kommen, dass die Benutzbarkeit eines Produktes mit dessen Sicherheit aneinander gerät. Das meint nicht, dass sie unmittelbar im Widerspruch zueinander stehen, jedoch in einer komplexen Weise miteinander agieren bzw. interagieren.
Denn ein leicht nutzbares System muss nicht sicher sein, so wie auch ein sehr sicheres System nicht mehr benutzbar sein kann.

Ebenso ist es leichter etwas über die Benutzbarkeit eines Produktes von einem Benutzer zu erfahren, als darüber wie sicher das Produkt wirklich ist.

Unabhängig von der individuellen Erfahrung des Nutzers, hat die Security immer auch einen Einfluss darauf, wie die Technologie genutzt wird oder ob denn überhaupt.

In einer Studie der Firma Sun Microsystems Inc.[28] [GL14, S. 5] sollte herausgefunden werden, warum Benutzer das single-sign-on System Open ID nicht nutzen wollten.

[28] wurde 2010 von Oracle übernommen, https://www.oracle.com/de/sun/index.html

Das Ergebnis war, dass viele Teilnehmer es als unsicherer empfanden, als das Eintippen eines Passwortes. Vertrauen spielt demnach auch ein sehr große Rolle. Weil aber die meisten Benutzer nicht richtig in der Lage sind präzise sagen zu könne, wo das Security Problem liegt [GL14, S. 5], erschwert dies eine substanzielle Auswertung enorm.

Den Grundstein legte hier 1969 die Veröffentlichung des Paper „ User-Centered-Security“ [ZS96] von Zurko und Simon. In dem Paper legten sie drei Faktoren fest, für ihre benutzerfreundliche Security:

> „applying usability Testing and techniques to secure systems; developing security models and mechanisch for user-friendly systems; and cosidering user needs as primary design Goal at the start of secure system development“ [ZS96].

Sie begründete dies wie folgt:

> „users will not purchase or use security products they cannot unterstand“ [GL14, S. 14].

Um dann diese verständlichen Systeme zu bauen, haben sie diese drei Faktoren bereits im Design des Produktes berücksichtigt, anstatt am Ende, nachdem das Produkt fertig war.

Damit Benutzerfreundlichkeit und Security miteinander vereinbart werden kann, wurde auf den folgenden Beobachtungen aufgebaut [GL14, S.87]:

- Benutzer legen keinen Fokus auf die Security . Sie wollen das System nutzen, um ihre jeweilige Aufgabe zu erledigen. Die Security ist üblicherweise zweitrangig.
- Der Benutzer ist die schwächste Stelle der Securitystrategie eines Projektes. Es ist daher unerheblich, wie gut das Verschlüsselungsverfahren ist, wenn dann der Benutzer doch das Passwort auf einen Zettel neben seinen Rechner notiert.
- Es ist wahnsinnig schwer sich wieder von Security Vorfällen zu erholen

Garfinkel und Lipford haben daher folgende Empfehlungen ausgesprochen:

- Reduziere möglichst die Entscheidungen, welche ein Benutzer innerhalb einer Anwendung anwenden kann. Es sollte den Benutzern nur möglich sein bestimmte Security Defaults zu ändern, nicht jedoch die Security Einstellungen insgesamt. Die einzelnen Defaults auch nur dann, wenn dies unbedingt nötig ist, um ihre Operationen ausführen zu können. Ansonsten könnte es passieren, dass die Benutzer von

einer Social Enginering Attacke[29] betroffen sind, welche sie dazu bringen könnten, ihre Security Einstellungen zu schwächen. Dafür müssen jedoch die gesetzten Defaults des Programms sicher sein. [GL14, S. 87]

- Benutzer sollten mit den besten Informationen versorgt werden und nicht mit den maximal möglichen Informationen. Um Security relevante Entscheidungen treffen zu können, muss der Benutzer diese verstehen und das spätestens, wenn er diese Entscheidungen treffen muss. Dafür ist auch der Kontext relevant. Entscheidungen, welche im Vorfeld getroffen werden sollen, obwohl diese noch nicht relevant sind, werden vom Benutzer anders empfunden, als diejenigen, welche im Kontext nun relevant sind. Ein Beispiel: ein Benutzer mag ein anderes Gefühl zu seiner Privatsphäre haben, wenn er seinen aktuellen Standort einfach so preisgeben soll oder wenn er ihn braucht, weil er ein Restaurant finden will). [GL14, S. 88, S. 89]

- Die Informationen, welche man den Benutzern zukommen lassen will, müssen in einer Art verfasst werden, in welcher so viele Benutzer wie möglich deren Inhalt erfassen können. [GL14, S. 90]

3.6 User Centered Design

Der Fokus von User Centered Design liegt darauf, interaktive Produkte in der Art zu formen, dass sie eine hohe Benutzerfreundlichkeit (siehe Kapitel 3.7) aufweisen. Don Norman [Nor02] hat hierzu sechs Design Prinzipien ausgearbeitet und auch begründet, was sie für die heutigen digitalen Produkte absolut notwendig macht.

- **Affordance:** Ein Objekt allein zeigt anhand seines Aussehens auf, wie es genutzt werden kann. (z.B eine Tür, welche einem aufzeigt, ob sie nach inne oder nach außen öffnet.)

- **Consistency:** Die gleiche Aktion hat immer die gleiche Reaktion auszuüben

- **Constraints:** Es sollen die Menge der Interaktionen, welche möglich sind begrenzt werden, um das Interface zu vereinfachen. Damit wird der Fokus auf das Wesentliche gerichtet

- **Feedback:** Jede Aktion benötigt auch immer eine Reaktion. Der Benutzer benötigt eine Rückmeldung, dass seine Aktion etwas bewirkt hat, das schließt auch visuelles, spürbares so wie hörbares Feedback ein.

[29] Sicherheitstechnisch relevante Daten werden durch Ausnutzung menschlicher Komponenten in Erfahrung gebracht [Ins]

- **Mapping:** Die Bezug zwischen der Aktion und dem resultierenden Effekt muss deutlich sein. Dabei soll es so selbst erklärend wie möglich sein.
- **Visability:** Dem Benutzer muss sichtbar gemacht werden, welche Bedienungsoptionen zur Verfügung stehen.

3.7 Was ist Benutzbarkeit

Benutzbarkeit oder auch Usability wird laut ISO Norm 9241-11 [fS] als

> „Extent to which a product can be used by specified users to achieve specified goals with effectiveness, effciency and satisfaction in a specified context of use. “

definiert. Sinngemäß übersetzt bedeutet dies, ob das Produkt von einem speziellen Benutzer so genutzt werden kann, dass ein bestimmtes Ziel effektiv, effizient und mit Zufriedenheit in einem festen Kontext erreicht wird. Trifft dies zu, so weist das Produkt eine hohe Usability auf [RF13, S. 4].
Das Ziel effektiv zu erreichen meint zum Beispiel, ob ein Benutzer überhaupt in der Lage ist seine Aufgaben zu erledigen und sein gewünschtes Ziel zu erreichen.
Die Effizienz wird oft in Zeit gemessen. Sie besagt wie viel Aufwand der Nutzer benötigt hat, um seine Aufgabe zu erledigen.
Die Zufriedenheit hält fest, was der Nutzer über die Benutzerfreundlichkeit denkt .

Einen anderen Ansatz verfolgte Shneiderman [GL14], in dem er sehr gute Benutzbarkeit unter Berücksichtigung folgender Kriterien beschrieben hat:

- **Die Lernfähigkeit:** Die Zeit für einen typischen Benutzer, um die wichtigsten Aktionen zu lernen, um seine Aufgabe zu erledigen.
- **Effizienz:** Die Zeit zur Erfüllung einer typischen Aufgabe.
- **Fehler:** Die Anzahl Fehler, welche ein Benutzer macht, während er eine Aufgabe ausführt.
- **Einprägsamkeit:** Die Methodik, wie ein Benutzer sein Wissen über das System über eine längere Zeit hinweg behalten kann
- **Subjektive Zufriedenheit:** Wie gut dem Benutzer die verschiedenen Aspekte des Systems gefallen.

Für die Benutzerfreundlichkeit darf nicht nur die Oberfläche betrachtet werden. Denn:

> „Usabilty steht dafür, wie gut Benutzer ein Werkzeug in ihrem Umfeld zur Bewältigung ihrer Aufgaben einsetzen können. “ [RF13, S.5]

Dies ist nicht nur eine allein stehende Eigenschaft. Die Benutzerfreundlichkeit eines Systems kann nur unter der Berücksichtigung von vier Bestandteilen (siehe Abb. 7) erreicht werden [RF13]:

- 1. Dem Benutzer
- 2. Das Werkzeug
- 3. Das Umfeld
- 4. Die Aufgabe

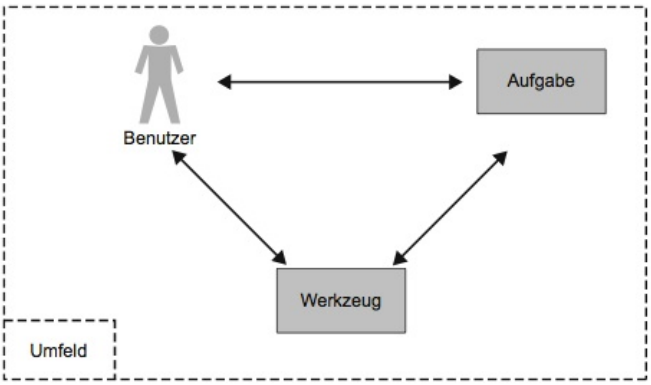

Abbildung 7: Die Komponenten eines Mensch-Maschinen-Systems [RF13, S.5]

Bereits die ISO Definition [fS] macht klar, was über die Effektivität, die Effizient und die Zufriedenheit hinaus noch zu beachten ist.

- „Specified users“ verrät einem, dass man wissen muss, wer der Benutzer des Produktes ist.
- „to achieve specified goals“ zeigt einem auf, dass beachtet werden muss, welches Ziel der Benutzer überhaupt erreichen möchte.
- „ effectiveness, effciency and satisfaction“ stellt dar, dass auch das Vorwissen der Person und dessen kognitiven Fähigkeiten berücksichtigt werden müssen. Denn ohne Berücksichtigung dessen, kann der Benutzer eines Produktes auch nicht zufrieden sein.

Es existiert keine feste Metrik, die anhand von immer wieder gleich reproduzierbaren Zahlen die Benutzbarkeit eines Systems belegen kann, um dann verschiedene Systeme miteinander vergleichen zu können [GL14, S.9].
Man muss sich demnach immer wieder damit auseinander setzen, um präzise den jeweiligen Benutzer, dessen Bedürfnisse und die Aufgabe zu verstehen.
Die Beachtung der Benutzerfreundlichkeit ist immer dort von Bedeutung

> „wo Benutzer mit interaktiven technischen Systemen zu tun haben und damit in irgendeiner Form eine Benutzerschnittstelle zum Einsatz kommt." [RF13, S.8]

Wie auch die Security, ist die Benutzerfreundlichkeit keine funktionale Anforderung, jedoch sollte sie eher während der Entwicklung eines Produktes beachtet werden, als am Ende, um entsprechende Konzepte der Umsetzung bereits zum Anfang zu realisieren. [GL14, S.9]

Im Ablauf des Usability Engineering, wird das Produkt konzipiert, getestet und bei nicht Erfüllung der Anforderungen erneut gestaltet. [RF13]

Für weitergehende Informationen zu diesem Thema, empfehle ich die Lektüre meiner Ausführungen aus meiner Bachelorarbeit „Untersuchung der Usability eines mobilen Prototypen für die Online-Ausweisfunktion" [Kos16], hier im Besonderen Kapitel 3.2 bis 3.5

3.8 Auswertung der Benutzerfreundlichkeit

Die Benutzerfreundlichkeit oder die Usability kann mit unterschiedlichen Untersuchungsmethoden ausgewertet werden.

Die Einteilung erfolgt in verschiedenen Kategorien, wobei diese nicht immer deutlich getrennt werden können, weil einzelne Methoden in differenten Varianten für unterschiedliche Zwecke verwendet werden können. [Bur03]

Es gibt eine Einteilung in formativer und summativer Evaluation, sowie in expertenbasierte und nutzerbasierte Verfahren.

Die Unterscheidung zwischen formativer und summativer Evaluation findet über den Zeitpunkt der Auswertung des Systems statt [Lab].

Eine formative Evaluation [Lab] findet sehr früh statt. Dabei können bereits Prototypen, welche z.B. aus Papier bestehen ausgewertet werden. Hierfür werden zum Beispiel die Beobachtungen und Beschreibungen der Personen festgehalten. Das Ziel ist es dadurch so viele Probleme wie möglich aufzudecken, um das Produkt zu verbessern [RF13, S.80].
Eine Beispiel Methodik wäre der Usability Test (siehe Kapitel 3.9).

Die summative Evaluationen [Lab] findet statt, wenn die Entwicklung des Produktes bereits beendet wurde. Das Ziel ist es eine Qualitätskontrolle als Gesamteinschätzung zu erhalten [RF13, S.80]. Es werden daher quantitative Daten erhoben, wie die Anzahl Fehler, welche eine Testperson gemacht hat oder wie viel Zeit die Person zur Bewältigung einer Aufgabe benötigt hat. Anders als bei der formativen Evaluation kann hier objektiv eine Messung vorgenommen werden.

Die Unterscheidung zwischen empirischer und expertenbasierter Methode findet über die Person statt, welche das Gutachten erstellt.

Bei der empirischen oder auch benutzerorientierte Methode [Lab] wird das zu untersuchende Produkt dem Benutzer vorgesetzt und seine Interaktionen mit diesem beobachtet. Anhand dieser Beobachtungen ist es möglich, sowohl die Stärken als auch die Schwächen des Produktes zu erfassen.
Zur Auswertung können ebenfalls Usability Tests (siehe Kapitel 3.9) verwendet werden.

Bei der expertenorientierten oder auch analytischen Methode [Lab] wird das Produkt von Experten getestet. Diese untersuchen das Produkt anhand von gegebenen Richtlinien oder ihrer eigenen fachlichen Expertise.
Ein mögliches Verfahren wäre die heuristische Evaluation, wo ein Experte anhand bestimmter Prinzipien des Usability Engineerings überprüft, ob das User Interface diesen gleicht. Diese Prinzipien sind dann die sogenannten Heuristiken.

Nielsen [Nie94] formulierte zehn heuristische Prinzipien aus, jedoch sind diese Heuristiken sehr allgemein formuliert, wodurch die Qualität der gesamte Auswertung von der Erfahrung des Experten abhängt. Der Vorteil der expertenorientierten Methode ist allerdings, dass sowohl die Durchführung des Tests als auch die Auswertung relativ schnell erfolgen können.

3.9 Usability Test

Bei einem Usability Test [DR93] werden Testpersonen dazu gebeten direkte mit einem Produkt zu interagieren, währenddessen sie beobachtet werden. Die Äußerungen sowie die Beobachtungen der Probanden sind die Grundlage zur Erkennung der Probleme in der Interaktion mit dem Produkt sowie der Anhaltspunkt zur Verbesserung des Produktes.

Die geeignetste Variante der Beobachtung ist es demzufolge die Testperson mit Hilfe einer Videokamera aufzuzeichnen, um deren Interaktion mit dem Produkt und deren Mimik zu erfassen und besser auswerten zu können.

Damit die Ergebnisse des Tests von allen Testpersonen miteinander verglichen werden können, muss sicher gestellt werden, dass der Ablauf des Testes bei allen Probanden gleich ist.

Weil ein Test nicht wiederholt werden kann, ohne dass eine Beeinflussung der Ergebnisse stattgefunden hat, muss der Usability Test genau geplant und Vorbereitet werden. Dazu gehört es für den Test einen Raum zu verwenden, welcher ohne Ablenkung des Probanden genutzt werden kann, damit der Test für alle Testpersonen unter gleichen Bedingungen stattfinden kann. Für solche Untersuchungen gibt es sog. Usability Labors.

Bei den Testpersonen sollte bedacht werden, dass diese bereits aus der Nutzergruppe stammen, welche auch das finale Produkt benutzen werden [RF13, S. 60]. Bei der Durchführung des Test ist es wichtig dem Probanden gegenüber explizit zu erklären, dass nicht er, sondern das Produkt getestet werden soll [Sny03, S. 204], denn die Testperson soll motiviert werden das Produkt offen und ohne scheu zu testen.

Für weitergehende Informationen zu diesem Thema, empfehle ich die Lektüre meiner Ausführungen aus meiner Bachelorarbeit 'Untersuchung der Usability eines mobilen Prototypen für die Online-Ausweisfunktion" [Kos16], hier im Besonderen Kapitel 3.4.

Beschluss

Für diese Abschlussarbeit habe ich mich dafür entschieden einen formative Evaluation zu erheben. Zum einen ist das entwickelte Werkzeug zur Codeanalyse als Prototyp konzipiert und damit als Produkt noch nicht abgeschlossen und zum anderen sollten qualitative Daten erhoben werden, um das Produkt zu verbessern.
Ich habe mit Hilfe der Thinking Aloud Methode [vSYFBJAS94] eine Usability Studie durchgeführt. Die Probanden wurden dabei beobachtet, während sie mit dem Produkt interagierten und verschiedene Aufgaben erfüllen sollten. Dabei wurde er darum gebeten laut zu Sprechen und dabei seine Gedanken, Meinungen und Empfehlungen zu äußern [vSYFBJAS94, Kapitel 4.3].
Es bot sich mir so die Möglichkeit jeden Schritt der Testperson gut beobachten zu können und ggf. die Probleme mit der Interaktion des Werkzeugs zu erkennen, bzw. herauszufinden, wo das Konzept falsch verstanden wurde.

Das Verfahren die eigenen Gedanken immer laut mitzuteilen, war für die Probanden zwar recht ungewohnt und ich musste sie auch häufiger dann

erinnern dies zu tun, aber ich beeinflusste die Probanden nicht hinsichtlich der von mir aufgebauten Fragestellungen. Das Ergebnis wurde daher auch nicht verfälscht.

3.10 Was sind die Trust Guidelines

Damit die Testperson nicht nur zufrieden mit der Nutzung des Werkzeugs zur Codeanalyse sind, sondern ihm vertrauen und seiner Auswertung vertrauen, habe ich bei der Gestaltung, soweit es mir möglich war, mich an die Trust Guidlines von Marsh und Briggs [Mar] orientiert.

Sie haben 15 Kriterien ausgearbeitet:

1. Sorge für gute Benutzerfreundlichkeit.
2. Verwende ein attraktives Design.
3. Erzeuge den Eindruck von Professionalität. Vermeide Rechtschreibfehler oder andere einfache Fehler.
4. Der Inhalt und Werbung sollte voneinander getrennt werden. Der Benutzer soll nicht das Gefühl bekommen, dass ihm etwas eingeredet oder aufgezwungen wird.
5. Mache dich selbst als Anbieter des Produktes erreichbar.
6. Verwenden hohe Konsistenz und Vorhersagbarkeit der Aktionen, sowohl in der Bedienung als auch in der Visualisierung
7. Verwende Gütesiegel und/oder Zertifikate.
8. Biete Erklärungen zu den Funktionen an.
9. Füge Referenzen von anderen Benutzern hinzu, um eine Community zu erschaffen.
10. Stelle klar definierte Sicherheits- und Datenschutzerklärungen bereit.
11. Füge Links zu anderen Webseiten hinzu, um Vertrauen aufzubauen.
12. Füge Hintergrundinformationen zu, um den Kunden ein Gesicht zu geben.
13. Mache die Verantwortlichkeiten deutlich. Der Kunde muss klar verstehen, was von ihm erwartet wird und was nicht
14. Die Kommunikation muss zwischen Anbieter und Kunden offen sein.
15. Biete einen personalisierten Kundenservice an, welcher sich um die Bedürfnisse des Kunden kümmert.

3.11 Welche Sicherheitskriterien gibt es

Innerhalb von Software stellt eine Sicherheitslücke [Gro, §Vulnerability] eine Schwachstelle dar, welche von einem Angreifer ausgenutzt werden kann, um unautorisierte Handlungen ausüben zu können, darunter fallen z.B der Diebstahl von Daten oder das Zerstören der Softwarearchitektur.

Die CWE[30] [CWEd] enthält eine Liste an Schwachstellen, welche von MITRE[31] für das NCSD[32] gepflegt wird und beschreibt diese Schwachstellen.

Um eine bessere Übersicht zu erhalten und nicht von der Menge der Liste erschlagen zu werden, habe ich im Rahmen dieser Abschlussarbeit die Schwachstellen grob in fünf Kategorien eingeteilt.

- Die erste Kategorie stellt Schwachstellen dar die den *Access Control* [GC] bzw. die Zugriffskontrolle betreffen. Dies bezeichnet den Prozess, welcher beschreibt, wem erlaubt ist was zu tun. Das reicht vom Kontrollieren eines Zugangs, um z.B. zu bestimmen wer auf welche Dateien Zugriff hat, bis dahin welche Funktionsberechtigung sie zu ihr haben (beispielsweise nur lesen).
- Die zweite Kategorie sind Schwachstellen, welche die *Memory Safety* betreffen, also den Schutz des Speichers. Dazu gehören Fehler wie:
  - Buffer Overflow [CWEa], also das Schreiben von Daten über den Puffer hinaus
  - Null Pointer Dereferenz [CWEb] (Dereferenzieren eines ungültigen Zeigers oder eines Zeigers auf Speicher, der nicht alluziert wurde
  - Gebrauch von uninitialisiertem Speicher [CWEc] (Daten werden aus dem Puffer gelesen, der zwar zugewiesen, aber nicht mit Anfangswerten gefüllt wurde).
- Die dritte Kategorie sind die *Race Conditions* [GC]. Eine Race Condition liegt vor, wenn Änderungen in der Reihenfolge von zwei oder mehr Ereignissen eine Änderung im Verhalten des Programms verursachen können. Wenn ein Angreifer die Situation zum Einfügen von bösartigen Code nutzen kann, das Ändern von Dateinamen oder den normalen Betrieb des Programms stört, ist die Race Condition eine Sicherheitslücke. Angreifer können manchmal die kleinen Zeitlücken

[30] Common Weakness Enumeration
[31] https://www.mitre.org
[32] National Cyber Security Devision

zur Verarbeitung von Code nutzen, um die Reihenfolge von Operationen zu stören. Ein Beispiel dafür sind SQL-Injections. Eine SQL-Injection [OWAc] tritt ein, wenn Benutzereingaben in eine SQL Eingabe eingebettet werden können. Damit kann der Angreifer die SQL Syntax ändern und das Ziel oder das Ergebnis verändern. Das kann zum Aufruf sensibler Informationen aus der Datenbank führen oder zu einem Angriff gegen den zugrunde liegenden Web-Server mit SQL-Datei-Operationen.

- Die vierte Kategorie sind *Unvalidated Inputs* [OWAd], also Eingaben, welche nicht geprüft wurden. In der Regel sollte man alle Eingaben des Programms überprüfen, um sicher zu gehen, dass die Daten begründet sind. Jede Eingabe, welche das Programm von einer nicht vertrauenswürdigen Quelle erhält, ist ein potenzieller Angriff. Beispiele von Eingaben einer nicht vertrauenswürdigen Quelle sind:
  - Befehle mittels eines URL, um ein Programm zu starten (Cross Site Scripting [OWAa])
  - Command Line Eingaben
  - Alle Daten, welche von einem nicht vertrauenswürdigem Server gelesen werden
- Die letzte Kategorie ist die *Schwachstelle in der Authentifikation* [GC]. Eine schwache Authentifikation beschreibt jedes Szenario, wo die Stärke des Authentifikation relativ schwach ist, im Vergleich zum Schutz. Es beschreibt auch ein Szenario, wo der Authentifikationsmechanismus fehlerhaft oder verwundbar ist. Die Authentifikation kann nicht ausreichend sein, wenn schwache Passwörter genutzt oder schlecht geschützt werden.

3.12 Wie wird ein Fragebogen aufgebaut

Die Erstellung eines Fragebogens sollte nicht unterschätzt werden und ist kein simples Unterfangen. Denn

> „Fragen im Fragebogen und damit die Antworten darauf sind [. . .] nicht nur kontextabhängig. Vielmehr sind die Formulierungen von Fragen bis ins kleinste Detail entscheidend wichtig für die Antworten, die wir auf sie erhalten werden.“ [Por14, S.13]

Anderes ausgedrückt; macht man sich keine großen Gedanken über die Formulierungen der Fragen, erhält man nicht die Antworten, welche man eigentlich erhalten möchte.

Zur Gestaltung eines Fragebogens formte Rolf Prost folgende 10 Regel [Por14, S.95] aus:

1. Du sollst einfache, unzweideutige Begriffe verwenden, die von allen Befragten in gleicher Weise verstanden werden!
2. Du sollst lange und komplexe Fragen vermeiden!
3. Du sollst hypothetische Fragen vermeiden!
4. Du sollst doppelte Stimuli und Verneinungen vermeiden
5. Du sollst Unterstellungen und suggestive Fragen vermeiden!
6. Du sollst Fragen vermeiden, die auf Informationen abzielen, über die viele Befragte mutmaßlich nicht verfügen!
7. Du sollst Fragen mit eindeutigem zeitlichen Bezug verwenden!
8. Du sollst Antwortkategorien verwenden, die erschöpfend und disjunkt (überschneidungsfrei) sind!
9. Du sollst sicherstellen, dass der Kontext einer Frage sich nicht (unkontrolliert) auf deren Beantwortung auswirkt!
10. Du sollst unklare Begriffe definieren!

Zusätzlich sollte laut Rolf Porst jeder Fragebogen folgende Elemente erhalten [Por14, S.36]

- Thematik des Fragebogens soll in einer kurzen Einleitung präsentiert werden
- Das Institut, welches die Umfrage durchführt soll vorgestellt werden
- Es soll auf die Wahrung der Anonymität sowie die Einhaltung der Datenschutzrichtlinien hingewiesen werden
- Anschließend folgt der eigentliche Frageblock
- Am Ende der Umfrage soll man sich noch den Teilnehmer für ihre Beteiligung an der Umfrage danken

Um diese Umfrageteilnehmer aus der Menge raus zu filtern, welche die Umfrage nach dem Zufallsprinzip oder vorsätzlich falsch ausgefüllt haben, sollten in einem Fragebogen sog. Kontrollfragen [Usa] eingefügt werden. Ein gängiges Verfahren ist es die gleiche Frage different, jedoch semantisch identisch, erneut zu stellen. Sollte man bei der Auswertung feststellen, dass der Umfrageteilnehmer diese beiden Fragen komplett different beantwortet hat, liegt es nahe, dass er die Frage entweder nach dem Zufallsprinzip beantwortet hat oder mit Absicht falsch, womit die Antworten für die Auswertung nicht mehr relevant sind.

Wenn alle Fragen ausformuliert sind unter Beachtung der Regelung und sämtliche Elemente des Fragebogens ausgearbeitet sind, kann mit einem sog. Pretest begonnen werden.

> „ Unter einem Pretest versteht man -ganz allgemein gesagt- die Testung und Evaluation eines Fragebogens oder einzelner seiner Teile vor ihrem Einsatz in der Haupterhebung“ [Por14, 190]

Es entspricht demnach einem Testlauf des Fragebogens, um ggf. noch unvorhersehbare Probleme zu erkennen und zu beseitigen. Nach Durchführung des Pretest und Verbesserung des Fragebogens kann mit der eigentlichen Umfrage begonnen werden, wobei die Ergebnisse des Pretest nicht in die Auswertung der Fragebögen selbst fallen.

Für weitergehende Informationen zu diesem Thema, empfehle ich die Lektüre meiner Ausführungen aus meiner Bachelorarbeit „Untersuchung der Usability eines mobilen Prototypen für die Online-Ausweisfunktion“ [Kos16], hier im Besonderen Kapitel 3.8.

Im Rahmen diese Abschlussarbeit wurde ein Fragebogen mit der Arbeitsgruppe Identity Management an der FU-Berlin entwickelt, bei dessen Erstellung die o.g Hinweise beachtet und eingehalten wurden. Ich habe sowohl geschlossene Fragen [Por14, 29] (Fragen mit vorgegeben Antwortmöglichkeiten), als auch offenen Fragen [Por14, 29] verwendet (Fragen, bei denen die Antworten selber ausformuliert werden müssen).
Die verwendeten Bewertungsskalen (siehe Abb. 8) wurden dem Fragebogen nach ISO 9241 /110 [PA] nachempfunden.

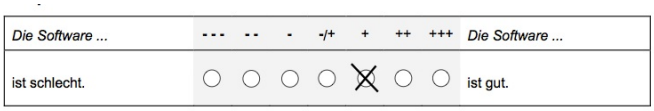

Abbildung 8: Bewertungsskala [PA]

3.13 Grundsätze der Dialoggestaltung

Die Grundsätze der Dialoggestaltung [PA] beziehen sich auf die Entwicklung von Benutzerschnittstellen.

Mit Einhaltung dieser Grundsätze soll verhindert werden, dass der Benutzer zusätzliche unnütze Schritte zur Bewältigung seine Aufgabe ausführen muss, irreführende Informationen abgegeben werden und das System nicht präzise so auf Handlungen des Benutzers reagiert, wie es das tun sollte.

Diese Grundsätze sind in der ISO Norm 9241/110 [PA] aufgeführt und bestehen aus sieben Punkten.

3.13.1 Aufgabenangemessenheit:

Diese ist dann gut, wenn dem Benutzer vom System dabei geholfen wird seine Aufgaben zu erledigen. Das meint z.B. das entsprechende Dialogfenster angezeigt werden sollen, welche dabei unterstützen die Aufgabe zu erledigen und keine nicht überflüssigen Informationen vorhalten, welche den Benutzer in die Irre führen könnten.

3.13.2 Selbstbeschreibungsfähigkeit:

Ohne Nutzung von externen Quellen ist der Benutzer in der Lage die Dialoge und Funktionen des Systems zu verstehen und sieht welche Handlungen von ihm erwartet werden.

3.13.3 Die Erwartungskonformität:

Aus dem Nutzungskontext heraus ist vorhersehbar, wie das System auf Handlungen des Benutzers reagieren wird. Das meint z.B , dass wenn ein Benutzer ein Bedienerelement betätigt, dann soll die dazu zutreffende Rückmeldung erreichen.

3.13.4 Die Lernförderlichkeit:

Eine gute Lernföderlichkeit ist vorhanden, wenn das System dem Benutzer dabei hilft den Umgang mit diesem zu erlernen. Das schließt das Anzeigen von Zwischenergebnissen oder Rückmeldungen ein, damit der Benutzer versteht, was an seinen Aktionen richtig oder falsch war.

3.13.5 Steuerbarkeit:

Wenn der Benutzer eine gute Kontrolle über die Geschwindigkeit der Interaktionen hat, bis er sein Ziel erreicht, dann ist eine gute Steuerbarkeit vorhanden. Das meint, dass der Benutzer selber beschließt, wenn und wann Dialoge fortgesetzt werden und diese auch wieder aufnehmen kann, wenn sie unterbrochen wurden.

3.13.6 Fehlertoleranz:

Die Fehlertoleranz sagt aus, dass, auch wenn der Benutzer eine fehlerhafte Eingabe vorgenommen hat, das System dennoch ihm zu seinem beabsichtigten Ziel führen kann. Dazu gehört auch, dass fehlerhafte Eingaben nicht zum Absturz des gesamten Systems führen.

3.13.7 Individualisierbarkeit:

Die Individualisierbarkeit besagt, dass der Benutzer die Möglichkeit hat die Darstellung der Informationen seinen individuellen Wünschen entsprechend anzupassen. Das meint zum Beispiel die Farbgestaltung, die Schriftart oder Schriftgröße. Für die Barrierefreiheit spielt dies eine nicht zu unterschätzende Rolle.

Diese Grundsätze können anhand des Fragebogens nach ISO Norm 9241/ 110 [PA] von Benutzern beantwortet werden. Das Ergebnis zeigt einem anschließend auf, wie benutzerfreundlich das getestete System ist.

4 Umfrage

4.1 Das Ziel

Ziel dieser Umfrage war es zu erfahren, inwiefern Programmierer Werkzeuge zur Codeanalyse überhaupt nutzen, was sie zur Nutzung dieser motivierte hat, welche Erfahrungen sie dabei gemacht haben und welche Anforderungen sie an solche Werkzeuge stellen. Dafür wurde die Umfrage gemeinsam mit der Arbeitsgruppen Identity Management an der FU Berlin ausgearbeitet.

Das Ergebnis dieser Umfrage lieferte mir die Ansätze dafür, welche Funktionen das Werkzeug abdecken und welche Bedienelement es bieten sollte.

4.2 Vorstellung

Weil die Umfrage eine quantitative Studie werden sollte, auf deren Basis ich den Prototypen des Werkzeugs zur Codeanalyse entwickeln könnte, entschied ist mich dafür eine Onlineumfrage zu starten.
Mit dieser war es mir möglich gleichzeitig schnell und viele Personen zu erreichen.

Die Onlineumfrage selbst habe ich mit der Plattform Umbuzoo[33] (siehe Abb. 9) erstellt. Sie erfüllt meine Kriterien. Es wird keine Werbung auf der Umfrage angezeigt, es lassen sich leicht und schnell Umfragen erstellen die darüber hinaus optisch sehr ansprechend sowie individuell gestaltbar sind. Sie bietet eine kostenlose Variante an und ermöglicht eine grenzenlose Erstellung von Fragen sowie Erhebungen von Antworten [Umb].

Es können jedoch pro Gratis-Account nur drei verschiedene Umfragen erstellt werden.

[33] https://www.umbuzoo.de

Abbildung 9: Startbildschirm der Onlineumfrage

Da ich sowohl deutsch als auch englisch sprachige Programmierer erreichen wollte (besonders unter Berücksichtigung von internationalen Foren), konnte ich die Option nutzen, dass die Umfrageteilnehmer innerhalb der gleichen Umfrage die Fragen sowohl in Deutsch als auch in Englisch beantworten können.

4.3 Aufbau

Die Onlineumfrage war in fünf thematischen Abschnitten unterteilt.

- Der erste Abschnitt beschäftigte sich mit der Frage, ob Personen bereits selber verfassten Code auf Plattformen wie z.B. Stack Overflow hochgeladen hatten. War dem so, wurde erfragt inwiefern sie diesen Quellcode nach Sicherheitskriterien überprüft hatten. Diese Frage zielten nicht darauf ab festzustellen, inwiefern sie auf dieser Plattform aktiv sind, sondern inwiefern sie die Notwendigkeit sahen, dass ein Quellcode sicherheitsrelevant untersucht werden müsste und, wenn ja, mit welchen Mitteln sie es getan haben.
- Der zweite Abschnitt sollte herausarbeiten, inwiefern sie bereits Code aus solchen Plattformen heraus kopiert und selber verwendet haben. Dabei wurde nicht nur gefragt mit welchen Mitteln sie ggf. eine Überprüfung vorgenommen hatten, bevor sie den Quellcode nutzten, sondern auch für welchen Zweck. Hier sollte herausgefunden werden, inwiefern sie die Notwendigkeit sahen eine Überprüfung vornehmen zu

müssen und, wenn ja, für welchen Zweck sie dies taten. Hierfür wollte ich herausfinden, ob eine grundsätzliche Einstellung der Testperson zur Prüfung oder nicht Prüfung existiert oder ob den Umständen entsprechend eine Unterscheidung gemacht wird. Damit hatte ich ein klares Bild der diesbezüglichen Grundeinstellung der Person erhalten.

- Der dritte Abschnitt der Onlineumfrage sollte die Wahrnehmung und Grundeinstellung der Umfrageteilnehmer bzgl. Plattformen wie Stack Overflow erfassen. Dafür fragte ich sie, anhand einer Bewertungsskala von gar nicht zu sehr, orientiert an der Skala des ISO Fragebogen 9241/110 [PA], für wie fehlerfrei sie die veröffentlichten Codes auf Stack Overflow empfinden und wie sie das Begründung. Die gleiche Skala kam zum Einsatz bei der Frage für wie sicher sie die Code auf Stack Overflow empfinden. Die Antworten auf diesen Fragen verrieten mir inwiefern sie aufgrund ihrer Grundeinstellung und dem Vertrauen welches sie dieser Plattform entgegenbringen, ein entsprechendes Verhalten bei der Prüfung des Code an den Tag legen. Beispielsweise vorausgesetzt eine Person vertraut der Plattform bzgl. deren Sicherheit im allgemeinen, dann sieht er vermutlich nicht mehr die Notwendigkeit seinen Code erneut zu überprüfen.

- Der vierte Abschnitt sollte ihre bereits gemachte Erfahrungen mit Werkzeugen zur Codeanalyse erfassen, sowie deren Anforderungen an sie. Dies zeigte mir die allgemeine Orientierung meiner Zielgruppe auf, welche Werkzeuge sie erwarten und was diese können sollen.

- Der letzte Frageblock entsprach den demographischen Fragen, um die Testpersonen besser einteilen zu können, entsprechend der Empfehlungen von Rolf Porst [Por14].

4.4 Zielgruppe

Bei der Gestaltung der Umfrage habe ich nicht nur darauf geachtet, wie ich die Fragen gestalte, sondern auch an wen sich diese Umfrage richtet. Dabei war mein Fokus ausschließlich auf Personen gerichtet, welche Programmiererfahrung haben. Die konkrete Frage nach der Anzahl Jahren an Programmiererfahrung bot mir die Möglichkeit die Umfrageteilnehmer ohne Programmiererfahrung aus den Ergebnissen heraus zu filtern.

Ich richtete mich ausschließlich an Personen mit Programmiererfahrung, weil nur diese sich in dem Milieu der Softwareentwicklung bewegen und sie mir daher gezielt sagen können, welche Anforderung sie an ein Werkzeug haben, welches sie ggf. nutzen wollen.

4.5 Kontrollfragen

Zusätzlich wurden sog. Kontrollfragen (siehe Kapitel 3.12) eingebaut, um die Umfrageteilnehmer heraus zu filtern, welche wahllos nach dem Zufallsprinzip die Onlineumfrage ausgefüllt hatten.

Diese Kontrollfragen versteckten sich unter anderem in der Frage: „Wie häufig haben Sie Code aus Stack Overflow (oder ähnlichen Plattformen) für welchen Zweck verwendet?“
Sollte eine Person angekreuzt haben, dass sie nie aus Stack Overflow Code kopiert und selber genutzt hat und anschließend angegeben, dass sie Plattform so und so oft für den jeweiligen Zweck benutzt hat, wäre dies ein Widerspruch und es muss damit eine Wahllosigkeit des Umfrageteilnehmers unterstellt werden.

Das Gleiche ist bei folgender Frage zu beobachten: „Wie ist Ihre Erfahrung im Umgang mit Codeanalyse Werkzeugen?“
Indem eine Person behauptet keine Erfahrungen mit Werkzeugen gemacht zu haben, zuvor jedoch behauptete, dass der kopierte Code mit einem bestimmten Werkzeug überprüft wurde, ist auch hier wieder eine deutliche Inkonsistenz sichtbar.

Unter Berücksichtigung von vollständig ausgefüllten Fragebögen und der Überprüfung der Kontrollfragen, war es mir möglich, unbrauchbare Fragebögen herauszufiltern und nur diejenigen in die Auswertung aufzunehmen, welche für die Studie tatsächlich relevant sind.

4.6 Erreichen der Testpersonen

Insgesamt haben 198 Personen an der Onlineumfrage teilgenommen. Ich habe diese über drei verschiedene Wege erreicht:

4.6.1 Forschungsplattformen

Der erste Weg war die Nutzung von sog. Forschungsplattformen, welche damit werben, dass sie Studienleitern dabei helfen Umfrageteilnehmer zu finden.

Das Prinzip dieser Plattformen besteht auf der Basis des gegenseitigen Helfen. Angemeldete Studienleiter dieser Plattformen erreichen andere Studienleiter als ihre Umfrageteilnehmer und helfen anderen Studienleitern in der Funktion als Umfrageteilnehmern ihrer Umfragen.
Das meint: wenn ich möchte, dass meine Umfrage beantwortet wird, dann

muss ich auch die Umfragen der anderen, welche auf dieser Plattform hochgeladen wurden, beantworten.

Ich habe zwei dieser Plattformen verwendet.

Eine Plattform nennt sich Survey Circle[34]. Das Prinzip besteht aus einem Ranking der gesamten Umfragen (siehe Abb. 10), welche auf dieser Plattform hochgeladen wurden. Die Nutzung ist kostenlos. Je mehr Punkte eine Umfrage hat, desto weiter oben findet sich die Umfrage in dem Ranking. Je weiter höher im Ranking, desto eher werden Teilnehmer auf diese Umfrage aufmerksam.

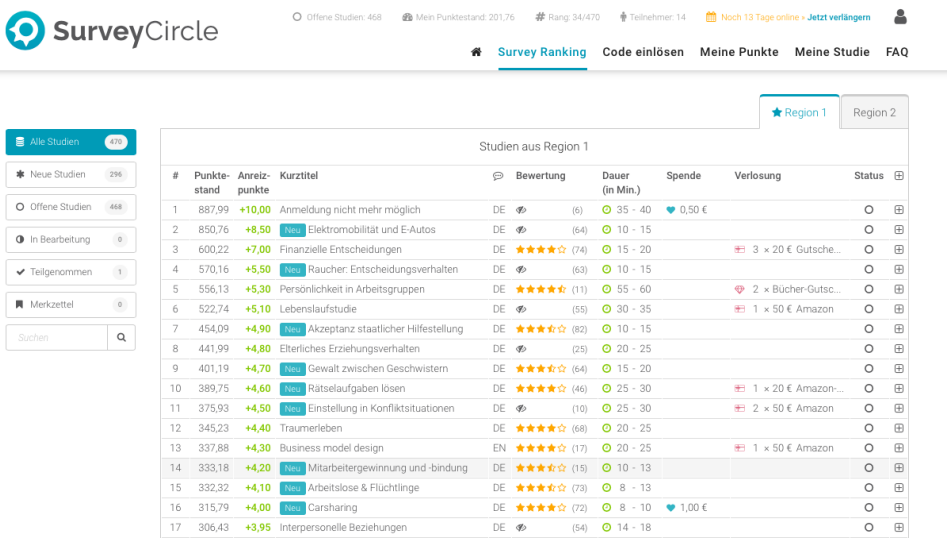

Abbildung 10: Liste der Onlineumfragen auf SurveyCircle [Cir]

Die Punkte werden durch die Beantwortung anderer Umfragen gesammelt. Je länger die Umfrage ist, desto mehr Punkte erhält man pro Umfrage. Das heißt, dass wenn man weiter oben im Ranking sein möchte, dann muss man selber auch eine adäquate Anzahl Umfragen beantworten.

Die zweite Plattform PollPool[35] funktioniert nach den selben Prinzip. Auch hier gilt das Prinzip des gegenseitigen Helfens. Auf dieser Plattform existiert jedoch kein Ranking, sondern eine Sammlung an Umfragen (siehe Abb.11). Um seine eigene Umfrage beantworten zu lassen, benötigt man Punkte sog. PollCoins (siehe Abb. 12). Diese Punkte werden durch das Beantworten anderer Umfragen auf dieser Plattform gesammelt. Mit jedem Teilnehmer, welcher die Umfrage beantwortet, werden Punkte vom eigenen Punktestand

[34] https://www.surveycircle.com
[35] https://www.poll-pool.com

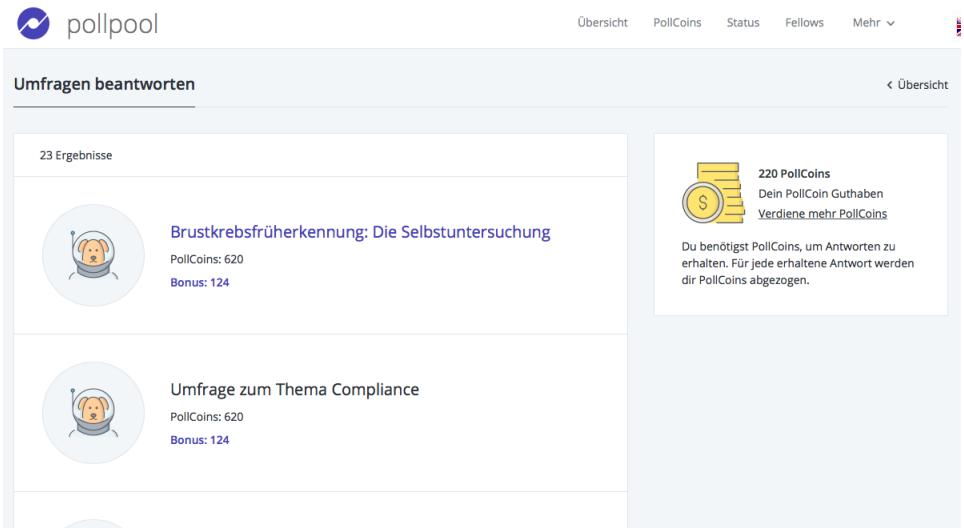

Abbildung 11: Onlineumfragen auf PollPool [Pol]

abgezogen. Wenn der Punktestand auf Null gerät, ist man gezwungen weitere Umfragen anderer Mitglieder zu beantworten oder im Store des Plattformbetreibers Punkte gegen Bezahlung zu erwerben.

Abbildung 12: PollCoins - Die Währung auf PollPool [Pol]

4.6.2 Onlineforen

Der zweite Weg war die Nutzung von Onlineforen, bei denen die Mitglieder zu meiner Nutzergruppe gehören. Dafür habe ich in den Foren Stack Overflow[36], das Heise Forum[37] und die Plattform XING[38] während Diskussionen um Codeanalyse Werkzeuge oder Security, darum gebeten meine Umfrage zu beantworten.

[36]https://stackoverflow.com
[37]https://www.heise.de/forum/startseite/
[38]https://www.xing.com

4.6.3 Mailverteiler der Universitäten

Der dritte Weg war die Nutzung der Mailverteiler der Universitäten für Informatik Studenten. Dabei begrenzte ich mich auf die Verteiler der Berliner Universitäten hier im Speziellen die Technische Universität (TU), die Humboldt Universität (HU) und die Freie Universität (FU).

Weil ich selber Studentin an der Freien Universität bin, war der Zugang zu diesem Mailverteiler einfacher, als zu denen der TU und HU. Dabei musste ich mich auf die Hilfe Dritter verlassen, welche mir gestatteten in ihren Mailverteiler auf meine Umfrage aufmerksam zu machen.
Insgesamt war der Faktor vorlesungsfreie Zeit deutlich spürbar. Als ich die Onlineumfrage terminbedingt bereits zu dieser Zeit erhob, war es mir gerade einmal möglich 25 Teilnehmer zur Beantwortung meiner Fragen zu erreichen.

Um ausschließen zu können, dass es nicht an der Qualität der Umfrage oder gar des Anschreibens lag, habe ich die Umfrage dann erneut zur Vorlesungszeit in die Mailverteiler setzen lassen. Eine Rückmeldung von 40 Teilnehmern am ersten Tag der erneut erhobenen Umfrage zeigte mir, dass diese Zeit deutlich berücksichtigt werden muss, sollte man Studenten erreicht wollen.

4.7 Ergebnis der Studie

4.7.1 Die Zahlen

Im Zeitraum eines Monats beteiligten sich 198 Teilnehmer an der Umfrage. Weil ich zum Vergleich der Daten nur diejenigen Umfragen berücksichtige, welche auch vollständig ausgefüllt wurden, hat sich die Anzahl dann auf 122 Teilnehmer reduziert. Unter Berücksichtigung meiner Nutzergruppe, wurden alle Testpersonen aus der Studie entfernt, welche keine Programmiererfahrung hatten. Damit wurden 19 weitere aus der Studie gestrichen. Zum Schluss wurden die Kontrollfragen untersucht, wodurch 5 weiter Umfrageteilnehmer aus der Studie ausgeschlossen wurden. Der Grund war in allen fünf Fällen Widersprüche in der Nutzung von Stack Overflow. Das führt zu einer Gesamtanzahl von immer noch 98 auswertbaren Umfragebögen.

Die Begründung zum Ausschluss einzelner Umfrageteilnehmer ist den Rohdaten (siehe Anhang) hinzugefügt.

Der größte Teil der Umfrageteilnehmer sind männliche Programmierer, rund 80%, im Alter zwischen 20 und 29 Jahren.

33% der Umfrageteilnehmer hat bereits selbst verfassten Code auf Plattformen wie Stack Overflow hochgeladen. Um die 40% haben diesen Code auch überprüft, bevor sie ihn auf die Plattform geladen hatten (siehe Abb.13).

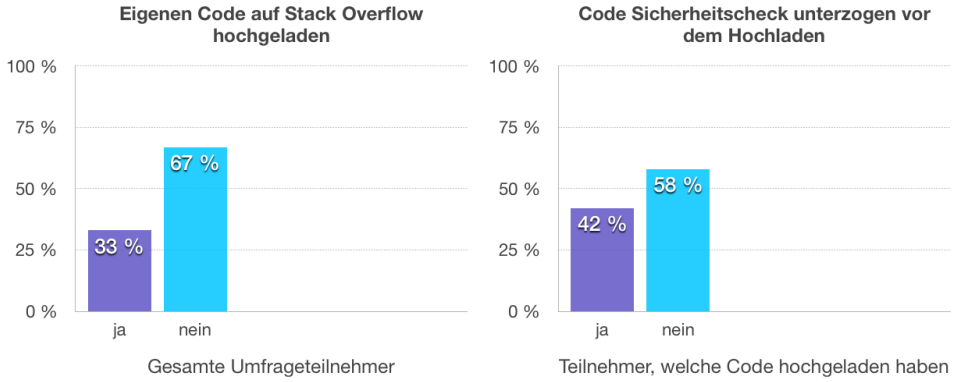

Abbildung 13: Ergebnisse zum Hochladen von Quellcode auf Stack Overflow

Doch nur 15% dieser 40% verwendeten ein Werkzeug, um diese Überprüfung vorzunehmen (siehe Abb. 14). Das entspricht gerade einmal 2% aller Umfrageteilnehmer. Der Rest überprüft den Code, indem sie sich auf ihre individuellen Erfahrungen verlaßen.

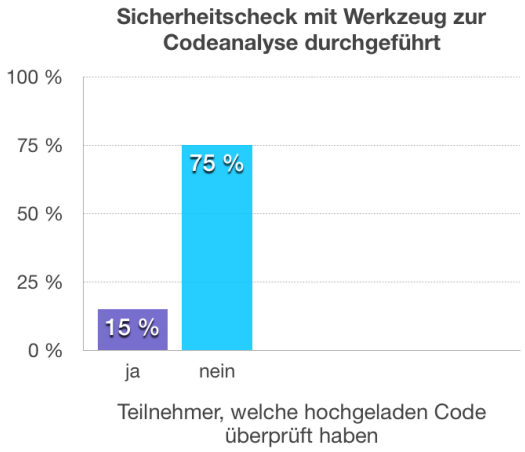

Abbildung 14: Sicherheitscheck mit einem Werkzeug durchgeführt

Unter den Begründungen wieso Umfrageteilnehmer den Code nicht getestet haben, bevor sie ihn hochgeladen hatten, fielen Aussagen wie, dass es sich nur um Codeschnipsel handelte und damit zu klein zum Überprüfen sei, es sich nicht um einen sicherheitsrelevanten Code handele oder es nur ein triviales Codebeispiel sei. Andere hingehen gaben an, dass ihnen solche Werkzeuge nicht bekannt seien, sie keine Zeit zum Prüfen gehabt hätten oder es gar nicht erst bedacht hätten.

Ganze 91% der Umfrageteilnehmer hatten bereits Code aus Stack Over-

flow heraus kopiert (siehe Abb. 15) und für den eigenen Code verwendet. Betrachtet man nur die befragten Studenten, sind es sogar 95% der Umfrageteilnehmer. 66% aller Umfrageteilnehmer (siehe Abb. 15), welche Code heraus kopiert hatten, haben den Code auch überprüft.
Hier zeigt sich klar die Tendenz von Anderen stammenden Code eher zu überprüfen, als den eigenen Code, welchen man der Community zur Verfügung stellt.

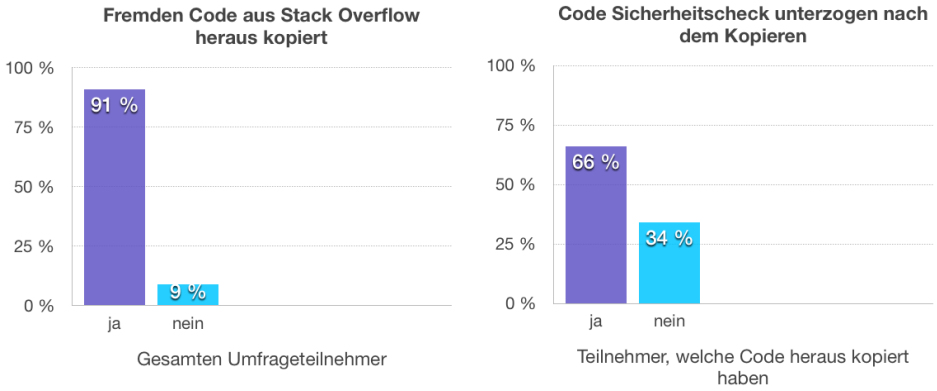

Abbildung 15: Code aus Stack Overflow kopieren und überprüfen

Doch auch hier haben nur knapp 16% (siehe Abb. 16)von diesen Teilnehmer eine Sicherheitsanalyse mit einem Werkzeug vorgenommen.

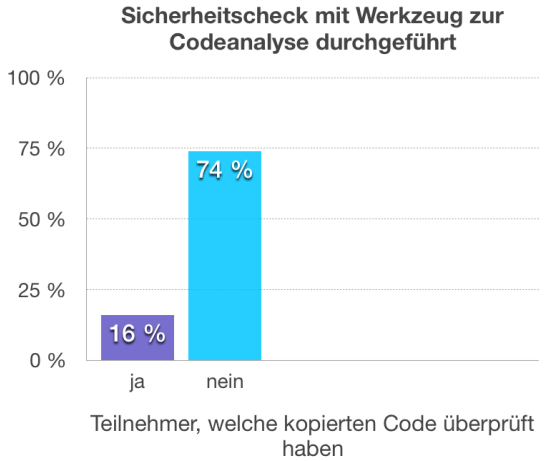

Abbildung 16: Testpersonen, welche Code kopiert und überprüft haben

Der größte Rest der Umfrageteilnehmer, hatte lediglich eine Untersuchung beruhend auf ihren eignen Kenntnissen vorgenommen. Dies ungeachtet des weithin bekannten Umstandes, dass nun wirklich jeder der codet ab und an

mal Fehler macht, oder simpel, bei der schier unüberschaubaren Menge an Einzelfaktoren sicherheitsrelevante Faktoren übersieht

Werkzeuge zur Analyse des Codes werden demnach noch sehr gering verwendet. Die Begründungen zur Nichtüberprüfung waren, wie oben beschrieben, stets recht ähnlich. Allein das Paper „Stack Overflow Considered Harmful“ (siehe Kapitel 2.2.2) zeigte jedoch deutlich auf, dass genau diese vermeintlichen kleinen harmlosen Codeschnipsel ein großes Problem beim Verursachen von Sicherheitslücken darstellen. Auch in diesem Paper wurde die mangelnde Zeit als Hauptargument angegeben. Andere hingegen teilten schlicht mit, dass sie simpel zu faul waren, um eine Überprüfung ihrer Arbeit durchzuführen.

Bei der Frage, mit welchem Werkzeug die Umfrageteilnehmer ihren Code getestet haben, habe ich eine begrenzte Menge an Werkzeugen zur Auswahl zur Verfügung gestellt. Ich wollte hierbei herausfinden, ob der Proband tatsächlich den Unterschied zwischen einem simplen Debugger und Werkzeugen zu Sicherheitsanalyse kennt.
Daher sind in dieser Liste nicht nur Werkzeuge zur Codeanalyse untergebracht.

Tatsächlich stellte sich heraus, dass in der gesamten Umfrage nur eine einzelne Person, bei der Frage mit welchen Werkzeugen er eine Sicherheitsanalyse durchgeführt hat, nur diejenigen Werkzeuge angegeben hatte, welche auch wirklich für eine Sicherheitsanalyse in Frage kommen. Die Analyse Werkzeuge waren Coverity[39], ein Werkzeug für die Sprachen C, C++ sowie Java und CPPCheck [40] ein Werkzeug für C und C++.

Zu den meist bekannten und den am meisten angegebenen „verwendeten“ Werkzeugen zur Analyse des Codes gehörten Valgrind[41], GBD Debugger[42](beides Debugger) und Visual Studio[43], welche eine integrierte Entwicklungsumgebung vom Microsoft[44] darstellt.

Dieses Ergebnis zeigte nicht nur auf, dass wenige Programmierer Werkzeuge zur Codeanalyse nach Sicherheitslücken verwenden, sondern auch, dass das Wissen über diese Werkzeuge sehr gering ist.

[39]https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/SAST-Coverity-datasheet.pdf
[40]https://sourceforge.net/projects/cppcheck/
[41]http://valgrind.org
[42] in Werkzeug zum Diagnostizieren und Auffinden von Fehlern im Quellcode
[43]https://www.visualstudio.com/de/
[44]https://www.microsoft.com/de-de

Es bot sich dabei insgesamt das Bild, dass, wenn ein Umfrageteilnehmer zu einer Prüfung seines Quellcodes bereit war, dann für jeden Code den er verfasste. Unerheblich dessen, welchem Zweck der Code dienen würde.

Die Grundeinstellung zum Thema „veröffentlichten Code auf Stack Overflow“ zeigt auf, dass die meisten Nutzer zu „+“ zwischen einem Ranking von „gar nicht“ zu „sehr“ fehlerfrei tendieren (siehe Abb. 17)

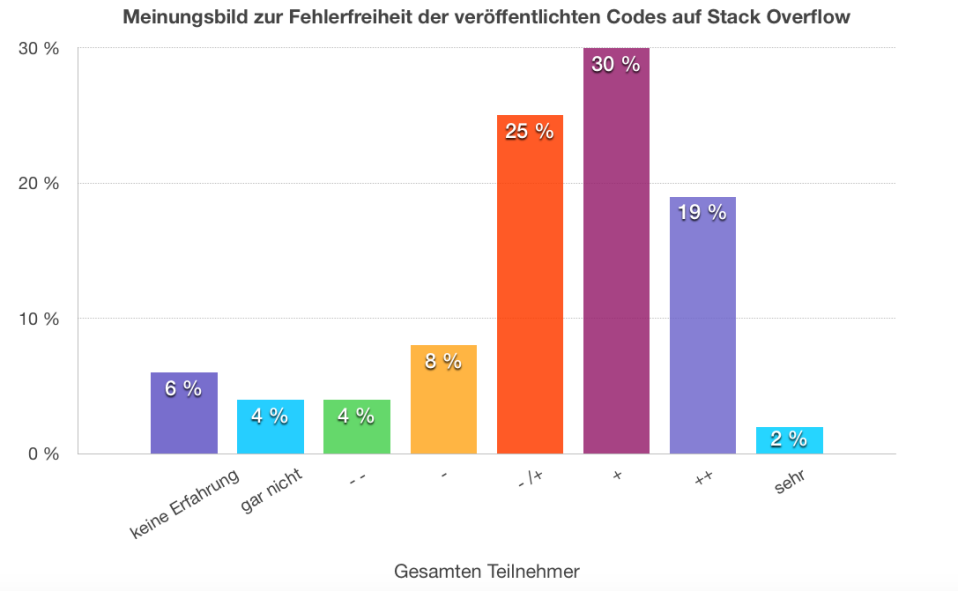

Abbildung 17: Wie fehlerfrei halten Sie den Code auf Stack Overflow?

Dazu äußerten viele Teilnehmer, dass die besagte Plattform eine große Community besitzt, weshalb man sich auf das „Viele-Augen-Prinzip“ verlassen kann und gemeinschaftlich am Code gearbeitet wird.
Es wurde zudem festgestellt, wie wichtig es ist, wer den Code veröffentlicht bzw. überarbeitet hat. Sind dies Programmierer die sich, anhand eines hohen Scores, einen guten Ruf innerhalb der Stack Overflow Community aufgebaut haben, dann kann man diesen auch vertrauen. Jedoch wurde auch bemerkt, dass im Prinzip jeder auf dieser Plattform in Lage ist Code hochzuladen und dieser nicht immer lauffähig ist.

Bzgl. des Sicherheitsgefühl der Codes und Codeschnipsel, welche auf Stack Overflow veröffentlich werden, haben die meisten Teilnehmer eine neutrale Einstellung (siehe Abb. 18). Hier war zu beobachten, dass die meisten (28%) keinerlei Tendenz aufzeigen konnten.
Auch wurde oft wieder das Vertrauen darin begründet, dass viele Mitglieder den veröffentlichen Code reviewen, was inhaltlich unterstellt, dass wieder und wieder nicht nur eine Sichtung, sondern dadurch auch eine stetige Ana-

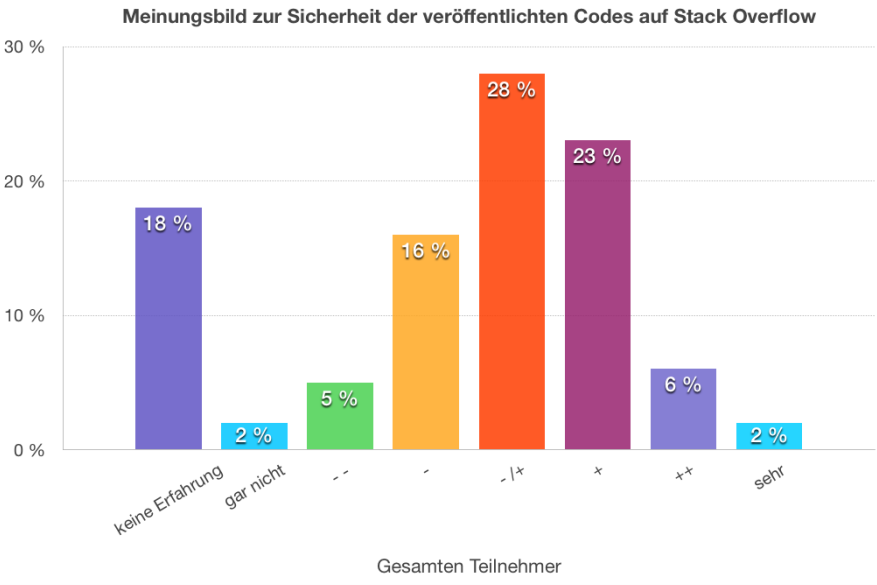

Abbildung 18: Wie sicher halten Sie den Code auf Stack Overflow?

lyse vorgenommen wird.
Es wurde allerdings auch mehrfach festgestellt, dass der Fokus der Programme nicht auf Sicherheit liegt, sondern viel eher eine Problemlösung anbietet. Letztlich wird dann wieder der Score eines Mitglieds als Vertrauensgrundlage herangezogen.

Als Gesamteindruck kann man den beiden Ergebnissen entnehmen, dass die Umfrageteilnehmer weder deutlich von der Fehlerfreiheit noch der Sicherheit der veröffentlichen Codes aus Stack Overflow überzeugt sind, aber auch nicht davon ausgehen, dass die Codes deutlich falsch und unsicher sind.

Erstaunlicher Weise zeigte die Personengruppe welche von der Sicherheit der Codes auf der Plattform sehr überzeugt waren, keine gesonderte Tendenz den Code, welchen sie aus der Plattform kopiert hatten, nicht zu testen. Tatsächlich hatten etwa gleich viele Programmierer aus dieser Gruppe den Code überprüft, wie auch nicht überprüft.

Auch bezüglich der Frage zur Untersuchung der Fehlerfreiheit war keinerlei deutliche Tendenz zu erkennen. Auch hier hatten etwa gleich viele Programmierer ihren Code nach dem Kopieren überprüft sowie nicht überprüft, dies gleichwohl sie stark davon überzeugt waren, dass die Codes und Codeschnipsel auf Stack Overflow sehr fehlerfrei sind.

4.7.2 Die Anforderungen

Die formulierten Anforderungen der Umfrageteilnehmer an ein Werkzeug zur Codeanalyse waren grob zusammengefasst:

- gefunden Fehler verständlich anzeigen
- eine einfache Bedienung
- schnelle Bedienung
- Hilfestellung zur Lösung des Problems bieten
- Beispiel zur Verbesserung angeben
- eine gute Dokumentation
- Integration in meine favorisierte IDE
- automatisierte Testen

Zudem wurden auch konkrete Lösungen zu mittlerweile Tag täglichen Problemen gewünscht, wie z.B. die Erkennung von Buffer Overflows und SQL Injektions.

Die eigentliche Herausforderung eines Werkzeug zur Erkennung von Sicherheitslücken hat Umfrageteilnehmer Nr. 83 treffend ausgedrückt:

> „Sicherheitslücken können nur mit Verstand gefunden werden. Man kann einige Pattern abfangen, aber das sind auch die, die man als Mensch i.A. am schnellsten findet. Wer denkt, dass der Code OK wäre, weil Valgrind[45] nicht gemeckert hat, hat's wirklich nicht verstanden."

4.8 Nebenerkenntnisse

Neben dem Ergebnis der eigentlichen Umfrage, war es mir noch möglich, die Strategie der Umfrage selbst auswerten zu können.

Unter der Zuhilfenahme der Mailverteiler der Universitäten war es mir möglich schnell und einfach viele Personen aus meiner Nutzergruppe zur erreichen. Mit dieser Methode hatte ich im Vergleich zu den anderen Varianten mit Abstand am meisten Teilnehmer (142) erreicht.
Leider konnte jedoch ein großer Teil der Umfrageblätter nicht ausgewertet werden. Allein 42% (siehe Abb. 19) der Umfragen mussten entfernt werden, weil die Umfrage nicht vollständig ausgefüllt wurde.

[45] ein einfacher Debugger

Das gleiche Verhalten wurde bei den Umfragen beobachtet, welche ich an die Foren gesendet hatte. Hier gestaltete es sich am schwierigsten, Personen dazu zu bewegen sich an meiner Umfrage zu beteiligen. Dort wurden mir lediglich 19 mal meine Umfrage beantwortet. Es wurden davon aber nur 11 vollständig ausgefüllt, also auch hier 43% (siehe Abb. 19) unbrauchbare Umfragen.

Die beste Quote an vollständig ausgefüllten Umfragen erhielt ich durch die Nutzung der Forschungsplattformen Survey Circle und PollPool. Ich erhielt 37 Umfragen zurück und davon wurden 28 vollständig ausgefüllt. Damit waren also 76% der Umfragen (siehe Abb. 19) nutzbar.

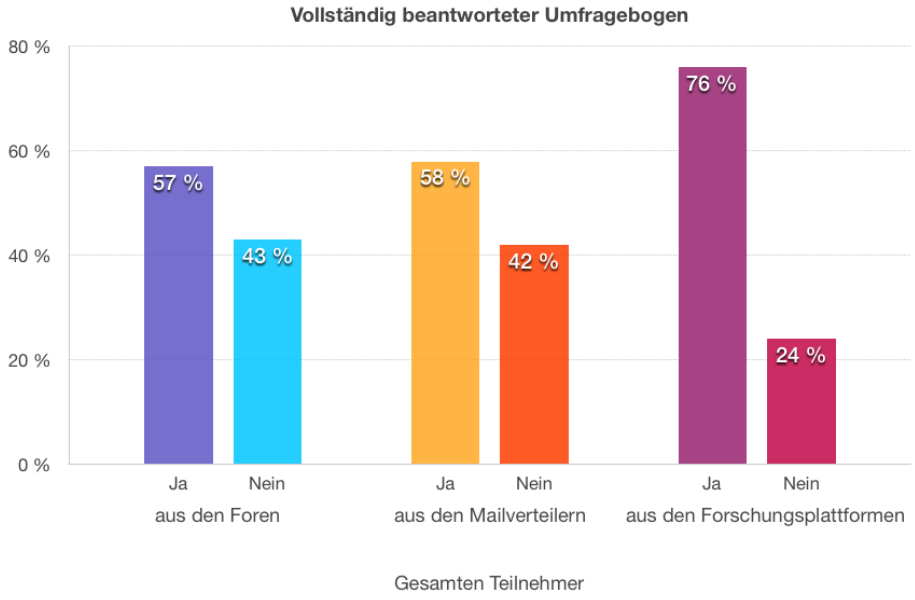

Abbildung 19: Beantwortung der Umfragen

Leider sind auch diese speziellen Forschungsplattformen nur bedingt nutzbar. Sie bieten nämlich nur begrenzt bzw. gar keine Möglichkeit gezielt Nutzergruppen zu erreichen. So kam er vermehrt dazu, dass ich Umfragen von Personen ausgefüllt bekommen habe, welche gar keine Programmierkenntnisse hatten. Weil das jedoch ein Kriterium meiner eigentlichen Nutzergruppe ist, musste ich diese Umfragebögen aus der Studie entfernen. Damit blieben mir 9 der ursprünglichen 37. Das entspricht einem Ausfall von 76% (siehe Abb. 20)

Im Ergebnis lässt sich feststellen, dass man diese speziellen Forschungs-

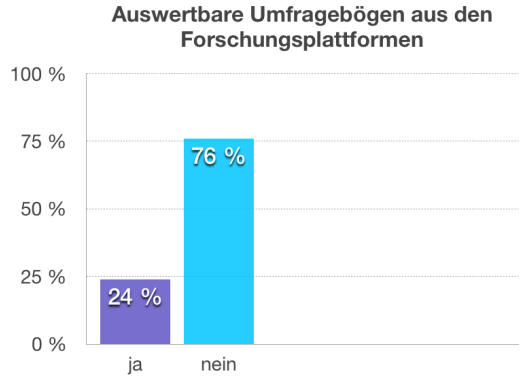

Abbildung 20: Auswertbare Umfragen aus den Forschungsplattformen

Umfrage-Plattformen nur dann sinnvoll einsetzen kann. Wer hier hoch spezialisierte Themenkomplexe anspricht, so wie in diesem Fall ich, der wird damit konfrontiert, dass es blankes Glück ist, wie viele Umfragen man von den jeweils ausgefüllten Umfragen überhaupt nutzen kann. Wenn man jedoch Umfragen möglichst allgemeiner Natur beantwortet haben möchte, mit beispielsweise Fragekomplexen aus dem täglichen Leben, der kann mit sehr hoher Gewissheit davon ausgehen, dass seine Umfragen vollständig und vernünftig ausgefüllt werden.

Sollte man Studenten erreichen wollen, liegt die naheliegender Variante immer noch bei der Nutzung der Mailverteiler, wobei dort dringend drauf geachtet werden muss, dass dies zur Vorlesungszeiten geschieht.

Die Strategie und das Erreichen von Mitglieder div. Foren auf gewerblichen Plattformen ist nur sehr bedingt aufgegangen. Ohne z.B. den Zugang eines guten Bekannten zur Karriere und Netzwerkplattform Xing[46] wäre ich dort vermutlich an gar keine Testpersonen geraten.

Selbst bei Plattformen, wie Stack Overflow und Stack Exchange[47] trugen meine Bemühungen kaum Früchte. Dort wurden meine Beiträge zur Teilnahme an meiner Umfrage direkt aus dem Verzeichnis gelöscht. Beim ersten Versuche auf der Stack Overflow Plattform, wurde ich über das Löschen meines Eintrags zumindest noch in Kenntnis gesetzt. Auf der Plattform Stack Exchange wurde über die Löschung meiner Einträge nicht einmal mehr informiert

[46] https://www.xing.com
[47] https://stackexchange.com

5 Das Werkzeug

5.1 Überlegungen und Konzept

Nachdem ich nun den Bedarf an Werkzeugen zur Codeanalyse, welche benutzerfreundlich sind, die Nutzergruppe, die Anforderungen an das Werkzeug sowie deren Szenarien zum Einsatz aufgezeigt habe, konnte ich mit der eigentlichen Ausarbeitung des Werkzeugs beginnen.

Hierfür habe ich natürlich als aller erstes Bezug auf die Anforderung genommen. Das Paper „Why Dont Software Developers Use Static Analysis Tools to Find Bugs“ (siehe Kapitel 2.2.2), aus dem Jahr 2013, lieferte mir die ersten Ansätze. Schon damals wünschten sich die Teilnehmer der Studie ein ansprechendes Design, sowie eine benutzerfreundliche Bedienung, eine deutliche Darstellung der angezeigten Fehler, sowie eine Quickfix Lösung, welche eine vorgeschlagene Lösung präsentiert.

Hinzu kamen die Ergebnisse und Anforderungen der Onlineumfrage, welche ich im Zusammenhang mit dieser Abschlussarbeit erhoben habe. Dabei bestätigten sich erneut, die noch immer vorhanden Anforderungen des o.g. Papers aus dem Jahr 2013. Durch die Onlineumfrage hinzugekommene Anforderungen sind, dass die Anzeige der Fehler im Quellcode nicht nur deutlich, sondern auch verständlich sein sollte, eine Integration an eine favorisierte IDE angeboten sein sollte und der Testvorgang automatisiert stattfinden sollte.

Damit hatte ich die ersten Anhaltspunkte, was das Werkzeug auf jeden Fall bieten musste.

Die eigentliche Gestaltung des Werkzeugs fand unter Berücksichtigung der Trust Guidelines (siehe Kapitel 3.10) sowie der Einhaltung der Design Guidelines von Don Norman (siehe Abschnitt 3.10) und den üblichen Qualitätsmerkmalen einer Software statt.

Diese Qualitätsmerkmale [Lig09] beurteilen die Qualität einer Software. Zu diesen Qualitätsmerkmalen gehören unter anderem die Security, Zuverlässigkeit, Verfügbarkeit, Speicher- und Laufzeiteffizienz sowie die Benutzbarkeit. Weil es sich bei dem entwickelten Werkzeug um einen Prototypen handelt, war es mir im Rahmen dieser Abschlussarbeit nur möglich die Faktoren Security und Benutzbarkeit zu berücksichtigen. Sämtliche anderen Faktoren blieben noch unberücksichtigt.

Es war mir nicht nur wichtig, dass das Design des Werkzeugs optisch ansprechend ist, sondern auch, besonders weil das Werkzeug ja die Security

betrifft, dass es das Vertrauen beim Benutzer erzeugt sich auf die Qualität des Werkzeugs verlassen zu können, sowie auf dessen Ergebnisse.

Nicht nur aus den Trust Guidelines von Marsh und Briggs [Mar], sondern auch aus dem Paper „Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes “ [RKB+13], aus dem Jahr 2013, lassen sich die Anhaltspunkte ziehen, worauf zu achten ist, wenn es um das Thema Vertrauen geht.
Hier war der Ansatz unter Beachtung der Usable Security (siehe Kapitel 3.5) so viele Security Details, wie möglich zu verstecken bei der Nutzung eines sicheren Webmail Systems, um damit die Verwendung für den Benutzer so einfach, wie möglich zu gestalten.
Diese vorgestellten Webmail Systeme hatten ein automatisches Schlüsselmanagement, sowie eine automatische Verschlüsselung. Damit sei es fast allen Benutzern möglich gewesen einfach und ohne Vorwissen dieses sichere Webmail System zu nutzen.

Das erstaunliche Ergebnis war, dass trotz der vorhandenen Automation ein kleiner Teil der Testpersonen dennoch versehentlich unverschlüsselte Nachrichten verschickten und andere Benutzer äußerten sich sogar insofern, dass sie sich nicht sicher seien, ob sie diesem Webmail System vertrauen könnten.

Als zum Vergleich eine alternative Software getestet wurde, welche eine manuelle Verschlüsselung erforderte, hatten die Testpersonen sowohl weniger Fehler begannen, als auch der Software mehr vertraut, weil mit der aktiven Verschlüsselung auch mehr Aufwand verbunden war.

Das Ergebnis dieser Studie zeigt auf, dass es, wenn man Vertrauen erreichen will, der richtige Weg ist mehr Transparenz aufzuzeigen. Das meint, dass z.B ein komplexes Verfahren nicht hinter einem Knopf versteckt werden sollte, sondern auch gezeigt werden sollte, was hinter diesem „simpel erscheinenden“ Betätigten des Knopfes geschieht und welcher Aufwand damit verbunden ist.
Sind Aufgabenstellungen offensichtlich so einfach gestaltet, dass ein komplexes Verfahren, wie die Verschlüsselung einer Mail, mit nur einem Knopf ausgeführt werden kann, ist es für technisch weniger versierte Personen nicht glaubwürdig und damit nicht vertrauensvoll.

5.2 Ansatz des verbesserten Werkzeugs

Aufbauend auf den beliebtesten Werkzeugen, welche in Abschnitt 3.1 betrachtet wurden, habe ich diese Orientierung zur Gestaltung des Werkzeuges verwendet und mit den entsprechenden ermittelten Anforderung aus dem Paper „Why Dont Software Developers Use Static Analysis Tools to Find

Bugs“ (siehe Kapitel 2.2.2) und jenen aus den Ergebnissen der Onlineumfrage ergänzt.

Unter Betrachtung aller Werkzeuge stellte ich fest, dass das vom Design und den Funktionen ausgereifteste Werkzeug RISP (siehe Kapitel 3.1.3), zwar nahezu alle Anforderung abdeckt, jedoch keinerlei Quickfix Funktion zur Verfügung stellt.

Was mir ebenso auffiel war, dass keines der untersuchten Werkzeuge, über jene hinaus, Rückmeldungen bietet, welche semantisch einwandfrei und psychologisch im positiven Sinne gestaltet sind.
Keines dieser Werkzeug untersucht den Quellcode nach Sicherheitslücken die noch möglich sind, und zeigt auch gleichzeitig auf, welche Lücken sie bereits abgedeckt haben.

Die Idee einer positiven Bestätigung [Psy], nach dem erfolgreichen Abdecken eines Gebietes ist es, dem Benutzer auch die Rückmeldung zu geben, dass er bereits gewisse Lücken abgedeckt hat und das dies auch vom Werkzeug wahrgenommen wurde.

Weil laut Anforderungen der Befragten aus der Onlineumfrage das Werkzeug in eine allgemein beliebte IDE eingebunden werden können sollte, habe ich mich an den in Abschnitt 3.2 vorgestellten Editoren orientiert, um darin das Werkzeug zur Codeanalyse einsetzen zu können.

5.3 Vorstellung des Prototyps

Das im Rahmen dieser Masterarbeit entwickelte Werkzeug nennt sich „SecTool“ (eine Kombination der Worte Security und Tool). Ein konzipierter Coding Editor, welcher dabei hilft den eigenen Quellcode auf Sicherheitslücken zu überprüfen.

Dabei werden fünf Kriterien geprüft: Access - Control Problem, Memory Safety, Race Condition, Unvalidated Input und Weakness in Authentication. Diese Kriterien entsprechen den Kategorien, worin ich die Sicherheitslücken in Abschnitt 3.11 eingeteilt habe. Damit der Benutzer demnach nicht von einer gesamten Liste an vorhandenen Sicherheitslücken erschlagen wird, wird neben dem angezeigten Fehler immer auch ein dazu gehörendes Kriterium angezeigt.

Weil es gängig ist, dass Editoren in Englisch bedient werden, habe ich sämtliche Bedienelemente in die englische Sprache umgesetzt. Die Erklärung der Fehlermeldung sowie zu den Kriterien sind ins Deutsche übersetzt worden. Ich hatte ausschließlich Testpersonen befragte, deren Muttersprache Deutsch

ist.
Selbst wenn der Industriestandard ein anderer ist, fällt es einem immer leichter erklärende Texte in der Muttersprache aufzunehmen, als in einer nativ fremden Sprache. Dies besonders dann, wenn die Thematik fachlich anspruchsvoll ist und man selbst darin nicht so tief vorgedrungen ist.

Das Werkzeug startet mit einem Willkommensbildschirm. Auf diesem wird nicht nur das Werkzeug selbst vorgestellt, sondern auch kurz erklärt, welchem Zweck das Werkzeug dient (siehe Abb. 21).

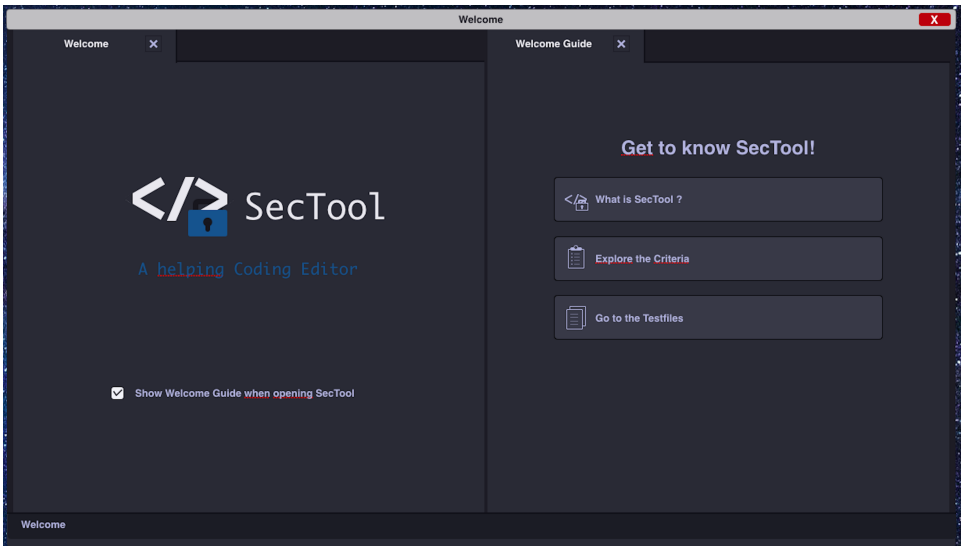

Abbildung 21: Willkommensbildschirm

Um den Testpersonen einen realistischen Eindruck des Werkzeugs zu vermitteln, wurden sämtliche Icons selber gestaltet.
Dazu gehören sowohl die Icons zur Beschreibung der Funktion, der Beschreibung der Kriterien und der Link zu den Testdateien (siehe Abb. 22) sowie auch das Logo (siehe Abb. 22) für das Werkzeug. Damit das Werkzeug einen Namen hat, entschied ich mich dazu es ”t’SecTool“ zu nennen (siehe Abb. 21).

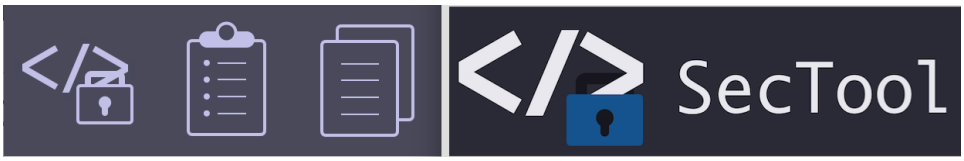

Abbildung 22: Icons und Logo

Mit einen Klick auf „What is SecTool“ erhält man eine kurze Beschreibung, was der Zweck dieses Werkzeugs ist. (siehe Abb. 23).

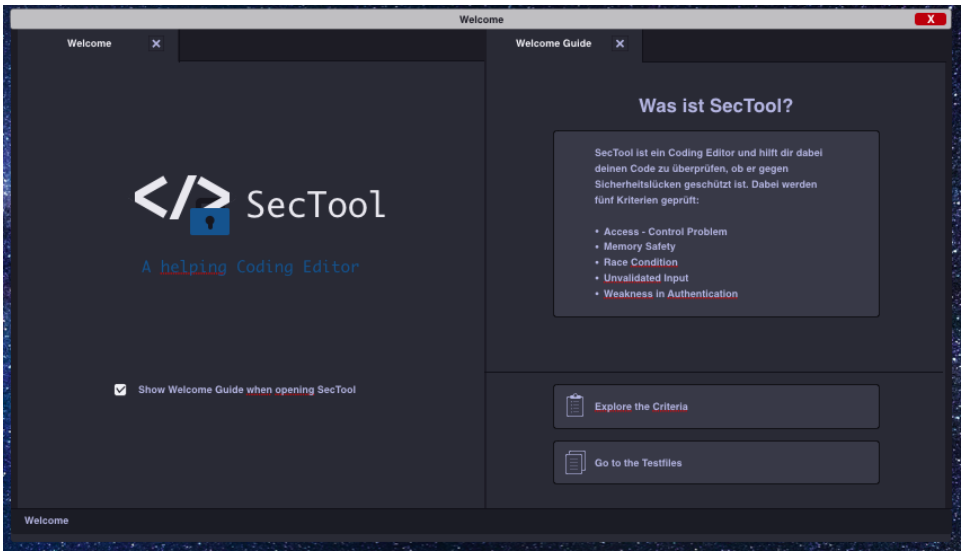

Abbildung 23: Was ist SecTool

Gleichzeitig hat man auch die Möglichkeit die Kriterien kennen zu lernen, nach welchen der Code vom Werkzeug untersucht wird. Klickt man also auf „Explore the Criteria“, wird jedes zu untersuchen Kriterium kurz vorgestellt, damit der Nutzer versteht was konkret überprüft wird und realisiert, was mit diesen Termini speziell gemeint ist. Klickt man eines dieser Kriterien an, um mehr zu erfahren, klappt sich das Fenster aus und liefert detaillierte Informationen (siehe Abb. 24).

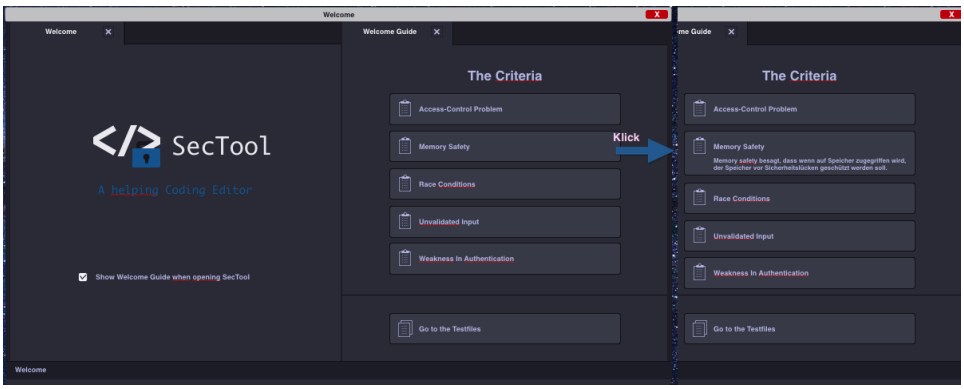

Abbildung 24: Die Vorstellung der Kriterien

Anschließend kann man sich die Testdateien ansehen.

Um die Funktionen von SecTool durch die Benutzer test zu lassen, habe ich fest vorgegebene Dateien in dieses Werkzeug eingepflegt. Es sind drei

Beispiele. Weil sich Teilnehmer der Onlineumfrage konkrete Behebungen von Problemen wünschten, habe ich diese in die Testdatei als Beispiele aufgenommen. Daher gibt es eine Datei, welche einen Buffer Overflow enthält und ein weiteres Beispiel, bei dem eine SQL Injection möglich ist.
Die Abbildung 25 zeigt die eigentliche Bedieneroberfläche, beim dem ein Quellcode kontrolliert werden kann. Auf der linken Seite sind in einem Verzeichnis die vorhandenen Dateien angezeigt, in der Mitte der eingegebene Quellcode, welcher kontrolliert wird und auf der rechten Seite die Kriterien der Sicherheitslücken.

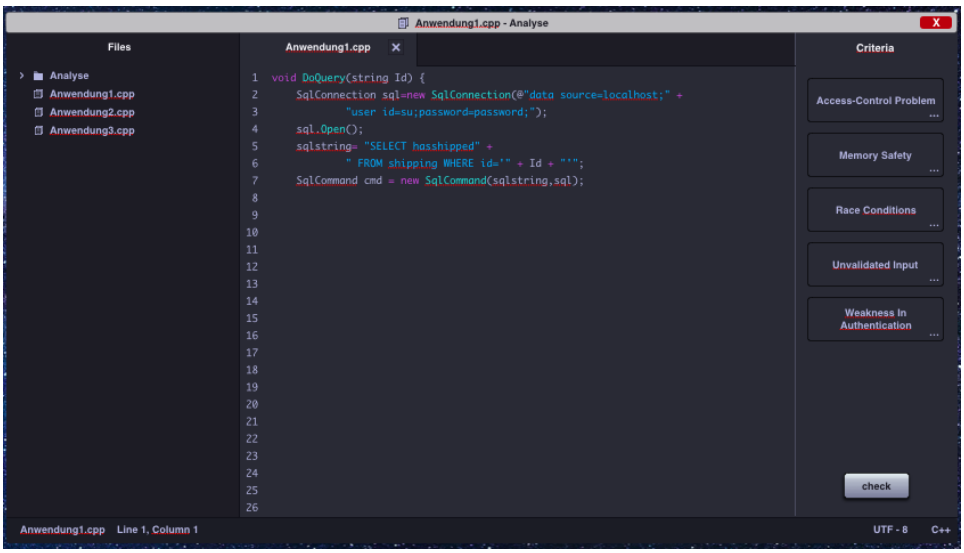

Abbildung 25: Bedieneroberfläche von SecTool

Wenn die Kriterien auf der rechten Seite angeklickt werden, werden detailliertere Informationen zu den jeweiligen Kriterien gegeben (siehe Abb. 26).

Die Überprüfung des Quellcodes wird mit dem Button namens „check“ gestartet. Orientierend an den Empfehlungen von Garfinkel und Lipford [GL14, S.84], gibt es nur einen Knopf, welcher den gesamten Quellcode nach allen Kriterien überprüft. Einstellungen an der Securityüberpüfung können nicht übernommen werden, weil sonst eine Gefährdung der Verlässlichkeit der Securityüberprüfung eintreten könnte. Sobald dieser Button betätigt wird, wird der jeweilige Code inhaltlich überprüft, Kriterium nach Kriterium. Wird ein Kriterium verletzt, erscheint ein rotes Kreuz neben dem Kriterium. Wird ein Kriterium nicht verletzt erscheint ein grüner Hacken (siehe Abb.27).

Orientierend an dem Paper „Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes“ [RKB+13], wird nicht versteckt, was nach

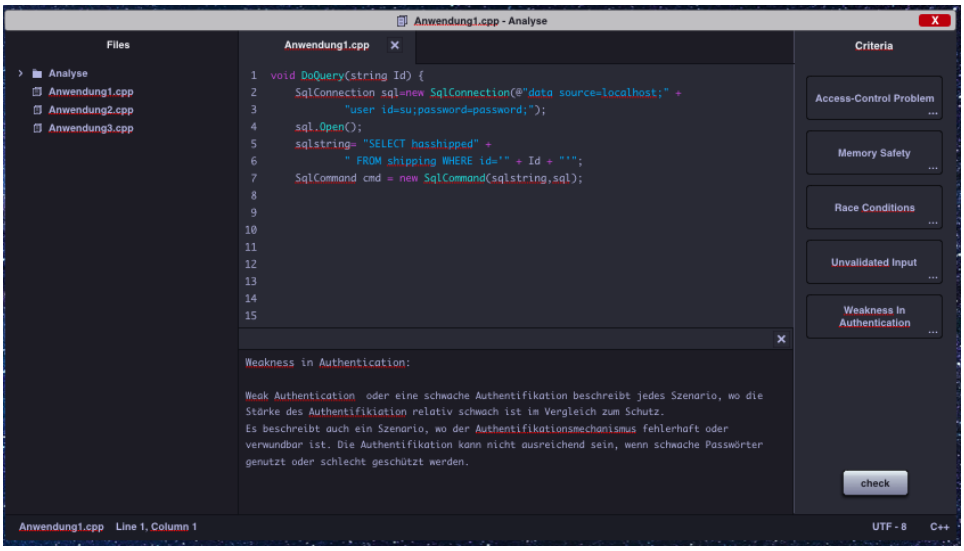

Abbildung 26: Erklärungen zu den Kriterien anzeigen lassen

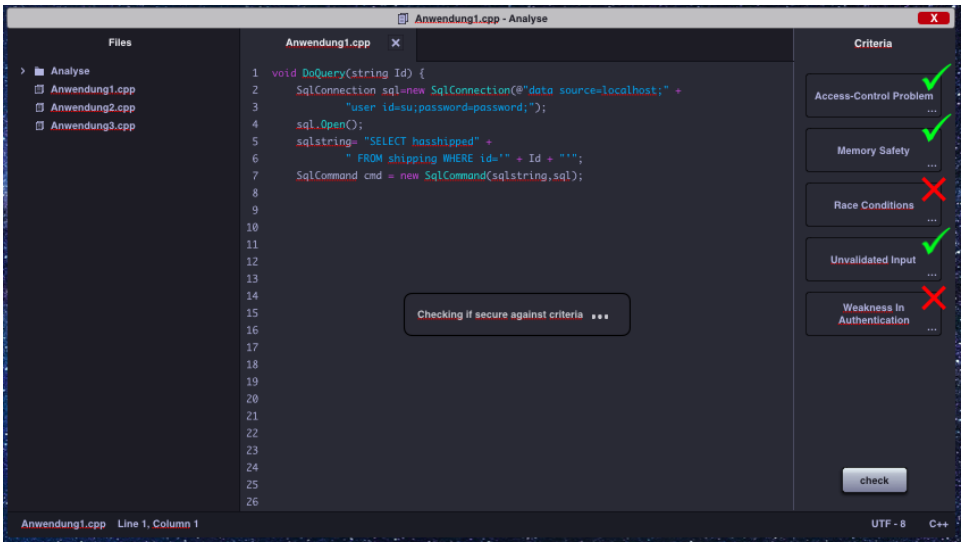

Abbildung 27: Der Checkvorgang

Betätigung des Check Buttons geschieht, sondern in einem Dialog der Reihe nach angezeigt, welches Kriterium, eines nach dem anderen, verletzt wurde oder eben auch nicht. Dies soll vor allem aufzeigen, dass im Hintergrund etwas sehr komplexes geschieht, wodurch automatisch das Vertrauen in die Software gestärkt wird.

Durch die differente Rückmeldung eines grünen Hackens und eines roten Kreuzes (siehe Abb.27), erhielten die Benutzer nicht nur die verschiedenen

Informationen, sondern auch eine sofortige Rückmeldung darüber, dass gewisse Kriterien bereits überprüft wurden bzw. nicht mehr überprüft werden müssen. Dies ist eine positive Bestätigung, welche aufzeigt was im Quellcode bereits richtig gemacht wurde.

Wurden Stellen im Quellcode entdeckt, welche Sicherheitslücken darstellen, werden sie direkt im Quellcode hervorgehoben (siehe Abb. 28).

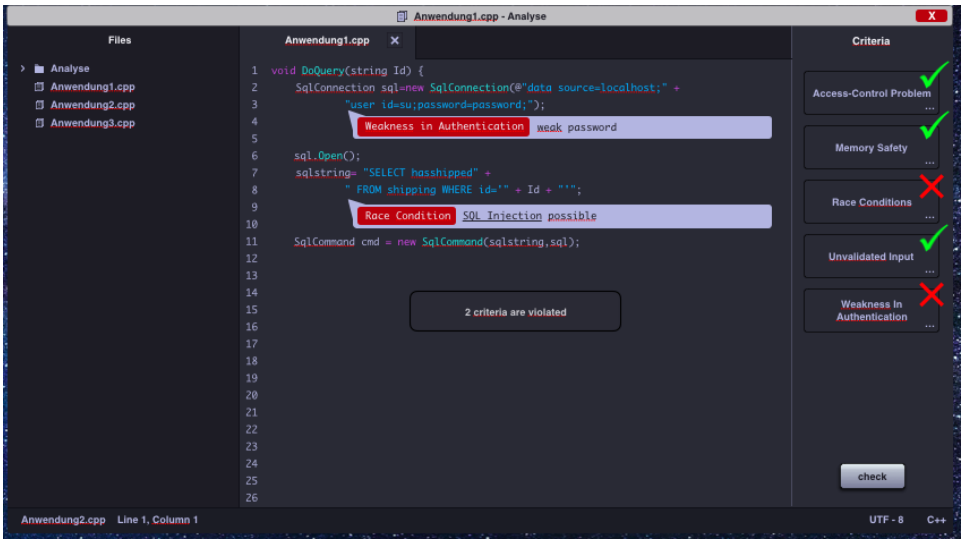

Abbildung 28: Sicherheitslücken erkannt

Beim Anklicken dieser Fehler wird die Erklärungen der Fehlermeldungen wiedergegeben (siehe Abb. 29).

Andere Beispiele enthielten die Option einen Quickfix nutzen zu können, also einen Vorschlag, wie die Sicherheitslücken behoben werden können (siehe Abb. 30).

Mit Betätigung des Knopfes Quickfix wird dann eine vorgeschlagene Lösung umgesetzt. Bei erneuter Überprüfung des Quellcodes, nachdem alle Fehler behoben wurden, erhielt der Benutzer die Rückmeldung „Good, no criteria were violated“. Zusätzlich wurden auch alle Kriterien mit grünen Hacken versehen, um eine weitere Bestätigung zu erhalten, dass nun keine, auf diese Art und Weise überprüfbaren, Sicherheitslücken mehr vorhanden sind (siehe Abb. 31).

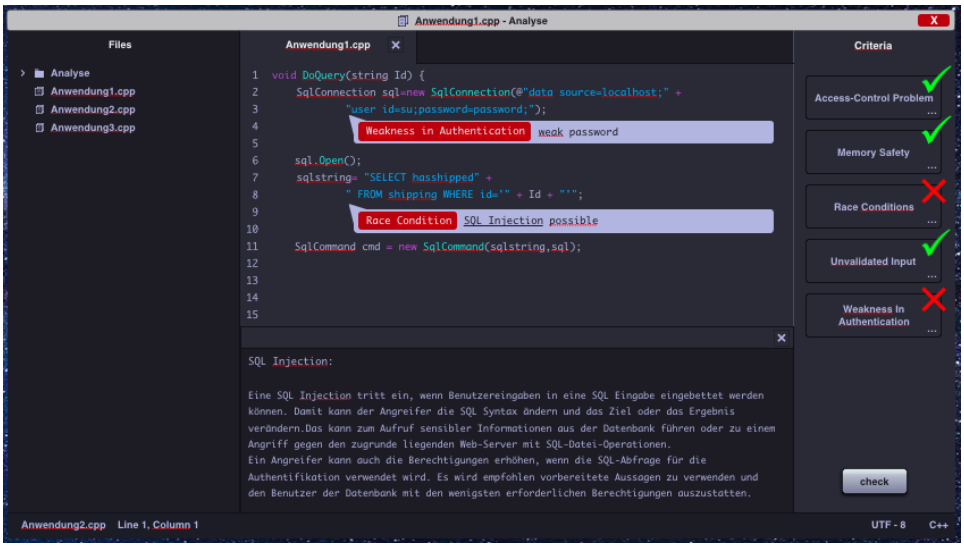

Abbildung 29: Erklärung der Sicherheitslücke

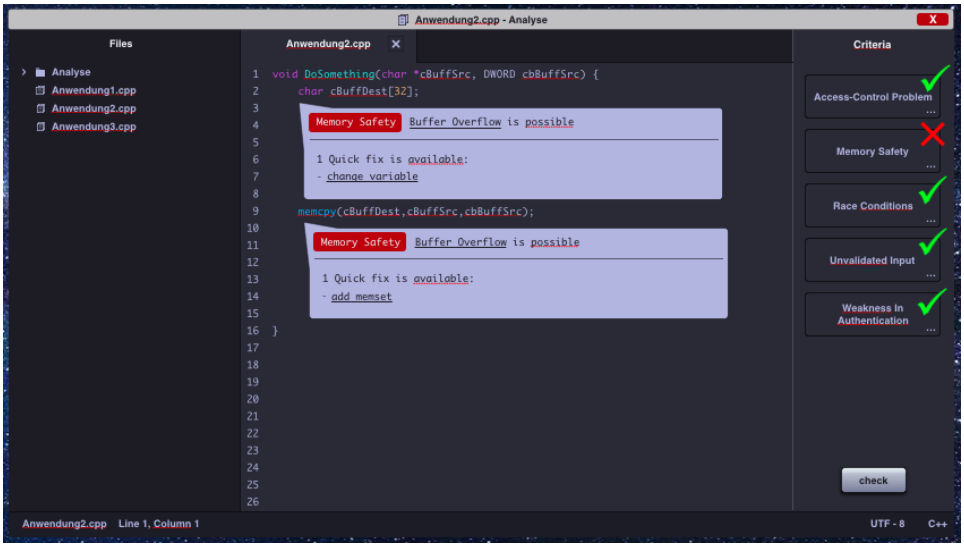

Abbildung 30: Angebot einer Quickfix Lösung

5.4 Abschlussbemerkung

Es war mein erklärtes Ziel ein Werkzeug zu konstruieren, welches intuitiv und einfach bedient werden kann, welches viel Erklärungen bietet, verständliche Fehlermeldung ausgibt sowie auch darstellt, wo diese genau entstehen und darüber hinaus auch eine Option offeriert, welche die Schließung von Sicherheitslücken durch vorgeschlagenen Lösung anbietet.

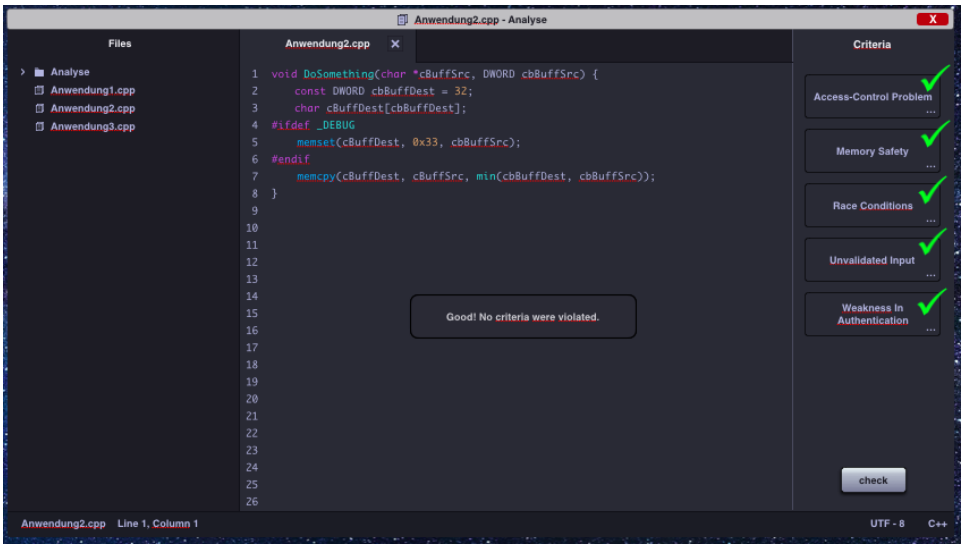

Abbildung 31: Keine Sicherheitslücken entdeckt

Neben der Idee des positiven Feedbacks zur Abdeckung einer Sicherheitslücke und der Schritt für Schritt Überprüfung jedes einzelnen Kriteriums, um Vertrauen in der Nutzung dieses Werkzeugs zu erhalten, sollte dass Durchgehen jedes einzelne Kriterium dazu führen, dass diese durch die häufige Wiederholung im Gedächtnis des Nutzers haften bleiben.
Nachweislich führen häufige Wiederholungen [Oeh17] dazu, dass sich Erlerntes im Gedächtnis verfestigt.

> „Lernen braucht Struktur, Lernen braucht Wiederholung! Nur Wiederholung führt dazu, dass aus flüchtigem Wissen Können wird“. [Oeh17, S.14]

Die Strategie dahinter war, dass diese stetigen Wiederholungen dazu führen, dass Programmierer auch ohne dieses Werkzeug bei betrachten jeglichen Quellcodes auch immer diese fünf Kriterien überprüfen, um sich ein Urteil über das Vorhandensein von Sicherheitslücken zu bilden.

5.5 Entwicklung drei verschiedener Werkzeuge

Im Rahmen dieser Abschlussarbeit wurden drei Varianten eines Werkzeuges mit gleicher Funktionalität entwickelt. Der Unterschied liegt nicht in der Bedienung, sondern darin in welcher Sprache das Werkzeug bedient werden soll.

Hierzu sei kurz erläutert das, obhin des internationalen Charakters der Informationstechnologie, es beobachtbar ist, dass Editoren üblicherweise in der englischen Sprache konzipiert werden. Genauso ist es üblich Bezeichnungen

von Fehlermeldungen, wie auch Sicherheitslücken in englischer Sprache anzugeben. Daher entwickelte ich ein Werkzeug, welches komplett auf Englisch ist, um den Testpersonen eine Bedienung, wie auch eine Bezeichnung zu geben, welche sie kennen und gewohnt sind.

Unter Berücksichtigung der möglichen Sprachbarrieren, sowie zugrunde legend, dass komplexe Sachverhalte in der Muttersprache immer am Besten verstanden werden, erstellte ich eine zweite Variante des Werkzeugs, welches gleich bedient, jedoch komplett auf Deutsch umgesetzt wurde (dabei ist hier anzumerken, dass ich ausschließlich deutsche Muttersprachler als Testpersonen hatte). Da ja eine der Anforderungen an das Werkzeug lautete „eine verständliche Erklärung zu Fehlermeldung zu erhalten“, konnte ich das nur gewährleisten, indem ich es auch in einer Sprache verfasste, welche vom Zielpublikum barrierefrei verstanden wird.

Um einen Kompromiss einzugehen und beiden Kriterien gerecht zu werden, habe ich noch eine dritte Variante des Werkzeugs erstellt, welche in englischer Sprache bedient wird. Auch werden Fehlermeldungen und die Namen der Sicherheitslücken in Englisch angezeigt. Doch sämtliche Erklärungen zu den Kriterien und der Beschreibung der Fehlermeldung findet man auf Deutsch. Damit erhoffte ich mir, dass ich beiden Erwartungen gerecht werden kann, nämlich ein erwartetes Bedienfeld, sowie auch eine verständliche Erklärung.

6 Benutzbarkeitsstudie

6.1 Einführung

Um anschließend heraus zu finden, wie benutzerfreundlich das Werkzeug und wie erfolgreich die Strategie des Werkzeugs beim Benutzer angekommen ist, wurde eine Studie durchgeführt.

Die Studie bestand aus vier Teilen:

1. ein Fragebogen zur Erfassung der Erfahrungen mit Werkzeugen zur Codeanalyse und ihrer Nutzung sowie deren Anforderungen an solch ein Werkzeug
2. das Testen des Werkzeugs
3. ein Nachgespräch
4. die Beantwortung des Fragebogens nach ISO Norm 9241

6.2 Bestandteile

6.2.1 Beginn

Zu Beginn der Studie wurden die Teilnehmer darüber aufgeklärt, in welchem Rahmen diese Umfrage stattfindet, zu welchem Zweck sie erhoben wird und wobei das Ergebnis dieser Studie hilft. Zusätzlich erhielten die Teilnehmer den Hinweis, dass sämtliche ihrer Antworten anonym ausgewertet und unter Einhaltung der Datenschutzbestimmungen behandelt würden.

6.2.2 Fragebogen

Der Fragebogen, mit dem die Studie zur Auswertung des Werkzeugs begonnen wurde, war der gleiche Fragebogen, mit dem ich auch die Onlineumfrage erhoben habe. Differierend von der Onlineumfrage, wurden die Fragen hier nicht von den Testpersonen selbstständig, sondern in Form eines Interview beantwortet. Damit war es mir möglich deutlich mehr und inhaltlich erweiterte Erklärungen zu erhalten.

6.2.3 Das Werkzeug

Das Werkzeug selbst wurde mit Hilfe eines „Thinking Aloud“ Tests getestet. Dabei erhielten die Testpersonen keine Aufgabenstellungen, sondern nur die Anweisung sich das Werkzeug anzusehen, es zu testen und sämtlichen Gedanken wie Kritik oder Anmerkungen laut auszusprechen.
Sobald ich feststellte, dass eine Testperson zu lange schwieg, bat ich nochmals darum die Vorhanden Gedanken laut auszusprechen.

Auch wenn es ein ungewohntes Verhalten sein mag, seine Gedanken stetig laut auszusprechen, schien keiner der Testpersonen mit dieser Anforderung überfordert oder gar verärgert. Zudem hatte dies den Vorteil, dass ich die Testpersonen nicht mit Fragen in ihrer Beurteilung des Werkzeugs beeinflusst habe.

Bevor sie sich dem Testen des Werkzeugs zugewendet hatten, wurden sie darauf hingewiesen, dass das Ziel des Tests nicht die Überprüfung ihrer eignen Fähigkeit zur Interaktion mit dem Werkzeug war, sondern der Fokus einzig auf der Verbesserung des Werkzeugs lag. Damit waren die Testpersonen nicht nur motivierter Anmerkung zu machen und hatten weniger Hemmungen Kritik auszuüben, sondern wussten auch worauf sie sich während des Testens zu konzentrieren hatten; nämlich ob das Werkzeug ihren Ansprüchen genügt und nicht anders herum.

Die Testpersonen testeten das Hauptmenü, die Erklärung der Kriterien, sowie die Überprüfung sämtlicher Testdateien nach Sicherheitslücken, ihrer

Fehlerbeschreibungen und die Quickfix Funktion.

6.2.4 Das Nachgespräch

Nachdem das Werkzeug getestet wurde, wurde ein Nachgespräch zum Erlebten, mit dem Umgang des Werkzeuges geführt. Dieses Nachgespräch umfasste 15 Fragen. Um zu Erfahren, wie nun die Überzeugungen der Testpersonen zu Security im Allgemeinen sind, wurden ihnen Fragen, unterschieden nach der Art wie Schnell[48] [Fra] es beschrieben hat, zur ihrer Einstellung und Meinung, ihrer Überzeugung und ihrem Verhalten gestellt.

Die Fragen nach der Einstellung lauteten:

- Sollte das Schließen von Sicherheitslücken bei Daten das wichtigste sein?
- Sollte der Hauptfokus von Editoren auf das Schreiben von sicherem Code liegen?

Die Frage nach der Überzeugung lautete:

- Weil ein Code immer geknackt werden kann, muss man ihn nicht sicher schreiben.

Die Frage nach dem Verhalten lautete:

- Haben Sie bereits mal Ihren Code nach Sicherheitslücken geprüft, auch wenn das nicht von Ihnen gefordert war? (Mit einem Ranking von nie, einmal, einige Male, schon oft nur regelmäßig)
- Schützen Sie Ihren Rechner?
- Schützen Sie Ihre mobilen Geräte?

Nach Beantwortung dieser Frage hatte ich eine Tendenz, welche Priorität Sicherheit für die Testperson einnimmt.

Anschließend wurden ihnen Fragen zum Werkzeug selbst gestellt. Unter anderem:

- Was ist Ihr Gesamteindruck?
- Würden Sie solch ein Tool verwenden?
- Was ist Ihre Meinung zum Konzept der positiven Bestätigung?
- Haben Sie etwas neues dazu gelernt?

[48] Assoziierte Prof. Dr. Tatjana Schnell, Universität Innsbruck

Weil die Testpersonen drei Testbeispiele hatten und damit immer wieder die Kriterien durchgelaufen sind, wollte ich erfahren, inwiefern sie diese Kriterien bereits verinnerlicht hatten und habe sie daher darum gebeten, die Kriterien nochmals aus dem Gedächtnis aufzulisten.

6.2.5 Beantwortung des ISO Fragebogen 9241 /110

Mit Hilfe der Beantwortung des Fragebogens zu den Grundsätzen der Dialoggestaltung war es mir möglich auch mit Zahlen zu belegen, für wie benutzerfreundlich die Testpersonen das Werkzeug erachteten. Ich entschied mich dazu diese Variante der Auswertung zu verwenden, weil sie mir aufzeigte, welche einzelnen Kriterien der Grundsätze besonders ausgeprägt gestaltet sind oder welche noch verbessert werden müssen. Damit konnten einzelne Probleme der Gestaltung besser herausgearbeitet werden.

6.3 Anzahl Testpersonen

Mir war es möglich acht Teilnehmer für die Studie zu finden. Weil es sich in diesem Fall um eine formative und qualitative Studie handelt, reichen bereits fünf Testpersonen [Nie], um die meisten Probleme im Prototypen zu finden. Mit weiteren Testpersonen wiederholen sich nur noch die genannten Probleme und es werden keine neuen mehr gefunden. Demnach hat die Anzahl von acht Teilnehmer deutlich gereicht. Die Testpersonen waren zwischen 17 und 58 Jahr alt und wiesen Programmiererfahrung zwischen $1\frac{1}{2}$ Jahren und 40 Jahren auf.

Die gesamte Studie hat pro Person durchschnittlich 75 Minuten beansprucht.

6.4 Ergebnisse der Studie

6.4.1 Ergebnis der ersten Umfrage

Das Verhältnis zur Nutzung der Plattform Stack Overflow, sowohl zum Hochladen des Codes, als auch zum Heraus-Kopieren von Quellcode, entsprach in etwa einem gleichem Verhältnis, welches auch bei der Onlineumfrage festgestellt wurde.

Anders hingegen waren die Erfahrungen der Testpersonen bzgl. der Werkzeuge zur Codeanalyse. Keiner von ihnen hatte eines der vorgegebenen oder ein anderes Werkzeug je verwendet. Nur einer Person (T7) war ein einzelnes Werkzeug aus der Liste bekannt, genutzt hatte sie es jedoch auch nicht. Die restlichen Personen kannten keines der angegebenen Werkzeuge.

Die Anforderungen an das Werkzeug glichen ebenso den Ergebnissen der Onlineumfrage. Auch hier wünschten sich die Teilnehmer ein Werkzeug, welches

leicht zu bedienen ist, einem deutlich aufzeigt, was genau das Problem im Quellcode ist, dieses verständlich erklärt und auch einen Lösungsvorschlag unterbreitet.

6.4.2 Aussagen zum Werkzeug

Erster Eindruck:

Sämtliche Testpersonen äußerten sich positiv zum ersten Eindruck. Sie bezeichneten das Werkzeug unter anderem als einladend und übersichtlich. Testperson T1 äußerte sich folgendermaßen.

> „Das Logo gefällt mir und durch das „Welcome“ habe ich ein sicheres Gefühl. Der Eindruck von Security ist sofort da, wegen der Bezeichnung des Editors und des Schlosses im Logo“.

Welcome Page:

Die Möglichkeit mehr über das Werkzeug zu erfahren, hatten sieben von acht Testpersonen genutzt. Nur Testperson T2 hatte diesen Schritt übersprungen und sofort damit begonnen die Testdateien überprüfen zu lassen. Sechs der genannten sieben Testpersonen merkten an, dass die Vorstellung der fünf Kriterien an der Stelle „Was ist SecTool“ (siehe Abb. 32) bereits als Link „anklickbar„ sein und die detaillierten Informationen dazu bieten sollte.

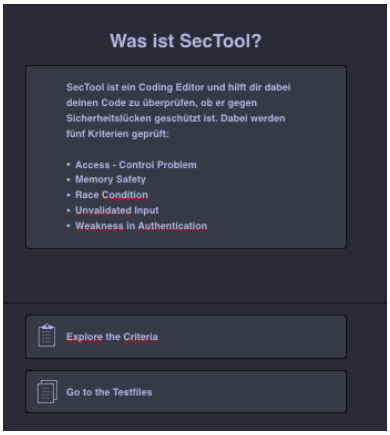

Abbildung 32: Kriterien auf der Willkommensseite

Auch haben sich sieben von acht Testpersonen den Punkt „Explore the Criteria“ genauer angesehen (siehe Abb. 33). Jedem von ihnen hat es gefallen, dass die Kritikerin nicht nur vorgestellt, sondern auch erklärt wurden. Testperson T8 merkte noch zusätzlich an, dass sich die Kriterien noch besser

von Bildern erklären lassen würden.

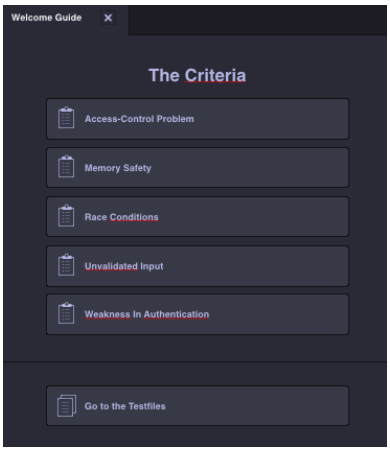

Abbildung 33: Die Kriterien

Vier von acht Personen merkten an, als sie das „Welcome Menü“ verlassen hatten, dass es keine Möglichkeit gäbe, da wieder zurück zu gelangen und sie einen „Zurück Knopf“ haben wollen würden.

Die Funktionsseite

Alle acht Testpersonen haben ohne Erklärung den Aufbau sowie die Bedienung des Werkzeugs verstanden. Sie erkannten die Auflistung der Beispielsprogramme, die Aufführung der Kriterien und verstanden die Funktion des „check“ Buttons.

Allen Testpersonen hat es gefallen, dass die Kriterien beschrieben wurden. Zwei der acht Testpersonen nahmen beim ersten Versuch der Überprüfen des Code an, dass die Betätigung der Kriterien, die Untersuchung nach einem gewissen Kriterium starten würde. Nach dem sie jedoch die Rückmeldung vom Werkzeug erhielten, dass damit die Erklärungen der Kriterien angezeigt wird, entdeckten und nutzen sie den check Button. Die Methodik des Checkverfahrens, Kriterium nach Kriterium zu untersuchen, beschrieb Testperson T1 mit den Worten:

> „Es ist sehr angenehm, dass alles Schritt für Schritt durchgegangen wird “

Beim der Betrachtung der Testdateien suchten vier von acht Testpersonen die Möglichkeit, eine Datei öffnen zu können. Sie gaben sich selbst als Nutzer vom Windows Betriebssystem an und hatten daher eine Menüleiste des Werkzeugs gesucht, welche sich bei der Vorstellung des Werkzeugs nicht an

dieser Stelle befindet.

Die Check Funktion:

Nachdem die Testdateien zum ersten Mal überprüft wurden, hatten alle acht Testpersonen eine positive Rückmeldung beim Erkennen, dass ein Kriterium mit einem grünen Hacken versehen wurde (siehe Abb. 34). Testperson T6 sagte dazu:

> „Es ist ein beruhigendes Gefühl und bestätigt mich, dass ich manche Aspekte schon berücksichtigt habe. “

Abbildung 34: Der Check Durchlauf

Testperson T7 merkte auch noch an, dass ein Abbruch Button fehlt, um das Checken stoppen zu können. Ansonsten würde sie gezwungen sein, jeden Checkdurchlauf immer abwarten zu müssen.

Fehlermeldungen:

Alle acht Testpersonen hatten die Anzeigeform der Fehlermeldung sehr positiv wahrgenommen (siehe Abb. 35). Sowohl dass sie die direkte Stelle angezeigt haben, als auch die konkrete Bezeichnung. Die Erklärungen zu den Fehlermeldungen waren jedoch allen Testpersonen entweder zu kurz oder nicht verständlich genug, ohne entsprechendes Vorwissen.

Quickfix:

Sieben von acht Personen haben die Quickfix Funktion verwendet. Testperson T3 sah zwar, dass eine Quickfix Funktion vorhanden war, erklärte jedoch nicht, wieso sie diese nicht genutzt hat. Alle Testpersonen haben sehr positiv auf das Vorhandensein der Quickfix Funktion reagiert. Testperson T7 sagte dazu:

> „Oh, das wird ja direkt gemacht. Sehr schön.“

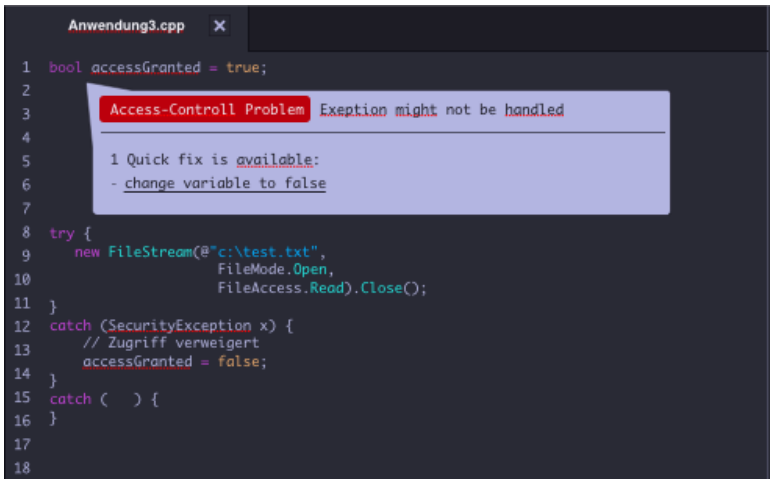

Abbildung 35: Anzeigen des Fehlers im Quellcode

Testperson T4 merkte jedoch an, dass sie sich mehr Erklärung zum Grund dieses Quickfixes wünsche. So sagte sie:

> „Es ist schön, dass der Fehler erkannt wird, aber warum soll ich das hinzufügen. Ich verstehe nicht warum das besser ist. Denn der Grund ist nicht erklärt. Es ist schade, dass man das nicht weiß. “

Trotz vorhanden sein des Quickfixbuttons überprüften fünf Testpersonen nochmals eigenständig das Ergebnis nach dem Quickfix. Testperson T1 sagte dazu

> „das Quickfix ist sehr gut, aber ich würde das gerne selber nochmal überprüfen, um das besser nach zu vollziehen“.

Vier von acht Testpersonen brachten sogar die Vertrauensfrage auf und fragten sich entweder, woher diese Quickfix Lösung stammt, woher sie mit Gewissheit nun sagen können, dass der korrigierte Code wirklich sicher ist oder ob das Werkzeug selbst nicht nach Sicherheitslücken überprüft werden solle. T1 sagte dazu:

> „Wer bietet die Lösungen an. Wer hat das überprüft?“

Vier Testpersonen hatten, nachdem sie den Quellcode mit der Quickfix Funkion korrigiert haben, erneut den check Button betätigt, um damit Gewissheit zu haben, dass der Quellcode nun wirklich keine Sicherheitslücken mehr aufweist.

Bei der Quickfix Korrektur war vielen nicht nachvollziehbar genug, was genau am Code verändert wurde. Daher hatten drei von acht Testpersonen

angemerkt eine Art der Vorschau vor dem Quickfix hinzuzufügen, welche dann bestätigt werden kann. Damit wäre ein besserer Vergleich zwischen den beiden Quellcodes vor dem Quickfix und nach dem Quickfix möglich.

Allgemeine Anmerkungen:

Um einen Kompromiss zu finden zwischen der üblichen Darstellung der Editoren in Englisch und einem guten Verständnis der Kriterien, wurde die Bedienung des Werkzeugs in Englisch gestaltet und sämtliche Erklärungen in Deutsch eingepflegt. Allgemein wurde jedoch von drei der acht Testpersonen angemerkt, dass der Mix von deutscher und englischer Sprache verwirrend sei. Testperson T4 empfahl:

> „Ich fände es besser in Deutsch. Ich glaube man bekommt auch eine gute Übersetzung zu den Kriterien hin".

Testperson T3 merkte auch etwas zum Design des Werkzeugs an:

> „Das Farbschema ist ganz toll. Ich mag es, wenn es dunkler ist. Es ist angenehm lesbar und die Farben sind gut zu erkennen."

Vier von acht Testpersonen fielen sogar die Kleinigkeit der Nennung der Sprache sowie der Zeichencodes auf und beherzigten es, dass dies angezeigt wurde.

Zusammenfassung:

Zusammenfassend lässt sich sagen, dass das Verständnis der Bedienung kein Problem darstellte. Was jedoch noch verbesserungswürdig ist, sind noch präzisere Erklärungen zu den Fehlermeldungen, weitere Beispiele zur Beschreibung der Kriterien und Erklärungen zu den Hintergründen, wer dieses Werkzeug entwickelt hat und die Quickfix Lösungen anbietet, um dem Werkzeug zu vertrauen.

6.4.3 Eindruck am Ende des Testes:

Sämtliche Testpersonen empfanden das Werkzeug, obgleich es nur ein Prototyp ist, von der äußeren Erscheinung, wie einen richtigen Editor. Dabei wurden selbst Kleinigkeiten hervorgehoben, wie das Anzeigen der Programmiersprache und der Kodierung. Niemand der Testpersonen hat gemerkt, dass es nur ein suggerierter Prototyp war, sondern nahmen an, dass es ein funktionierendes Werkzeug war. Genau das hat mir den Eindruck gegeben, dass mein Werkzeug, trotz des Zustand als Prototyp, überzeugte und relevante Ergebnisse geliefert hat.

6.4.4 Ergebnis des Nachgesprächs

Die Ergebnisse des Nachgesprächs zeigten auf, dass keiner der Person die Security vernachlässigte bzw. außer Acht ließ. Durchschnittlich bewertet die Testpersonen die Priorität der Security in einem Ranking zwischen „gar keine“, „- -“ „„ -“ „„ -/+“ „„ +“, „+ +“ , „sehr hohe“ mit „++“. Es zeigte auf, dass Security für sie zwar nicht die höchste Priorität hat, jedoch stark berücksichtigt wird.

Besonders Wert legte ich auf die Antworten zur Aussage: „Weil ein Code immer geknackt werden kann, muss man ihn gar nicht erst sicher schreiben“. Sollte jemand diese Aussage befürworten, zeigt es deutlich auf, dass keinerlei Ambitionen vorhanden sind, sicher programmieren zu wollen. Jedoch negierte jede der Testpersonen diese Aussagen und äußerten sich in etwa folgendermaßen dazu:

> „Es wäre gut, wenn die meisten Hacker abgewehrt werden. Alle werden sicher nicht abgewehrt, aber es sollte immer eine Prävention vorgenommen werden.“

Zur Frage, ob die Testpersonen dieses Werkzeug verwenden würden, stimmten alle acht von acht Testpersonen diesem zu. Dabei hoben sie hervor, dass die Bedienung sehr angenehm war, es schlicht designt wurde und sehr hilfreich ist, besonders weil auch Lösungen zu den Problemen angeboten wurden.

Auch die Darstellung der Rückmeldung der Erfüllung einer Kategorie, gekennzeichnet durch ein rotes Kreuz oder einen grünen Hacken, empfanden alle acht Testpersonen durchweg positiv. Sie sagten, dass es einer verständlichen sowie auch gut lesbare Darstellung entsprach, sowie auch die „Schritt für Schritt“ Überprüfung eines Kriteriums sehr nachvollziehbar waren.

Eines der Kriterien, welches ich besonders mit diesem Werkzeug untersuchen wollte, war inwiefern die Testpersonen auf eine Rückmeldung einer positiven Erfüllung eines Kriteriums reagieren. Tatsächlich hatten alle acht Testpersonen sehr gut darauf reagiert. Sie äußerten sich sinngemäß mit Meinungen wie:

> „Es hat einen guten Kontrast und es sagt mir, dass ich da nicht nochmal schauen muss.“

oder

> „Es sagt mir, dass ich den Code so weiter schreiben kann. Es sagt mir, dass ich gewisse wichtige Kriterien schon bedacht habe.“

und

„Ah das habe ich schon richtig gemacht“.

Die letzte Frage zum Wiedergeben der fünf Kriterien aus dem Gedächtnis hatte darauf abgezielt, inwiefern die wiederholte Überprüfung der drei Testdateien, nach diesen fünf Kriterien, dazu geführt hatte, dass diese auch im Gedächtnis des Programmierer hängen bleiben. Damit könnte der Programmierer das erlernte Wissen bei jedem Quellcode den er sich ansieht nutzen, und diesen nach den fünf genannten Kriterien, ohne ein Werkzeug zu haben, überprüfen.
Wenn sie die Kriterien im Kopf haben, könnte das ein Hinweis sein, dass sie darauf sensibilisiert wurden.

Das Ergebnis war, dass sieben von acht Testpersonen in der Lage waren mindestens vier von fünf Kriterien wieder zu geben, obgleich sie sich mit dem Werkzeug im Durchschnitt nur 30 Minuten beschäftigt hatten.
Teilnehmer T4 hatte darüber hinaus nicht nur alle Kriterien genannt, sondern auch nochmal deren Erklärungen wieder gegeben.
Drei Teilnehmer waren sogar in der Lage alle fünf Kriterien benennen zu können.
Das am meisten vergessene Kriterium der sieben Testpersonen war bei allen das Kriterium des „Unvalidated Input“. Weil ich in den drei Testbeispiele keinen Code verwendet habe, welches Sicherheitslücken in diesem Kriterium aufweisen, könnte das der Grund sein.

Zusammengefasst:

- 100% der Testpersonen reagierten sehr gut auf die positive Bestätigung
- 88% der Testpersonen waren in der Lage mindestens vier von fünf Kriterien frei aus dem Kopf wiederzugeben

6.4.5 Ergebnis der ISO Studie

Nach der Auswertung[49] der Fragebögen nach ISO Norm 9241/110 zu den Grundsätzen der Dialoggestaltung kam ich zu folgendem Ergebnis:

Anhand der Auswertung der ISO Studie ist zu erkennen, dass das Werkzeug nahezu alle Grundsätze gut erfüllt (siehe Abb. 36).

Das Werkzeug hat die Testpersonen bei der Erledigung ihrer Aufgabe unterstützt (Aufgabenangemessenheit), es erklärt sich selbst gut (Selbstbeschreibungsfähigkeit), die Funktionen sind gut vorhersehbar in Form ihrer Dialoge wie Rückmeldungen beim Bedienen der Oberfläche (Erwartungskonformität) und es hilft ebenso sehr gut dabei den Umgang mit dem System zu erlernen

[49] Die Auswertung entspricht gerundeten Ergebnissen

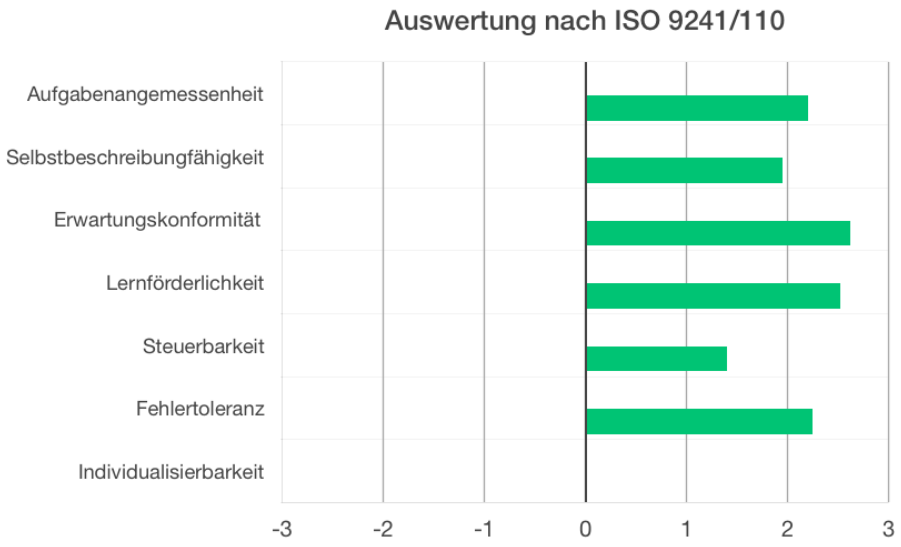

Abbildung 36: Auswertung der gesamten Teilnehmer

(Lernförderlichkeit).

Die Steuerbarkeit, also die Kontrolle über die Interaktion lässt sich noch verbessern. Dort wurde bemängelt, dass in dem Prototyp keine Funktion des Speicherns des Quellcodes vorhanden war, sowie kein erneuter Zugriff auf die Willkommensseite.

Die Fehlertoleranz war nur teilweise auswertbar. Es entspricht der Funktion, wie das Werkzeug auf fehlerhafte Eingaben des Benutzers reagiert. Weil mir das in diesen Prototypen nicht möglich war zu testen, entspricht die Fehlertoleranz einem Resultat, bei dem nur die Einträge berücksichtigt wurden, welche auch vollständig von allen Testpersonen ausgefüllt wurden. Die wenigen Kriterien, wie der konkrete Hinweis zur Fehlerhebung, wurden jedoch sehr gut bewertet.

Weil keine Form der Individualisierung vorgenommen wurde, entfernte ich sie aus der Auswertung des Fragebogens nach ISO Norm 9241/110.

Als Gesamtergebnis lässt sich, beruhend auf der Auswertung der ISO 9241/110 sagen, dass das entwickelte Werkzeug im Rahmen dieser Abschlussarbeit als benutzerfreundlich zu betrachten ist.

6.5 Verbesserungsvorschläge:

Auf der Grundlage der Bemerkungen der Studienteilnehmer und als Ergebnis des ISO Tests wurden folgende Verbesserungen im Design ausgearbeitet:

1. Ein Vorschaufenster für das Quickfix:

Weil es nicht nachvollziehbar und schwierig zu erfassen war, welche konkreten Änderungen durch das Betätigen des Quickfix Button ausgelöst wurden, sollte eine Art der Vorschau eingesetzt werden, welcher einem deutlich aufzeigt, wie der Code verändert werden soll und dieser dann bestätigt werden kann. Ein möglicher Entwurf könnte folgendermaßen aussehen (siehe Abb. 37):

Abbildung 37: Vorschaufenster im Quickfix

Hierbei erscheint beim Mouse Over über der Quickfix Empfehlung ein Pop Up Fenster, welches einem die entsprechende Änderung anzeigt. Sobald man dann, in diesem Fall „add memset“ betätigt, wird die im Pop Up Fenster angezeigte Änderung umgesetzt.

2. Ein Abbruchknopf des Checkens:

Auch wenn nur eine Testperson (T7) angemerkt hatte, dass der Checkvorgang abgebrochen werden können muss, stellt das eine berechtigte Anmerkung dar, welche als Änderung dringend aufgenommen werden sollte. Ein möglicher Entwurf könnte folgendermaßen aussehen (siehe Abb. 38):

Sobald der Check Button betätigt wird und der Überprüfungsvorgang noch

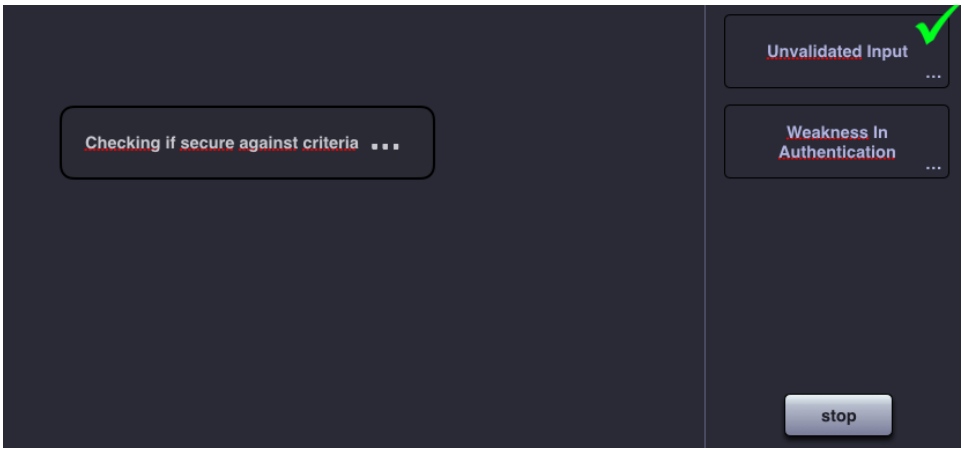

Abbildung 38: Check-Vorgang stoppen

nicht beendet ist, wird der Check Button solange zu einem „stop Button“ und der Vorgang kann unterbrochen werden.

3. Zurück ins Menü:

Um nochmals die „Willkommenstour“ zu durchlaufen, um ein weiteres mal die Kriterien und das Werkzeug selber kennenlernen zu können, sollte ein Button vorhanden sein, welcher einem wieder dorthin zurück führen kann.

4. Kategorie verlinken aus der Willkommensseite:

Um bereits im „Willkommensmenü“ mehr über die Kriterien zu erfahren, sollten sie im Abschnitt „Was ist SecTool“ zur Erklärungsseite verlinkt sein. (Siehe Abb. 39)

5. Spracheinstellung:

Um selber entscheiden zu können, welche Sprache man in der Bedienung des Werkzeugs bevorzugt, sollte ein Button zu dieser Einstellung vorhanden sein. Mit dieser Einstellung würde auch ein Kriterium für die Individualisierbarkeit hinzugefügt werden.

Ein möglicher Entwurf könnte folgendermaßen aussehen(siehe Abb. 40):

Mit diesem Kippschalter kann nun zu beliebiger Zeit zwischen deutscher und englischer Sprache gewechselt werden.

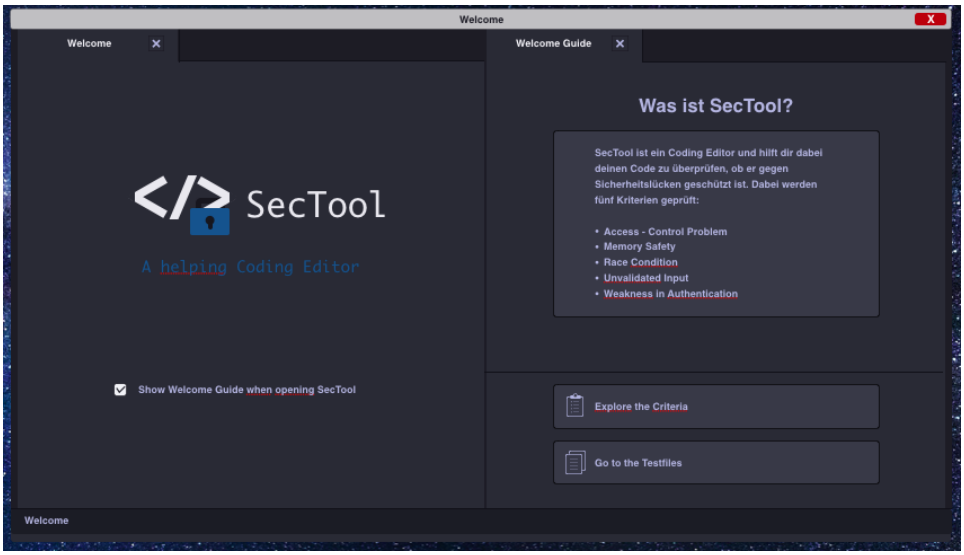

Abbildung 39: Verlinkte Kriterien für mehr Informationen

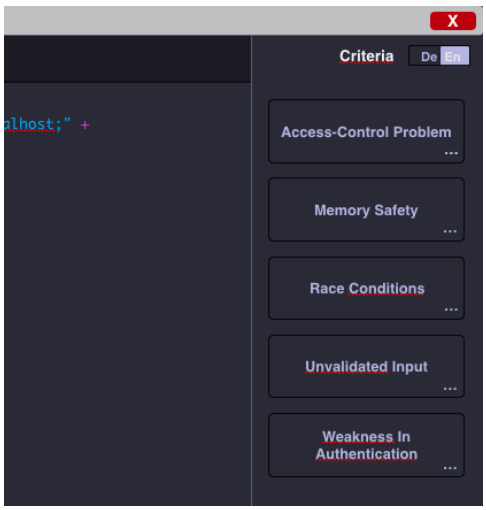

Abbildung 40: Spracheinstellung

6. Allgemeine Verbesserungen:

Die Fehlermeldung und Kriterien müssen ausführlicher erklärt werden, besonders für Programmierer ohne Vorwissen in Sachen Security. Eine Anmerkung von Testperson T4 bestand durch die Ergänzung weiterer Beispiele, um anhand dieser eine Erklärung vorzunehmen.
Um ein besseres Vertrauensgefühl zu erhalten sollte, wie in den Trust Guidelines beschrieben, dem Werkzeug ein Zertifikat hinzugefügt werden. Damit

hätten die Benutzer die Gewissheit, vom wem dieses Werkzeug erstellt wurde.

7 Fazit

In dieser Arbeit wurden Werkzeuge zur Codeanalyse betrachtet und festgestellt, wie weitreichend die Konsequenzen des Kopierens von Codeschnipseln aus z.B. Stack Overflow sein können.
Mit Hilfe einer eigenen Onlinestudie wurde aufgezeigt, wie selten Personen Werkzeuge zur Codeanalyse kannten und noch seltener diese selbst benutzen. Wenn sie denn ein vermeintliches Werkzeug verwendeten, dann stellt es sich meist als simpler Debugger heraus, der keinerlei Sicherheitslücken aufdeckt. Heutzutage wird besonders unter Studenten die Plattform Stack Overflow genutzt, ohne jedoch den kopierten Code zu überprüfen. Mit Hilfe dieser Onlineumfrage war es dann möglich Anforderung an ein benutzerfreundliches Werkzeug zu ermitteln.

Im Verlauf dieser Arbeit hat sich herausgestellt, dass ein neues Konzept eines Werkzeugs zur Codeanalyse sehr gut von Nutzern angenommen wird. Unterteilt in fünf Sicherheitskriterien wurde festgestellt, dass dieses Werkzeug einfach zu nutzen, verständlich und intuitiv zu bedienen ist. Mit Hilfe der positiven Rückmeldung einer Erfüllung eines Sicherheitskriteriums, zeigt es an, das bereits gewisse Lücken abgedeckt und erledigt wurden. Auf diese haben die Testpersonen sehr gut reagiert. Das Vorhandensein einer Quickfix Lösung wurde sehr begrüßt und sorgte für Erleichterung bei der Lösung des Sicherheitsproblems.

Nach Auswertung anhand der Grundsätze der Dialoggestaltung stellte sich das Werkzeug als sehr benutzerfreundlich heraus. Gleichwohl die Erklärungen zu den Kategorien und den beschriebenen Fehlermeldungen noch ausführlicher sein könnten, sowie eine nachvollziehbarere Quickfix Lösung implementiert werden sollte, bestätigten alle Testpersonen, solch ein Werkzeug zukünftige nutzen zu wollen.

Zur Festlegung inwiefern die Testpersonen die Kriterien verinnerlicht hatten, waren erstaunliche sieben von acht Personen in der Lage mindestens vier der fünf Kriterien aus dem Gedächtnis wiedergeben zu können und das bei einer Beschäftigung mit dem Werkzeug von gerade einmal ca. 30 min. Das ließ darauf schließen, dass eine intensivere Beschäftigung mit dem Werkzeug zu mehr Einfluss beim Nutzer führen wird und ihn bzgl. der Security kontinuierlich sensibilisiert.

Um dem Vertrauensproblem bzgl. der Quelle der Quickfix Lösungen entge-

gen zu wirken, sollte eine Zertifizierung des Werkzeugs vorgenommen werden, um zu bestätigen, dass sämtlicher Inhalt aus seriösen Quellen stammt.

Auch wenn noch Verbesserungspotential vorhanden ist, zeigt diese Abschlussarbeit auf, dass dieses neue Konzept eines Werkzeugs zur Codeanalyse eines ist, in welches weiter investiert werden kann, weil es gleichzeitig benutzerfreundlich, wie auch sensibilisierend bzgl. der Sicherheit ist. Denn gleich wie gut ein Werkzeug benutzt werden kann, nutzt es gar nichts, wenn ein Nutzer nicht die Notwendigkeit zur Achtung der Security sieht und es würde sich nichts an den heutigen Problemen ändern.

8 Ausblick

Als Endresultat hat sich herausgestellt, dass sich das Konzept dieses Werkzeugs zur Codeanalyse auf dem richtigen Weg befindet. Mit der Umsetzung der Verbesserungsvorschlägen könnte damit begonnen werden, dieses Werkzeug richtig zu implementieren. Damit wäre es möglich sowohl sämtliche Aspekte der Fehlertoleranz sowie der Individualisierbarkeit zu testen.

Erst diese reale Umsetzung würde aufzeigen, wie gut die Möglichkeiten eines Quickfixes sind und wie viel Zeit ein Checkvorgang einnehmen würde.

Die Wiedergabe der Kriterien zeigte bereits auf, dass einige Aspekte der Sicherheitskriterien bereits ins Gedächtnis aufgenommen wurden. In zukünftiger Arbeit wäre es interessant heraus zu finden, wie stark die Testpersonen bzgl. der Sicherheit sensibilisiert wurden. Eine Langzeitstudie könnte aufzeigen, wie lange die Nutzer die Kriterien im Kopf behalten haben und ob sie damit begonnen haben bei Untersuchungen ihres eigenen Quellcode nach dem gleichen Muster vorzugehen.

Um Gewissheit zu erhalten, ob bereits die Hinzugabe eines Zertifikates reicht, um genug Vertrauen in den Vorschlag der Quickfix Lösung oder in die gesamte Software zu haben, müsste eine erneute Untersuchung initiiert werden.

A Inhalt der CD-Rom

A.1 Masterarbeit

Pfad:/
MasterarbeitKostic2018.pdf

A.2 Werkzeuge

Pfad:/Werkzeuge/

- Prototyp_1_English
- Prototyp_2_German
- Prototyp_3_Mixt

Pfad:/Werkzeuge_PDF/

- Prototyp_1_English.pdf
- Prototyp_2_German.pdf
- Prototyp_3_Mixt.pdf

A.3 Umfrage

Pfad: /Umfrage/

- Ergebnisse
- Fragekatalog

A.4 Online_Umfrage

Pfad: /Online_Umfrage/

- Fragekatalog
- Rohdaten
- Rohdaten_PDF

Literaturverzeichnis

[ABF+16] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 289–305, May 2016.

[Aka17] Akamai. state of the internet-sicherheitsbericht. `/https://www.akamai.com/de/de/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-executive-summary.pdf`, 2017. Zugriff: 23.04.18.

[Alb03] Eirik Albrechtsen. Safety vs. security. `/http://www.iot.ntnu.no/users/albrecht/rapporter/notat%20safety%20v%20security.pdf`, 2003. Zugriff: 08.05.18.

[Atoa] Atom. A hackable text editor. `/https://atom.io`. Zugriff: 08.05.18.

[Atob] Atom. Improved language integration. `/https://ide.atom.io`. Zugriff: 08.05.18.

[Atoc] Atom. Introducing github for atom. `/https://github.atom.io`. Zugriff: 08.05.18.

[BLO+15] Zinaida Benenson, Gabriele Lenzini, Daniela Oliveira, Simon Parkin, and Sven Uebelacker. Maybe poor johnny really cannot encrypt: The case for a complexity theory for usable security. In *Proceedings of the 2015 New Security Paradigms Workshop*, NSPW '15, pages 85–99, New York, NY, USA, 2015. ACM.

[Bun16] Bundeskriminalamt. *Cybercrime - Bundeslagebild 2016*. Bundeskriminalamt, 2016.

[Bur03] M. Burmester. *Ist das wirklich gut? Bedeutung der Evaluation für die benutzerzentrierte Gestaltung.* In Machate, J. und Burmester, 2003.

[C.e17] C.ebra. Itk-benchmark: Unternehmen investieren verstärkt in it-security. `/https://www.cebra.biz/news/markt/13-09-2017-unternehmen-investieren-verstaerkt-in-it-security/`, 2017. Zugriff: 09.05.18.

[Che] Checkmarx. The ultimate list of open source static code analysis security tools. /https://www.checkmarx.com/2014/11/13/the-ultimate-list-of-open-source-static-code-analysis-security-tools/. Zugriff: 12.01.18.

[Cir] Survey Circle. Survey ranking. /www.surveycircle.com. Zugriff: 08.05.18.

[Coda] Visual Studio Code. Getting started. /https://code.visualstudio.com/docs. Zugriff: 08.05.18.

[Codb] Visual Studio Code. Programming languages. /https://code.visualstudio.com/docs/languages/overview. Zugriff: 08.05.18.

[Codc] Visual Studio Code. Using version contro in vs code. /https://code.visualstudio.com/Docs/editor/versioncontrol. Zugriff: 08.05.18.

[Con] The Web Application Security Consortium. Static code analysis list. /http://projects.webappsec.org/w/page/61622133/StaticCodeAnalysisList. Zugriff: 14.02.18.

[Cov] Coverity screenshot. /http://twimgs.com/ddj/images/article/2013/1213/COVERITY_adrian.gif. Zugriff: 08.05.18.

[Cro] G2 Crowd. The top 9 static code analysis software. /https://www.g2crowd.com/categories/static-code-analysis#highest_rated. Zugriff: 08.05.18.

[Cur] Curlie. Static checkers. /https://curlie.org/Computers/Programming/Languages/Java/Development_Tools/Performance_and_Testing/Static_Checkers. Zugriff: 20.02.18.

[CWEa] CWE. Buffer overflow. /http://cwe.mitre.org/data/definitions/120.html. Zugriff: 08.05.18.

[CWEb] CWE. Null pointer dereferenz. http://cwe.mitre.org/data-/definitions/476.html. Zugriff: 08.05.18.

[CWEc] CWE. Uninitialisierter speicher. /http://cwe.mitre.org/data/definitions/824.html. Zugriff: 08.05.18.

[CWEd] CWE. View the list of weaknesses. /http://cwe.mitre.org/index.html. Zugriff: 08.05.18.

[DB13] Gabriel Díaz and Juan Ramón Bermejo. Static analysis of source code security: Assessment of tools against samate tests. *Information & Software Technology*, 55(8):1462–1476, 2013.

[DR93] Joseph F. Dumas and Janice C. Redish. *A Practical Guide to Usability Testing*. Greenwood Publishing Group Inc., Westport, CT, USA, 1st edition, 1993.

[DW18] DW. Cyberangriffe aus drei jahrzehnten. `/http://www.dw.com/de/cyberangriffe-aus-drei-jahrzehnten/a-42779306`, 2018. Zugriff: 20.04.18.

[e.V16] Bitkom e.V. Spionage, sabotage und datendiebstahl wirtschaftsschutz im digitalen zeitalter. `/https://www.bitkom.org/noindex/Publikationen/2015/Studien/Studienbericht-Wirtschaftsschutz/150709-Studienbericht-Wirtschaftsschutz.pdf`, 2016. Zugriff: 23.04.18.

[e.V18] Bitkom e.V. Live security studie 2017/2018- eine repräsentative untersuchung von bitkom research im auftrag von f-secure. `/http://images.secure.f-secure.com/Web/FSecure/%7bff67047d-8c0d-418c-86dd-af4d8b1cbc5f%7d_F-Secure_Live_Security_Studie_2017_2018.pdf`, 2017/2018. Zugriff: 23.04.18.

[Exc] Stack Exchange. Stack overflow. `/https://stackexchange.com/sites`. Zugriff: 28.03.18.

[FBX+17] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy and paste on android application security. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017.

[Fra] Fragebogen.de. Aufbau der fragen. `/https://www.fragebogen.de/aufbau-der-fragen-bei-umfragen.htm`. Zugriff: 08.05.18.

[fS] ISO International Organization for Standardization. Ergonomic requirements for office work with visual display terminals (vdts). `/https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en`. Zugriff: 18.04.18.

[fSidIa] Bundesamt für Sicherheit in der Informationstechnik. Phishing. `/https://www.bsi-fuer-buerger.de/BSIFB/`

DE/Risiken/SpamPhishingCo/Phishing/phishing_node.html. Zugriff: 09.05.18.

[fSidIb] Bundesamt für Sicherheit in der Informationstechnik. Sicher informiert. /https://www.bsi-fuer-buerger.de/SharedDocs/Newsletter/DE/BSIFB/BuergerCERT-Newsletter/14_Sicher-Informiert_06-07-2017.html. Zugriff: 08.05.18.

[fSidIc] Bundesamt für Sicherheit in der Informationstechnik. Verteilte denial-of-service-attacken (ddos). /https://www.bsi-fuer-buerger.de/BSIFB/DE/Risiken/DoS/DDoS/DDoS.html;jsessionid=CE59743598080C5E91D6A2F91309EC5E.1_cid360. Zugriff: 08.05.18.

[fSidI17] Bundesamt für Sicherheit in der Informationstechnik. Die lage der it-sicherheit in deutschland 2017. /https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2017.pdf?__blob=publicationFile&v=3, 2017. Zugriff: 23.04.18.

[GC] Guide and Sample Code. Types of security vulnerabilities. /https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html#//apple_ref/doc/uid/TP40002529-SW2. Zugriff: 08.05.18.

[GL14] Simson Garfinkel and Heather Richter Lipford. *Usable Security: History, Themes, and Challenges*. Morgan & Claypool Publishers, 2014.

[Gro] Network Working Group. Internet security glossary. /https://tools.ietf.org/html/rfc2828. Zugriff: 08.05.18.

[hBN] hp Business Now. 35 statistiken zum thema internetsicherheit, die jeder it-manager 2017 kennen sollte. /https://www.hpbusinessnow.de/index.php/2017/06/29/35-statistiken-zum-thema-internetsicherheit-die-jeder-manager-2017-kennen-sollte/. Zugriff: 29.03.18.

[Hel] Software Testing Help. Top 40 static code analysis tools. /https://www.softwaretestinghelp.com/

tools/top-40-static-code-analysis-tools/. Zugriff: 12.01.18.

[Heu18] ZDF Heute. Hacker-gruppe äpt28". /https://www.zdf.de/nachrichten/heute/hacker-gruppe-apt28-100.html, 2018. Zugriff: 20.04.18.

[IGM] Tiago Gomes Ivo Gomes, Pedro Morgado and Rodrigo Moreira. An overview on the static code analysis approach in software development. /http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.734.6506&rep=rep1&type=pdf. Zugriff: 23.02.18.

[Inf15] TNS Infratest. D21-digital-index 2015 die gesellschaft in der digitalen transformation. /https://initiatived21.de/app/uploads/2017/01/d21_digital-index2015_web2.pdf, 2015. Zugriff: 09.05.2018.

[Ins] Security Insider. Was ist social engineering? /https://www.security-insider.de/was-ist-social-engineering-a-633582/. Zugriff: 08.05.18.

[ISO] Iso/iec 27000:2009 (e). (2009). information technology security techniques information security management systems overview and vocabulary. iso/iec.

[Jac] Life Jacker. Five best text editors. /https://lifehacker.com/five-best-text-editors-1564907215. Zugriff: 08.05.18.

[JSMHB13] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.

[Kos16] Sandra Kostic. Untersuchung der usability eines mobilen prototypen fu r die online-ausweisfunktion. Master's thesis, Freie Universität Berlin, 2016.

[Lab] Cheval Lab. Usabilitymethoden. /Link: http://www.cheval-lab.ch/was-ist-usability/usabilitymethoden/. Zugriff: 08.05.18.

[Lap] Lapseplus screenshot. /https://www.owasp.org/images/8/80/LapsePlusScreenshot.png. Zugriff: 08.05.18.

[Lig02] P. Liggesmeyer. *Software-Qualität*. Spektrum-Verlag, Heidelberg, Germany, 2002.

[Lig09] Peter Liggesmeyer. *Software-Qualität - Testen, Analysieren und Verifizieren von Software (2. Aufl.)*. Spektrum Akademischer Verlag, 2009.

[Liv] Liveedu. 10+ best text editors for programming 2016/2017. `/http://blog.liveedu.tv/10-best-text-editors-programming-2016/`. Zugriff: 08.05.18.

[Mar] Briggs Marsh. Designing systems that people will trust. Zugriff: 08.05.18.

[Nie] Jakob Nielsen. Why you only need to test with 5 users. `/https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`. Zugriff: 08.05.18.

[Nie94] Jakob Nielsen. Usability inspection methods. chapter Heuristic Evaluation, pages 25–62. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[NIS] NIST. Source code security analyzers. `/https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html`. Zugriff: 23.02.18.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002.

[Oeh17] Armin Born / Claudia Oehler. *Lernen mit Grundschulkindern*. kohlhammer, 2017.

[Onl] Heise Online. It-investitionen der deutschen unternehmen steigen. `/https://www.heise.de/newsticker/meldung/IT-Investitionen-der-deutschen-Unternehmen-steigen-101954.html`. Zugriff: 08.05.18.

[Onl17] Zeit Online. Merkel und der schicke bär. `/https://www.zeit.de/2017/20/cyberangriff-bundestag-fancy-bear-angela-merkel-hacker-russland`, 2017. Zugriff: 20.04.18.

[Ove] Stack Overflow. Welcome to stack overflow. `/https://stackoverflow.com/tour`. Zugriff: 27.02.18.

[OWAa] OWASP. Cross site scripting. `/https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`. Zugriff: 08.05.18.

[OWAb] OWASP. Cross-site scripting (xss). `/https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`. Zugriff: 25.04.18.

[OWAc] OWASP. Sql injection. `/https://www.owasp.org/index.php/SQL_Injection`. Zugriff: 08.05.18.

[OWAd] OWASP. Unvalidated input. `/https://www.owasp.org/index.php/Unvalidated_Input`. Zugriff: 08.05.18.

[OWAe] OWASP. Vulnerability scanning tools. `/https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools`. Zugriff: 20.02.18.

[Owaf] Owasp.org. Owasp lapse project. `/https://www.owasp.org/index.php/OWASP_LAPSE_Project`. Zugriff: 12.01.18.

[Owag] Owasp.org. Source code analysis tools. `/https://www.owasp.org/index.php/Source_Code_Analysis_Tools`. Zugriff: 12.02.18.

[Oxfa] Oxforddictionaries. safety. `/https://en.oxforddictionaries.com/definition/safety`. Zugriff: 08.05.18.

[Oxfb] Oxforddictionaries. Secure. `/https://en.oxforddictionaries.com/definition/secure`. Zugriff: 08.05.18.

[Oxfc] Oxforddictionaries. Security. `/https://en.oxforddictionaries.com/definition/security`. Zugriff: 08.05.18.

[PA] Jochen Prümper and Micael Anft. Beurteilung von software auf grundlage der internationalen ergonomie-norm din en iso 9241-110. Zugriff: 08.4.18.

[Pee] Peerly. Resource: A list of dynamic analysis tools for software. `/https://www.peerlyst.com/posts/resource-a-list-of-dynamic-analysis-tools-for-software-susan-parker`. Zugriff: 22.02.18.

[Pol] PollPool. Pollpool fragen beantworten. `/www.poll-pool.com`. Zugriff: 08.5.18.

[Por14] Rolf Porst. *Fragebogen - Ein Arbeitsbuch*. Springer-Verlag, Berlin Heidelberg New York, 4. aufl. edition, 2014.

[Psy] Lern Psychologie. Operantes konditionieren | details zu verstärkung und verstärkern. `/http://www.lern-psychologie.de/behavior/verstaerker.pdf`. Zugriff: 08.05.18.

[RF13] Michael Richter and Markus.D Flückiger. *Usability Engineering kompackt-Benutzbare Produkte gezielt entwickeln*. Springer-Verlag, Berlin Heidelberg New York, 3.aufl. edition, 2013.

[RiS] Risp screenshot. `/https://demo.ripstech.com/main/(issues/39/54/filter//sidebar:types/39/54/1)`. Zugriff: 08.05.18.

[RKB+13] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy W. van der Horst, and Kent E. Seamons. Confused johnny: when automatic encryption leads to confusion and mistakes. In *SOUPS*, 2013.

[Sch] Schneier. About bruce schneier. `/https://www.schneier.com/blog/about/`. Zugriff: 20.03.18.

[Sch00] Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2000.

[Seaa] SearchSecurity. Cookie poisoning. `/https://searchsecurity.techtarget.com/definition/cookie-poisoning`. Zugriff: 08.05.18.

[Seab] Searchsecurity. Static source code analysis tools: Pros and cons. `/https://searchsecurity.techtarget.com/answer/Static-source-code-analysis-tools-Pros-and-cons`. ZUgriff: 20.02.18.

[Sec] Committee on national security systems: National information assurance (ia) glossary, cnss instruction no. 4009, 26 april 2010.

[Ser16] IDG Research Service. Ciso security studie. `/https://www.infopoint-security.de/medien/idg-cisco-ciso_security_studie.pdf`, 2016. Zugriff: 08.05.18.

[Sny03] Carolyn Snyder. *Paper Prototyping - The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann, San Francisco, Calif, 2003.

[Staa] Statista. Bitkom. n.d. wie häufig ist ihr unternehmen it-angriffen ausgesetzt, durch die ihr unternehmen ausspioniert oder geschädigt werden soll? /https://de.statista.com/statistik/daten/studie/348989/umfrage/haeufigkeit-von-cyberattacken-auf-unternehmen/. Zugriff: 28.04.18.

[Stab] Statista. Bundeskriminalamt. n.d. polizeilich erfasste fälle von cyberkriminalität im engeren sinne* in deutschland von 2000 bis 2016. /https://de.statista.com/statistik/daten/studie/295265/umfrage/polizeilich-erfasste-faelle-von-cyberkriminalitaet-im-engeren-sinne-in-deutschland/. Zugriff: 09.05.2018.

[Sub] Sublime. Sublime text. /www.sublimetext.com. Zugriff: 08.05.18.

[Syn] Synopsys. Coverity scan static analysis. /https://scan.coverity.com. Zugriff: 08.05.18.

[t3n] t3n. Die beliebtesten 4 code-editoren im Überblick. /https://t3n.de/news/code-editoren-ueberblick-1007349/. Zugriff: 08.05.18.

[Tag18a] Tagesschau.de. Hackerangriff mit internationaler dimension. /https://www.tagesschau.de/inland/hacker-angriff-103.html, 2018. Zugriff: 09.05.18.

[Tag18b] Tagesschau.de. Von der uni ins ministerium? /https://www.tagesschau.de/inland/hackerangriff-bundesregierung-101~_origin-1c642af0-2d7e-4900-9559-a57662ad489c.html, 2018. 22.04.18.

[Tag18c] Tagesschau.de. Was wissen wir über den #bundeshack? /https://www.tagesschau.de/inland/faq-bundeshack-101~_origin-b4bd9e87-1c7e-45a3-baef-55b91b365162.html, 2018. Zugriff: 23.04.18.

[Teca] RIPS Tech. Poduct. /https://www.ripstech.com. Zugriff: 08.05.18.

[Tecb] RIPS Tech. Use cases. /https://www.ripstech.com/use-cases/. Zugriff: 08.05.18.

[Tecc] Techopedia. Dynamic application security testing (dast). /https://www.techopedia.com/definition/30958/dynamic-application-security-testing-dast. Zugriff: 12.02.18.

[tecd] techopedia. Static application security testing (sast). /https://www.techopedia.com/definition/30521/static-application-security-testing-sast. Zugriff: 12.02.18.

[Umb] Umbuzoo. Allgemeins. /https://www.umbuzoo.de/preise/. Zugriff: 10.01.18.

[Usa] Usabilityblog. Onlineforschung: Antworttendenzen, durchklicker und kontrollfragen. /https://www.usabilityblog.de/onlineforschung-antworttendenzen-durchklicker-und-kontrollfragen/. Zugriff: 08.05.18.

[vSYFBJAS94] Maarten W. van Someren Yvonne F. Barnard Jacobijn A.C. Sandberg. The think aloud method - a practical guide to modelling cognitive processes. /http://echo.iat.sfu.ca/library/vanSomeren_94_think_aloud_method.pdf, 1994. Zugriff: 08.05.18.

[Zei15] Süddeutsche Zeitung. Gesamtes it-netz des bundestages muss ausgetauscht werden. /http://www.sueddeutsche.de/politik/hackerangriff-auf-den-bundestag-gesamtes-it-netz-des-bundestages-muss-ausgetauscht-werden-1.2519934#redirectedFromLandingpage, 2015. Zugriff: 20.04.18.

[ZS96] Mary Ellen Zurko and Richard T. Simon. User-centered security. In *Proceedings of the 1996 Workshop on New Security Paradigms*, NSPW '96, pages 27–33, New York, NY, USA, 1996. ACM.

Abbildungsverzeichnis

1 Der Prozess des Code Review [DB13] 3
2 OWASP LAPSE Plus Screenshot [Lap] 13
3 Coverity Screenshot [Cov] 14
4 RISP Screenshot [RiS] 15
5 Sublime Screenshot [Sub] 16
6 Atom Screenshot [Atob] 17
7 Die Komponenten eines Mensch-Maschinen-Systems [RF13, S.5] 23
8 Bewertungsskala [PA] 31
9 Startbildschirm der Onlineumfrage 34
10 Liste der Onlineumfragen auf SurveyCircle [Cir] 37

11 Onlineumfragen auf PollPool [Pol] 38
12 PollCoins - Die Währung auf PollPool [Pol] 38
13 Ergebnisse zum Hochladen von Quellcode auf Stack Overflow 40
14 Sicherheitscheck mit einem Werkzeug durchgeführt 40
15 Code aus Stack Overflow kopieren und überprüfen 41
16 Testpersonen, welche Code kopiert und überprüft haben . . . 41
17 Wie fehlerfrei halten Sie den Code auf Stack Overflow? 43
18 Wie sicher halten Sie den Code auf Stack Overflow? 44
19 Beantwortung der Umfragen 46
20 Auswertbare Umfragen aus den Forschungsplattformen 47
21 Willkommensbildschirm 51
22 Icons und Logo 51
23 Was ist SecTool 52
24 Die Vorstellung der Kriterien 52
25 Bedieneroberfläche von SecTool 53
26 Erklärungen zu den Kriterien anzeigen lassen 54
27 Der Checkvorgang 54
28 Sicherheitslücken erkannt 55
29 Erklärung der Sicherheitslücke 56
30 Angebot einer Quickfix Lösung 56
31 Keine Sicherheitslücken entdeckt 57
32 Kriterien auf der Willkommensseite 62
33 Die Kriterien 63
34 Der Check Durchlauf 64
35 Anzeigen des Fehlers im Quellcode 65
36 Auswertung der gesamten Teilnehmer 69
37 Vorschaufenster im Quickfix 70
38 Check-Vorgang stoppen 71
39 Verlinkte Kriterien für mehr Informationen 72
40 Spracheinstellung 72