Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Einsatz von Machine-Learning-Methoden zur adaptiven Darstellung von Software-Metriken

Matthias Hermann

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. Stefan Wagner
Betreuer/in:	Dr. Michael Gebhart Jasmin Ramadani, M.Sc.
Beginn am:	02.11.2016
Beendet am:	02.05.2017
CR-Nummer:	D.2.8, D.2.10, G.1.6, I.2.6

Kurzfassung

Auf manchen SonarQube-Instanzen wird die verfügbare Fläche der Webseite nicht effizient genutzt und große Teile der Seite enthalten Leerflächen. Damit diese Flächen genutzt werden können, um genau die Informationen darzustellen, weswegen der Benutzer die Webseite aufgerufen hat, wurde im Rahmen dieser Arbeit mit *DeepSonar* eine adaptive Benutzeroberfläche für die Codeanalyse-Plattform SonarQube entwickelt. Diese erlernt mittels Machine-Learning die für den aktuellen Benutzer und Nutzungskontext relevantesten Informationen, d. h. die aus einer Programmcodeanalyse resultierenden Software-Metriken. Anhand der Ergebnisse des Machine-Learnings wird die Weboberfläche von SonarQube angepasst, sodass diese Metriken in der davor ungenutzten Fläche auf der Startseite angezeigt werden.

Inhaltsverzeichnis

1 Einleitung

In der Softwareentwicklung kommt ein Entwickler heutzutage mit vielen Tools, die ihn bei der Arbeit unterstützen sollen, und deren Weboberflächen in Kontakt. Ziel dieser Anwendungen ist es, dem Entwickler die Informationen, welche er im Moment braucht – z. B. verschiedene Software-Metriken – und weswegen er die jeweilige Anwendung aufgerufen hat, schnell und übersichtlich zu vermitteln. Eine weitere Gemeinsamkeit ist, abgesehen von der Informationsvermittlung, dass auf den Startseiten dieser Anwendungen nur sehr allgemeine Informationen zu finden sind. Benötigt ein Benutzer detailliertere Informationen, muss er erst durch einige Seiten navigieren, denn erst auf den von den Startseiten aus erreichbaren Seiten findet ein Entwickler die für ihn im Moment wichtigen Informationen, die er auch sucht. Dabei bedeutet das Navigieren zu der gesuchten Seite jedoch einen Mehraufwand, welcher vor allem bei kleinen Mengen an Informationen, unverhältnismäßig erscheint. Aus diesem Grund bietet es sich für einen Entwickler an, wichtige Informationen zusätzlich schon auf der Startseite anzuzeigen, um das Navigieren durch eine Weboberfläche zu minimieren. Dennoch wäre eine derartige Anpassung der Oberfläche, selbst wenn die Einstellungsmöglichkeiten dafür gegeben wären, ein zusätzlicher Arbeitsaufwand. Dadurch ergibt sich der Wunsch ein Programm einzusetzen, welches erlernt auf welche Informationen häufig zugegriffen wird und diese Anpassung von selbst durchführt.

Die Lernfähigkeit ist von noch größerer Bedeutung unter dem Gesichtspunkt, dass jedes Projekt und jeder Benutzer unterschiedlich sind. Deswegen kann bei der Betrachtung der Relevanz bestimmter Informationen nicht immer der gleiche Maßstab angelegt und auch keine allgemeingültige Lösung produziert werden. Zusätzlich ist das Bestimmen der momentanen Relevanz von Informationen ein sehr dynamisches Problem, da sich der Zustand eines Projektes andauernd ändern kann und die Krisenherde, die der Benutzer mithilfe des Tools finden möchte, wechseln können. Je nachdem wo die Krisenherde aktuell liegen sind andere Software-Metriken von Bedeutung. Sämtliche Informationen bereits auf einer Startseite anzuzeigen ist dabei auch keine Lösung, denn

werden zu viele Informationen auf einer Seite angezeigt, ist es schwer die Übersicht zu behalten und die wirklich Wichtigen zu finden.

Wenn, wie oben beschrieben, auf den Startseiten einer Webanwendung nur sehr allgemeine Informationen mit relativ niedriger Relevanz für den Benutzer angezeigt werden und leerer Platz vorhanden ist, bietet es sich ohnehin an, diesen Platz für besonders relevante Informationen zu nutzen. Damit zeigt sich, dass nicht nur um den Navigationsaufwand zu senken, sondern auch um eine effizientere Nutzung des Platzes auf der Startseite zu ermöglichen, versucht werden sollte Informationen von verlinkten Seiten zusätzlich auf der Startseite anzuzeigen, um diese früher verfügbar zu machen. Falls den Benutzer im Moment ohnehin nur etwas Spezielles interessiert, würde er sich dadurch einen weiteren Seitenaufruf sparen.

Für solche Anpassungen ist es jedoch nötig Arbeitszeit aufzuwenden. In Projekten herrscht jedoch immer Zeitmangel, wodurch eine Hürde für solche Anpassungen existiert. Selbst wenn ein Anwender sich die Mühe gemacht hat seine Startseite einmal für sich einzurichten, kann sich die Relevanz der einzelnen Informationen durch die Weiterarbeit am Projekt, wie oben bereits beschrieben, verändern. Um diesen wiederkehrenden, manuellen Arbeitsaufwand zu vermeiden, wird eine im Hintergrund, automatisch ablaufende, lernende Softwarelösung, die eine Anpassungen der Weboberfläche selbstständig durchführt, benötigt.

1.1 Problemstellung

An dieser Stelle sollen die angedeuteten Probleme eindeutig definiert werden, um daraus Ziele und Anforderungen für die Beiträge dieser Arbeit ableiten zu können.

PS1 **Beeinflussende Kontextfaktoren bzgl. der Relevanz von Software-Metriken**

Es ist unklar, durch welche Kontextfaktoren die Relevanz einer Software-Metrik für einen Softwareentwickler bestimmt wird. Die Relevanz entscheidet jedoch darüber, ob das Widget einer Metrik bereits auf der Startseite angezeigt werden soll oder nicht. Die Vermutung ist, dass es verschiedene Gruppen von Faktoren gibt, welche teilweise abhängig, teilweise unabhängig voneinander sind. Um eine zufriedenstellende Anpassung an einen Benutzer und effektives Lernen durchführen zu können, ist ein grundlegendes Verständnis der zugrunde liegen-

den Einflüsse nötig. Dazu zählen die Anzahl der Kontextfaktoren, die spezifischen Einflüsse selber sowie ihre Bedeutung.

Dabei muss jedoch auch die Verwendbarkeit berücksichtigt werden, denn nicht alle Faktoren lassen sich auf triviale Weise erfassen oder sind für eine Anwendung in einem Kontext mit fehlenden Sensoren sogar schlichtweg unmöglich zu erfassen.

PS2 **Methoden des Machine Learning**

Die Auswahl eines Machine-Learning-Algorithmus hat abhängig von der Struktur des zugrunde liegenden Problems erheblichen Einfluss auf die Qualität der Ergebnisse. Die Vorgehensweise des Algorithmus muss daher auch zu der Struktur des Problems und den verfügbaren Eingaben passen.

Feature Selection und Preprocessing sind weitere Punkte, die es im Zusammenhang mit der Methodik des Machine Learnings zu beachten gilt. Irrelevante Merkmale der zu klassifizierenden Objekte stellen Störeinflüsse dar und verschlechtern die Performanz des Algorithmus. Daher kann es nötig sein diesem Effekt durch Preprocessing und Feature Selection entgegenzuwirken.

PS3 **Anpassung einer Benutzeroberfläche**

Eine Anpassung einer Benutzeroberfläche hat einen Einfluss auf die Benutzbarkeit für einen Anwender. Um diese Anpassung angemessen und vorteilhaft zu gestalten, muss ein Konzept ausgearbeitet werden, damit Änderungen keine negativen Auswirkungen auf die Benutzbarkeit haben sowie die Erkenntnisse des Machine Learning widerspiegeln und auch effektiv ausnutzen. Außerdem sollen Anpassungen vom Benutzer nicht als optisch störend wahrgenommen werden.

1.2 Beiträge dieser Bachelorarbeit

Ziel dieser Arbeit ist es, eine wie zu Beginn des Kapitels beschriebene adaptive Benutzeroberfläche zu entwickeln, welche die soeben definierten Problemstellungen löst. Diese soll aus einer Menge zur Verfügung stehender Software-Metriken die für den Benutzer relevantesten auswählen und die entsprechenden Widgets bereits auf der Startseite der Anwendung anzeigen. Daraus ergeben sich die folgenden Beiträge, welche diese Arbeit leisten:

B1 **Bestimmung der Kontextfaktoren**

Um im Regelfall der alltäglichen Arbeit die normalerweise nützlichsten Informationen anzeigen zu können, muss der Nutzungskontext inkl. Benutzerverhalten bei der Interaktion mit der Weboberfläche des Tools beobachtet und analysiert werden. Dafür muss ergründet werden, welche Benutzerdaten dafür von Bedeutung sind und wie diese erfasst werden können.

Auf die Wichtigkeit einer Information haben womöglich auch noch weitere Faktoren einen Einfluss. Beispielsweise könnte das Alter der Information, wie auch externe Faktoren, wie z. B. das aktuelle Datum oder die Tageszeit eine Rolle spielen. Daher müssen auch diese nicht-benutzerbezogenen Kontextfaktoren in Erfahrung gebracht werden.

B2 **Machine-Learning-Konzept zur Verarbeitung der Daten**

Es muss ein Konzept ausgearbeitet werden, wie durch Machine-Learning die von einem Benutzer gewonnenen Daten sowie Daten zu nicht-benutzerbezogenen Kontextfaktoren verarbeitet und daraus für den aktuellen Kontext Bewertungen für die verschiedenen Software-Metriken abgeleitet werden können.

B3 **Implementierung des Demonstrators**

Das Lernen von Benutzerpräferenzen und Einflüssen der Kontextfaktoren soll in einem lauffähigen Programm implementiert werden. Das Programm soll die dazu benötigten Daten einlesen, mit dem dafür entwickelten Machine-Learning-Konzept verarbeiten und für die verfügbaren Metriken Bewertungen für deren Relevanz ausgeben.

Die durch das Machine-Learning erhaltenen Ergebnisse sollen im letzten Schritt dazu verwendet werden anhand dieser Bewertungen die Weboberfläche der Anwendung so anzupassen, dass die wichtigsten Informationen bereits auf der Startseite angezeigt werden. Dabei sollen nicht ausreichend wichtige Informationen überhaupt nicht auf der Startseite angezeigt werden, um die Seite übersichtlich gestalten zu können.

Der Fokus dieser Arbeit liegt darauf, anhand der voreingestellt in ihren Widgets angezeigten Metriken zu entscheiden, welche davon für den Benutzer im Moment am

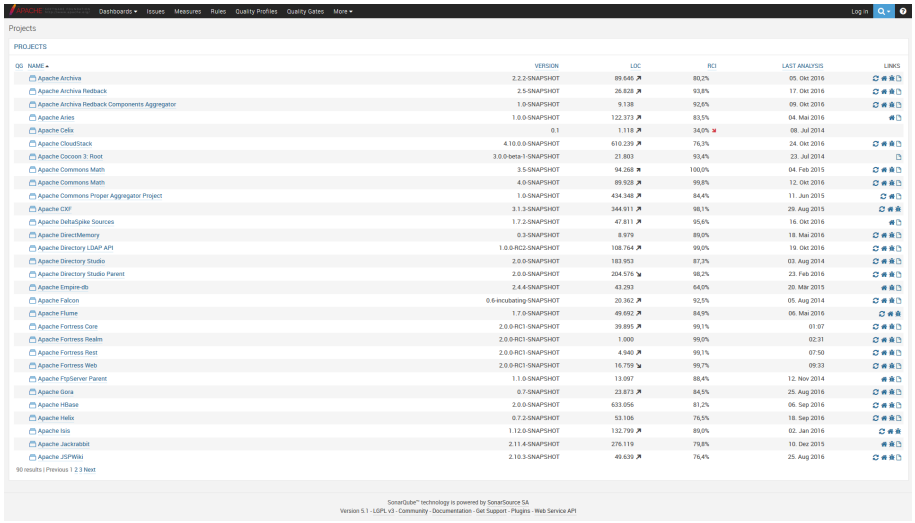


Abbildung 1.1: SonarQube Startseite der Apache Software Foundation

relevantesten zu sein scheinen. Dabei ist es ausdrücklich nicht das Ziel auch Widgets zu beachten, welche zwar durch das Ändern von Einstellungen angezeigt werden könnten, aber ansonsten nicht angezeigt werden. Da diese für den alltäglichen Gebrauch nicht zwangsläufig wichtig sein müssen und es auch das Ziel ist nur die wichtigsten Daten, aber dafür übersichtlich, darzustellen, würde das Ausprobieren von ungenutzten Widgets, welche erst nach einer Änderung der Einstellungen angezeigt würden, dem entgegenwirken und die angezeigte Seite unübersichtlicher werden lassen.

1.3 Beschreibung des Demonstrators

Für die Zwecke dieser Arbeit dient SonarQube [3] als Beispielanwendung. Die Open-Source Software-Plattform SonarQube vereint mehrere Werkzeuge zur statischen Codeanalyse, um dabei zu helfen die technische Qualität von Programmcode auf Grundlage von Software-Metriken zu verbessern und die sog. technische Schuld zu minimieren.

Der Quellcode wird dazu auf einem Server analysiert und die aus der Analyse gewonnenen Daten werden optisch aufbereitet und wie in Abbildung 1.3 auf einer Webseite dargestellt. Diese enthält zahlreiche Widgets, wie in Abbildung 1.2 zu sehen, welche die einzelnen Software-Metriken, thematisch gruppiert, beinhalten. Auf diesem sogenannten „Projekt-Dashboard“ können die einzelnen Metriken ausgewählt und dadurch

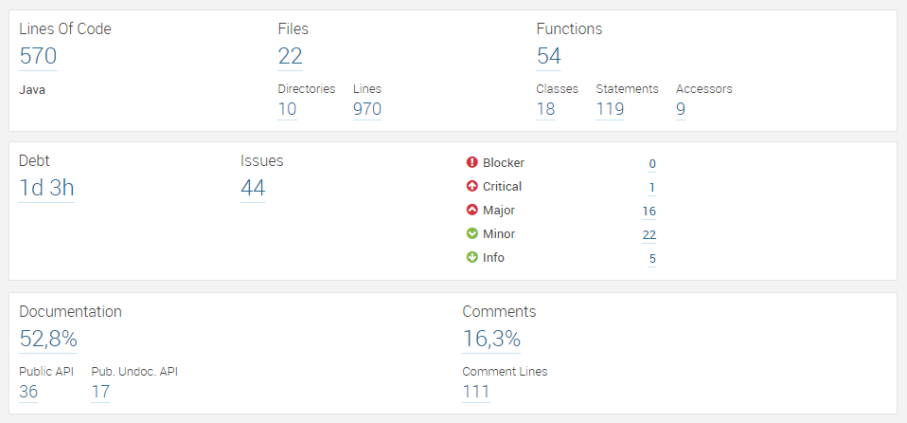


Abbildung 1.2: Drei Widgets von einem Projekt-Dashboard. Die dargestellten Widgets enthalten u.a. die Metriken „Lines of Code“, „Debt“ und „Documentation“

detailliertere Informationen und Erklärungen angezeigt werden. Ein solches Dashboard ist in Abbildung 1.3 zu sehen. Das erklärte Ziel von SonarQube ist es dabei, seinen Benutzern mit diesen Hilfsmitteln zu ermöglichen die Softwarequalität in ihren Projekten zu verbessern.

Auf der Startseite der Weboberfläche werden, wie bereits erwähnt, typischerweise allgemeine Informationen präsentiert. Viele öffentlich einsehbare Instanzen von Open-Source-Projekten und -Organisationen [1] zeigen z. B. nur eine Liste der verfügbaren Projekte bzw. Module und teilweise noch eine Übersichtsgrafik über den allgemeinen Zustand des Programmcodes der einzelnen Projekte und Module. Am Beispiel der SonarQube-Instanz der Apache Software Foundation in Abbildung 1.1 ist das sehr gut zu sehen, da hier die gesamte Startseite aus einer Auflistung der Projekte besteht.

Das Dashboard der einzelnen Projekte enthält typischerweise mehrere unterschiedliche, voneinander getrennte Widgets. Da auf einem solchen Dashboard einige solcher Widgets vorzufinden sind und sich somit eine ausgeprägte Modularität ergibt, können dadurch sinnvoll feingranulare Anpassungen an der Benutzeroberfläche vorgenommen werden. Diese Flexibilität macht es daher zu einem guten Anschauungsobjekt für diese Arbeit. Die grundlegenden Konzepte dieser Arbeit sind jedoch auch auf andere Anwendungen übertragbar.

Zur Demonstration der in dieser Arbeit beschriebenen Konzepte, werden diese auf die Software-Plattform SonarQube angewandt. Dafür werden Einflussfaktoren für die Relevanz einzelner Widgets eines Projekt-Dashboards analysiert. Die Machine-Learning-

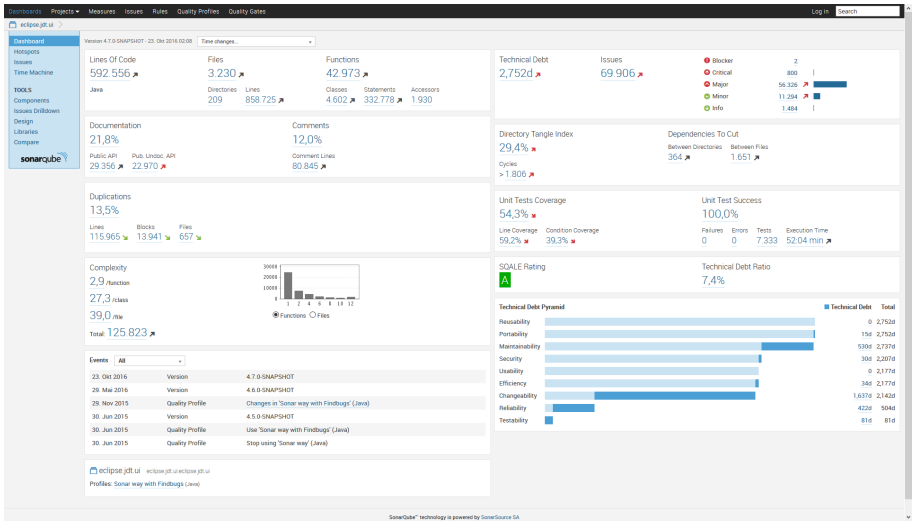


Abbildung 1.3: Übersicht von Widgets mit Software-Metriken des Programmoduls „eclipse.jdt.ui“ des Eclipse-Projektes in SonarQube.

Komponente lernt anhand des gewonnenen Wissens über die Einflussfaktoren die aktuelle Relevanz der einzelnen Metriken bzw. Widgets. Die erlernte Relevanz wird im nächsten Schritt dafür verwendet die Startseite der Webanwendung so anzupassen, dass die Widgets der relevantesten Metriken – zusätzlich zum Projekt-Dashboard – bereits auf der Startseite angezeigt werden.

In Abbildung 1.4 sind diese Anpassungen beispielhaft manuell durchgeführt worden. Der ursprüngliche Inhalt, die Liste an Projekten, wurde auf eine Spalte verschmälert, um Platz für Softwaremetriken zu schaffen. In diesem Beispiel wird angenommen, dass der Benutzer durch sein Verhalten hat darauf schließen lassen, dass die dargestellten Metriken im aktuellen Kontext am relevantesten für ihn sind. Die Grundlage für diese Annahmen werden die in Kapitel 4 beschriebenen Kontextfaktoren und das in Kapitel 5.1.1 beschriebene implizite Feedback sein, anhand derer ein Machine-Learning-Algorithmus erlernt, welche Software-Metriken den Benutzer am meisten interessieren könnten. Basierend auf dessen Auswahl an Metriken, werden die entsprechenden Widgets wie in Abbildung 1.4 auf der Startseite angezeigt. Wie in der Abbildung ebenfalls zu sehen ist, fügen sich die eingefügten Metriken nahtlos in das Erscheinungsbild der Seite ein. Außerdem ist zu beachten, dass die Hyperlinks in den einzelnen Widgets, wie auf dem Projekt-Dashboard, zu weiteren, detaillierteren Ansichten führen und somit keine Funktionalität verloren geht.

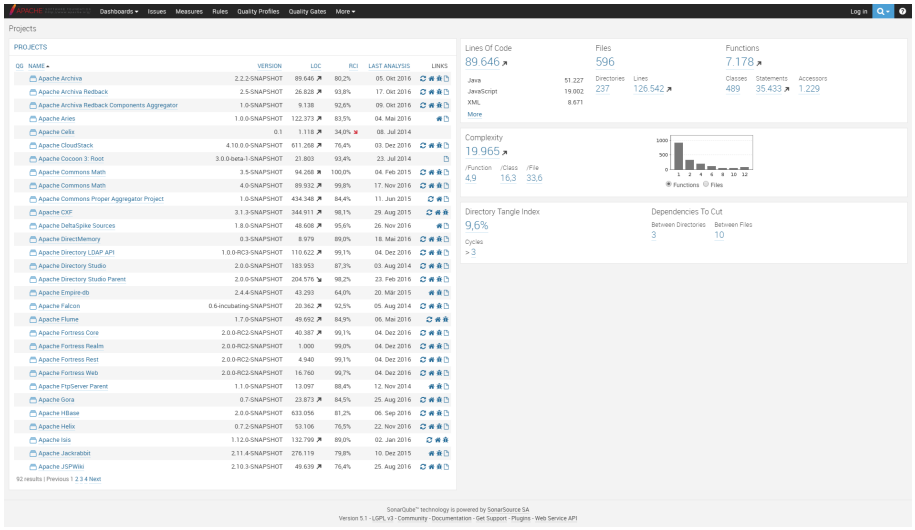


Abbildung 1.4: SonarQube Startseite der Apache Software Foundation mit eingefügten Widgets aus dem Projekt Apache Archiva.

1.4 Gliederung der Arbeit

Der folgende Abschnitt gibt einen kurzen Überblick über diese Arbeit. Kapitel 2 erklärt einige Grundlagen, welche für das Verständnis dieser Arbeit hilfreich sind. Kapitel 3 enthält einen Überblick über bisherige Forschungsarbeiten zum Thema adaptive Benutzeroberflächen, Anforderungen an das Ergebnis dieser Arbeit und inwiefern die bisherige Forschung diesen Anforderungen entspricht. In Kapitel 4 wird ein Konzept zur Erfassung der Benutzerpräferenzen und des Nutzungskontexts erarbeitet. Kapitel 5 behandelt das Machine-Learning-Modul, welches die Daten zum Benutzerverhalten sowie den Nutzungskontext als Eingabe nimmt, und daraus eine nach Relevanz sortierte Liste von Widgets ausgibt. In Kapitel 6 wird die Anpassung der Weboberfläche auf Basis solcher Listen beschrieben. In Kapitel 7 wird demonstriert, dass die in den vorherigen Kapiteln erarbeiteten Konzepte auch tatsächlich so in der Praxis umgesetzt werden können und effektiv funktionieren. Mit Kapitel 8 folgt eine Zusammenfassung der Arbeit und ein Ausblick auf Ansatzpunkte für weitere Forschungsarbeiten.

2 Grundlagen

Die zentralen Themen dieser Arbeit sind Machine-Learning und die Adaption einer Benutzeroberfläche, genauer gesagt einer Weboberfläche. Auch das Konzept allgemeiner Optimierungsprobleme wird in dieser Arbeit benötigt. Damit die in den nächsten Kapiteln verwendeten Konzepte leichter verständlich sind, bieten die folgenden Abschnitte grundlegende Erklärungen zu diesen Themen, denn sie werden dazu verwendet, um die Weboberfläche von SonarQube, wie in Kapitel 1 beschrieben, an den Benutzer anzupassen.

2.1 Machine-Learning

Ein Machine-Learning-Algorithmus approximiert eine Funktion $y(x)$, welche versucht Eingabedaten entsprechend ihrer Attribute in Klassen einzuteilen. Die konkrete Zuordnung von Werten x und der Klassifikation der Werte $y(x)$ wird durch den Lernprozess bestimmt. Hierbei ist zwischen überwachtem und unüberwachtem Lernen zu unterscheiden [13]. Um diese Klassifikatorfunktion $y(x)$ zu erlernen, wird bei einem überwachten Lernprozess eine kleine Teilmenge der möglichen Eingabedaten, deren korrekte Klassifikation bekannt ist, das sog. Trainingsset, verwendet. Dabei werden Parameter der Funktion variiert, um die für das Trainingsset erreichte Rate an korrekten Klassifikationen zu maximieren. Bei einem unüberwachten Lernprozess ist das Ziel vielmehr das Finden von davor unbekannten Gemeinsamkeiten in den Daten, als das Einteilen in vorgegebene Klassen. Daher enthalten die Trainingsdaten bei dieser Art von Machine-Learning-Verfahren auch keine vorgegebenen Zielklassen [13].

2.2 Allgemeine Optimierungsprobleme

Als allgemeines Optimierungsproblem – auch nichtlineares Optimierungsproblem – wird das Finden bzw. Approximieren eines Minimums einer reellwertigen Funktion

$$f(x_1, x_2, \ldots, x_n)$$

bezeichnet, wobei optionale Nebenbedingungen den Lösungsraum einschränken können [47]. Hierbei ist zwischen lokalen und globalen Minima zu unterscheiden, denn in Abhängigkeit der optionalen Nebenbedingungen kann ein globales Minimum der Funktion f diese Nebenbedingungen eventuell nicht erfüllen, aber ein lokales Minimum ist unter Umständen schlechter als das globale Minimum [47]. Durch Umkehren der Vorzeichen der Funktion f kann das Optimierungsproblem auch als Maximierungs- statt Minimierungsproblem definiert werden.

2.3 Webtechnologien

Die Ergebnisse des Machine-Learnings sollen schließlich dafür verwendet werden die Weboberfläche von SonarQube anzupassen. Eine Weboberfläche besteht heutzutage oftmals aus HTML-, CSS- und Javascript-Anteilen. HTML steht für *Hypertext Markup Language*. Die HTML-Dokumente einer Webseite kodieren die logische Struktur und den Aufbau der Seite, das sogenannte *Document Object Model* (DOM). Sie bestimmen welche Komponenten auf der Seite existieren und legen auch fest wie diese relativ zueinander positioniert sind. Weitere Anpassungen des Stylings der Seite können mithilfe von *Cascading Style Sheets* (CSS) vorgenommen werden. Diese CSS-Dateien enthalten genauere Vorgaben dazu, wie gewisse Elemente auszusehen haben. Dazu zählen beispielsweise die Farbe oder Größe von Elementen, Schriftarten und -größen sowie Abstände zwischen Elementen [27]. Client-seitiges Javascript ermöglicht andernfalls statischen Webseiten, auf Aktionen des Benutzers zu reagieren und Inhalte der Seite dynamisch nachzuladen oder zu verändern [26].

3 Verwandte Arbeiten

3.1 Anforderungen

Das Ergebnis dieser Arbeit soll eine adaptive Benutzeroberfläche sein, die aus dem Benutzerverhalten und möglichen weiteren Kontextfaktoren lernt, welche Informationen – konkret sind das die Software-Metriken aus SonarQube – für den Benutzer von Bedeutung sind. Die folgenden Anforderungen spezifizieren Bewertungsmaßstäbe anhand derer andere Forschungsarbeiten zu diesem Thema bewertet werden. Tabelle 3.1 fasst diese Anforderungen in einem Anforderungskatalog zusammen.

A1 **Unbemerktes Erfassen des Benutzerverhaltens und -kontexts**

Daten über das Benutzerverhalten sowie den Nutzungskontext werden ohne Zutun des Benutzers und ohne dass dieser etwas davon merkt gesammelt. Daher geschieht das Erfassen des Benutzerverhaltens sowie anderer Kontextfaktoren im Hintergrund ohne dabei Einbrüche der Performanz zu verursachen, um den entsprechenden Punkt in Problemstellung PS1 zu erfüllen.

A2 **Lernen anhand der Daten zu Benutzerverhalten und Nutzungskontext**

Die Relevanz der Informationen wird nur anhand des Benutzerverhaltens und -kontextes erlernt werden. Diese Einschränkung ist nötig, um nur möglichst relevante und auch erfassbare Daten entsprechend Problemstellung PS1 zu sammeln. Die zum Lernen verwendete Machine-Learning-Methode soll dabei die in Problemstellung PS2 genannten Kriterien erfüllen.

A3 **Anpassung der Benutzeroberfläche an den Benutzer**

Die Weboberfläche wird auf Basis der vom Machine-Learning-Modul gelieferten Ergebnisse an den spezifischen Benutzer angepasst, um dessen Interaktion mit der Anwendung, wie in Problemstellung PS3 erwähnt, effektiver zu gestalten.

A4 **Erhalten des Seitendesigns**

Bei Anpassungen der Benutzeroberfläche wird das bisherige Design erhalten und es werden keine Brüche eingefügt. Änderungen fügen sich nahtlos in den Rest der Benutzeroberfläche ein, sodass der Benutzer sie nicht als störend wahrnimmt, um Problemstellung PS3 gerecht zu werden.

A5 **Lernen & Adaption während des laufenden Betriebs**

Die Anpassung der Benutzeroberfläche geschieht im laufenden Betrieb und in Echtzeit, um schnell agieren zu können. Das Lernen der Benutzerpräferenzen findet auch während des laufenden Betriebs statt und muss nicht aufgrund von Rechenleistungsintensität zu einem anderen Zeitpunkt, in dem die eigentliche Anwendung nicht verwendet werden kann, durchgeführt werden. Dabei soll die Performanz der Anwendung selbst dauerhaft erhalten bleiben und darf nicht merklich abnehmen.

Name	Beschreibung
A1	Unbemerktes Erfassen des Benutzerverhaltens und Nutzungskontexts
A2	Lernen anhand der Daten zu Benutzerverhalten und Nutzungskontext
A3	Anpassung der Benutzeroberfläche an den Benutzer
A4	Erhalten des Seitendesigns
A5	Lernen & Adaption während des laufenden Betriebs

Tabelle 3.1: Anforderungskatalog zur Bewertung existierender Forschungsarbeiten zu adaptiven Benutzeroberflächen

3.2 Bewertung verwandter Arbeiten

Dieses Unterkapitel widmet sich dem Vergleich bisheriger Arbeiten im Bereich adaptiver Benutzeroberflächen. Als Bewertungsmaßstab werden die in Kapitel 3.1 beschriebenen

Anforderungen verwendet, um festzustellen inwieweit die vorgestellten Arbeiten und Teile davon der Zielformulierung dieser Arbeit entsprechen.

3.2.1 Learning Styles Diagnosis Based on User Interface Behaviors for the Customization of Learning Interfaces in an Intelligent Tutoring System

Der Kontext der Arbeit von Cha et al. [16] ist die Entwicklung einer intelligenten Benutzeroberfläche für eine Lernumgebung. Die Grundlage dafür, wie die Oberfläche angepasst wird, ist das *Index of Learning Style*-Modell (ILS) nach Felder & Silverman [25]. Dieses Modell beschreibt verschiedene Lerntypen mit unterschiedlichen Lernweisen. Durch eine adaptive Benutzeroberfläche und die Anpassung an die individuellen Lernweisen wird ein effizienterer Lernprozess für den Lernenden erwartet.

Das ILS besteht aus insgesamt vier Dimensionen [16]:

- Global (G) versus sequentiell (Q) in Bezug auf den Verstehensprozess von Informationen
- Visuell (V) versus auditiv (A) in Bezug auf die Aufnahme von Informationen
- Sensorisch (S) versus intuitiv (N) In Bezug auf die Wahrnehmung von Informationen
- Aktiv (C) versus reflexiv (R) in Bezug auf die Verarbeitung von Informationen

Demnach sind „globale“ Lerner an einer Übersicht über alle Themen interessiert, wohingegen „sequentielle“ Lerner sich an die Reihenfolge der Lektionen halten und dadurch andere Navigationsmöglichkeiten bevorzugen. Ähnliche Unterschiede gibt es auch in der Dimension von „visuellen“ und „auditiven“ Lernern, wobei erstere Bilder und Demonstrationen und letztere Texte oder Erklärungen bevorzugen. In der dritten Dimension unterscheidet sich ein „sensorischer“ Lerner durch ein höheres Interesse an Details und ein gründlicheres, aber dafür langsameres, Vorgehen vom „intuitiven“ Typ. Zu guter Letzt zeichnet sich ein „aktiver“ Lerner durch eine Vorliebe für Experimente und Diskussionen aus, während ein „reflektiver“ Lerner es bevorzugt für sich Meinungen anderer Lerner oder Experten zu reflektieren.

Um festzustellen in welche Kategorien ein Lerner fällt, werden die Art der Interaktion, also z. B. welche Navigationswege oder Typen von Lernressourcen verwendet werden, als Eingabesequenz bzw. als quantitativer Wert, wie z. B. die Klickfrequenz auf ein bestimmtes GUI-Element, erfasst und in einer XML-Datei gespeichert. Für die initiale Menge an Trainingsdaten wird der Fragebogen des ILS-Modells verwendet, um für die erhaltenen Daten eine Klassifikation zu ermöglichen.

Die quantitativen Daten sind dabei besser dazu geeignet anhand eines Entscheidungsbaums (engl.: Decision Tree, kurz DT) klassifiziert zu werden, während die sequentiellen Daten prädestiniert dafür sind von einem Hidden-Markow-Modell (HMM) klassifiziert zu werden. Für die Analyse wurden dennoch beide Methoden in allen Dimensionen getestet und dann überprüft welche Methode in welchen Dimensionen zuverlässiger ist. Demnach sind HMMs besser für die „G vs. Q"-Dimension, während DTs besser für die „V vs. A"-Dimension geeignet sind. In der „S vs. N"-Dimension hängt die Entscheidung hingegen von den Werten des zu klassifizierenden Lerners ab und in der „C vs. R"-Dimension ist eine Einordnung aufgrund von zu wenig Trainingsdaten sehr schwer.

Bewertung der Arbeit Cha et al. [16] erfassen die Aktionen des Benutzers dadurch, dass automatisch softwareseitig die Benutzereingaben verfolgt werden. Das Sammeln der Daten geschieht dabei vollkommen transparent und unbemerkt, womit die Anforderung A1 erfüllt ist. Das Machine-Learning arbeitet jedoch nur auf dem Benutzerverhalten und lässt den Kontext, in dem sich der Benutzer befindet, außer Acht. Weiterhin werden auch noch Vergleichsdaten durch den ILS-Fragebogen benötigt, um die Benutzerdaten klassifizieren zu können, weshalb es sich um ein überwachtes Lernen handelt. Daher wird Anforderung A2 nicht erfüllt. Der Fokus der Arbeit lag auf der Diagnose der Lerntypen, weshalb die Anpassung der Benutzeroberfläche (A3 & A4) nicht beurteilt werden kann. Ein Prototyp des verwendeten Systems wird jedoch in [31] vorgestellt. Außerdem ist nicht klar, ob die Klassifizierung eines Benutzers während des laufenden Betriebs (A5) oder erst danach stattfindet.

3.2.2 Design for Adaptive User Interface for Modeling Students' Learning Styles

Mbiliny et al. [34] beschreibt eine adaptive Benutzeroberfläche, um damit das Finden von Navigationspfaden in e-Learning-Systemen zu erleichtern und nicht um den Inhalt auf den Benutzer zuzuschneiden. Dafür wird der Lerntyp des Benutzers gelernt und dadurch die Reihenfolge bestimmt, in der Lernmodule angezeigt werden. Dabei fokussierten sie sich auf die Fragen, welche Daten der Benutzerinteraktion zum Erkennen des Verhaltensmusters und damit des Lerntyps benötigt werden, sowie welche Eigenschaften des Benutzerinterfaces diese Lerntypen besser adaptieren kann.

Als Modell für die Lerntypen der Benutzer wird das Modell von Kolb [51] verwendet. Es kennt im Gegensatz zum Modell aus [16] nur zwei Dimensionen und somit vier Lerntypen. Außerdem basiert es auf der Idee, dass Menschen mit ihrer Erfahrung lernen und sich Lerntypen daher auch mit der Zeit ändern können.

In der Lernumgebung soll anhand mehrerer Module zum Thema Statistik auf Grundschulniveau gelernt werden. Die Inhalte werden immer in vier Module - Beispiele, Theorie, Übungen und Problemlösungsaufgaben - aufgeteilt, wobei jedes Modul ein vollständiges Verstehen des jeweiligen Themas ermöglichen soll. Somit bieten diese jeweils vollständige Alternativen zueinander. Auf einem Auswahlbildschirm werden die Module entsprechend des Lerntyps sortiert und in verschiedenen Größen angezeigt, wobei das am besten passende Modul am größten und oben dargestellt wird.

Bei der erstmaligen Verwendung wird für den Benutzer ein Profil erstellt, in dem alle nutzerbezogenen Daten gespeichert werden. Die initialen Informationen sind dabei die Antworten auf einen Fragebogen für das Modell von Kolb, durch die der Benutzer einem Lerntyp zugewiesen wird. Durch weiteres Benutzen der Anwendung wird dieses Profil mit den während der Nutzung gesammelten Verhaltensmustern erweitert. Damit hat die Anwendung immer einen Grundstock an Informationen über den Benutzer.

Das Vorgehen zum Erfassen der Verhaltensmuster sieht dabei wie folgt aus: Werden das erste Mal alle Module zu einem Thema angezeigt, wird das Modul, welches als erstes ausgewählt wird, unabhängig von der Reihenfolge, in der es angezeigt wird, als das vom Benutzer bevorzugte interpretiert. Weiterhin wird erfasst wie viel Zeit in den jeweiligen Modulen in Relation zu den anderen verbracht wird. Dabei wird angenommen, dass ein Modul mehr gefällt, je länger der Benutzer darauf verweilt. Zum

Schluss gibt es einen Test, bei dem immer wieder zu den Modulen zurückgesprungen werden kann, um Informationen nachzuschlagen. Es wird angenommen, dass je öfter zu einem bestimmten Modul zurückgesprungen wird, desto mehr hat ebenjenes Modul zum Verständnis des Themas beim Benutzer beigetragen.

Bewertung der Arbeit Wie auch schon Cha et al. [16] verwenden Mbilinyi et al. [34] einen hybriden Ansatz zur Erfassung des Benutzerverhaltens (A1). Es wird zwar im Experiment gefordert einen Fragebogen zu beantworten, jedoch sind die daraus gewonnenen Informationen nur anfangs von Bedeutung, wenn noch keine Verhaltensmuster bekannt sind. Letztere werden tatsächlich unbemerkt vom Benutzer erfasst. Da anhand der Verhaltensmuster die Reihenfolge der Module an den Benutzer angepasst wird ist Anforderung A3 erfüllt, nicht jedoch A2, da nicht wirklich Machine-Learning betrieben wird, sondern die Module lediglich anhand der Klickfrequenz und Verweildauer sortiert werden. Das Seitendesign bleibt durch die Anpassungen in sich stimmig (A4), da lediglich die Sortierung und die Größe der Links zu den Modulen verändert wird. Auch wenn kein wirkliches Lernen stattfindet, wird zumindest die Anpassung der Benutzeroberfläche (A5) im laufenden Betrieb vorgenommen, was diese Anforderung zumindest teilweise erfüllt.

3.2.3 Machine learning techniques to make computers easier to use

Motoda et al. [35] beschäftigten sich weniger mit adaptiven Benutzeroberflächen, als mit der Vorhersage des Benutzerverhaltens. Das Forschungsobjekt *ClipBoard* ist eine UNIX Shell mit graphischer Benutzeroberfläche, die um die verschiedene Funktionalitäten erweitert wurde. In einer Multitasking-Umgebung versucht sie den nächsten Befehl vorherzusagen, aus häufigen Befehlssequenzen Skripte zu erstellen und benötigte Dateien vorzuladen, bevor sie benötigt werden. Dafür zeichnet *ClipBoard* benutzerbezogene Daten auf und erstellt daraus einen gerichteten Graphen, aus dem oft auftretende Muster für Vorhersagen extrahiert werden. Die Anzeige des vorgeschlagenen Befehls geschieht durch die Einblendung eines zum Befehl gehörigen Icons.

Für jeden Befehl werden neben dem vorhergehenden Befehl auch die E/A-Operationen gespeichert, d. h. welche Dateien als Eingabe bzw. Ausgabe benutzt werden, da die Befehlsreihenfolge alleine nicht ausreicht, um zuverlässige Vorhersagen zu treffen.

Die gesammelten Daten mit relationalen Informationen zwischen Befehlen untereinander sowie mit Eingabe-/Ausgabedateien werden in einem Graphen zusammengefasst, indem alle Befehle und Dateien als Knoten interpretiert werden. Für aufeinanderfolgende Befehle wird eine gerichtete Kante vom vorhergehenden zum nachfolgenden Befehl eingefügt. Ebenso werden Kanten von Eingabedateien zum zugeordneten Befehl und von Befehlen zu den jeweiligen Ausgabedateien eingefügt. Für jeden Befehl im Verlauf wird dafür so ein Graph mit begrenzter Tiefe, welche der Länge der Befehlssequenz entspricht, und begrenztem Verzweigungsgrad erstellt. Dabei entspricht der Wurzel-Knoten dem Befehl selber und die restlichen Knoten werden als verschachtelte Attribute angesehen. Jeder dieser Graphen stellt dabei ein Trainingsbeispiel dar. Der von Motoda et al. [35] verwendete Algorithmus kann dabei sowohl überwacht, als auch unüberwacht lernen.

Sind die Daten in dieser Graphstruktur können im nächsten Schritt, der *Graph-based Induction* (GBI), häufig vorkommende Muster aus den Graphen extrahiert werden. GBI benutzt nur eine Heuristik: „Alles, was häufig auftaucht, verdient Aufmerksamkeit“ [35]. Wenn sich ein kleiner Subgraph in den Daten oft wiederholt, erkennt die GBI dieses Muster und ersetzt es durch einen neuen Knoten, welcher die zugehörigen Befehle und E/A-Operationen abstrahiert. Auf diese Weise wird der Graph immer weiter minimiert bis nur noch ein einziger Knoten übrig ist. Die Ausgabe des Algorithmus ist in dem Fall eine Menge an Mustern, die, wenn sie im Graphen zusammengefasst werden, diesen Graphen minimieren.

Die Auswahl, mit welchem anderen Knoten ein Knoten zusammengefasst werden soll, hängt dabei von der Definition der „Größe“ des Graphen ab und lässt sich nur empirisch bestimmen [35], wobei sie so gewählt werden sollte, dass damit gleichzeitig die prädiktive Fehlerrate minimiert wird. Motoda et al. [35] wählten als Kriterium die „Informationszunahme“, welche anhand der Häufigkeiten der einzelnen Knoten und Verbindungen berechnet wird. Damit nicht direkt im ersten Schritt der gesamte Graph zusammengefasst wird, bedarf es außerdem auch einer Terminierungsbedingung für die Clusterbildung z. B. die Anzahl der Iterationen, Größe des Musters oder Änderungsrate des Selektionskriteriums.

Testergebnisse zeigten, dass die E/A-Operationen eine wichtige Informationsquelle zur Vorhersage des nächsten verwendeten Befehls darstellt. Fehlt diese Information für den Wurzelknoten, also für den aktuellen Befehl, dann beträgt die Vorher-

sagegenauigkeit abhängig vom Rauschen, wie z. B. unzusammenhängenden Befehlen, zwischen 47.2 – 52.1%. Sind diese Informationen, also die Eingabeparameter des Befehls, bekannt, dann steigt die Vorhersagegenauigkeit auf 72.1 – 73.7% [35]. Außerdem zeigten sich Erkenntnisse der Informationstheorie als sehr bedeutsam bei der Auswahl der zu zusammenfassenden Knoten. Als weitere mögliche Einflussfaktoren werden der Rückgabestatus des Programms oder die Tageszeit vermutet. Ersterer kann beispielsweise Aufschluss darüber geben, ob eine Datei fehlerhaft ist und neu bearbeitet werden muss oder schon weiterverarbeitet werden kann, während beispielsweise die Tageszeit Rückschlüsse auf Gewohnheiten zulässt.

Versuche die Graphminimierung mit Prädikatenlogik durchzuführen erwiesen sich als ineffizient, da die Laufzeit mehrere Stunden betrug. Deshalb konnte der Ansatz nicht weiterverfolgt werden. Die Aussagekraft der Ergebnisse der GBI sind im Vergleich zur Prädikatenlogik als schwächer einzuordnen. Als eine eingeschränkte Form der Aussagenlogik ist das Lernpotential zwar auch viel schwächer als das der Prädikatenlogik, aber stärker als in der Präsentation als Attribut-Wert-Paare und dabei trotzdem so effizient [35].

Bewertung der Arbeit *ClipBoard* erfasst die Aktionen des Benutzers automatisch ohne dessen Zutun (A1). Durch die *Graph-based Induction* werden die relationalen Daten abstrahiert und wiederkehrende, strukturelle Muster erlernt (A2). Anpassungen der Benutzeroberfläche (A3) finden jedoch kaum statt. Die einzigen Änderungen sind die neben Dateien angezeigten Icons, welche andeuten welcher Befehl dafür vorgesehen ist und somit nicht die Adaption der Oberfläche an den Benutzer zur Verbesserung der Bedienung in den Fokus stellen. Bedingt dadurch wird Anforderung A4 erfüllt, da das Seitendesign bei kleinen Änderungen leicht erhalten bleibt. Motoda et al. [35] schreiben explizit, dass das Lernen in Echtzeit geschieht (A5) und keine handgeschriebene Wissensbasis benötigt wird (A2). Ein weiterer bedeutsamer Punkt, der nicht in den Anforderungen gefasst ist, ist die Tatsache, dass der Benutzer bei allem in Kontrolle ist und kein Zwang besteht den vom Programm geäußerten Empfehlungen zu folgen.

3.2.4 A Personal Learning Apprentice

CAP ist ein sog. *Learning Apprentice*-System, welches einen selbstlernenden Kalender-Manager implementiert. Der Begriff *Learning Apprentice* ist dabei als ein persönlicher Assistent zu verstehen, der dem Benutzer bei alltäglichen Aufgaben hilft. Dieser soll die Gewohnheiten des Benutzers bei der Kalenderverwaltung lernen und darauf basierend Vorschläge beim Erstellen, Verschieben, Kopieren oder Löschen von Meetings bereithalten. Beispiele für solche Vorschläge wären Zeit und Ort des Meetings oder ob Erinnerungen via E-Mail an die Teilnehmer verschickt werden sollen. Ein anderer Anwendungsfall ist das Versenden von automatischen Antworten bei Meeting-Anfragen.

Für die Entwicklung von CAP stellten sich Dent et al. [20] folgende Fragen:

- Kann ein solcher Assistent, selbst wenn er anfangs uninformiert ist, ausreichend nützlich sein, um einen Benutzer zur Nutzung zu überzeugen und somit überhaupt erst Trainingsdaten zu sammeln?
- Gibt es ausreichend mächtige Methoden zur Generalisierung, um automatisch generelle Entscheidungsstrategien aus den Trainingsdaten zu erlernen?
- Ist die Menge der gewählten Attribute der Trainingsdaten stabil oder ändert es sich mit der Zeit?

Anhand dieser Fragen zeigen sie, dass Learning-Apprentice-Systeme die praktische Entwicklung von persönlichen Software-Assistenten ermöglichen können, welche bis dahin zu teuer in Entwicklung und Wartung waren.

Der Kalender-Manager CAP bringt auch ohne eine erlernte Wissensbasis bereits die grundlegenden Funktionen zur Verwaltung eines Kalenders, wie das Hinzufügen und Löschen von Terminen, und ein E-Mail-Interface zum Versenden von Erinnerungen mit. Die aus der Benutzung gesammelten Daten enthalten den aktuellen Zustand des Kalenders, Informationen zu 250 Personen (inkl. Abteilung, Büro und E-Mail-Adresse), eine sich weiterentwickelnde Menge an Entscheidungsregeln und Künstlichen Neuronalen Netzen für Empfehlungen an den Benutzer, sowie Beschreibungen der letzten Befehle des Benutzers, welche die Trainingsdaten zum Lernen darstellen. Diese Trainingsdaten können jedoch auch ein gewisses Rauschen enthalten, welches durch Eingabefehler oder verwendete Synonyme (z. B. zwei Bezeichnungen für denselben Raum) verursacht wird.

Die Trainingsdaten werden von CAP im laufenden Betrieb gesammelt, wenn z. B. Termine erstellt werden, jedoch wird das Machine-Learning „offline“ betrieben, um Verzögerungen bei der Nutzung zu vermeiden. Für das Lernen werden dabei eine Entscheidungsbaum-Methode ähnlich wie ID3 [44, 45], aus der eine Menge an Regeln extrahiert werden, sowie ein Künstliches Neuronales Netz im Wettbewerb zueinander verwendet. Außerdem wird jedes Attribut des Datensatzes, wie z. B. der Ort oder die Zeit eines Meetings, als seperates Lernproblem betrachtet, wodurch Korrelationen zwischen diesen Attributen nicht beachtet werden.

In Tests zeigte sich, dass CAP nach durchschnittlich 1-2 Monaten Training im alltäglichen Betrieb nützliche Vorschläge liefern kann. Die Akzeptanzrate der Benutzer für diese Vorschläge variierte dabei von 45% bis 70% bei Vorschlägen zu Ort und Zeit eines Meetings. Diese Zahlen können jedoch verbessert werden, wenn nur Vorschläge gemacht werden, bei denen CAP sich sicher ist und zuverlässigere Regeln, für die mehr Trainingsbeispiele zur Verfügung stehen, verwendet. Dabei muss jedoch ein Kompromiss zwischen Genauigkeit und Abdeckung der Trainingsbeispiele geschlossen werden. Die Qualität der erlernten Regeln entspricht in etwa der von manuell festgelegten Regeln, wodurch Grund zur Annahme besteht, dass CAP die Aufgabe der Bestimmung dieser Entscheidungsregeln zuverlässig übernehmen kann.

Dent et al. [20] sehen somit in CAP einen erfolgreichen Kalender-Manager, der dem Anspruch genügt nützlich zu sein, auch wenn noch kein Wissen über den Benutzer erlernt werden konnte. Sie kamen außerdem zu dem Schluss, dass die Auswahl der zu analysierenden Attribute eines Meetings sehr wichtig ist, solange die verwendeten Machine-Learning-Methoden nicht mit einer zu großen Menge Attribute umgehen können. Dennoch werden die manuell ausgewählten Attribute als ausreichend stabil angesehen, um auch auf längere Zeit und mit verschiedenen Benutzern gute Ergebnisse zu erzielen. Ein weiterer Punkt, auf den sie hinweisen sind mögliche Kontextfaktoren, die (noch) nicht von CAP beobachtet werden können, aber dennoch von Bedeutung sind z. B. die Kalender anderer Personen oder Raumbelegungen. Eine wichtige Bedingung für die erfolgreiche Anwendung des Kalender-Managers sei außerdem, dass dieser Änderungen in der Welt schneller lernt, als sie geschehen, d. h. er muss sich an die Gegebenheiten anpassen, bevor sie sich erneut ändern.

Bewertung der Arbeit Die Arbeit von Dent et al. [20] enthält neben der offensichtlichen Beschreibung ihres Prototyps ebenso viele wichtige Faktoren, die es allgemein bei der Entwicklung eines lernenden Systems zu beachten gibt. Einer davon ist die Tatsache, dass ein Benutzer sich nur dann mit der Anwendung auseinandersetzt, wenn diese auch ohne erlerntes Wissen einen gewissen Mehrwert hat und für den Benutzer beim Erfassen der Benutzerdaten keine signifikante Belastung bedeutet, was CAP auch gelingt (A1). Wie auch Motoda et al. [35] in ihren Tests, haben Dent et al. [20] auf Störeinflüsse in den Eingabedaten z. B. durch Schreibfehler beim Tippen hingewiesen, welche nach Möglichkeit erkannt und bereinigt werden sollten.

Das Ziel des Kalender-Managers CAP war nicht die Verbesserung der Bedienung durch eine Anpassung der Benutzeroberfläche, sondern durch das Präsentieren von nützlichen Vorschlägen, weshalb Anforderung A3 nicht erfüllt wird und Anforderung A4 hier nicht anwendbar ist. Mit dem Lernen durch Entscheidungsbaum-Methoden sowie Künstlichen Neuronalen Netzen anhand der vom Benutzer eingetragenen Termine erfüllt CAP jedoch die Anforderung A2. Da der Lernvorgang aber offline durchgeführt wird und dadurch auch keine Anpassung des Verhaltens im laufenden Betrieb erfolgt, ist Anforderung A5 nicht erfüllt.

3.2.5 TFMAP: Optimizing MAP for Top-N Context-aware Recommendation

Im Bereich des kollaborativen Filterns und der kontextsensitiven Empfehlungen haben Shi et al. [49] mit TFMAP einen Ranking-Algorithmus entwickelt, welcher unter Einbezug von implizitem Feedback die besten n Elemente für einen Benutzer in einem gegebenen Kontext – nach einem erlernten Ranking sortiert – ausgibt. Bei diesem Algorithmus wird ein latentes Modell der Benutzer, Items und verschiedenen Kontexte gelernt, indem im Gradientenverfahren die sog. *Mean Average Precision* (MAP), die sich direkt aus diesen Modellen ergibt, optimiert wird.

Das implizite Feedback des Benutzers wird in einem binären Tensor gespeichert, was in [49] einer dreidimensionalen Matrix entspricht. Frühere Arbeiten verwendeten eine einfache Matrix, um Verbindungen zwischen Benutzern und zu bewertenden Items darzustellen. Um Kontextinformationen miteinzubeziehen wird diese Matrix um eine Dimension erweitert. Für jede Kombination aus Benutzer, Item und Kontext ist hierbei

auch nur gespeichert, ob eine Interaktion zwischen diesem Benutzer und diesem Item in dem spezifizierten Kontext stattfand oder nicht. Daher ist der verwendete Tensor auch nur binär.

Die latenten Modelle der Benutzer, Items und Kontexte werden respektive als die zweidimensionalen Matrizen $U \in \mathbb{R}^{MXD}$, $V \in \mathbb{R}^{NXD}$ und $C \in \mathbb{R}^{KXD}$ mit zufälligen Werten initialisiert. M, N und K stehen hierbei für die Anzahlen von Benutzern, Items und Kontexten. Die jeweiligen Matrizen speichern für jedes Objekt eine feste Anzahl D an latenten Eigenschaften. Die Eigenschaftsvektoren einer Kombination aus Benutzer, Item und Kontext werden faktorisiert und auf Grundlage aller möglichen Faktorisierungen wird der MAP-Wert errechnet.

Die *Mean Average Precision* ist dabei als die „Qualität über alle Empfindlichkeitslevel“ bzw. als durchschnittliche Fläche unter der *precision-recall*-Kurve zu verstehen. Sie gibt Auskunft darüber wie gut ein Ranking ist, mit der Besonderheit, dass Items, die fälschlicherweise weit oben im Ranking stehen, stärker bestraft werden, als Items, die fälschlicherweise zu weit unten im Ranking stehen. Dadurch sollen keine schlechten Items auf guten Plätzen im Ranking stehen und es ist nur zweitrangig, dass gute Items auch auf guten Plätzen stehen.

Es zeigt sich, dass bereits ohne Verwendung des Kontexts der TFMAP Algorithmus signifikant besser ist, als andere Algorithmen auf dem Stand der Forschung [49]. Durch Miteinbezug des Kontextes konnte das Ergebnis noch weiter um 5% des MAP-Wertes verbessert werden [49]. Somit kann TFMAP trotz nur allgemeiner, impliziter Feedback-Daten gute Ergebnisse erzielen.

Bewertung der Arbeit Der Ansatz des kollaborativen Filterns ist eine Methode aus dem Bereich des Data Mining, die typischerweise darauf ausgerichtet ist, Daten von vielen Benutzern zu analysieren, um daraus Informationen über einen konkreten Benutzer abzuleiten. Daher bleibt offen wie gut der von Shi et al. [49] entwickelte TFMAP-Algorithmus bei einem oder nur wenigen Benutzern funktioniert.

TFMAP ist bei den benötigten Eingabedaten vergleichweise genügsam, da keine nähere Information über die Interaktion eines Benutzers mit einem Element im jeweiligen Kontext benötigt wird, sondern nur ob diese stattfand oder nicht. Dadurch, dass nur solches implizites und kein explizites Feedback gesammelt wird, ist Anforderung A1 erfüllt. Anhand dieser Daten erfolgt durch die Optimierung der Benutzer-, Item-

und Kontextmodelle eine Form des Lernens (A2), die die gesammelten Daten in Eigenschaftswerte übersetzt.

Der Fokus von [49] lag darauf einen Ranking-Algorithmus zu entwerfen und nimmt daher keinen Bezug auf Anpassung einer graphischen Benutzeroberfläche. Dadurch werden Anforderungen A3 und A4 nicht erfüllt. Lernen & Adaption während des laufenden Betriebs (A5) ist hingegen möglich, obwohl für jede Iteration im Gradientenverfahren laut [49] mehrere Sekunden benötigt werden, indem Lernen und Adaption der Oberfläche entkoppelt voneinander ablaufen. Somit wäre eine Adaption der Benutzeroberfläche auch in Echtzeit möglich.

3.3 Zusammenfassung und Handlungsbedarf

Im vorhergehenden Abschnitt wurden verschiedene Arbeiten vorgestellt, die sich mit dem Thema Nutzungskontext und Adaption an den Benutzer auseinandersetzen. Die Arbeiten von Cha et al. [16] und Mbilinyi et al. [34] haben erste Ansätze bereitgestellt, um Oberflächen von Lernplattformen an verschiedene Typen von Benutzern, entsprechend eines festen Benutzermodells, anzupassen. Jedoch war es bei beiden Arbeiten notwendig die Benutzer einen Fragebogen ausfüllen zu lassen.

In der Arbeit von Motoda und Yoshida [35] wurde ein auf einem Graphenmodell beruhender Ansatz verfolgt. Dabei wird des Verhalten des Benutzers als Graph, in dem Aktionen und die daran gekoppelten Ressourcen voneinander abhängen, interpretiert. Anhand dieser strukturierten Wissensbasis wird versucht, Wissen über den Benutzer in einem bestimmten Kontext, der Verhaltenshistorie, abzuleiten.

Das Konzept eines *Learning Apprentice*, wie der von Dent et al. [20], zeigt wichtige Punkte auf, die es beim Design von adaptiven Systemen zu beachten gilt. Da CAP jedoch auf Basis von Kalendereinträgen als Datensätze arbeitet und damit Empfehlungen bei neuen Einträgen liefert, ist das gelöste Problem ein anderes. Im Gegensatz dazu versucht diese Arbeit aus einer gegebenen Menge von Widgets die relevantesten davon zu finden.

Shi et al. [49] widmen sich genau der Problematik eines Top-n-Rankings und verwenden dazu nur implizites Wissen über das Benutzerverhalten. Einziger Nachteil ist der hohe Rechenaufwand, der Lernen & Adaption im laufenden Betrieb schwer macht.

	Cha et al.	Mbilinyi et al.	Motoda et al.	Dent et al.	Shi et al.
A1	✓	✓	✓	✓	✓
A2	✗	✗	✓	✓	✓
A3	❖	✓	❖	✗	✗
A4	❖	✓	❖	✗	✗
A5	❖	❖	✓	✗	✓

Tabelle 3.2: Übersicht über die Bewertung der untersuchten Arbeiten. Ein „✓“ steht dabei für eine erfüllte Anforderung, ein „✗“ für eine nicht erfüllte Anforderung und ein „❖“ steht für eine teilweise erfüllte Anforderung.

Wie Tabelle 3.2 zeigt, konnte keine der untersuchten Arbeiten alle Anforderungen erfüllen. Daher besteht der Handlungsbedarf eine neue Lösung zu entwickeln, die die oben genannten Anforderungen erfüllt. Dabei ist der Ansatz von Shi et al. [49] besonders vielversprechend, da die Problemstellung, aus einer Menge Elemente die für einen Benutzer besten n Elemente anhand des Kontextes zu bestimmen, mit der in Kapitel 1 beschriebenen eng verwandt ist. Aus diesem Grund wird sich diese Arbeit den von Shi et al. [49] beschriebenen TFMAP-Algorithmus als Vorbild nehmen und an das vorliegende Problem anpassen.

4 Kontextfaktoren

Als erster Schritt auf dem Weg zur Anwendung eines Machine-Learning-Algorithmus und der Implementierung einer adaptiven Benutzeroberfläche müssen zuerst die zu verarbeitenden Kontextinformationen, welche Rückschlüsse auf die Relevanz der verschiedenen Software-Metriken in den einzelnen Widgets zulassen, erfasst werden. Die wichtigste Frage dabei ist es, welche Kontextfaktoren überhaupt existieren und relevant sind. Hierfür werden in Kapitel 4.2 nacheinander Gruppen dieser Faktoren abgehandelt, um dem Problem mit einem Top-down-Ansatz zu begegnen.

Die gewonnenen Erkenntnisse werden bzgl. ihres möglichen Einflusses bewertet. Außerdem muss analysiert werden, wie gut diese in einem realen Einsatzszenario erfasst werden können. Da es sich beim Demonstrator um eine Webanwendung handelt, stehen Sensoren wie z. B. Eyetracker, Kameras oder Mikrofone im Normalfall nicht zur Verfügung. Deshalb können manche erwähnten Einflüsse, wie beispielsweise die Stimmung des Benutzers, nicht auf diese Weise erfasst werden, sondern müssten indirekt z. B. über den Zustand der Programmqualität inferiert werden.

Zum Schluss wird ein Konzept vorgestellt, wie die in diesem Kapitel erlangten Erkenntnisse in der zu entwickelnden adaptiven Benutzeroberfläche verwendet werden. Es wird dargestellt welche Kontextinformationen mit welcher Begründung verwendet werden, aus welchen direkt verfügbaren Informationen diese inferiert werden und aus welchen Quellen diese Informationen stammen.

4.1 Bewertungskriterien

Wie bereits in Kapitel 3 geschehen, muss auch für die auszuwählenden Kontextfaktoren definiert werden, nach welchen Kriterien diese bewertet und ausgewählt werden. Ein wichtiger Anspruch an einen Faktor ist es dabei ein einfaches Erfassen der entsprechenden Werte des Faktors zu ermöglichen, da dieser andernfalls keinen Nutzen mit sich bringt, wenn es zu schwer ist die entsprechenden Werte zu bestimmen. Ein wei-

Name	Beschreibung
K1	Einfaches Erfassen der Daten
K2	Relevante Erklärung von Varianz
K3	Kontexteigenschaft

Tabelle 4.1: Anforderungskatalog zur Bewertung potentieller Kontextfaktoren

terer Anspruch ist es, dass ein Kontextfaktor auch dazu beitragen soll Vorhersagen zu verbessern.

Die folgende Aufzählung beschreibt all die Anforderungskriterien, anhand derer die potentiellen Kontextfaktoren bewertet und ausgewählt werden:

K1 **Einfaches Erfassen der Daten**

Die Werte eines potentiellen Kontextfaktors müssen in einfacher Weise von der Anwendung bestimmt und verwendet werden können. Dazu darf kein erheblicher Aufwand für den Benutzer entstehen, da dieser möglichst wenig beeinträchtigt werden soll. Darüber hinaus können zusätzliche Geräte und Sensoren zum Erfassen von Kontextfaktoren, welche nicht zur standardmäßigen Peripherie eines Arbeitscomputers gehören, nicht vorausgesetzt werden.

K2 **Relevante Erklärung von Varianz**

Irrelevante Kontextfaktoren, welche keinen signifikanten Anteil der Varianz in den Eingabedaten – dem impliziten Feedback des Benutzers – erklären können, könnten die Vorhersagen des Algorithmus verschlechtern, da sie als Rauschen wirken können [38]. Daher sind vor allem Faktoren, die in allen Situationen gleiche Werte haben, wie z. B. das Geschlecht bei ein und demselben Benutzer, nicht in der Lage Varianz in den Eingabedaten zu erklären.

K3 **Kontexteigenschaft**

Ein Kontextfaktor muss die Situation einer Interaktion zwischen Benutzer und Software-Metrik näher beschreiben. Daher muss auch ein Bezug zu dieser Interaktion bestehen, um Korrelationen zwischen Kontextfaktor sowie Benutzern und Metriken plausibel zu erklären.

4.2 Identifikation von Kontextfaktoren

Das folgende Zitat soll einleitend ein erstes Verständnis darüber geben, was unter dem Begriff „Kontext“ zu verstehen ist:

> We define context as any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object.
> – Gregory D. Abowd [6]

G. Adomavicius und A. Tuzhilin kategorisieren Kontextfaktoren in [7] und betrachten zur Einteilung von Kontextfaktoren, was ein *Recommender System* über die Faktoren weiß und wie sich die Informationen mit der Zeit ändern [54]. Durch diese Einteilung können bereits manche Faktoren aussortiert werden, denn es können in dieser Arbeit nur Kontextfaktoren verwendet werden, die auch beobachtbar und deren Werte feststellbar sind [55].

Für die Auswahl der Kontextfaktoren ist es nötig noch weitere Punkte zu beachten. Einer davon ist es deren Anzahl zu begrenzen. Das ist zum einen deshalb notwendig, da durch eine steigende Anzahl von Kontextfaktoren auch die Komplexität der Berechnung und die Menge der zu erfassenden Daten steigt; zum andern kann darunter die Qualität der Ergebnisse leiden [53]. Irrelevante Kontextfaktoren, d. h. Faktoren, die kaum Varianz in den Daten erklären können, können die Vorhersagen verschlechtern, da sie in dem Fall als Rauschen in den Daten wirken [9, 38].

Ein weiterer Punkt sind Eigenschaften, die einen relevanten Kontextfaktor ausmachen. Gewissermaßen eignen sich Attribute der beteiligten Benutzer und Software-Metriken als Kontext, aber nur unter der Bedingung, dass sie sich zwischen zwei Interaktionen verändern, was auch auch dem Trend bei *Recommender Systems* entspricht [54]. Dies schließt Faktoren wie z. B. das Alter oder das Geschlecht aus, da diese zwischen zwei Interaktionen im Normalfall als konstant angenommen werden können.

Die folgende Gruppierung der Kontextfaktoren richtet sich nach der jeweiligen Quelle der Informationen und wurde nur beispielhaft gewählt, um diese Faktoren übersichtlich und thematisch zu gruppieren. Dabei sind unter softwarebezogenen Faktoren diejenigen zu verstehen, die aus der Software direkt stammen. Dazu zählen v. a. Software-Metriken, da sie im Kontext dieser Arbeit die Hauptinformationsquellen bzgl. des Zu-

stands der vom Benutzer zu entwickelnden Software sind. Ähnlich verhält es sich mit benutzer-, projekt- und umgebungsbezogenen Kontextfaktoren, die jeweils respektive durch Benutzer, das Projekt bzw. die Umgebung bedingt werden.

Durch diese Gruppierung wird bereits eine Unterscheidung der verschiedenen Qualitäten vorgenommen. Das bedeutet, dass verschiedene Typen von Kontextfaktoren verschiedene Arten von Aussagen treffen können. Es kann also beispielsweise nicht vom aktuellen Wetter auf die Qualität des Quelltextes geschlossen werden.

Da die konkrete Auswahl von Kontextfaktoren in Recommender Systemen immer auch vom konkreten System abhängt [39], ist es nicht möglich eine allgemeingültige Liste relevanter Faktoren aus der Literatur zu gewinnen, auch wenn sie Vorschläge für ein paar häufig verwendete Faktoren liefert. Eine Befragung der Benutzer ist außerdem keine zuverlässige Methode Kontextfaktoren zu finden, da diese die Relevanz im Allgemeinen schlecht einschätzen können [40]. Aus diesen Gründen wird in den folgenden Kapiteln versucht zu ergründen, welche potentiellen Kontextfaktoren überhaupt existieren. Inwieweit diese geeignet und relevant sind, wird anhand der Erfassbarkeit der entsprechenden Kontextinformationen und der von Odić et al. in [39] gewonnenen Erkenntnisse bewertet. Diese sagen aus, dass sich ein interessanter Kontextfaktor durch Variationen seines Zustands zwischen zwei Benutzer-Metrik-Interaktionen auszeichnet, anhand derer eine variierende Relevanz der entsprechenden Software-Metriken erklärt werden kann.

4.2.1 Softwarebezogene Kontextfaktoren

Bei der täglichen Arbeit muss ein Entwickler über den Zustand der Software im Bilde sein, um angemessen reagieren zu können. Wie bereits erwähnt, sind Software-Metriken ein bedeutendes Werkzeug, um dieses Ziel zu erreichen. Sie werden direkt aus dem Quelltext abgeleitet und beschreiben bestimmte Attribute der Software [5]. Aus diesem Grund sind die Eigenschaften der konkreten Software auch wichtige Einflussgrößen für diese Metriken. Durch diese enge Bindung lassen sich daher aus den Metriken selbst auf direktem Wege Informationen über die Software - und damit auch über die Relevanz der einzelnen Metriken - gewinnen.

Bei der Betrachtung einer Software-Metrik sind vor allem die folgenden Typen von Informationen naheliegend:

- Welchen *Wert* hat eine bestimmte Metrik?
- Wie ist die *Werteänderung* einer bestimmten Metrik?
- Wie lange *stagniert* der Wert einer bestimmten Metrik?

Diese verändern sich typischerweise im Verlauf der Weiterentwicklung einer Software und sind deswegen bei einem Aufruf als potentiell relevante Kontextfaktoren zu betrachten [38].

Durch den Wert selber erhält man die direkte Information über den Zustand einer Software. Wird dieser Wert in Relation zu anderen Projekten gesetzt – falls solche Vergleiche sinnvoll sind, wie in sehr ähnlichen Projekten, oder anderen Modulen im gleichen Projekt, aber auch durch bloße Erfahrung – kann daraus bestimmt werden, wie kritisch der Zustand ist und ob Handlungsbedarf besteht. Je schlechter der Zustand ist und je mehr Handlungsbedarf besteht, desto wichtiger ist es, dies den Entwickler wissen zu lassen, damit dieser darauf reagieren kann. Falls alles in Ordnung ist, muss der Entwickler nicht damit abgelenkt werden.

Aus dem Werteverlauf lassen sich auf ähnliche Weise Indizien bezüglich der Relevanz einer Metrik ableiten. Ähnlich wie beim Vergleich aktueller Absolutwerte, kann der Verlauf eines Wertes ebenfalls auf den Handlungsbedarf an der jeweiligen Stelle hinweisen. Der Vorteil dabei liegt an der Eigenschaft, dass vielmehr das Ausmaß der Werteänderung, als der Wert selbst zur Aussagekraft beiträgt, d. h. ein Vergleich mit anderen Projekten oder Modulen ist weniger von Bedeutung. Dennoch genügt es möglicherweise nicht, nur die Werteänderung zu betrachten, da ein steigender Wert in einem kritischen Wertebereich nicht zwangsläufig eine Entwarnung bedeutet. Auf der anderen Seite kann analog eine sich verschlechternde Software-Metrik bereits eine frühzeitige Warnung ermöglichen, selbst wenn der Wert selbst noch in einem vertretbaren Bereich ist. Bei der Betrachtung der Werteverläufe könnte außerdem kleinen Fluktuationen weniger Beachtung geschenkt werden, als länger anhaltenden Trends, da ansonsten auf Veränderungen, die an und für sich unerheblich sind, reagiert werden müsste.

In Bezug auf die Software-Metriken können auch noch dauerhaft stagnierende Metriken eine Rolle spielen, da sie einen Hinweis auf Stillstand sind. Hierbei ist jedoch mit Vorsicht zu walten, denn selbst eine Metrik mit dauerhaft schlechten Werten kann

auf zweierlei Weisen gedeutet werden. Einerseits kann dies bedeuten, dass dieser Zustand zu einem blinden Fleck geworden und aus dem Fokus der Entwickler gewandert ist, aber andererseits kann es auch bedeuten, dass diese Metrik lediglich eine geringe Priorität hat und andere Aspekte mehr Aufmerksamkeit verdienen.

Bewertung

Sollen die absoluten Werte von Software-Metriken verwendet werden, sind zusätzlich Vergleichswerte notwendig, da aus einem bloßen Wert im Allgemeinen deutlich weniger Informationen herausgezogen werden kann. Welche Werte für welches Projekt akzeptabel sind, ist u.a. deshalb schwer zu berechnen, weil bei jedem Projekt andere Anforderungen gestellt werden, andere Personen mitarbeiten und eventuell auch eine andere Architektur vorliegt. Daher ist eine Einschätzung und Bewertung eines konkreten Wertes schwer, sodass die Einordnung konkreter Werte in eine auf andere Projekte übertragbare Kontextdefinition ebenfalls schwer möglich ist und wird aus diesem Grund in dieser Arbeit auch nicht versucht. Die entsprechenden Daten können zwar einfach erfasst werden, aber die sinnvolle Verwendung ist zu aufwendig.

Stagnierende Werte haben ein ähnliches Problem, denn es wäre zwar als Entwickler interessant zu wissen, welche Metriken bei verhältnismäßig schlechten Wertebereichen stagnieren. Im umgekehrten Fall, wenn sie in guten Wertebereichen stagnieren, ist dies jedoch vergleichsweise uninteressant und ohne Relativierung des Wertes ist diese Einteilung ebenso schwer zu vollziehen.

Ein geeigneterer Ansatz ist es lediglich das Vorzeichen der letzten Veränderung einer Metrik zu betrachten. Auch wenn dadurch weniger spezifische Erkenntnisse transportiert werden, wird dennoch die grundlegende Information über den Veränderungstrend übermittelt. Diese Information kann auch ein in Hinsicht auf Software-Metriken unerfahrener Benutzer nutzen, da keine Erfahrung über die Bedeutung der konkreten Absolutwerte nötig ist. Daran lässt sich zwar noch nicht erkennen, ob der Zustand der jeweiligen Metrik „gut“ oder „schlecht“ ist, jedoch genügt es für die Anwendung dieser Arbeit zu wissen, ob sich der Zustand verbessert oder verschlechtert hat.

Somit bleibt von den softwarebezogenen Kontextfaktoren die Richtung der Werteveränderung als vielversprechender und vergleichsweise leicht zu extrahierender Faktor übrig und wird im Rahmen der Kontextdefinition in Kapitel 4.3 verwendet.

	Absoluter Wert	Werteänderung	Stagnation
K1	✓	✓	✓
K2	❖	✓	❖
K3	✓	✓	✓

Tabelle 4.2: Übersicht über die Bewertung der softwarebezogenen Kontextfaktoren. Ein „✓“ steht dabei für ein erfülltes Kriterium, ein „✗“ für ein nicht erfülltes Kriterium und ein „❖“ steht für ein teilweise erfülltes Kriterium.

4.2.2 Benutzerbezogene Kontextfaktoren

Der Benutzer stellt in dieser Arbeit den Bewertungsmaßstab und damit auch nach Ansicht von Zheng et al. [55] einen relevanten Kontextfaktor dar. Gleichermaßen ist es schwer, lediglich durch implizites Feedback wie in der Arbeit von Shi et al. [49] bei der Interaktion mit einer Anwendung Informationen über ihn zu erhalten.

Die Stimmung, Wachsamkeit und Konzentrationsfähigkeit des Benutzers können eine große Wirkung auf dessen Verhalten, die Informationen, welche ihn interessieren, und auch wie diese aufgenommen werden können haben. Ein erschöpfter Entwickler wird potentiell ein anderes Verhalten an den Tag legen, als jemand, der wach und konzentriert ist. Außerdem sind Emotionen auch bei vielen kognitiven Prozessen, wie z. B. kausalen Argumentationen, Nachdenken, Abschätzungen oder Planaufgaben, beteiligt, weshalb davon ausgegangen werden kann, dass Anwendungen, die Emotionen beachten, besser nutzbar sind [21].

Im Gegensatz dazu lässt sich das Verhalten bei der Interaktion mit der Anwendung im Rahmen der Peripherie des Computers problemlos erfassen. Es muss z. B. lediglich gespeichert werden, auf welche Widgets wie oft geklickt wurde oder wie lange ein Benutzer danach auf einer Seite verweilt. Daraus lassen sich dann das Interesse und potentiell auch Gewohnheiten des Benutzers ableiten [34]. Dabei ist nicht zu vergessen, dass

ebenjene für den Benutzer subjektiv interessanten Metriken nicht zwangsläufig auch gemäß der in Kapitel 4.2.1 genannten Kriterien relevant sein müssen.

Die Stellung bzw. die Aufgabe des Entwicklers kann weitere Informationen darüber liefern, was für ihn relevant sein kann, denn davon ist abhängig welche Metriken im Moment überhaupt von Bedeutung sind und einen Mehrwert mit sich bringen. Während ein Tester an der Testüberdeckung des Quelltextes interessiert ist, interessiert sich ein Entwickler in der Qualitätssicherung womöglich eher für die Komplexität der einzelnen Funktionen und Methoden, um zu versuchen diese zu reduzieren.

Bewertung

Das Befinden eines Benutzers zu erfassen gestaltet sich allein schon aufgrund der dafür fehlenden Sensorik als nicht praktisch realisierbar. Diese Einflüsse sind schwer zu analysieren, wenn kein erheblicher, zusätzlicher Aufwand betrieben wird, um physiologische Merkmale des Benutzers zu erfassen und zu analysieren, wobei diese selbst dann nicht zuverlässig erkannt und gedeutet werden können, da solche Daten typischerweise starkes Rauschen und keine klaren Grenzen besitzen [18, 21]. Im Falle von SonarQube findet zwischen Benutzer und Anwendung außerdem kaum Interaktion statt, aus der auf z.B Stimmung, Wachsamkeit oder Konzentrationsfähigkeit geschlossen werden kann, denn der Großteil der Navigation findet typischerweise nur mit der Maus oder Tastaturkürzeln statt. Möchte man jedoch beispielsweise Ermüdung des Benutzers zuverlässig erkennen, wären z. B. EEG-Diagramme, Kameras oder Mikrofone notwendig [15, 21], was für die alltägliche Nutzung nicht zumutbar ist.

Der Ansatz auf die Aufgabe des Entwicklers zu schließen könnte Aufschluss darüber geben, welche Teile des Programmcodes interessant sind, aber diese Schlussfolgerung zu machen ist ebenfalls schwer. Da sich die Rolle eines Entwicklers schnell ändern kann und Entwickler manchmal auch gleichzeitig Tester sind, ist eine solche Einteilung nicht zuverlässig machbar. Selbst wenn versucht werden soll eine andere Aufgabenaufteilung als Grundlage zu verwenden, ist es nicht trivial die charakteristischen Verhaltensindikatoren zu identifizieren. Angenommen Entwickler- und Tester-Rollen wären in der Tat strikt getrennt, würde es nicht ausreichen Zugriffe auf Test-Coverage einzelner Klassen zu beobachten, denn ein Entwickler kann genauso daran interessiert sein zu wissen,

welche Funktionalitäten bereits ausreichend getestet sind, wenn er diese Funktionalitäten in seinem Code verwenden möchte.

Das Interesse an bestimmten Metriken lässt sich jedoch indirekt über das Verhalten des Benutzers erfassen. Interessiert sich ein Benutzer für spezielle Metriken, wird er häufiger auf die entsprechenden Links klicken, um sich weitere Details anzeigen zu lassen [16]. Die Frequenz, mit der eine Metrik im Vergleich zu anderen Metriken ausgewählt wird, kann ein erster Indikator sein, ist aber noch nicht ausreichend, denn der Benutzer kann auch nur aus Versehen darauf geklickt haben. Um solche Versehen ausschließen zu können würde zusätzlich noch die Verweildauer auf der Seite benötigt, denn wenn ein Benutzer nach dem Klicken auf eine Metrik auf der Seite bleibt, ist davon auszugehen, dass er an den Informationen wirklich interessiert ist [34]. Die Klickfrequenz bietet damit die Möglichkeit direkt das Interesse des Benutzers zu erfassen und ist deshalb weniger eine Form von Kontext, als eine feingranularere Form von implizitem Feedback. Wird das binäre implizite Feedback, welches nur angibt, ob ein Benutzer eine bestimmte Metrik angeklickt hat oder nicht, zusätzlich mit der relativen Klickfrequenz im Vergleich zu anderen Metriken gewichtet, könnte somit die persönliche Bevorzugung verschiedener Metriken noch stärker in das implizite Feedback eingebracht werden.

Da Klickfrequenz und Verweildauer des Benutzers auch in der Literatur [16, 34] die verbreitetsten Messgrößen sind, werden sie zwar zur Verfeinerung des impliziten Feedbacks verwendet, nicht jedoch als Kontextfaktor. Dies ist darin begründet, dass sie semantisch mehr dem Feedback entsprechen und weniger einen diskreten Kontext für eine einzige Interaktion beschreiben, sondern das Verhalten über eine längere Zeit hinweg, wodurch die Kontexteigenschaft nicht erfüllt ist.

4.2.3 Projektbezogene Kontextfaktoren

Bei der Betrachtung des Projektkontextes ist neben den einzelnen Teilen des Projektes, wie z. B. Planungen, Aufgaben und die daran beteiligten Personen, auch das große Ganze, das Projekt selber, als Kontext relevant. Die gröbste Einteilung von Software-Metriken in SonarQube besteht in der Einteilung nach Projekten, wobei jedes Projekt einen separaten Einzelkontext darstellt. Bezüglich der Relevanz verschiedener Metriken für einen Entwickler spielt das insofern eine Rolle, dass ein Entwickler sich im Regelfall

	Stimmung	Wachsamkeit	Klick-verhalten	Aufgabe
K1	✗	✗	✓	✗
K2	✓	✓	✓	✓
K3	✓	✓	✗	✓

Tabelle 4.3: Übersicht über die Bewertung der benutzerbezogenen Kontextfaktoren. Ein „✓“ steht dabei für ein erfülltes Kriterium, ein „✗“ für ein nicht erfülltes Kriterium und ein „❖“ steht für ein teilweise erfülltes Kriterium.

hauptsächlich für die Metriken der Projekte, an denen er auch zur Zeit arbeitet, interessiert. Dadurch ermöglicht der Einbezug des Projekts, an dem der Entwickler aktuell arbeitet, bereits ein ausgiebiges Herausfiltern von wahrscheinlich irrelevanten Metriken aus anderen Projekten.

Eine weitere, erhebliche und auch unnachgiebige Einflussgröße in Projekten ist die Zeit. Daher hat auch der zeitliche Kontext, relativ zu Deadlines oder Projektterminen gesehen, das Potential Prioritäten zu verschieben. Dies geschieht vor allem dadurch, dass im Verlauf der Zeit die Ziele und Erwartungen an den aktuellen Ist-Stand sowie die noch verfügbare Zeit angepasst werden müssen. Nimmt man einen agilen Softwareentwicklungsprozess als Beispiel, würde sich beispielhaft am Anfang eines Sprints ein Ziel zum Abbau der technischen Schuld, wie „Alle größeren Fehler beseitigen“, vorgenommen. Vergeht etwas Zeit und der Sprint neigt sich dem Ende zu, wird jedoch möglicherweise klar, dass stattdessen noch mehr Arbeit in die neue, geforderte Funktionalität gesteckt werden muss und deswegen die Testüberdeckung für diese neuen Funktionen hochgehalten werden sollte, damit diese möglichst fehlerfrei funktionieren. Im Umkehrschluss können sich Prioritäten dann auch genauso verändern, wenn vor einer Deadline mehr Zeit übrig ist, als es erwartet wurde. Damit derartige Kausalitäten in Bezug auf wichtige Projekttermine beachtet werden können, bietet es sich an wichtige Termine, wie Releasetermine, Meilensteine oder Präsentationstermine in den Kontext miteinzubeziehen.

Unabhängig von Terminen ist es jedoch allgemein wünschenswert Mehrwert schaffende Funktionalitäten an den Kunden auszuliefern [11]. Gibt es unter den zu liefernden Funktionalitäten welche, die verhältnismäßig schlechte Werte in den Software-Metriken erhalten, wäre dies daher ein weiterer Faktor, der die Relevanz ebenjener Metriken beeinflusst. Im Unterschied zur ähnlichen Situation aus Kapitel 4.2.1, in der lediglich der schlechte Wert der Metrik ausschlaggebend ist, liegt hier der Fokus darauf eine auszuliefernde, noch fehlerbehaftete Funktionalität zu entdecken, sodass Metriken, welche zwar einen schlechten Wert haben, für die nächste Deadline aber nur eine untergeordnete Rolle spielen, dadurch vorerst nicht an Relevanz gewinnen.

Bewertung

Das aktuelle Projekt aus Anwendungssicht zu erfassen, gestaltet sich als sehr leicht, da SonarQube standardmäßig die Möglichkeit bietet jedes Projekt eindeutig über eine ID zu identifizieren. Dafür genügt es bereits diese ID aus der URL des Projekt-Dashboards auszulesen und abzuspeichern. Auf diese Weise kann sehr einfach festgestellt werden in welchem Projektkontext ein Entwickler arbeitet. Aus diesem Grund bietet sich die Projekt-ID als relevanter und einfach zu erfassender Kontextfaktor an.

Den feingranulareren Projektkontext aus Anwendungssicht zu erfassen, ist dafür deutlich schwieriger, weil es in SonarQube üblicherweise ohne Plugins nicht möglich ist zusätzlich zum Programmcode auch Projektdaten, wie z. B. zukünftige Deadlines, Meilensteine oder dergleichen, einzupflegen. Somit stünden abgesehen von der Projekt-ID noch weniger Informationen zur Verfügung, als bei den benutzerbezogenen Kontextfaktoren. Selbst wenn diese Funktionalität für SonarQube nachgerüstet wird, ist es ein signifikanter Aufwand die Projektdaten auch einzupflegen und aktuell zu halten.

Dass es wünschenswert wäre beispielsweise Deadlines zur Verfügung zu haben, sieht man an der dadurch ermöglichten Fähigkeit zur Planung, denn wenn aktueller Zustand, Fortschritt und zu erreichendes Ziel verglichen werden können, kann die Planung an die Realität angepasst werden [11]. Die verfügbare Zeit kann so beeinflussen, welche Metriken priorisiert werden müssen, wenn wenig oder viel Zeit für bestimmte Aufgaben zur Verfügung stehen. Wenn kurz vor einer Deadline beispielsweise noch ein paar größere Probleme und damit potentielle Programmfehler vorhanden sind, sollten Metriken, die

darauf hinweisen, gegenüber nicht funktional orientierten Metriken, wie z. B. „Lines of Code“ oder Code-Duplikate, priorisiert werden.

Ein ähnliches Problem wie bei der Einpflegung von Projektterminen ergibt sich auch wenn beachtet werden soll, welche Funktionalität als nächstes Ziel gesetzt wird. Das und der Zusammenhang zwischen einer Funktionalität und der dafür relevanten Metriken, die den Zustand der Funktion darstellen sollen, müssten ebenso manuell eingepflegt werden, denn die Wahl der zu beachtenden Metriken hängt von der „quantifizierbaren Frage“ ab und unterscheidet sich von Projekt zu Projekt [10]. Da hierfür ebenfalls ein signifikanter Aufwand nötig wäre, eignen sich die auszuliefernden Funktionalitäten ähnlich wie Projekttermine ebenfalls nicht als Kontextfaktor.

Aus denen in diesem Kapitel erwähnten Kandidaten haben sich aufgrund der Eigenschaften der Kontextfaktoren und der Schwierigkeiten bei der Erfassung zwar nützliche Erkenntnisse ergeben, aber nur die Projekt-ID eignet sich auch, um als Kontextfaktor verwendet zu werden.

	Deadlines	Projekt-ID	geforderte Funktionalitäten
K1	✗	✓	✗
K2	✓	✓	✓
K3	✓	✓	✓

Tabelle 4.4: Übersicht über die Bewertung der projektbezogenen Kontextfaktoren. Ein „✓“ steht dabei für ein erfülltes Kriterium, ein „✗“ für ein nicht erfülltes Kriterium und ein „❖“ steht für ein teilweise erfülltes Kriterium.

4.2.4 Umgebungsbezogene Kontextfaktoren

In diesem Kapitel werden zuletzt noch all die Faktoren erwähnt, die von der physischen Arbeitsumgebung bedingt werden. Ein potentieller Kandidat dafür ist die Tageszeit, mit der sich auch Baltrunas et al. [8] auseinandergesetzt haben. So kann ein Anwender

abhängig von der Tageszeit verschiedene Verhaltensweisen zeigen. Im Falle eines Entwicklers kann das bedeuten, dass dieser sich abhängig von der Tageszeit anderen Aufgaben widmet, sei es wegen des Lärmpegels im Büro, der spät am Abend ein anderer als am Vormittag ist, oder aufgrund von Terminen. Die aktuelle Zeit ist daher potentiell relevant, um zeitbedingte Varianz in den persönlichen Präferenzen zu erklären.

Informationen über die Umgebung können auch Anhand des Ortes, an dem der Benutzer sich befindet, inferiert werden. Anhand des Ortes kann beispielsweise in verteilten Projekten, bei denen beteiligte Personen an unterschiedlichen Standorten auch unterschiedliche Rollen wahrnehmen, auf die Zugehörigkeit zu einer bestimmten Gruppe geschlossen werden. Tatsächlich können auf Basis von Raumdaten noch detailliertere Informationen über einen Benutzer erschlossen werden, wie z. B. politische Einstellungen, der Gesundheitszustand oder persönliche Vorlieben [23]. Da verschiedene Rollen mit verschiedenen Aufgaben in Projekten auch verschiedene Bedürfnisse an Informationen haben, kann der Aufenthaltsort des Benutzers in gleicher Weise ein relevanter Kontextfaktor sein.

Die Internetverbindung kann eine Rolle in der Darstellung von Software-Metriken spielen, wenn größere Datenmengen übertragen werden oder der Empfang schlecht ist. Zwar können bestimmte Informationen wichtig sein, wenn sie jedoch z. B. Grafiken sind und eine Übertragung der Daten eine lange Zeit in Anspruch nähme, sodass der Benutzer seinen Versuch wahrscheinlich abbricht bevor die Übertragung erfolgreich vollendet wurde, kann es sinnvoller sein, anderen Informationen den Vortritt zu gewähren, um die verfügbare Bandbreite besser zu nutzen.

Bewertung

Erfahrungsgemäß lässt sich der Aufenthaltsort eines Benutzers über eine Webschnittstelle nur sehr ungenau bestimmen, solange kein Zugriff auf GPS-Daten, welche zumindest bei heutigen Smartphones verfügbar sind, besteht. Die Spezifikation der *Geolocation API* des W3C [4] sieht nämlich keine vorgeschriebene Genauigkeit vor und kann bei Desktop-PCs je nach Browser-Implementierung nach eigener Erfahrung mehrere Kilometer betragen. Daher ist davon auszugehen, dass in vielen Fällen eine Einteilung in z. B. Kunde und Dienstleister aufgrund ungenauer Daten nicht in die Realität umgesetzt werden kann. Auch wenn bei allen Geräten genaue Daten vorhanden wären, müssten

die Rohdaten vorverarbeitet werden, da sie sonst als Rauschen wirken und damit die Qualität der Empfehlungen reduzieren können [38].

Möchte man die Geschwindigkeit der Internetverbindung wissen, bleibt nach eigenem, gegenwärtigem Kenntnisstand nichts anderes übrig, als einen Download zu starten und zu messen wie lange dieser andauert. Dieses Verfahren kann jedoch nur eine grobe Abschätzung geben, da viele Faktoren, wie z. B. der Netzzustand oder andere Downloads, die Bandbreite zu einem Zeitpunkt bestimmen, weshalb die Verbindungsgeschwindigkeit von Messung zu Messung variieren kann. Eine Besonderheit von SonarQube ist es außerdem, dass bevor etwas von der Webseite angezeigt wird erst die benötigten Stylesheets und Browserskripte en bloc heruntergeladen werden. Erst danach wird mit dem Rendern der Seite begonnen, wovon das meiste ohne Internetverbindung abläuft. Daher ist es gleich aus zwei Gründen nicht praktikabel die Verbindungsgeschwindigkeit zu messen:

1) Dadurch, dass erst ein Download (z. B. eines Bildes) benötigt wird, um die Geschwindigkeit zu erfassen, würde sich der Seitenaufruf verzögern oder es bliebe keine Zeit entsprechend auf die Geschwindigkeitsmessung zu reagieren.

2) Im Fall von SonarQube liegt eine Alles-oder-Nichts-Situation vor. Die in Kapitel 4.2.4 erwähnten Repriorisierungen würden selbst bei schlechten Internetverbindungen keinen großen Einfluss auf den Darstellungszeitpunkt einer bestimmten Metrik haben, da das Rendern einer Metrik erst nach dem Herunterladen aller benötigten Ressourcen geschieht und das vergleichsweise wenig Zeit in Anspruch nimmt.

Daher ist es nicht nur schwer einen zuverlässigen Wert für die aktuelle Verbindungssituation zu erlangen, sondern bringt auch keinen großen Mehrwert mit sich.

Aus den in diesem Kapitel genannten Gründen eignet sich daher nur die aktuelle Tageszeit als Faktor zur sinnvollen Erweiterung der Kontextinformationen, da sie durch den Einfluss von Verhaltensmustern oder wiederkehrenden Terminen geeignet, ist zeitliche Regelmäßigkeiten erkennbar zu machen. Deswegen fließt sie auch in die folgende Kontextdefinition mit ein.

	Ort	Zeit	Internet-verbindung
K1	❖	✓	❖
K2	❖	✓	✗
K3	✓	✓	✓

Tabelle 4.5: Übersicht über die Bewertung der umgebungsbezogenen Kontextfaktoren. Ein „✓“ steht dabei für ein erfülltes Kriterium, ein „✗“ für ein nicht erfülltes Kriterium und ein „❖“ steht für ein teilweise erfülltes Kriterium.

4.2.5 Ergebnisse

In diesem Kapitel wurde eine Vielzahl von möglichen Kandidaten für Kontextfaktoren besprochen. Die Wahl fällt dafür auf die Werteänderung der jeweiligen Metrik, die Projekt-ID und die Tageszeit. Aus den Überlegungen in diesem Kapitel ergeben sich aber auch abgesehen von der Auswahl der konkreten Faktoren einige interessante Erkenntnisse. Zum einen scheitert eine Verwendung als Kontextfaktor bei den meisten hier vorgestellten Kandidaten, weil die entsprechenden Daten schwer in einer praktisch anwendbaren Art und Weise erfasst werden können. Eine andere Erkenntnis ist, dass Kontextfaktoren sich von Interaktion zu Interaktion verändern müssen, um Veränderungen im Verhalten erklären und damit einen Mehrwert erzielen zu können [39].

4.3 Einordnung der Faktoren in eine Kontextdefinition

Aus den bisher erlangten Erkenntnissen darüber, welche Kontextfaktoren verwendet werden, soll an dieser Stelle der Begriff des „Kontexts“ auch formal definiert werden. Für das von Shi et al. [49] beschriebene Verfahren ist es nicht möglich tatsächliche Zahlenwerte zu verwenden. Stattdessen müssen diese Werte zuerst in je eine Ordinalskala pro Faktor eingeordnet werden, um dann jede mögliche Kombination dieser Ordinal-Werte mit einer natürlichen Zahl, die diese spezifische Kontextsituation identifiziert, zu kodieren. Bei der Qualität der Werteänderung ergibt sich diese Eigenschaft von selbst,

da es bereits seitens SonarQube lediglich die Einteilung in „steigend“, „konstant“ und „fallend“ gibt, während bei anderen Faktoren erst die entsprechenden Werte umgeformt werden müssen.

Jeder Kontextfaktor f wird als eine Dimension D des Kontextes C aufgefasst. Aus Kapiteln 4.2.1 bis 4.2.4 ergeben sich damit die folgenden Dimensionen:

1) Projekt-ID

2) Werteänderung

3) Zeit

Damit alle daraus resultierenden Kontextbeschreibungen eindeutig durch eine natürliche Zahl k identifiziert werden können, darf es keine zwei Interpretationen für k geben. Diese Anforderung ist dadurch sichergestellt, dass die Anzahl der möglichen Dimensionswerte n_D für die Dimensionen der Zeit und der Werteänderung konstant ist, wodurch immer dieselben Bit-Stellen von k für die Kodierung dieser zwei Dimensionen verwendet werden können. Werden zusätzliche Bits benötigt, um zusätzliche Projekt-IDs kodieren zu können, können größere Zahlen k verwendet werden ohne Mehrdeutigkeiten einzuführen.

Die Kodierung soll, um ein selektives Auslesen der Kontextinformationen zu ermöglichen, bitweise erfolgen. Jeder Dimension D stehen dafür entsprechend ihrer Größe n_D die folgende Anzahl an Bits zur Verfügung:

$$\lceil log_2(n_D) \rceil$$

Die Zahl k zur Identifizierung einer Kontextbeschreibung ergibt sich aus der Konkatenation der einzelnen kodierenden Bitstrings jeder Dimension, entsprechend der obigen Sortierung der Dimensionen.

Die Dimension der Werteänderung D_w soll, wie oben bereits erwähnt, die möglichen Werte *steigend*, *konstant* und *fallend* besitzen. Zur Kodierung der drei Zustände werden zwei Bit benötigt. Für die Zeit wird eine grobe Unterteilung in acht Blöcke á drei Stunden vorgenommen. Dementsprechend werden zur Kodierung der Dimension der Zeit D_z drei Bit benötigt. Die restlichen Bits können zur Kodierung der Projekt-IDs verwendet werden. Dafür lässt sich keine feste Zahl nennen, da mit der Zeit auch neue Projekte zu einer SonarQube-Instanz hinzugefügt werden können.

Projekt-ID	DeepMind	YouTube	GMail	Fibre
	00	01	10	11
Werteänderung	fallend	konstant	steigend	
	00	01	10	
Zeit	Morgen	Vormittag	Mittag	Nachmittag
	000	001	010	011
	Abend	später Abend	Nacht	früher Morgen
	100	101	110	111

Tabelle 4.6: Kodierungen der Werte in den jeweiligen Dimensionen.

Anhand dieser Definitionen lässt sich ein Kontext C mit dem Identifikator k als 3-Tupel C_k mit folgenden Eigenschaften schreiben:

$$C_k \in \{(p, w, z) \mid p \in D_p,\, w \in D_w,\, z \in D_z\,\}$$

Um die beschriebene Kodierungsmethode zu veranschaulichen folgt nun ein Beispiel: Das Tupel C_b = (*DeepMind*, *steigend*, *Abend*) ist ein möglicher Kontext. Aus Tabelle 4.6 kann nun die Kodierung des Tupels abgelesen werden und es gilt für dieses Beispieltupel C_b:

$$b = (0010100)_2 = (20)_{10}$$

Damit kann das Beispieltupel C_b mit der Zahl 20 identifiziert werden und es gilt

$$C_b = C_{20} = (DeepMind,\, steigend,\, Abend)$$

4.4 Erfassung und Speicherung der Daten

In den vorhergehenden Kapiteln lag der Fokus auf den Kontextfaktoren und den Grundlagen, wie die erfassten Daten strukturiert werden sollen. Dieses Kapitel beschäftigt sich mit der Datenerfassung und -speicherung. Bisher wurde zwar berücksichtigt, ob es grundsätzlich möglich ist bestimmte Daten zu erfassen, jedoch wurde dies nicht detailliert betrachtet. Daher soll das folgende Kapitel 4.4.1 detaillierteren Aufschluss über das genaue Vorgehen bringen und dazu nacheinander Ansätze zum Erfassen der einzelnen Kontextfaktoren liefern. In Kapitel 4.4.2 wird danach beschrieben wie die gesammelten Daten abgespeichert werden.

4.4.1 Erfassung der Kontextinformationen

Für die in einer SonarQube-Instanz verfügbaren Projekte wird jedem Projekt eine Zahl – beginnend mit 0 – zugewiesen. Die konkrete Zuordnung ergibt sich durch die lexikographische Ordnung der Projektnamen. Das Projekt, welches in einer alphabetisch sortierten Liste, an oberster Stelle steht erhält somit die ID 0, das Projekt an zweiter Stelle erhält die 1 usw. Wird im Laufe der Zeit ein weiteres Projekt der SonarQube-Instanz hinzugefügt muss lediglich darauf geachtet werden die IDs der Projekte zu aktualisieren.

Eine Werteänderung, wie im letzten Kapitel beschrieben, kann drei mögliche Zustände haben. Dies spiegelt sich auch in SonarQube wieder. Dort wird eine Veränderung durch Pfeile ausgedrückt. Bei quantitativen Metriken, also solche die nicht die Qualität des Codes widerspiegeln, werden schwarze Pfeile benutzt; bei qualitativen Metriken, die sehr wohl die Qualität des Codes widerspiegeln, sind das grüne Pfeile für positive und rote Pfeile für negative Entwicklungen. Fand keine Veränderung statt, wird auch kein Pfeil angezeigt [2]. Welcher Pfeil angezeigt wird, kann aus dem HTML-Quellcode der Dashboard-Seite ausgelesen werden. Wird ein grüner bzw. nach oben gerichteter Pfeil ausgelesen, dann wird für die Werteänderung des aktuellen Kontextes der Wert *steigend* gesetzt. Wird hingegen ein roter bzw. nach unten gerichteter Pfeil ausgelesen, dann wird der Wert *fallend* gesetzt. Kann kein Pfeil gefunden werden, muss angenommen werden, dass es keine Änderung gab und es wird der Wert *konstant* gesetzt.

Die Uhrzeit, zu der eine Interaktion stattfand, kann ebenfalls viel mehr verschiedene Werte annehmen, als hier unterschieden werden sollen. Da eine minutengenaue Auflösung darüberhinaus wenig sinnvoll ist, wird eine gröbere Einteilung, ähnlich zu [50], in acht gleich große Blöcke vorgenommen: *Morgen*, *Vormittag*, *Mittag*, *Nachmittag*, *Abend*, *später Abend*, *Nacht*, *früher Morgen*. Jeder Block umfasst dabei drei Stunden des Tages.

Die zu den entsprechenden Blöcken zugeordneten Zeiten sind die folgenden:

- *Morgen*: 6 Uhr - 9 Uhr
- *Vormittag*: 9 Uhr - 12 Uhr
- *Mittag*: 12 Uhr - 15 Uhr
- *Nachmittag*: 15 Uhr - 18 Uhr
- *Abend*: 18 Uhr - 21 Uhr
- *Später Abend*: 21 Uhr - 24 Uhr
- *Nacht*: 0 Uhr - 3 Uhr
- *Früher Morgen*: 3 Uhr - 6 Uhr

Bei einer Interaktion wird der zur Tageszeit, zu der die Interaktion stattfindet, passende Wert für die Dimension der Zeit, entsprechend der hier definierten Blöcke, eingetragen.

4.4.2 Struktur der Datensätze

Die auf diese Art gesammelten Kontextinformationen bilden zwar angemessen den Kontext, in der eine Interaktion zwischen Benutzer und Software-Metrik stattfindet, ab, jedoch müssen für wirklich aussagekräftige Datensätze diese Informationen mit einer Identifikation des Benutzers und der entsprechenden Metrik zusammengebracht werden.

Ein Datensatz, wie er auch in der Arbeit von Shi et al. [49] verwendet wird, der dem impliziten Feedback einer gemessenen Interaktion zwischen einem Benutzer und einer Metrik in einem Kontext entspricht, muss diese drei daran beteiligten Identitäten

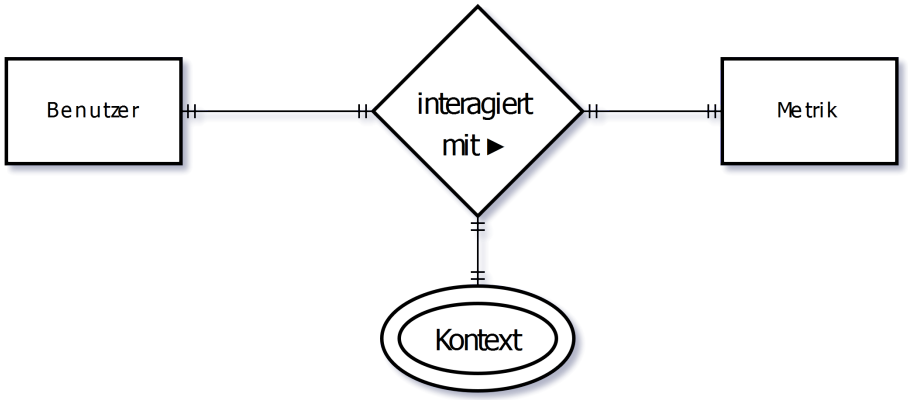


Abbildung 4.1: ER-Diagramm zur Modellierung der Interaktion zwischen Benutzern und Metriken

enthalten. Daher wird ein solcher Datensatz D ein 3-Tupel, das folgendermaßen strukturiert ist, darstellen:

$$D = (UID,\ MID,\ CID)$$

wobei UID für eine eindeutige Benutzerkennung (User-ID), MID für eine eindeutige Metrikkennung (Metric-ID) und CID (Context-ID) für die eindeutige Kodierung eines Kontextes, wie in Kapitel 4.3 beschrieben, steht [49]. Ein solcher Datensatz entspricht dabei der „Interagiert mit“-Relation in Abbildung 4.1.

5 Lernen der relevanten Software-Metriken

Nachdem in den letzten Kapiteln die formalen Grundlagen gelegt wurden, widmet sich dieses Kapitel dem Kern dieser Arbeit: Machine-Learning mittels des TFMAP-Algorithmus. Bereits in Kapitel 3.2.5 wurde ein kurzer Überblick über die Arbeit von Shi et al. [49] geboten. Die dort erwähnten Konzepte werden in diesem Kapitel nun detaillierter behandelt und erklärt. Daraus wird schließlich ein umfassendes Konzept ausgearbeitet, welches beschreibt wie aus den in Kapitel 4 charakterisierten Kontextfaktoren die Relevanz der einzelnen Software-Metriken erlernt werden kann. Die in diesem Kapitel verwendeten Formeln, Notationen und Kurzschreibweisen stammen dabei alle aus [49].

5.1 Datenstrukturen

Für den TFMAP-Algorithmus sind zwei Datenstrukturen essentiell. Zum einen ist dies das implizite Feedback, welches die Eingabedaten der Benutzer darstellt. Es ist die grundlegende Informationsquelle für den Lernprozess und liefert die Daten anhand derer gelernt wird.

Zum anderen sind dies die sogenannten latenten Modelle. Diese Modelle sind Matrizen, welche versuchen die Eigenschaften von Benutzern, Metriken und Kontexten zu beschreiben. Sie sollen durch den Lernprozess so angepasst werden, dass sie das implizite Feedback möglichst optimal erklären und dementsprechend gute Vorhersagen für Metriken machen können. Somit sind die latenten Modelle die Datenstrukturen, die beim Lernen angepasst werden und das erlernte Wissen speichern.

Diese beiden Datenstrukturen werden in den folgenden beiden Kapiteln näher vorgestellt und erläutert.

5.1.1 Implizites Feedback

Wird das Thema Benutzerfeedback behandelt, dann ist damit oft explizites Feedback gemeint. Ein Beispiel dafür sind Käuferbewertungen, die ein Kunde in einem Online-

Shop abgeben kann, um ein gekauftes Produkt für andere Kunden zu bewerten. Dabei muss ein Kunde immer selbst aktiv werden und überlegen, wie er das Produkt bewerten möchte. Der Aufwand kann dabei nur eine Bewertung mit Sternen sein oder auch ein geschriebener Kommentar. Dies bedeutet für den Benutzer jedoch in allen Fällen einen Mehraufwand während seinem Aufenthalt im Online-Shop. Da dieser Mehraufwand ein Hindernis darstellen kann, gibt der Kunde für ein Produkt möglicherweise überhaupt keine Bewertung ab.

Daher kann es erwünscht sein, stattdessen nur implizites Feedback zu sammeln. Im Gegensatz zu explizitem Feedback wird dabei die reguläre Interaktion zwischen Benutzer und Anwendung beobachtet, weshalb kein zusätzlicher Aufwand für den Benutzer verursacht wird. Hierbei wird z. B. erfasst auf welche Elemente in der Benutzeroberfläche geklickt wurde oder wie der Benutzer auf der Seite navigiert. Aus diesen Beobachtungen werden Schlüsse gezogen, die als Feedback interpretiert werden. Im Beispiel des Online-Shops könnte das Klicken auf ein Produkt, um genauere Informationen darüber zu erhalten, als Interesse interpretiert werden. Zwar stellt der Vorgang des Klickens einen Aufwand dar, jedoch geschieht dies im Rahmen der regulären Interaktion, da davon ausgegangen werden kann, dass das Aussuchen und Kaufen interessanter Produkte das beabsichtigte Ziel des Benutzers ist. Nach dem Kauf eine Bewertung abzugeben ist hingegen zusätzlicher Aufwand, welcher den Kunden seinem Ziel, die von ihm gewünschten Produkte zu finden und zu kaufen, nicht näher bringt.

Bei der Verwendung von SonarQube ist das primäre Ziel des Benutzers die Ergebnisse der Code-Analyse zu überprüfen. Er soll dabei nicht zusätzlich angeben müssen, welche Informationen dabei am hilfreichsten für ihn sind, da dies eine Ablenkung darstellen würde. Stattdessen soll, genauso wie im Beispiel des Online-Shops, implizites Feedback darüber, was für den Benutzer relevant ist, gesammelt werden. Dafür werden Interaktionen mit Software-Metriken – genauer gesagt: auf welche Metrik der Benutzer klickt – analysiert und dabei Kontextinformationen, wie es in Kapitel 4 beschrieben ist, erfasst. Jede Interaktion in Form von Anklicken einer Metrik, um detailliertere Informationen davon zu erhalten, kann dabei als potentielles Interesse gewertet werden.

Dieses implizite Feedback, wie auch schon von Shi et al. in [49] verwendet, ist es, das in den weiteren Schritten benötigt wird, um entsprechend der latenten Benutzerpräferenzen die zur Verfügung stehenden Software-Metriken nach ihrer Relevanz für den Benutzer bewerten zu können. Gespeichert wird dieses implizite Feedback in ei-

nem binären Tensor Y. Die Elemente von Y werden mit y_{mik} bezeichnet, wobei m einen Benutzer, i ein Item, was im Kontext dieser Arbeit einer Metrik entspricht, und k einen Kontext identifiziert. Es gilt: $y_{mik} = 1$ genau dann, wenn für einen Benutzer m eine Interaktion mit einem Item i im Kontext k stattfand; andernfalls ist $y_{mik} = 0$. Ein Vektor Y_{mk} bezeichnet das implizite Feedback eines Benutzers m im Kontext k für alle Items.

5.1.2 Latente Benutzer-, Metrik- und Kontextmodelle

Die latenten Modellen für Benutzer, Metriken und Kontexte versuchen jene Entitäten bestmöglich zu beschreiben, um damit Vorhersagen bezüglich der Relevanz einer Kombination von Benutzer, Metrik und Kontext treffen zu können. Jede dieser Entitäten bekommt dafür einen Eigenschaftsvektor fester Größe zugewiesen, dessen Werte initial zufällig gewählt werden, da anfangs kein Wissen über diese latenten Eigenschaften verfügbar ist. Diese Eigenschaftsvektoren werden für jeweils alle Benutzer, Metriken und Kontexte zu Matrizen, den latenten Modellen, zusammengefasst.

Diese latenten Eigenschaften sind, wie der Name schon sagt, jedoch nicht offensichtlich und müssen erst aus vorhandenen Daten, hier dem impliziten Feedback des Benutzers, erlernt werden. Ferner sind dies oft auch keine direkt messbaren Größen, wie z. B. Körpergröße eines Benutzers, Preis eines Gegenstandes oder die Tageszeit in einem Kontext; sie sind eher mit persönlichen, womöglich unterbewussten, Vorlieben eines Menschen oder beispielsweise der empfundenen Stimmung eines Liedes zu vergleichen und werden deshalb auch als latent bezeichnet. Matrix-Faktorisierungs-Verfahren haben den Vorteil latente Eigenschaften der darin modellierten Benutzer und Items zu offenbaren, was Shi et al. [49] durch die Erweiterung auf Tensoren auch gleichzeitig für Kontextinformationen ermöglichen. Aus diesem Grund wird mit der Tensor-Faktorisierung ein mit der Matrix-Faktorisierung verwandtes Verfahren verwendet, um die latenten Modelle erlernen zu können.

Da diese verborgenen Eigenschaften nicht benannt werden können, d. h. es ist nicht klar, was sie konkret bedeuten, können sie auch nicht einfach gemessen werden. Stattdessen wird eine gewisse Anzahl latenter Eigenschaften vermutet und durch Machine Learning werden deren Werte approximiert. Diese vermuteten Eigenschaften beschreiben damit Entitäten, wie z. B. Benutzer. Zwar kann im Allgemeinen nicht mit Sicherheit gesagt werden, welche Bedeutung diese latenten Eigenschaften haben, aber das ist auch

nicht nötig, denn sie müssen nur erlernt und in Vorhersagen verwendet werden können; sie zu interpretieren ist nicht erforderlich. Es genügt vollkommen zu wissen, dass die Modelle tatsächlich eine Aussagekraft haben. Dieses Wissen wird durch den Lernprozess gewonnen, indem die Werte der latenten Modelle angepasst werden, um das beobachtete implizite Feedback, was der Interaktion des Benutzers mit der Benutzeroberfläche entspricht, möglichst gut erklären zu können. [49]

Nach dem Lernprozess können diese latenten Modelle dazu verwendet werden Empfehlungslisten zu generieren. Das sind nach Relevanz sortierte Listen von Metriken, welche für eine Kombination aus Benutzer und Kontext angeben, welche Metriken am relevantesten sind. Die Relevanz einer Metrik für einen Benutzer in einem bestimmten Kontext ergibt sich aus dem *Predicted Relevance Score* aus Formel (5.2), welche im nächsten Abschnitt genauer beschrieben wird. Anhand der so erlernten Modelle können nach dem Lernprozess auf Basis der Erfahrungen bzw. Trainingsdaten des impliziten Feedbacks Vorhersagen für einen Benutzer gemacht werden. Damit erfüllen die latenten Modelle den Zweck, ein abstraktes Verständnis über die beteiligten Entitäten zu erlangen und diese in einer für Vorhersagen verwendbaren Art und Weise zu beschreiben.

In dieser Arbeit wird der Notation von [49] entsprochen. Das bedeutet, dass $U \in \mathbb{R}^{M \times D}$, $V \in \mathbb{R}^{N \times D}$ und $C \in \mathbb{R}^{K \times D}$ Matrizen sind, die den eben beschriebenen latenten Modellen für Benutzer, Items bzw. Metriken sowie Kontexte entsprechen. Hierbei ist D die Anzahl der vermuteten Eigenschaften einer Entität. Diese Modellmatrizen bestehen aus einzelnen, D-dimensionalen Eigenschaftsvektoren $U_m \in \mathbb{R}^D$, $V_i \in \mathbb{R}^D$ bzw. $C_k \in \mathbb{R}^D$, welche die latenten Eigenschaften für einzelne Benutzer m, Items i oder Kontexte k repräsentieren.

5.2 Die Zielfunktion

Für die Optimierung der latenten Modelle wird eine Zielfunktion benötigt, welche ein direktes Maß dafür ist, wie gut diese Modelle und die durch sie generierten Empfehlungslisten sind. Gelernt wird, indem der Wert der Zielfunktion optimiert wird. Kapitel 5.2.1 beschreibt die in [49] gewählte Zielfunktion *Mean Average Precision*. Danach wird in Kapitel 5.2.2 darauf eingegangen, warum eine „geglättete“, stetige Version dieser Funktion für die Optimierung benötigt wird.

5.2.1 Mean Average Precision

Die *Mean Average Precision* (MAP) ist in ihrer Essenz ein Maßstab zur Bewertung von Listen von Dokumenten [33] bzw. in dem hier vorliegenden Fall von Metriken. Beispiele für solche Listen stellen Ergebnisse einer Suchmaschinenanfrage dar. Bei deren Qualität ist vor allem die Relevanz der einzelnen Einträge, aber auch die Sortierung in welcher sie in der Liste auftreten, von Bedeutung. Bei einer Suchmaschine sind diese Anforderungen nötig, da oft viele tausend Ergebnisse zurückgeliefert werden, welche ein Benutzer nicht alle überprüfen möchte. Stattdessen sollen die relevanten Ergebnisse am Anfang der Liste stehen. Dieses Verhalten ist immer dann gewollt, wenn es nicht wünschenswert ist alle Elemente einer solchen „Empfehlung“ zu verwenden oder zu überprüfen, was auch in dieser Arbeit bei der Bestimmung der Menge der relevantesten Software-Metriken der Fall ist. Die *Average Precision* (AP) ist ein Maß für Empfehlungslisten, mit der Eigenschaft irrelevante Elemente auf guten Listenpositionen stärker zu bestrafen als auf schlechten Listenpositionen [32, 49], und ist definiert als die Fläche unter dem Precision-Recall-Diagramm für eine einzelne Empfehlungsliste. Dieses PR-Diagramm ist wiederum ein Maß für die bei verschiedenen Trefferquoten erzielte Genauigkeit [33], also der prozentuale Anteil relevanter Ergebnisse bei der Betrachtung von verschieden großen Teilen der Liste. So werden beim AP-Maß relevante Elemente auf guten Listenpositionen auch automatisch belohnt, da irrelevante Elemente dadurch automatisch weiter hinten platziert sein müssen. Für den TFMAP-Algorithmus wird der AP-Wert dafür wie folgt berechnet:

$$AP_{mk} = \frac{1}{\sum_{i=1}^{N} y_{mik}} \sum_{i=1}^{N} \frac{y_{mik}}{r_{mik}} \sum_{j=1}^{N} y_{mjk} \mathbb{I}(r_{mjk} \leq r_{mik}) \tag{5.1}$$

Hierbei bezeichnen die Indizes m, i und k jeweils einen Benutzer, eine Metrik bzw. einen Kontext. Hier ist y_{mik} ein Wert aus dem binären Tensor $Y \in \{0,1\}^{M \times N \times K}$, welcher dem in Kapitel 5.1.1 beschriebenen impliziten Feedback entspricht, und gibt dementsprechend an, ob eine Interaktion zwischen einem Benutzer m und einer Metrik i in einem Kontext k stattfand. Die Indikatorfunktion $\mathbb{I}(\cdot)$ hat den Wert 1, wenn ihre Bedingung wahr ist und ansonsten den Wert 0. r_{mik} entspricht dem Ranking einer Metrik i in einer Empfehlungsliste für einen festgelegten Kontext k und Benutzer m. Dieses Ranking ergibt sich durch eine Sortierung der Software-Metriken gemäß dem *Predicted Relevan-*

ce Score f, welcher durch das Skalarprodukt der Eigenschaftsvektoren von Benutzer, Metrik und Kontext in Formel (5.2) beschrieben ist.

$$f_{mik} = \langle U_m, V_i, C_k \rangle = \sum_{d=1}^{D} U_{md} V_{id} C_{kd} \tag{5.2}$$

In dieser Formel bezeichnen U, V und C die in Kapitel 5.1.2 beschriebenen latenten Modelle. U_m, V_i und C_k sind die jeweiligen Eigenschaftsvektoren der konkreten Benutzer, Metriken und Kontexte. Der *Predicted Relevance Score* ist ein Maß dafür, wie relevant eine Metrik i für einen Benutzer m im Kontext k ist [49].

Bei der *Average Precision* bezieht sich ein Wert auf genau eine Empfehlungsliste, was in dieser Arbeit einer sortierte Liste an Software-Metriken für einen festen Benutzer in einem spezifischen Kontext entspricht. Da für die benötigte Zielfunktion jedoch eine Aussage über alle Benutzer und alle möglichen Kontextsituationen notwendig ist, damit die optimierten latenten Modelle auch eine externe Validität besitzen, muss das AP-Maß weiter zur *Mean Average Precision* verallgemeinert werden. Wie der Name bereits verrät, wird dazu der arithmetische Mittelwert der AP-Werte über alle Kombinationen aus Benutzer und Kontext gebildet [49]. Dadurch ergibt sich die Formel für MAP direkt aus Formel (5.1) mit:

$$MAP = \frac{1}{MK} \sum_{m=1}^{M} \sum_{k=1}^{K} AP_{mk} \tag{5.3}$$

Aufgrund dieses Zusammenhangs zwischen AP- und MAP-Wert kann MAP als die durchschnittliche Fläche unter den Precision-Recall-Diagrammen aller Empfehlungslisten verstanden werden [33].

Damit wird der Name „TFMAP“ auch etwas klarer, denn das Skalarprodukt aus Formel (5.2) wird auch als Tensorfaktorisierung bezeichnet, was der Ursprung der ersten beiden Buchstaben ist, wodurch sich der Name TFMAP aus „tensor factorization for MAP“ ergibt.

Abschließend soll noch auf den Zusammenhang zwischen den latenten Modellen und dem daraus resultierenden MAP-Wert hingewiesen werden, da dieser Zusammenhang das bestimmende Kriterium ist, warum sich die *Mean Average Precision* als Zielfunktion eignet. Die Tensorfaktorisierung in Formel (5.2), welche die Kernoperation des TFMAP-Algorithmus darstellt, zeigt, dass der *Predicted Relevance Score* direkt und

ausschließlich von den latenten Modellen abhängt. Außerdem ist er eine direkte Bewertung für die Relevanz einer Kombination aus Benutzer, Item und Kontext. Dieser Wert ist das einzige Kriterium, das die Itemrankings r_{mik} für einen festgelegten Benutzer m und Kontext k bestimmt. Da MAP ein direktes Bewertungsmaß für die Qualität aller Empfehlungslisten ist, wird damit auch die Qualität der latenten Modelle gemessen, da die Empfehlungslisten bzw. die Item-Rankings direkt von den Modellen abhängen.

5.2.2 Geglättete MAP

Aus der Formel für die *Average Precision* geht hervor, dass diese Funktion nicht stetig ist, denn die Rankings r_{mik} einzelner Metriken ändern sich in nicht-stetiger Weise, da sie nur ganzzahlige Werte annehmen können. Aufgrund dieser Eigenschaft lassen sich in der Form keine Standartoptimierungsalgorithmen, wie das Gradientenverfahren, anwenden, da diese eine stetige Zielfunktion benötigen [49, 19]. Um diesem Problem zu begegnen, müssen die nicht-stetigen Anteile der AP-Funktion mit einer stetigen Funktion approximiert werden. Konkret bedeutet das, dass die Rankings r_{mik}, sowie die Indikatorfunktion $\mathbb{I}(\cdot)$ angenähert werden müssen. Nach Chapelle et al. [17] bietet sich für diese Approximation die stetige und außerdem streng monoton steigende logistische Funktion (5.4) an.

$$g(x) = \frac{1}{1 + e^{-x}} \tag{5.4}$$

Diese ist eine Sigmoidfunktion, die wegen ihrer Form besonders dafür geeignet ist die Indikatorfunktion $\mathbb{I}(\cdot)$ anzunähern. Dahinter steht die Annahme, dass je höher der *Predicted Relevance Score* einer Metrik i in Relation zu einer anderen Metrik j ist, desto besser sollte diese Metrik i platziert werden. Dabei werden Metriken jedoch nie umsortiert, da die logistische Funktion streng monoton ist. Die Approximation der Indikatorfunktion sieht damit folgendermaßen aus [49]:

$$\mathbb{I}(r_{mjk} \leq r_{mik}) \approx g(f_{mjk} - f_{mik}) = g(\langle U_m,\, V_j - V_i,\, C_k \rangle) \tag{5.5}$$

Die zweite Substitution, die vorgenommen werden muss betrifft das Ranking r_{mik}. Es gibt zwar eine durchdachte Approximation dafür, diese wird jedoch nicht benötigt, da

nur der Kehrwert $\frac{1}{r_{mik}}$ verwendet wird [17, 49]. Deswegen soll $\frac{1}{r_{mik}}$ ebenfalls mit derselben logistischen Funktion auf folgende Weise approximiert werden:

$$\frac{1}{r_{mik}} \approx g(f_{mik}) = g(\langle U_m, V_i, C_k \rangle) \tag{5.6}$$

Durch die Verwendung der logistischen Funktion werden die Abstände zwischen den einzelnen Rankingplätzen zwar verzerrt, aufgrund der Monotonie werden diese aber nie umgeordnet, was bei der Approximation der Ranking-Plätze besonders wichtig ist. Davon abgesehen führen große Werte für f_{mik} zu Werten nahe 1 für die darauf angewandte logistische Funktion, was einen kleinen und damit guten Ranking-Wert r_{mik} bedeutet. Umgekehrt führen kleine Werte für f_{mik} zu großen und damit schlechteren Ranking-Werten r_{mik} [49].

Führt man die in diesem Kapitel beschriebenen Substitutionen in der AP-Funktion durch, erhält man die folgende, geglättete und stetige Version der *Average Precision*-Funktion [49]:

$$AP_{mk} = \frac{1}{\sum_{i=1}^{N} y_{mik}} \sum_{i=1}^{N} y_{mik} g(\langle U_m, V_i, C_k \rangle) \sum_{j=1}^{N} y_{mjk} g(\langle U_m, V_j - V_i, C_k \rangle) \tag{5.7}$$

5.3 Lernen mit TFMAP

Die vorherigen Kapitel haben die letzten Vorbereitungen behandelt, die benötigt werden, um den Lernprozess mithilfe des TFMAP-Algorithmus zu beschreiben. Das folgende Kapitel hat die Optimierung der latenten Modelle durch die stetige MAP-Funktion mithilfe des Gradientenverfahrens zum Inhalt. Es wird beschrieben wie dieses Verfahren funktioniert und wie die bisherigen Erkenntnisse genutzt werden können, um es auf das vorliegende Problem anzuwenden.

5.3.1 Optimierung durch das Gradientenverfahren

Zum Lösen allgemeiner Optimierungsprobleme gehört das Gradientenverfahren (Engl.: *method of steepest decent*) zu den einfachen Standardansätzen [52]. Es kann bei reellwertigen, differenzierbaren Funktionen verwendet werden, um Extremstellen zu fin-

den. Diese sog. Zielfunktion muss differenzierbar sein, damit für den zu untersuchenden Definitionsbereich der Funktion auch die benötigten Gradienten berechnet werden können [19]. Beim Gradientenverfahren wird in jedem Schritt der Gradient der Funktion an einer Stelle x berechnet. Der Gradient gibt die „Richtung“ an, in der der Wert der Zielfunktion am stärksten steigt, also dem Optimum näherkommt. Die Variable x wird daraufhin entsprechend dem Gradienten verändert, sodass im nächsten Schritt eine Stelle x' betrachtet wird, an der der Wert der Zielfunktion näher am Optimum ist als an der Stelle x. Diese Schritte werden wiederholt, bis sich der Wert der Zielfunktion nicht mehr verbessern lässt, woraufhin angenommen wird, dass das Optimum erreicht wurde.

Das Verfahren lässt sich mit der Metapher eines Bergsteigers an einem Berg gut veranschaulichen. Es sei angenommen die Positionskoordinaten des Bergsteigers, beispielsweise als Längen- und Breitengrad angegeben, sind die Variablen einer Funktion, die die Höhe eines Ortes über dem Meeresspiegel beschreibt. Der Graph dieser Funktion ist die Oberfläche des Berges, auf dem sich der Bergsteiger bewegen kann. Eine Beispielfunktion zur Veranschaulichung kann Abbildung 5.1 entnommen werden. Das Ziel des Bergsteigers ist es nun vom Gipfel des Berges zurück ins Tal zu finden. Er hat jedoch die Orientierung verloren und kann nicht sehr weit sehen, weshalb er beschließt immer in die Richtung zu gehen, in der es am steilsten bergab geht. Auf diese Weise wird er immer tiefer gelangen, bis er ein Plateau, was einem lokalen Optimum entspricht, oder das Tal, das globale Optimum, erreicht.

Wie durch die Metapher erkennbar ist, besitzt dieses Verfahren keine Weitsicht, da es nur den Gradienten der aktuellen Position betrachtet und damit anfällig für lokale Optima ist. Das muss jedoch nicht zwangsweise ein Problem darstellen, da das Verhalten der Funktion von der Struktur des Problems abhängt. Im betrachteten Definitionsbereich können auch schlichtweg keine lokalen Optima, in denen der Algorithmus vorzeitig terminieren könnte, vorhanden sein.

Das Gradientenverfahren wurde auch von Shi et al. in [49] für das Erlernen der latenten Modelle verwendet. Da es sich dabei als effektiv bewährt hat, wird es auch im Rahmen dieser Arbeit für die Optimierungsphase verwendet. Die dabei verwendete Zielfunktion ist im Grunde die durch Kapitel 5.2.2 erhaltene geglättete Version der MAP-Funktion. Für den Einsatz als Zielfunktion haben Shi et al. [49] eine kleine Anpassung vorgenommen und zwar wird zur Regularisierung die Frobenius-Norm der la-

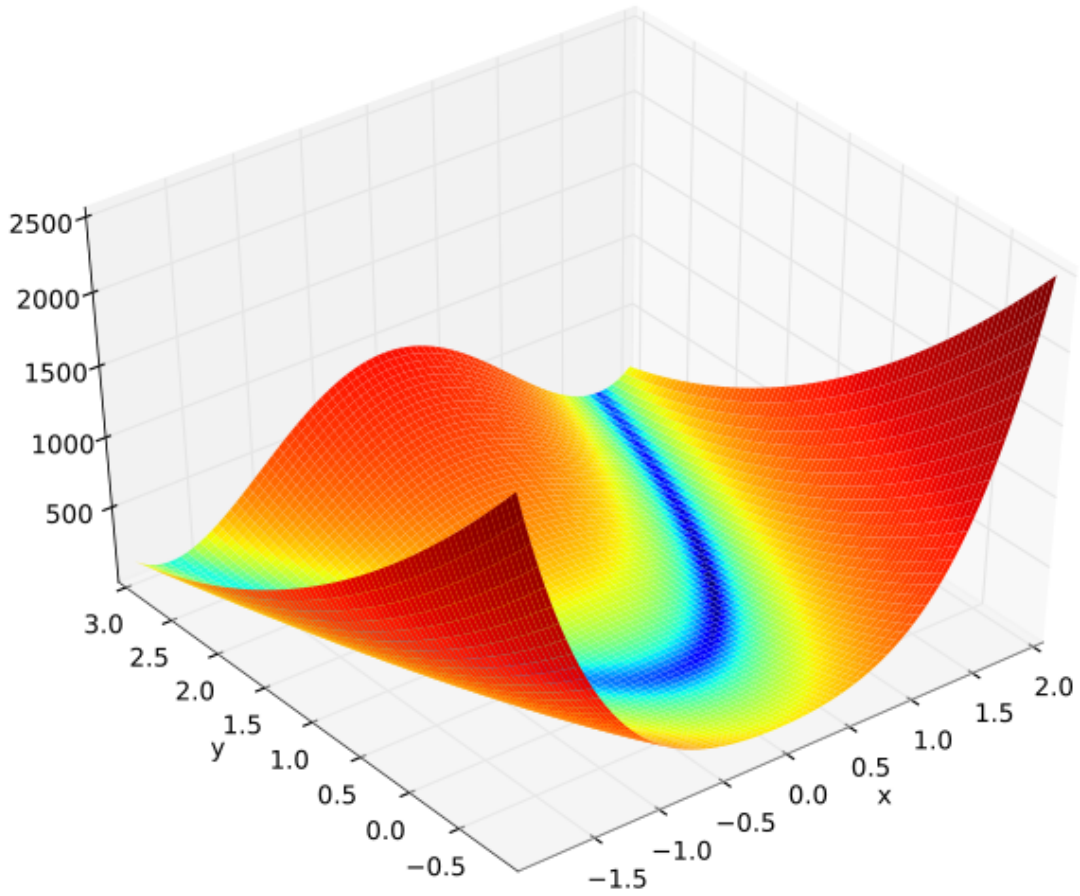


Abbildung 5.1: Rosenbrock-Funktion visualisiert von *Morn the Gorn* (eigenes Werk) [Public domain], via Wikimedia Commons. Die Rosenbrock-Funktion ist eine gängige Benchmark-Funktion für Optimierungsalgorithmen.

tenten Modelle, multipliziert mit einem Gewichtungsfaktor $-\lambda$, hinzuaddiert. Dies ist nötig, um eine Überanpassung der Modelle zu verhindern, da Vorhersagen mit Realdaten andernfalls fehleranfälliger wären [48, 49]. Daraus ergibt sich die folgende, auf MAP basierende Zielfunktion:

$$\begin{aligned} L(U,\, V,\, C) = & \sum_{m=1}^{M} \sum_{k=1}^{K} \frac{1}{\sum_{i=1}^{N} y_{mik}} \\ & \sum_{i=1}^{N} \Big[y_{mik} g(\langle U_m,\, V_i,\, C_k \rangle) \times \sum_{j=1}^{N} y_{mjk} g(\langle U_m,\, V_j - V_i, C_k \rangle) \Big] \\ & - \frac{\lambda}{2} (\|U\|^2 + \|V\|^2 + \|C\|^2) \end{aligned} \tag{5.8}$$

Die für das Gradientenverfahren benötigten Ableitungen sind in den Formeln (5.10), (5.11) und (5.12) [49] zu sehen. Um die Notation der Ableitungen übersichtlicher zu gestalten, wurden auch folgende Kurzschreibweisen aus [49] übernommen:

$$\begin{aligned} f_{mik} &:= \langle U_m,\, V_i,\, C_k \rangle \\ f_{m(j-i)k} &:= \langle U_m,\, V_j - V_i,\, C_k \rangle \\ \delta &:= g'(f_{mik}) \Sigma_{j=1}^{N} y_{mjk} g(f_{m(j-i)k} - g(f_{mik}) \Sigma_{j=1}^{N} y_{mjk} g'(f_{m(j-i)k}) \end{aligned}$$

Dabei bezeichnet $g'(x)$ hier und in den folgenden Formeln die Ableitung der logistischen Funktion $g(x)$ (5.4) und es gilt:

$$g'(x) = \frac{e^x}{(e^x + 1)^2} \tag{5.9}$$

Darüber hinaus wird mit dem Symbol $\odot$ die elementweise Multiplikation bezeichnet.

$$\begin{aligned} \frac{\partial L}{\partial U_m} = & \sum_{k=1}^{K} \frac{1}{\sum_{i=1}^{N} y_{mik}} \sum_{i=1}^{N} y_{mik} \\ & \Big[\delta * (V_i \odot C_k) + g(f_{mik}) \sum_{j=1}^{N} y_{mjk} g'(f_{m(j-i)k}) (V_j \odot C_k) \Big] - \lambda U_m \end{aligned} \tag{5.10}$$

$$\frac{\partial L}{\partial C_k} = \sum_{m=1}^{M} \frac{1}{\sum_{i=1}^{N} y_{mik}} \sum_{i=1}^{N} y_{mik} \Big[\delta * (U_m \odot V_i) + g(f_{mik}) \sum_{j=1}^{N} y_{mjk} g'(f_{m(j-i)k})(U_m \odot V_j)\Big] - \lambda C_k \tag{5.11}$$

$$\frac{\partial L}{\partial V_i} = \sum_{m=1}^{M} \sum_{k=1}^{K} \frac{y_{mik}(U_m \odot C_k)}{\sum_{i=1}^{N} y_{mik}} \sum_{j=1}^{N} y_{mjk} \Big[g'(f_{mik}) g(f_{m(j-i)k}) + \Big(g(f_{mjk}) - g(f_{mik})\Big) g'(f_{m(j-i)k})\Big] - \lambda V_i \tag{5.12}$$

Eine detaillierte Beschreibung der Ableitung $\frac{\partial L}{\partial V_i}$ kann dem Anhang von [49] entnommen werden, da sich die Ableitung aufgrund der Abhängigkeiten zu anderen latenten Eigenschaftsvektoren schwieriger als bei den anderen beiden Gradienten gestaltet.

An dieser Stelle soll auch auf die höhere Berechnungskomplexität der Ableitung $\frac{\partial L}{\partial V_i}$ hingewiesen werden. Unter der Annahme, dass die Anzahl der Benutzer-Item-Interaktionen $|Y|$ deutlich größer als die Anzahl der Items bzw. Metriken selbst ist, ist die Komplexität mit $\mathcal{O}(DN|Y|)$ mehr als quadratische bezüglich der Anzahl an Items. Für dieses Problem ist eine *Fast Learning*-Version des Algorithmus, der die Komplexität auf $\mathcal{O}(D|Y|)$ reduziert, in der Arbeit von Shi et al. [49] beschrieben. Da die erwartete Anzahl der zu bewertenden Metriken im Rahmen von SonarQube jedoch als weitgehend konstant und klein betrachtet werden kann, wird aufgrund des Overheads beim *Fast Learning TFMAP*-Algorithmus in dieser Arbeit auf diese Optimierung verzichtet. Für andere Szenarien, in denen die Anzahl der zu bewertenden Items wesentlich größer ist, ist es ausdrücklich empfohlen aufgrund der niedrigeren Komplexität die *Fast Learning*-Optimierung zu verwenden.

5.3.2 Parameter

Aus den Formeln und Definitionen zu TFMAP sind noch zwei Variablen nicht behandelt worden. Dies ist zum einen die Dimensionalität der Eigenschaftsvektoren D und zum anderen ist das der Regularisierungsfaktor λ. Für die Anwendung des Gradientenver-

fahrens ist auch noch der Lernfaktor γ, der noch nicht erwähnt wurde, von Bedeutung. Durch ihn werden die Gradienten bei der Berechnung eines Schrittes des Verfahrens gewichtet. Er reguliert damit wie stark sich die Werte der Modelle zwischen zwei Verfahrensschritten verändern können. Davon hängt ab, ob die Veränderungen klein genug sind, damit ein Optimum auch gefunden werden kann oder zu groß sind, sodass der Algorithmus das Optimum verfehlt und um das Optimum oszilliert.

Der Regularisierungsfaktor λ kontrolliert hingegen wie stark sich die latenten Modelle an den Trainingsdaten orientieren, indem es als Gewicht für die Frobeniusnormen in der Zielfunktion aus Formel (5.8) fungiert. Damit wird die Tendenz zur Überanpassung der erlernten, latenten Modelle reguliert [48], was die externe Validität und Qualität der produzierten Rankings bei der Verwendung von Realdaten nach der Lernphase verbessert.

Die Dimensionalität der Eigenschaftsvektoren - und damit auch der Modelle - beschreibt in grober Weise die Ausdruckskraft derselbigen, indem es die Granularität der zuweisbaren Eigenschaften bestimmt. Dementsprechend sind die resultierenden Modelle komplexer und können auch feingranulare, zugrundeliegende Einflüsse im Lernprozess aufgreifen oder sind simpel und lassen sich mit weniger Trainingsdaten dennoch angemessen erlernen. Wie anhand der vorherigen Abschnitte auch zu sehen ist, beeinflusst die Dimensionalität D auch direkt die Berechnungs-Komplexität des TFMAP-Algorithmus, da die Anzahl der notwendigen Multiplikationen proportional zur Größe der Modelle ist.

Für die eben beschriebenen Parameter können i. Allg. keine optimalen Werte angegeben werden. Stattdessen müssen diese empirisch ermittelt werden, da sie auch stark vom zugrundeliegenden Problem abhängen. Als Referenz geben Shi et al. [49] die Werte $\lambda = 0.001$ und $\gamma = 0.001$ an. Für die Dimensionalität wurde ein Wert von $D = 10$ veranschlagt. Abhängig von der Komplexität der zu erfassenden latenten Eigenschaften ist es dabei womöglich wünschenswert diese Zahl zu erhöhen, steht hingegen die Berechnungskomplexität im Vordergrund kann D auch kleiner gewählt werden, wobei jedoch möglicherweise Abstriche in der Qualität der Ergebnisse in Kauf genommen werden müssen.

5.3.3 Bedeutung der Daten

Die aus der Weboberfläche von SonarQube gewonnenen Trainingsdaten sind die bestimmenden Einflüsse in Bezug auf die Ergebnisse des Algorithmus. Deren Qualität und Quantität beeinflussen direkt die Qualität der produzierten Voraussagen. Je mehr aussagekräftige Datensätze vorhanden sind, desto genauer und vollständiger können alle möglichen Interaktionsszenarien erfasst und miteinbezogen werden, was ein umfassenderes Gesamtbild ermöglicht. Des Weiteren wird durch eine große Informationsbasis, welche auf einem Trainingsdatensatz basiert, der alle Eventualitäten ausschöpft, die externe Validität des Rankings von Software-Metriken gesteigert und damit die Ergebnisse für unbekannte, reale Daten bzgl. des impliziten Feedbacks verbessert.

Die Größen der in anderen Arbeiten verwendeten Datensätze unterscheiden sich teilweise beachtlich. Der *Appazaar*-Datensatz enthält Beispielsweise 300.469 wie in Kapitel 4.4.2 beschriebene 3-Tupel mit 1.767 Benutzern, 7.701 Items und 9 verschiedenen Kontextsituationen [14, 49]. Weitaus bekannter und noch wesentlich größer ist der *Netflix Prize*-Datensatz [12], welcher aus dem gleichnamigen Wettbewerb stammt. Er enthält 100.480.507 Einträge von ungefähr 480.000 Benutzern und 17.770 Items [22, 36]. Dabei ist jedoch anzumerken, dass dem *Netflix Prize*-Datensatz die Kontextinformationen fehlen. Der TFMAP-Algorithmus wurde in [49] daher nur anhand des *Appazaar*-Datensatz getestet. In dem hier behandelten Anwendungsszenario mit SonarQube ist die zu erwartende Größe der Benutzerbasis in einem Anwendungskontext, welcher im Extremfall aus nur einer einzigen Instanz bestehen kann, jedoch um mehrere Größenordnungen kleiner abzuschätzen.

Bei der Verwendung von Datensätzen vieler verschiedener Benutzer ergibt sich noch ein weiterer Vorteil. Durch die Möglichkeit eine große Anzahl an Benutzern in einem Datensatz zu analysieren, können Verallgemeinerungen gefunden werden, die eine Bedeutung für alle Benutzer haben. Die Eigenschaften der Interaktionen können dadurch, dass sie unabhängig von einzelnen Benutzern sind, als Eigenschaften der jeweiligen Metriken bzw. Kontexte interpretiert werden. Auch um diese Zusatzinformationen durch Verallgemeinerungen nutzen zu können, ist es vorteilhaft eine möglichst große Benutzerzahl in einem Datensatz vertreten zu haben.

6 Adaptive Darstellung von Software-Metriken

Der Fokus dieser Arbeit lag bisher darauf die Ergebnisse von Shi et al. [49] auf den Anwendungsfall mit SonarQube anzupassen und die notwendigen Definitionen klarzustellen. Dieses Kapitel widmet sich nun dem letzten Schritt hin zur fertigen adaptiven Benutzeroberfläche. Es wird zuerst die Verwertung der im letzten Kapitel erlangten Ergebnisse für die Verwendung bei der letztendlichen Anpassung der Weboberfläche besprochen. Im darauffolgenden Kapitel werden Designüberlegungen behandelt, die einen Rahmen für die vorzunehmenden Anpassungen abstecken und Vorgehensweisen für deren Umsetzung vorschlagen. Das letzte Kapitel hat schließlich die Manipulation der Weboberfläche zum Thema und dient auch als Erklärung für manche der verwendeten Methoden.

6.1 Verwertung der Ergebnisse des TFMAP-Algorithmus

Die Ausgabe des TFMAP-Algorithmus sind die optimierten und damit erlernten Modelle für Benutzer, Metriken sowie Kontexte. Sie enthalten alle Informationen, die durch den Lernprozess aus dem impliziten Feedback extrahiert werden konnten. Werden die drei Modelle in einer Tensorfaktorisierung miteinander kombiniert, können damit die durch Formel (5.2) definierten *Predicted Relevance Scores* f berechnet werden. Wie der Name bereits verrät, handelt es sich dabei um Vorhersagen zur Relevanz einer Metrik für einen Benutzer in einem gegebenen Kontext.

Ein reales Einsatzszenario sieht somit nun folgendermaßen aus: Ein Benutzer m ruft die SonarQube Weboberfläche auf. SonarQube bestimmt die gegebene Kontextsituation k über die ihm gegebenen Erfassungsmöglichkeiten. Möglichst schnell, optimalerweise während dem Ladevorgang, sollen die für den aktuellen Benutzer m und Kontext k relevantesten Software-Metriken bestimmt und nachgeladen werden, um diese Informationen so schnell wie möglich präsentieren zu können. Es wird angenommen, dass der Lernprozess zu diesem Zeitpunkt bereits abgeschlossen ist, da der Aufbau der Seite so schnell wie möglich abgeschlossen sein soll, um das Benutzererlebnis zu optimieren.

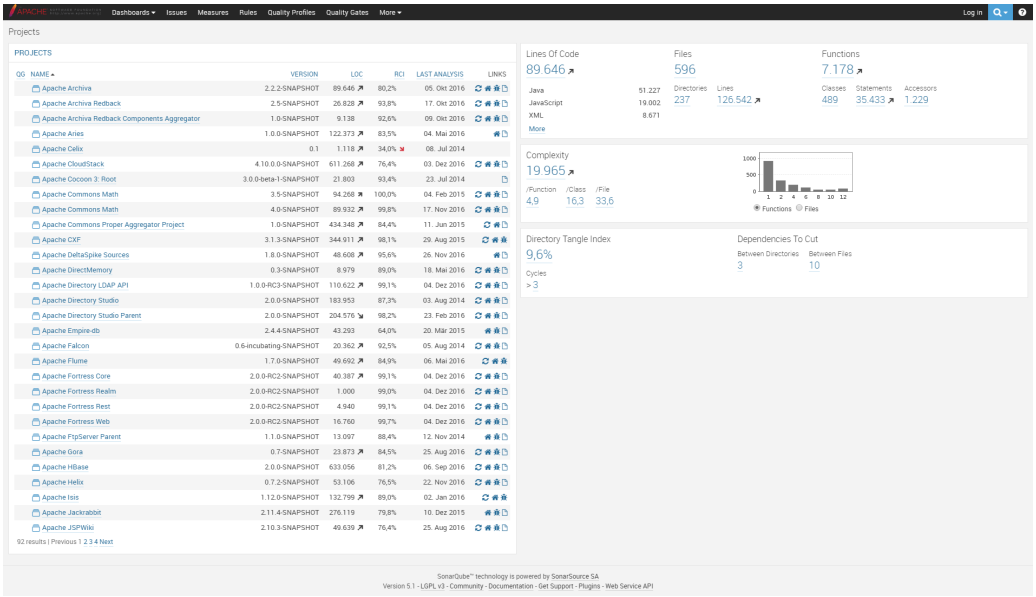


Abbildung 6.1: SonarQube Startseite der Apache Software Foundation mit eingefügten Beispielmetriken aus dem Projekt Apache Archiva.

Damit die relevantesten Software-Metriken bestimmt werden können, werden die *Predicted Relevance Scores* aller Items für den Benutzer m im Kontext k berechnet und die resultierende Liste nach diesen Werten sortiert. Das Resultat enthält die als am relevantesten vermuteten Software-Metriken am oberen Ende der Liste.

6.2 Designüberlegungen

Bei der Manipulation der Weboberfläche stellt der Umgang mit den restlichen Inhalten der Startseite ein Problem dar. Ein rücksichtsloses Entfernen würde für den Benutzer ein ernstzunehmendes Ärgernis darstellen und würde ihn sehr einschränken, wenn z. B. die Projektliste zur Navigation unvorhergesehen entfernt werden würde.

Es ist daher leicht zu sehen, dass es unerwünscht ist Inhalte von der Startseite zugunsten von Widgets für Software-Metriken zu entfernen. Jedoch kann es bereits genauso unerwünscht sein diese Inhalte lediglich zu verdrängen, sodass sie nicht mehr auf den ersten Blick zu sehen sind. Die voreingestellten Oberflächen-Elemente haben immer Vorrang, denn sie wurden vom Benutzer eventuell bewusst so eingestellt und

haben dadurch eine – aus Sicht der adaptiven Benutzeroberfläche – unbestreitbare Relevanz.

Aus dieser Situation heraus ergibt sich die Herausforderung den zur Anzeige notwendigen Platz auf andere Weise zu beanspruchen, um zumindest einen guten Kompromiss zu finden und sowohl voreingestellte Anzeigen, als auch neue Zusatzinformationen in Form von Metriken anzeigen zu können. Das Verschmälern der ursprünglichen Startseite durch das Verwenden mehrerer Spalten erscheint dafür die natürlichste Lösung zu sein, da somit keine Informationen entfernt oder aus dem initial sichtbaren Bereich verdrängt werden müssen.

Der nächste, daraus resultierende Punkt wäre damit die Dimensionierung des Spaltendesigns. Sowohl Spaltenanzahl, als auch Größenverhältnisse zwischen den Spalten spielen dabei eine Rolle. Mögliche Varianten sind das reguläre Design der Startseite mit einer einzigen Spalte, wie es bei vielen SonarQube Instanzen üblich ist, ein Design mit zwei Spalten, wie es in jedem Projekt-Dashboard zu finden ist und ein Design mit drei Spalten. Dies sind jedoch nicht die einzigen Möglichkeiten, da sich mithilfe von Webtechnologien auch noch viele andere Designs implementieren lassen. Diese werden hier jedoch nicht behandelt, da Designoptimierungen nicht der Schwerpunkt dieser Arbeit sind.

Die Verwendung von zwei Spalten zeigt bereits in Abbildung 6.1, dass durch die Komprimierung der ursprünglichen Seite keine Informationen verloren gehen. Der Platzverbrauch durch übermäßig breite Zeilen ist vor der Anpassung unnötig hoch und der dadurch eingesparte, leere Platz kann dadurch besser genutzt werden. In diesem Fall, bei dem lediglich eine Projektliste komprimiert werden muss, ist das noch ein einfaches Unterfangen, doch in anderen Situationen müssen eventuell ungewollte Zeilenumbrüche oder gestauchte Elemente in Kauf genommen werden.

Wie diese Effekte aussehen, kann man gut in Abbildung 6.2 beim Design mit drei Spalten sehen. Die Spalten sind dabei zu klein zur Darstellung der Software-Metriken. Dadurch kommt es zu ungewollten Stauchungen und Verschiebungen, die die Darstellung unästhetisch aussehen lassen. Dies ist bei den Metriken darin begründet, dass ihr Design und ihre Ausmaße auf das zweispaltige Design des Projekt-Dashboards ausgelegt sind und nicht ohne Weiteres in ein dreispaltiges Design übernommen werden können.

Anhand dieser Überlegungen bildet sich ab, dass ein zweispaltiges Design, mit potentiell dynamisch veränderbaren Spaltenbreiten, die vielversprechendste Designalter-

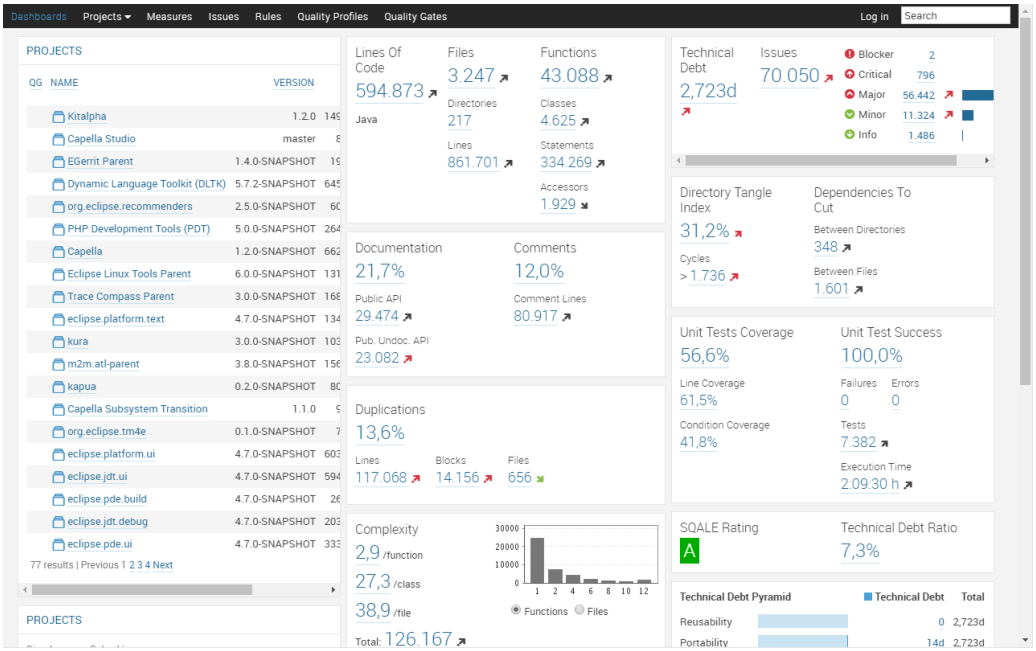


Abbildung 6.2: Seitenlayout mit drei Spalten am Beispiel der öffentlichen SonarQube-Instanz des Eclipse-Projekts. Die Spalten sind zu eng für den darzustellenden Inhalt, weshalb dieser umstrukturiert und teilweise abgeschnitten wird.

native ist, da viele Teile der Benutzeroberfläche ohnehin schon auf ein zweispaltiges Design ausgelegt ist. Dadurch sind, wenn überhaupt, auch nur kleinere, dynamische Anpassungen nötig.

Es ist noch anzumerken, dass in der Realität jede SonarQube Instanz aufgrund der Einstellungsmöglichkeiten unterschiedlich viel Platz auf der Startseite verwendet. Deswegen muss es nicht zwangsläufig so sein, dass lediglich eine Projektliste mit viel Potential zur Platzeinsparung vorhanden ist. Im Gegenteil, der Administrator der Instanz hat zahlreiche Möglichkeiten den Platz der Startseite selber einzuteilen und dessen Nutzung zu bestimmen. In diesen Fällen soll ebenfalls der oben beschriebene Grundsatz gelten, dass Voreinstellungen des Benutzers absoluten Vorrang haben. Ist also beispielsweise bereits sämtlicher Platz auf der Startseite im initial sichtbaren Bereich vergeben und kann nicht ohne Verlust von Information oder der Ästhetik freigeräumt werden, dann soll der nicht-initial sichtbare Bereich der Seite, der nur durch Verschieben des Bildschirminhalts erreichbar ist, verwendet werden. Mit diesem Kompromiss erhält der Benutzer weiterhin alle von ihm explizit eingestellten Informationen, kann aber trotz-

dem schnell auf Zusatzinformationen in Form der eingefügten Metriken zugreifen, ohne einen weiteren Seitenaufruf starten zu müssen.

Eine letzte Designüberlegung ist die Anzahl einzufügender Metriken. Wie bereits in Kapitel 1.2 zur Implementierung des Demonstrators erwähnt wurde, sollen nicht einfach alle verfügbaren Metriken - lediglich nach Relevanz sortiert wie bei einer Suchmaschine - auch tatsächlich auf der Startseite angezeigt werden. Zusätzlich muss eine Auswahl getroffen werden, um die Weboberfläche übersichtlich zu halten. Dafür kann u.a. beachtet werden, wie viel Platz zur Verfügung steht und wie viele Metriken darin untergebracht werden können, um ein andernfalls notwendiges Scrollen zu verhindern. Dies könnte entweder über die direkte Berechnung des noch verfügbaren Platzes geschehen oder durch Ausnutzen einer Faustregel. Eine andere Möglichkeit ist es, nur die Metriken mit den höchsten *Predicted Relevance Scores* anzuzeigen.

6.3 Manipulation der Weboberfläche

Da in den bisherigen Abschnitten dargelegt wurde, wie die zu verwendenden Software-Metriken ausgesucht und dargestellt werden sollen, behandelt dieses Kapitel den noch fehlenden Teil: die Anpassung in der Weboberfläche.

Zur Manipulation der Weboberfläche sind nur einfache Manipulationen des *Document Object Model* (DOM) – der logischen, baumartigen Struktur eines HTML-Dokumentes, auf dem die Webseite basiert – nötig. In einem ersten Schritt wird auf der Startseite eine Fläche reserviert, in der die einzufügenden Widgets platziert werden können. Wie in Kapitel 6.2 erwähnt, wird dazu bei Bedarf entweder eine zweite Spalte erstellt und der restliche Seiteninhalt in die erste Spalte komprimiert oder die Widgets werden in den bestehenden Spalten unten angehängt. Um dieses Ziel zu erreichen, werden die DOM-Elemente, welche die Widgets auf der Projekt-Dashboard Seite darstellen, in diesen reservierten Bereich kopiert. Sobald das DOM des HTML-Dokumentes verändert wurde, werden die eingefügten Elemente automatisch vom Webbrowser gerendert. Dadurch fällt es für den Benutzer kaum auf, dass diese Inhalte keine native Funktionalität von SonarQube sind.

Damit die Software-Metriken beim Aufrufen der Startseite von einem Projekt-Dashboard übernommen werden können, ist es davor notwendig die benötigten Informationen von SonarQube abzufragen. Die Widgets müssen dafür, während die Startseite

aufgebaut wird, parallel mittels des *AJAX*-Konzeptes geladen werden. AJAX steht für *Asynchronous JavaScript And XML* und bezeichnet ein Programmierkonzept zum asynchronen Versenden und Verarbeiten von HTTP-Anfragen, um somit ein Nachladen von Inhalten, ohne ein Neuladen der kompletten Seite, zu ermöglichen [24, 26]. Während die SonarQube-Startseite geladen und aufgebaut wird, wird eine solche AJAX-Anfrage mit dem Ziel eines Projekt-Dashboards abgesendet. Die Antwort des Servers enthält das komplette HTML-Dokument des Dashboards, aus dem die zugehörigen DOM-Elemente der Widgets von Software-Metriken extrahiert und per JavaScript in der Startseite eingefügt werden können.

In den Anforderungen dieser Arbeit aus Kapitel 3.1 wurde mit Anforderung A4 das Erhalten des Seitendesigns verlangt. Dieses Kriterium wird von der hier verwendeten Methode automatisch erfüllt, denn durch das direkte Kopieren vom Projekt-Dashboard werden die Informationen bzgl. des Stylings ebenfalls übernommen. Die verwendeten Stylesheets der Dashboard-Seite gleichen denen auf der Startseite, weshalb die Widgets nach dem Einfügen automatisch richtig dargestellt werden. Die Voraussetzung dafür ist jedoch, dass wie auf dem Projekt-Dashboard ein Layout mit zwei Spalten verwendet wird, denn andernfalls wirken die Widgets stark gestreckt bzw. gestaucht. Außerdem ist es für den Benutzer auf den ersten Blick unklar, aus welchem Projekt die eingefügten Widgets stammen, wodurch es nötig wird auch das zugehörige Projekt anzugeben. Das ist jedoch die einzige Anpassung, die vorgenommen werden muss.

Eine alternative Möglichkeit die entsprechenden Daten von SonarQube zu erlangen wäre die SonarQube-API zu verwenden. Die davon zurückgelieferten Daten sind jedoch nicht im HTML-Format, weshalb zusätzlicher Aufwand nötig wäre, um diese Daten optisch ansprechend darzustellen. Diese Methode ist dann zu bevorzugen, wenn ohnehin ein anderes Styling zu verwenden wäre.

7 Demonstration

Die in den letzten Kapiteln vorgestellten Erkenntnisse und Konzepte müssen nun auch zeigen, dass sie nicht nur theoretisch vielversprechend sind, sondern auch in der Praxis beweisen können, dass sie wie vorgestellt auch realisierbar sind und dabei einen Mehrwert liefern. Dafür widmet sich der erste Teil dieses Kapitels der Umsetzung des Machine-Learning-Konzepts sowie der adaptiven Benutzeroberfläche im Zusammenspiel mit SonarQube und zeigt darüber hinaus am Beispiel verschiedener Benutzungsszenarien wie sich die Benutzeroberfläche an den Benutzer anpasst. Das aus diesen Komponenten bestehende System wird für die Zwecke dieser Arbeit *DeepSonar* genannt. Im zweiten Teil des Kapitels wird darauf eingegangen inwiefern die Anforderungen aus Kapitel 3 von den in dieser Arbeit präsentierten Ergebnissen erfüllt werden.

7.1 Komponenten & Funktionsweise von *DeepSonar*

Um zuerst einen Einblick in die Funktionsweise des entwickelten Systems mit Machine-Learning-Modul und dem Modul zur Anpassung der Oberfläche – im Folgenden „Adapter“ genannt – werden im Folgenden die verwendeten Komponenten beschrieben und ihr Zusammenwirken sowie ihre Arbeitsweise erklärt. Dadurch soll ein Überblick über das *DeepSonar*-System gegeben werden, welches in Abbildung 7.3 auch bildlich dargestellt ist.

7.1.1 Grundlegender Aufbau

Die Basis, auf der der Adapter aufbaut, ist SonarQube, genauer gesagt dessen Weboberfläche. Diese ist für die Zwecke dieser Arbeit zwei Bereiche aufgeteilt: Zum einen das Home-Dashboard und zum andern die verschiedenen Projekt-Dashboards.

Auf dem Home-Dashboard (s. Abbildung 7.1), welches die Startseite von SonarQube darstellt, wird meistens eine Liste der zur Verfügung stehenden Projekte und je nach Konfiguration noch weitere Widgets angezeigt, welche beispielsweise einen kurzen

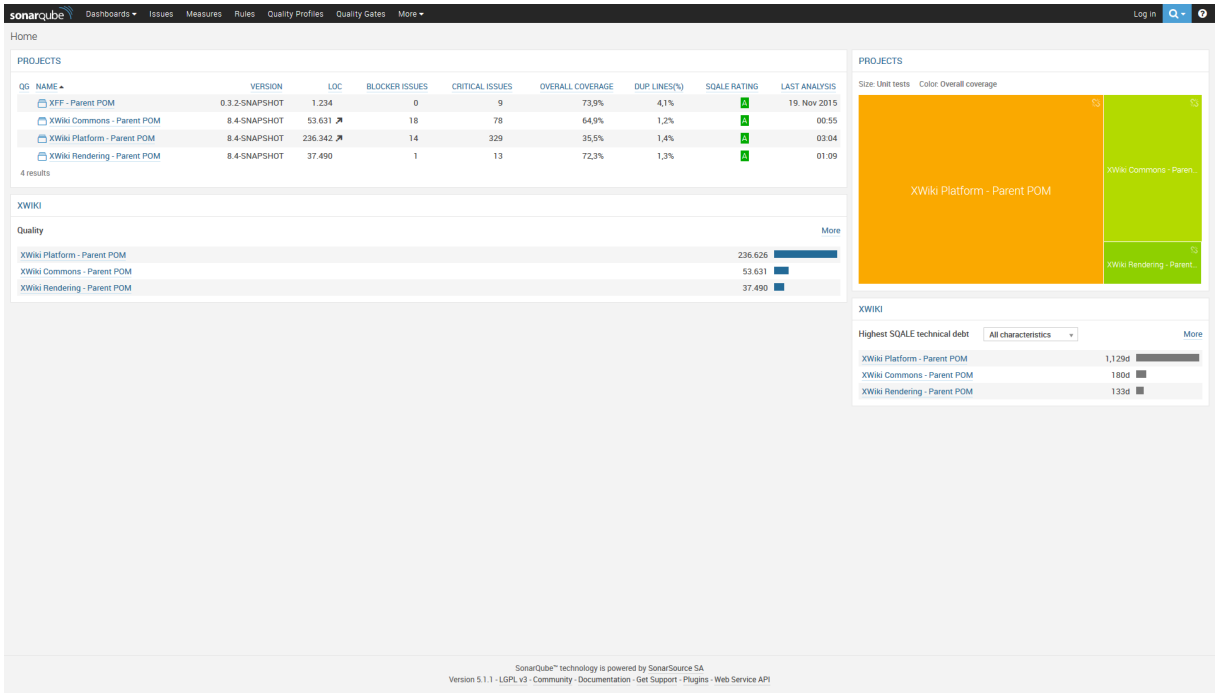


Abbildung 7.1: SonarQube Startseite des XWiki-Projekts

Überblick über die technische Qualität der Projekte bietet. Da diese Seite, wie in Kapitel 1 beschrieben, in vielen Fällen den zur Verfügung stehenden Platz nicht effizient nutzt, soll diese Seite für die Demonstration dieser Arbeit auf die Bedürfnisse des Benutzers angepasst werden, indem Widgets mit Software-Metriken darauf angezeigt werden.

Die Widgets mit Metriken dafür sollen aus dem Projekt-Dashboard (s. Abbildung 7.2) stammen, welches für den Nutzer am relevantesten ist. Diese Projekt-Dashboards enthalten typischerweise viele verschiedene Widgets mit unterschiedlichen Metriken, um einen umfassenderen Überblick über ein einzelnes Projekt und die Entwicklung dessen technischer Qualität zu bieten. Auf dieser Seite werden daher die Informationen dargestellt, weswegen der Benutzer SonarQube aufgerufen hat.

Der Adapter ist als Erweiterung der regulären Weboberfläche von SonarQube zu sehen, da er sich als Browserskript nahtlos darin integriert und diese entsprechend des Nutzerverhaltens verändert, welches ebenfalls von ihm erfasst wird. Das bedeutet, dass der Adapter als Mittelsmann zwischen SonarQube, Benutzer und Machine-Learning-Modul fungiert, was auch in Abbildung 7.3 dargestellt ist.

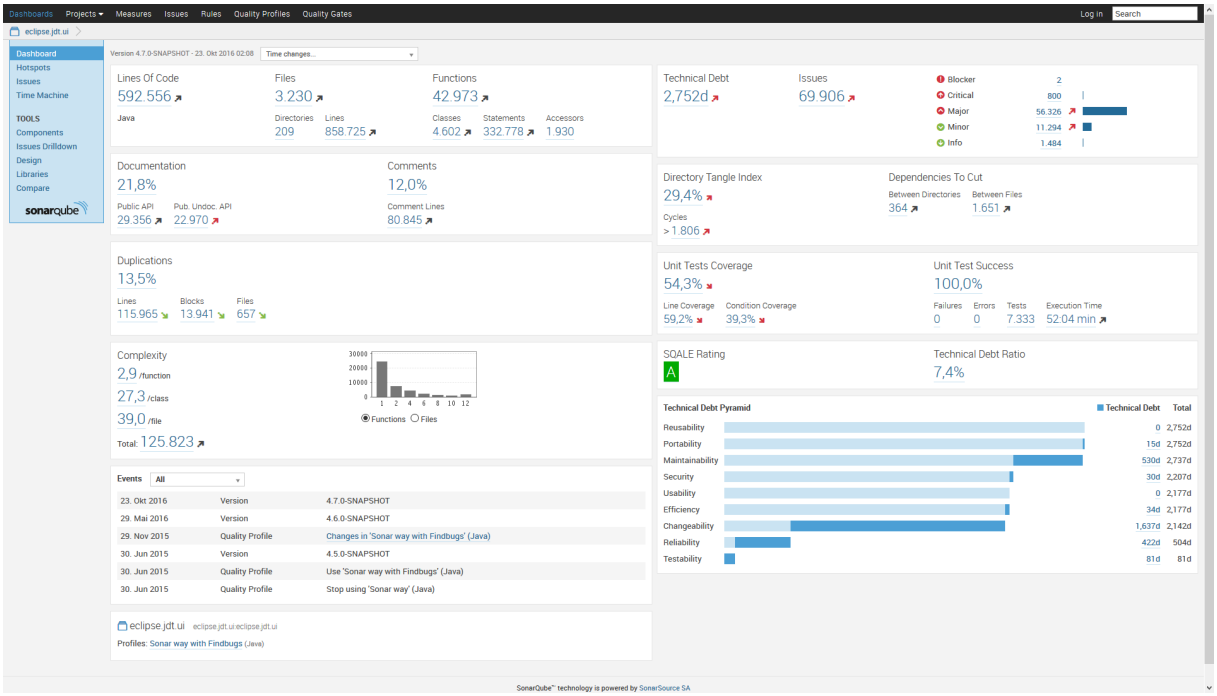


Abbildung 7.2: Übersicht von Widgets mit Software-Metriken eines Programmoduls in SonarQube. Die hier gezeigten Widgets stammen aus dem Modul „eclipse.jdt.ui“ des Eclipse Projekts.

Das Skript des Adapters wird beim Aufruf der SonarQube Webseite mitgeladen und erkennt selbstständig, ob es sich dabei um das Home- oder ein Projekt-Dashboard handelt. Daraufhin führt es die notwendigen Aktionen sofort aus. Im Fall eines Home-Dashboards bedeutet das, dass die HTML-Struktur der Seite geändert wird, um freien Platz zu schaffen, in den Widgets geladen werden. Die Information welche Widgets geladen werden sollen erhält der Adapter dafür vom Machine-Learning-Modul. Auf einem Projekt-Dashboard wird hingegen für jedes Widget erfasst, ob der Benutzer darauf klickt und im Falle einer solchen Interaktion wird die gegenwärtige Kontextsituation, wie in Kapitel 4 definiert, erfasst und in Verbindung mit dem Widget und dem aktuellen Benutzer abgespeichert. Diese Daten stellen das implizite Feedback des Benutzers dar, so wie es von dem TFMAP-Algorithmus von Shi et al.[49] erwartet wird.

Das Machine-Learning-Modul verwendet das gespeicherte Feedback, um wie in Kapitel 5 beschrieben die latenten Modelle der Benutzer, Metriken und Kontexte zu erlernen. Diese Daten macht der Adapter für das Machine-Learning-Modul verfügbar. Da-

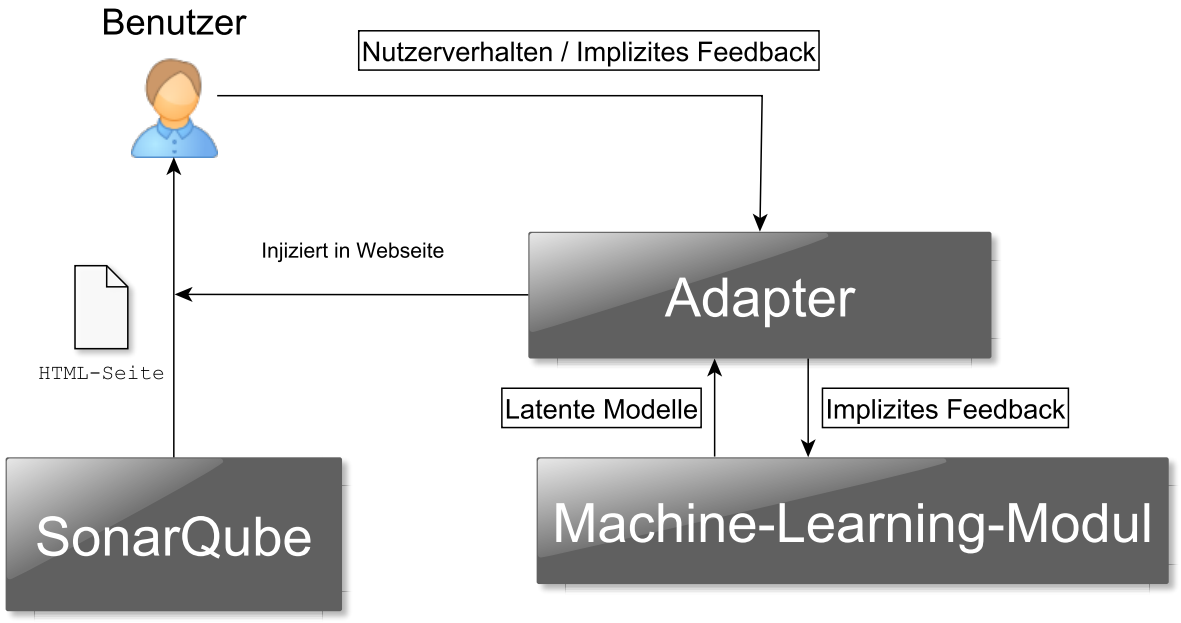


Abbildung 7.3: Übersicht über das beschriebene System

durch kann der Lernprozess, abgesehen vom Verfügbarmachen der Daten, unabhängig und entkoppelt vom Adapter die latenten Modelle erlernen. Dies ist notwendig, da das Lernen ein lange andauernder Prozess ist und andernfalls vom Benutzer unterbrochen werden könnte, wenn er die Webseite von SonarQube verlässt.

Wenn der Adapter nun zur Anpassung des Home-Dashboards wissen muss, welche Metriken für den aktuellen Benutzer im derzeitigen Kontext relevant ist, fordert er die erlernten latenten Modelle beim Machine-Learning-Modul an. Mithilfe der Formel des *Predicted Relevance Score* (5.2) können die verfügbaren Metriken bezüglich ihrer Relevanz zur Laufzeit der SonarQube-Webandwendung bewertet werden.

7.2 Veranschaulichung der Funktion

Um ersichtlich zu machen, wie sich *DeepSonar* an verschiedene Nutzer anpasst, beschreiben die nächsten Abschnitte verschiedene Szenarien wie sich Beispielbenutzer verhalten könnten und welche Anpassungen von *DeepSonar* die unterschiedlichen Interaktionen mit der Weboberfläche bewirken.

7.2.1 Nutzerszenarien

Die folgenden Szenarien sollen beschreiben, wie Verhaltensmuster von Benutzern aussehen können. Das Augenmerk liegt dabei auf der Erläuterung welche Merkmale beim Benutzerverhalten eine Rolle spielen. Die hier beschriebenen Szenarien müssen dabei nicht zwangsweise repräsentativ für die Realität sein, sondern sollen die Eigenschaften der Anpassungen hervorheben.

Situation A Das simpelste Verhaltensmuster ergibt sich dadurch, dass ein einziger Benutzer A existiert, welcher nur eine Metrik α und das auch ausschließlich in einem Kontext κ aufruft. Der Tensor aus implizitem Feedback hat demnach nur einen einzelnen Eintrag ungleich null, und zwar den für Benutzer A und Metrik α im Kontext κ.

In dieser Konstellation wird erwartet, dass der Benutzer A, wenn er die Weboberfläche von SonarQube aufruft und sich dabei im Kontext κ befindet, die Metrik α auf der Startseite angezeigt bekommt. Ist das der Fall, dann werden tatsächlich auch vor allem *relevante* Metriken angezeigt.

Situation B Für den nächsten Fall soll Situation A erweitert werden. Benutzer A verhält sich wie in Situation A, ruft wenn er sich in einem anderen Kontext λ befindet, jedoch eine Metrik β auf. Der Tensor aus implizitem Feedback hat nun zwei Einträge ungleich null für zwei verschiedene Kontextsituationen.

Wenn die von TFMAP empfohlenen Metriken, die auf der SonarQube-Startseite angezeigt werden, von der aktuellen Kontextsituation abhängen, dann muss im Kontext κ die Metrik α und im Kontext λ die Metrik β angezeigt werden.

Situation C Auch für Situation C sei die gleiche Ausgangssituation A angenommen. Der Benutzer A ruft nun zusätzlich im Kontext κ auch die Metrik γ auf. Dadurch erhält der Tensor aus implizitem Feedback einen zweiten Eintrag mit dem Wert eins für den Benutzer A und den Kontext κ

Das erwartete Verhalten für diese Situation ist, dass der Benutzer A im Kontext κ nun beide Metriken – α und γ – angezeigt bekommt. Das würde bedeuten, dass *DeepSonar* auch fortlaufend neue Informationen über den Benutzer miteinbezieht und seine Empfehlungen darauf anpasst.

Situation D Da es sich bei TFMAP um einen Algorithmus aus dem Bereich des kollaborativen Filterns handelt, behandelt Situation D Mehrbenutzersysteme. Es existieren nun Benutzer A, B und C. Benutzer A ruft wie bisher im Kontext κ die Metrik α auf. Benutzer B verhält sich ähnlich und ruft die gleiche Metrik α, jedoch im Kontext λ auf. Benutzer C ist ein neuer Benutzer, zu dem noch kein Benutzerverhalten erfasst werden konnte.

Es wird hierfür erwartet, dass das Verhalten der Benutzer auch Auswirkungen für andere oder neue Benutzer hat, wenn der Lernprozess Daten aller Benutzer verarbeitet. Da Metrik α sowohl für Benutzer A, als auch Benutzer B von Interesse zu sein scheinen, kann die Annahme getroffen werden, dass die Metrik α auch für den neuen Benutzer C möglicherweise von Interesse ist [41]. Daher sollte diese Metrik auch bei Benutzer C auf der Startseite angezeigt werden.

7.2.2 Adaption des Systems

Anhand der soeben beschriebenen Nutzerszenarien, soll nun in diesem Abschnitt illustriert werden, welche Auswirkungen unterschiedliche Verhaltensmuster in der Anpassung der Weboberfläche von SonarQube durch *DeepSonar* nach sich ziehen.

Die Tests wurden dafür auf einer lokalen, nicht-öffentlichen SonarQube-Instanz in Version 5.1.2 durchgeführt. In Abbildung 7.4 ist die unveränderte Startseite zu sehen, welche im Folgenden als Referenz dient. Zu sehen ist lediglich eine Projektliste, in der ein großer Teil der Fläche nicht genutzt wird.

Für jede der in Kapitel 7.2.1 beschriebenen Situationen wird jeweils die durch Anpassung resultierende Seite dargestellt. Der Lernprozess wurde für die Situationen A bis D mit den in [49] gewählten Parametern $D = 10$, $\gamma = 0.001$ und $\lambda = 0.001$ durchgeführt, wobei D der Anzahl der Eigenschaften, γ dem Lernfaktor und λ dem Regularisierungsfaktor entspricht.

Situation A Im einfachsten Beispiel wurde davon ausgegangen, dass der aktuelle Benutzer in der Vergangenheit ausschließlich das Widget der Metrik „Lines of Code“ im Projekt Apache Tomcat angesehen hat, während es Morgen und der Wert der Metrik „steigend“ war. Daraus ergibt sich die Kontextsituation κ = („Apache Tomcat“, „steigend“, „Morgen“).

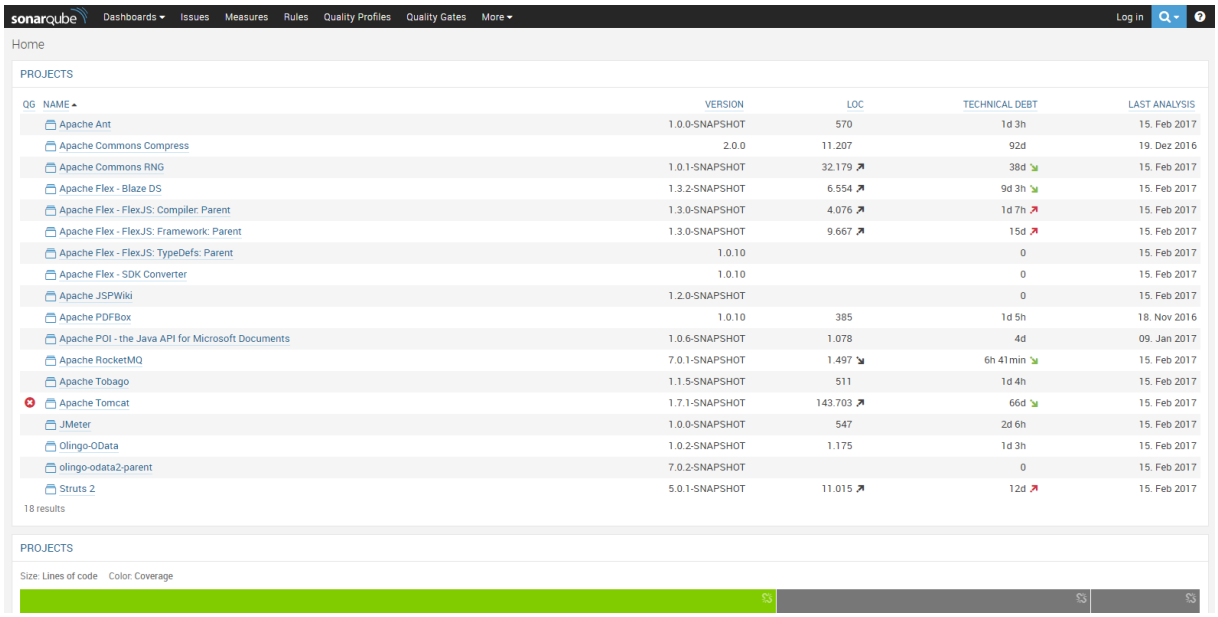


Abbildung 7.4: Unveränderte Startseite der lokalen Testinstanz von SonarQube.

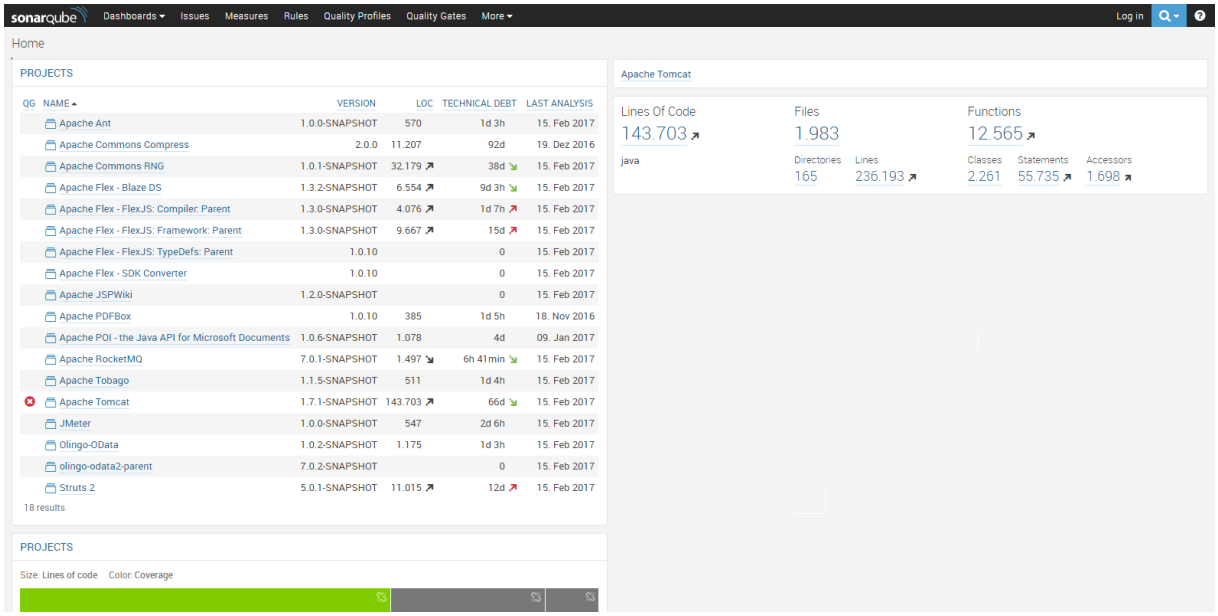


Abbildung 7.5: Für Situation A angepasste Startseite der lokalen SonarQube-Instanz

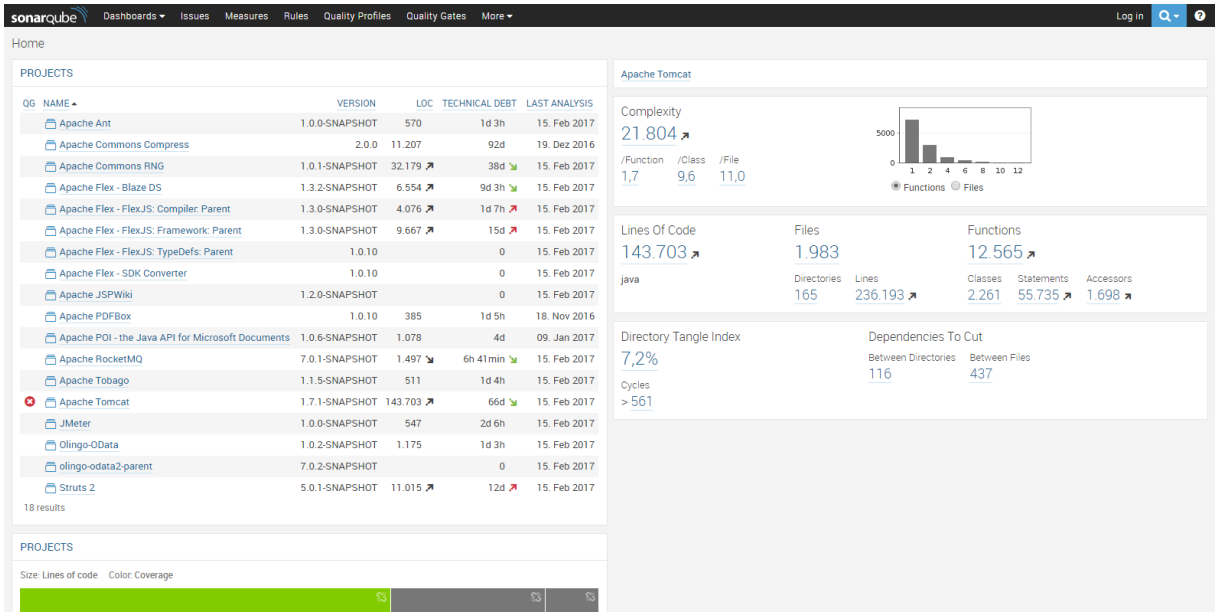


Abbildung 7.6: Für Situation B angepasste Startseite der lokalen SonarQube-Instanz

Ruft der Benutzer, während er sich im gleichen Kontext befindet, die Startseite von SonarQube nach Abschluss des Lernprozesses auf, bekommt er die Webseite, wie sie in Abbildung 7.5 dargestellt ist, zu sehen. Dafür muss er als letztes das Projekt Apache Tomcat betrachtet haben, der Wert der Metrik „Lines of Code“ muss immer noch „steigend“ sein und die Tageszeit muss „Morgen“ sein.

Zu sehen ist, dass der bisherige Inhalt der Seite verkleinert wurde und sich in der linken Spalte befindet. Die rechte Spalte ist neu hinzugekommen und enthält als erstes Element den Namen des Projekts, aus dem das angezeigte Widget stammt. Als nächstes werden die vom TFMAP-Algorithmus als wichtig erachtenden Widgets angezeigt. In diesem Fall ist das nur das Widget der Metrik „Lines of Code“, da das das einzige Widget ist, das der Benutzer je angeklickt hat und damit als relevant erachtet werden kann.

Situation B Das Beispiel von Situation B soll zeigen, dass die vom TFMAP-Algorithmus vorgeschlagenen Widgets auch aufgrund der jeweils vorliegenden Kontextsituation ausgesucht werden. Dafür hat der Benutzer in dieser Situation zusätzlich in einem Kontext $\lambda =$ („Apache Tomcat“, „steigend“, „Abend“) die Metrik „Complexity“ angeklickt.

Befindet sich der Benutzer wie in Situation A im Kontext κ, wird auch die SonarQube-Startseite wie in Abbildung 7.5 angezeigt. Befindet sich der Benutzer jedoch im Kontext

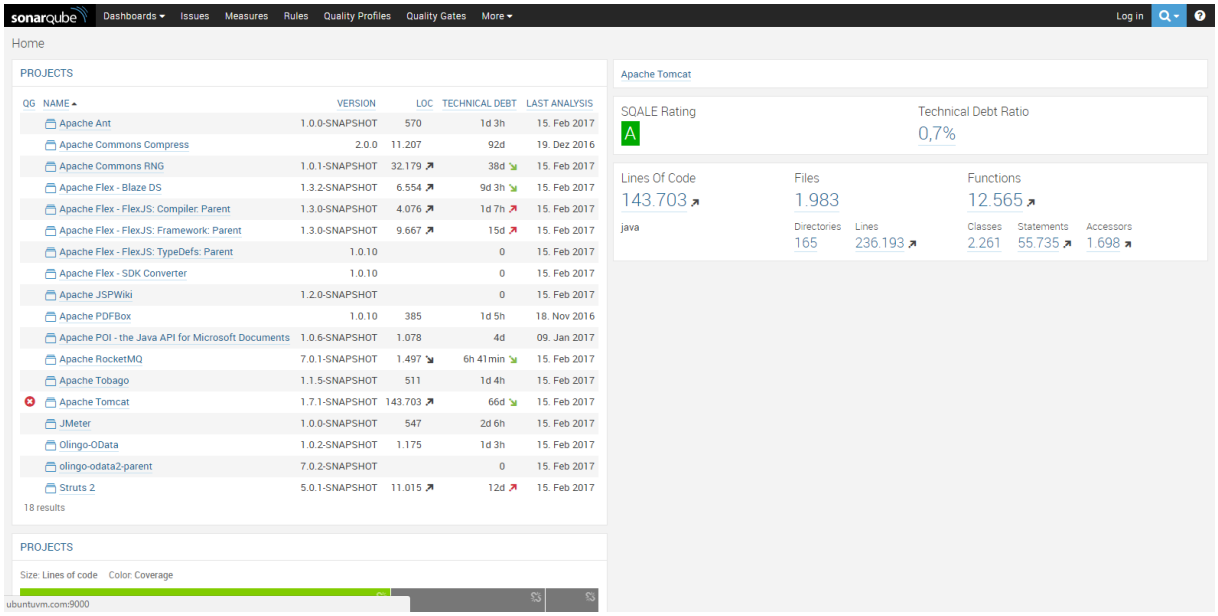


Abbildung 7.7: Für Situation C angepasste Startseite der lokalen SonarQube-Instanz

λ, wird die Startseite ähnlich wie in Situation A verändert, mit dem Unterschied, dass zusätzlich zu dem Widget für die Metrik „Lines of Code“, das Widget für die Metrik „Complexity“ angezeigt wird. Dadurch wird ersichtlich, dass die nach dem Lernprozess angezeigten Metriken auch vom Kontext des Benutzers abhängen.

Tatsächlich wird in Abbildung 7.6 das entsprechende Widget der Metrik „Complexity“ neben zwei weiteren Widgets angezeigt, welche jedoch niedrigere *Predicted Relevance Scores* haben.

Situation C Wenn mehrere Metriken für einen Benutzer relevant sind sollen diese auch alle angezeigt werden. In dieser Situation hat der Benutzer im Kontext κ zusätzlich zum Widget der Metrik „Lines of Code“ auch das Widget „Debt Overview“ angeklickt, welches das sog. SQALE-Rating und den Prozentsatz der technischen Schuld des Projektes anzeigt.

Ruft der Benutzer wie in Situation A die SonarQube-Startseite nach Abschluss des Lernprozesses auf, wird ihm die Startseite wie in Abbildung 7.7 angezeigt. Diese sieht ähnlich aus, wie die Version aus Situation A, nur dass nun zusätzlich das Widget „Debt Overview“ angezeigt wird.

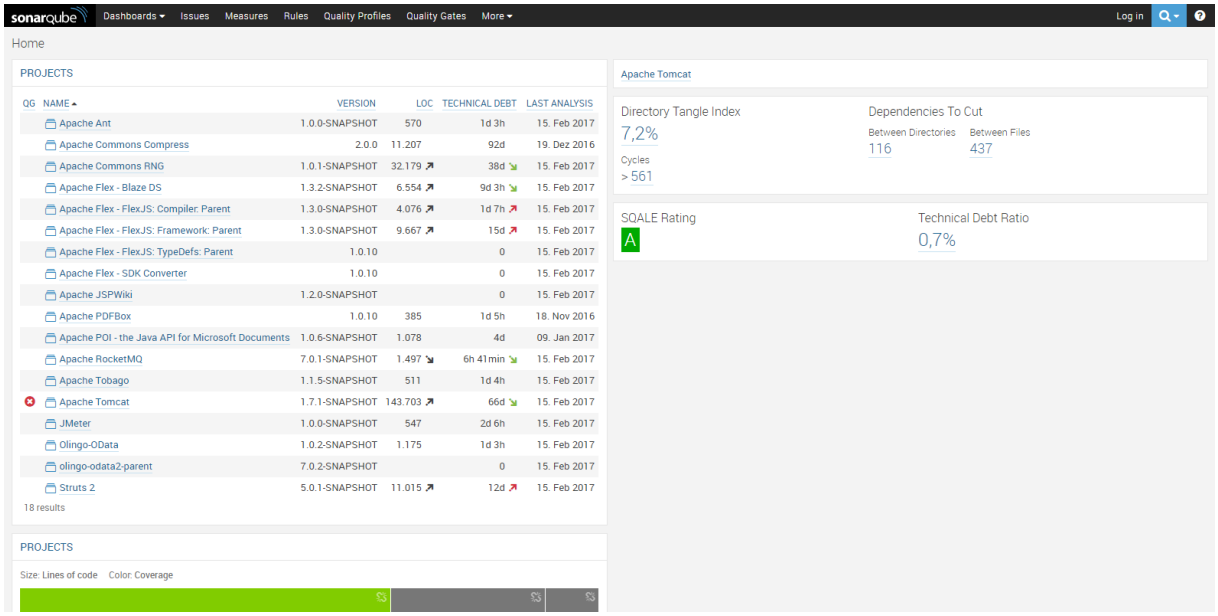


Abbildung 7.8: Für Situation D angepasste Startseite der lokalen SonarQube-Instanz

Wie viele Widgets angezeigt werden, hängt vom jeweiligen *Predicted Relevance Score* der Kombination aus Benutzer, Metrik und Kontext ab. Die Metriken, welche als relevant angenommen werden können, werden jedoch alle angezeigt.

Situation D Diese Situation soll die externe Validität der gefundenen Modelle nachweisen. Für diese Situation haben zwei Benutzer A und B in zwei unterschiedlichen Kontextfaktoren das gleiche Widget „Lines of Code“ angeklickt. Der Tensor mit implizitem Feedback enthält daher zwei Einträge ungleich null. Beide Einträge beziehen sich auf dasselbe Widget, aber sowohl auf unterschiedliche Benutzer, als auch auf verschiedene Kontextsituationen.

Wenn nun ein neuer Benutzer C die SonarQube-Startseite das erste Mal, also bevor *DeepSonar* die Möglichkeit hat implizites Feedback vom Benutzer zu erfassen, in einer beliebigen Kontextsituation aufruft, dann wird erwartet, dass die Machine-Learning-Komponente erlernt hat das Widget „Lines of Code“ als relevant einzustufen, da es von den anderen Nutzern des Systems unabhängig vom Kontext angeklickt wurde. Wie in Abbildung 7.8 zu sehen ist, ist das jedoch leider nicht der Fall. Eine mögliche Ursache dafür kann eine zu kleine Menge an implizitem Feedback sein, sodass die Benutzer-

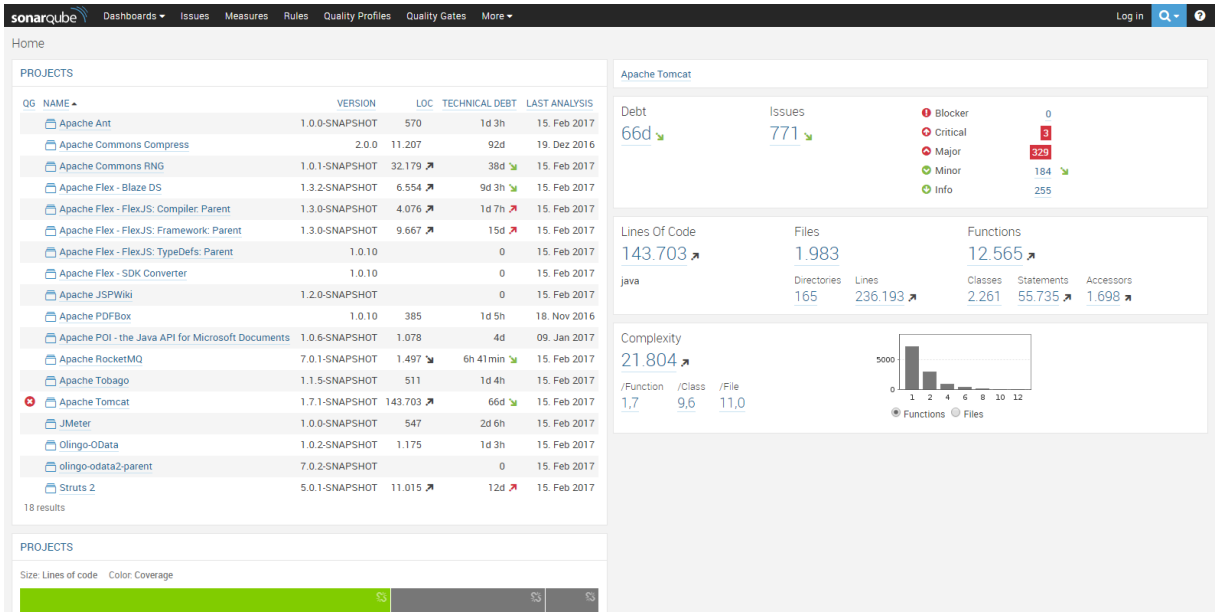


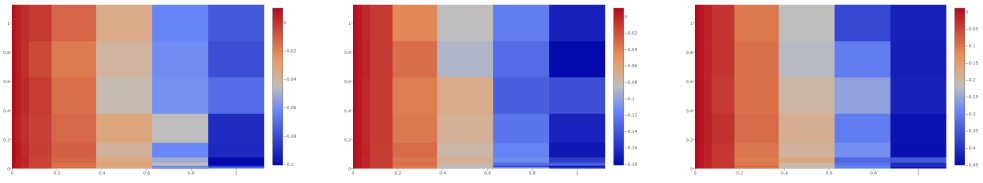
Abbildung 7.9: Angepasste Startseite der lokalen SonarQube-Instanz in einem weiteren Test ähnlich zu Situation D. Das dafür verwendete implizite Feedback enthält sieben statt zwei Benutzer, welche in verschiedenen Kontexten die Metrik „Lines of Code“ angeklickt haben.

präferenzen vom Rauschen bei der pseudozufälligen Initialisierung der latenten Modelle überdeckt werden und der Algorithmus in einem lokalen Optimum terminiert.

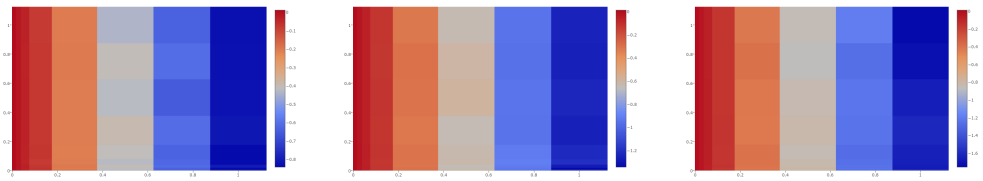
Ein weiterer Test, in dem sieben Benutzer A bis G die Metrik „Lines of Code“ in verschiedenen Kontexten ausgewählt haben, zeigt in Abbildung 7.9, dass für einen neuen Benutzer H unter anderem auch die Metrik „Lines of Code“ angezeigt wird. Daher ist davon auszugehen, dass das Machine-Learning-Modul tatsächlich durch das implizite Feedback der zusätzlichen Benutzer die Relevanz der Metrik „Lines of Code“ besser erlernen konnte.

7.2.3 Parameter der *Mean Average Precision*

Abbildung 7.10 zeigt am Beispiel von Situation D mit sieben Benutzern wie die MAP-Werte bei der Wahl verschiedener Parameter ausfallen. Jedes Diagramm steht dabei für eine unterschiedliche Anzahl an vermuteten Features für die latenten Modelle. Auf der x-Achse ist jeweils der verwendete Regularisierungsfaktor und auf der y-Achse der Lernfaktor abgebildet. Die Farbe gibt den zugehörigen MAP-Wert für die Kombination der



(a) 1 Feature (b) 2 Features (c) 5 Features



(d) 10 Features (e) 15 Features (f) 20 Features

Abbildung 7.10: *Mean Average Precision*-Werte für das implizite Feedback von sieben Benutzern gemäß Situation D für unterschiedliche Zahlen an Features D, Lernfaktoren $\gamma \in (0; 1]$ und Regularisierungsfaktoren $\lambda \in (0; 1]$. Die Regularisierungsfaktoren sind auf der x-Achse und die Lernfaktoren auf der y-Achse geplottet.

Parameter an, wobei rot einen hohen und blau einen niedrigen, meist sogar negativen Wert, Wert bedeutet.

Die größte Gemeinsamkeit zwischen allen Fällen ist, dass ein hoher Wert des Regularisierungsfaktors λ ($\lambda > 0.1$) zu einem raschen und starken Abfall des MAP-Wertes führt, während eine Variation des Lernfaktors zu kaum einem Unterschied führt. Eine weitere Gemeinsamkeit ist der Werteverlauf der Diagramme, welcher für verschiedene Anzahlen an verwendeten Features relativ ähnlich ist. Die Diagramme 7.10d, 7.10e und 7.10f enthalten zwar weniger sichtbares rauschen, das liegt jedoch daran, dass die Bereiche rechts, mit hohen Werten für λ, viel stärker in den negativen Bereich hineinreichen und die Fluktuationen in Relation zu den absoluten MAP-Werten viel weniger ins Gewicht fallen.

Da der Lernfaktor γ die Qualität der resultierenden latenten Modelle – gemessen an den berechneten MAP-Werten – kaum beeinflusst, bietet es sich an, diesen aus Effizienzgründen möglichst groß zu wählen. Grund dafür ist, dass der Lernfaktor γ gewichtet, wie stark sich die latenten Modelle pro Iteration des TFMAP-Algorithmus verändern können. Bei kleinen Werten für γ sind daher mehr Iterationen nötig, um das gleiche

Optimum zu finden. Durch Variation des Lernfaktors γ kann somit die Laufzeit, um ein Vielfaches reduziert werden, ohne dass die Ergebnisse des Algorithmus darunter leiden.

7.3 Erfüllung der Anforderungen

Nachdem die Funktionalität der adaptiven Benutzeroberfläche *DeepSonar* erklärt wurde, besteht das Ziel der nächsten Abschnitte darin niederzulegen, wie diese die in Kapitel 3 bestimmten Anforderungen erfüllen.

Zusammengefasst, lauten die Anforderungen, dass die adaptive Benutzeroberfläche die für den Lernprozess benötigten Daten vom Benutzer unbemerkt erfasst, um ihn nicht zu stören, und ausschließlich anhand dieser Daten lernt. Des Weiteren soll die Benutzeroberfläche von SonarQube anhand der Ergebnisse des Lernprozesses angepasst werden, wobei das Design der Oberfläche erhalten bleiben soll. Darüber hinaus muss während des Lernprozesses die Weboberfläche von SonarQube auch weiterhin verfügbar sein.

7.3.1 Unbemerktes Erfassen des Benutzerverhaltens und Nutzungskontexts

Durch Verwendung des *Observer-Patterns* [42] und sog. *Event-Listener*, welche eine vorher festgelegte Aktion durchführen, sobald ein bestimmtes Ereignis stattfindet, wird bei der Erfassung des Benutzerverhaltens für jedes Widget festgestellt, wenn darauf geklickt wurde. Daraufhin wird die aktuelle Kontextsituation, in der die Interaktion stattgefunden hat, erfasst, wobei keine zusätzlichen Eingaben des Benutzers, die über die standardmäßige Verwendung der Weboberfläche hinausgehen, benötigt werden. Daher geschieht die Erfassung des Benutzerverhaltens und des aktuellen Benutzerkontexts unbemerkt, sodass der Benutzer davon nichts mitbekommt.

7.3.2 Lernen anhand der Daten zu Benutzerverhalten und Nutzungskontexts

Als Eingabe erhält der zum Machine-Learning verwendete TFMAP-Algorithmus, wie in Abbildung 7.3 dargestellt, lediglich das vom Benutzer gesammelte implizite Feedback, welches das Stattfinden von Interaktionen zwischen Benutzern und Software-Metriken in bestimmten Kontexten beschreibt. Dieses implizite Feedback enthält nur die Informationen welche Benutzer welche Software-Metriken aufgerufen haben und in welcher Kontextsituation sie sich dabei befanden. Aus diesem Grund wird die Relevanz der Software-Metriken ausschließlich anhand des Benutzerverhaltens und -kontexts erlernt.

7.3.3 Anpassung der Benutzeroberfläche an den Benutzer

Das Kapitel 7.2.2 zeigt, dass sich das *DeepSonar*-System dynamisch an den Benutzer anpasst, abhängig davon in welchen Kontextsituationen er welche Software-Metriken aufgerufen hat. Die dafür benötigten, erlernten latenten Modelle werden vom Machine-Learning-Modul auf Basis des vom Adapter gesammelten impliziten Feedback erlernt. Durch den Lernprozess wird sichergestellt, dass die angezeigten Metriken auch relevant für den Benutzer sind und die Benutzeroberfläche somit auch wirklich an den *Benutzer* angepasst wird.

7.3.4 Erhalten des Seitendesigns

Bei Veränderungen der Weboberfläche verwendet der Adapter ausschließlich Gestaltungsmaßnahmen, die auf der ursprünglichen Webseite bereits verwendet werden, um sicherzustellen, dass der Gesamteindruck stimmig ist. Es wird auch darauf geachtet, dass selbst kleine Details und Feinheiten übernommen werden. Ein Beispiel für ein solches Detail ist die Verwendung eines Randes von -1 Pixeln für Spalten in einem mehrspaltigen Layout [1]. Durch Einbezug auch solcher Details soll es so schwer wie möglich sein Unstimmigkeiten im Design der Webseite zu finden.

Für die Anpassung der Platzverhältnisse auf der Startseite wird darauf geachtet, dass das Hinzufügen einer neuen Spalte genauso aussieht, als wenn der Benutzer die Startseite selber mit den Bordmitteln von SonarQube angepasst hätte. Das bedeutet vor

allem, dass kein Widget zu stark verkleinert wird und die darin enthaltenen Informationen immer lesbar bleiben.

Der vermutlich wichtigste Grund, warum das Design durch die vom Adapter vorgenommenen Anpassungen nicht zerstört wird, ist jedoch die Verwendung der originalen Styling-Attribute. Das wird erreicht, indem die angezeigten Metriken als HTML-Elemente inkl. Styling-Attribute von dem jeweiligen Projekt-Dashboard geladen und ohne Veränderung auf der Startseite in die HTML-Struktur der Webseite eingebaut werden. Dadurch werden dieselben Styling-Informationen verwendet, da sowohl die Startseite, als auch die Projekt-Dashboards die gleichen Stylesheets verwenden.

Das Ergebnis sieht deswegen wie eine native Funktion von SonarQube aus und es ist nicht direkt ersichtlich, dass sie erst nachträglich eingebaut wurde.

7.3.5 Lernen & Adaption während des laufenden Betriebs

Das Machine-Learning-Modul kann unabhängig vom Adapter, der die Anpassungen an der SonarQube-Weboberfläche vornimmt und das Benutzerverhalten erfasst, arbeiten, da es funktional nur schwach an den Adapter gekoppelt ist. Die einzige Kommunikation zwischen den beiden Modulen besteht in dem Austausch von implizitem Feedback und erlernten latenten Modellen. Somit kann das Machine-Learning-Modul unabhängig vom Adapter arbeiten, sobald es das implizite Feedback erhalten hat und blockiert nicht den Adapter während des Lernprozesses.

Der asynchrone Ablauf des Lernprozesses ist sehr wichtig, da dieser abhängig davon wie nah das tatsächliche Optimum der *Mean Average Precision*-Funktion erreicht werden soll und wie viele Benutzer, Metriken oder Kontextsituationen betrachtet werden sollen auf einem einzelnen, handelsüblichen Computer durchaus mehrere Stunden benötigen kann.

Durch diese schwache Kopplung ist garantiert, dass die SonarQube-Webanwendung ungehindert funktioniert, während im Hintergrund die latenten Modelle vom Machine-Learning-Modul gelernt werden, sodass dieser Lernprozess den laufenden Betrieb und den Benutzer bei der Verwendung von SonarQube nicht stört.

8 Zusammenfassung & Ausblick

Ziel dieser Arbeit war es eine adaptive Benutzeroberfläche für die Softwareanalyse-Plattform SonarQube unter Zuhilfenahme von Machine-Learning-Methoden zu entwickeln. Adaptiv bedeutet dabei, dass die Benutzeroberfläche sich in Abhängigkeit vom Benutzerverhalten an einen Benutzer persönlich anpasst, indem sie die Widgets von Software-Metriken, die für den aktuellen Benutzer im gegenwärtigen Nutzungskontext am relevantesten sind, bereits auf der Startseite der Anwendung anzeigt. Zusätzlich sollten die Anforderungen aus Kapitel 3.1 erfüllt werden.

Das Ergebnis dieser Arbeit ist *DeepSonar*, eine selbstlernende, adaptive Benutzeroberfläche, welche aus zwei Komponenten besteht: dem sog. Adapter und dem Machine-Learning-Modul. Der Adapter wird in die Weboberfläche von SonarQube intergriert und nimmt an ihr Veränderungen vor, um das Erfassen des Benutzerverhaltens und des Nutzungskontextes sowie das Anzeigen relevanter Widgets von Software-Metriken zu ermöglichen. Dabei wird auch nur erfasst, ob ein Benutzer das Widget einer bestimmten Metrik in einer Kontextsituation angeklickt hat oder nicht. Das Machine-Learning-Modul hingegen erlernt auf Basis des Benutzerverhaltens mittels des TFMAP-Algorithmus von Shi et al. [49] welche Software-Metriken für den Benutzer in einem bestimmten Kontext relevant sind.

DeepSonar erfüllt alle der in Kapitel 3.1 erwähnten Anforderungen, da es zum einen die für den Lernprozess benötigten Daten zum Benutzerverhalten und Nutzungskontext vom Benutzer unbemerkt erfasst und ihn somit nicht in seiner Arbeit stört und zum anderen auch ausschließlich anhand dieser Daten die Relevanz der verschiedenen Software-Metriken für verschiedene Benutzer und Kontexte erlernt. Darauf aufbauend passt *DeepSonar* die Weboberfläche von SonarQube an den jeweiligen Benutzer an, indem die für den Nutzer im gegenwärtigen Kontext relevanten Widgets von Metriken auf der Startseite angezeigt werden. Bei dieser Anpassung wird außerdem das Design der Seite übernommen, sodass diese wie eine native Funktionalität von SonarQube erscheint. Auch die letzte Anforderung wird erfüllt, da der Lernprozess nahezu vollständig vom SonarQube-Prozess entkoppelt ist und SonarQube somit auch

während der Ausführung des TFMAP-Algorithmus ohne Einschränkungen verwendet werden kann.

Um diese Ziele zu erreichen, wurde in dieser Arbeit zuerst eine Übersicht über verwandte Arbeiten gegeben. Diese behandelten bestehende Lösungsansätze für den Entwurf sowie die Umsetzung einer adaptiven Benutzeroberfläche, aber auch Machine-Learning-Methoden, welche Kontextinformationen ausnutzen, um bessere Ergebnisse zu erzielen. Hierbei eignete sich vor allem die Arbeit von Shi et al. [49] als Grundlage für die Machine-Learning-Komponente.

Im nächsten Schritt wurden Abwägungen getroffen, wie der Kontext einer Interaktion mit der Weboberfläche von SonarQube definiert werden kann, welche Kontextfaktoren existieren und wie diese im normalen Gebrauch der Anwendung ohne großen Ressourceneinsatz erfasst werden können.

Darauf aufbauend wurde ein Konzept zur adaptiven Darstellung der Software-Metriken erarbeitet, welches die Ergebnisse des TFMAP-Algorithmus verwertet. Dabei flossen auch Designüberlegungen mit ein, um die Anpassungen an der Weboberfläche in einer nicht negativ auffallenden Weise zu gestalten.

In den vorhergehenden Kapiteln wurde schließlich beschrieben, wie das *DeepSonar*-System aufgebaut ist und anhand konkreter Nutzerszenarien demonstriert, dass die Konzepte dieser Arbeit realisierbar sind.

Obwohl der in dieser Arbeit behandelte Ansatz sehr gute Ergebnisse liefert, bleiben dennoch Ansatzpunkte für weitere Forschungen auf diesem Gebiet vorhanden. Das zur Optimierung der *Mean Average Precision*-Funktion verwendete Gradientenverfahren konvergiert unter Umständen sehr langsam [52]. Außerdem ist es dafür notwendig, eine Approximation der MAP mithilfe der logistischen Funktion durchzuführen. Andere Optimierungsverfahren können hier möglicherweise schneller gute Ergebnisse liefern, lokale Optima besser vermeiden und auch auf nicht-stetischen Funktionen angewendet werden, wodurch die eben erwähnte Approximation nicht mehr benötigt würde. Mögliche Kandidaten solcher Optimierungsalgorithmen wären z. B. der *Artificial Bee Colony*-Algorithmus [28], der *Gravitational Search Algorithm* [46] oder die *Particle Swarm Optimization* [43, 30].

Ein weiterer Ansatzpunkt für mögliche Verbesserungen sind die verwendeten Eingaben in Form des impliziten Feedbacks, da diese nur erfassen, ob eine Interaktion zwischen Benutzer und Metrik für einen Kontext stattgefunden hat oder nicht. Es ist

denkbar, dass zusätzlich auch erfasst werden kann, wie oft Interaktionen für die jeweiligen Kombinationen aus Benutzer, Metrik und Kontext stattgefunden haben (vgl. Kapitel 4.2.2). Vergleiche auf Basis dieser Häufigkeiten ergeben feinere Abstufungen des impliziten Feedbacks und somit potentiell die Möglichkeit aussagekräftigere Informationen über den Benutzer zu sammeln. Eine Übersicht über weitere Möglichkeiten implizites Feedback zu sammeln kann aus den Arbeiten von Oard et al. [37] und Kelly et al. [29] entnommen werden.

Eine Verbesserung der von *DeepSonar* produzierten Empfehlungslisten könnte auch durch weitere Forschungsarbeit für die Ableitung der benutzerspezifischen Relevanz der dargestellten Software-Metriken in Abhängigkeit von den gewählten Kontextfaktoren erreicht werden. Zwar haben sich Odic et al. [38, 39] bereits mit der Auswahl von relevanten Kontextfaktoren beschäftigt, jedoch schieden viele dieser Möglichkeiten aufgrund von Schwierigkeiten bei der Erfassung in einer Webanwendung aus. Im Gegensatz zu modernen Smartphones haben klassische Desktop-PCs üblicherweise keine vergleichbare Sensorik eingebaut, auf die sich verlassen werden kann, um den Nutzungskontext zu erfassen.

Literaturverzeichnis

[1] Public sonarqube instances. http://www.sonarqube.org/resources/public-sonarqube-instances/. Zugriff: 25.10.2016.

[2] Sonarqube blog post about tendencies in metrics. http://www.sonarqube.org/tendencies-in-sonar/. Zugriff: 19.12.2016.

[3] SonarQube project homepage. http://www.sonarqube.org/. Zugriff: 25.10.2016.

[4] W3c draft on geolocation api. https://dev.w3.org/geo/api/spec-source.html. Zugriff: 19.12.2016.

[5] Ieee standard for a software quality metrics methodology. *IEEE Std 1061-1998*, pages i–, Dec 1998.

[6] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. *Towards a Better Understanding of Context and Context-Awareness*, pages 304–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[7] Gediminas Adomavicius and Alexander Tuzhilin. *Context-Aware Recommender Systems*, pages 191–226. Springer US, Boston, MA, 2015.

[8] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, 2009.

[9] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 301–304, New York, NY, USA, 2011. ACM.

[10] Victor R Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, 1992.

[11] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[12] James Bennett and Stan Lanning. The netflix prizes. Categories and Subject Descriptors I.2.6 [Machine Learning]: Engineering applications -applications of techniques. General Terms Experimentation, Algorithms.

[13] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[14] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 47–56, New York, NY, USA, 2011. ACM.

[15] Gianluca Borghini, Laura Astolfi, Giovanni Vecchiato, Donatella Mattia, and Fabio Babiloni. Measuring neurophysiological signals in aircraft pilots and car drivers for the assessment of mental workload, fatigue and drowsiness. *Neuroscience & Biobehavioral Reviews*, 44:58 – 75, 2014. Applied Neuroscience: Models, methods, theories, reviews. A Society of Applied Neuroscience (SAN) special issue.

[16] Hyun Jin Cha, Yong Se Kim, Seon Hee Park, Tae Bok Yoon, Young Mo Jung, and Jee-Hyong Lee. *Learning Styles Diagnosis Based on User Interface Behaviors for the Customization of Learning Interfaces in an Intelligent Tutoring System*, pages 513–524. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[17] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13(3):216–235, 2010.

[18] Cristina Conati and Samad Kardan. Student modeling: Supporting personalized instruction, from problem solving to exploratory open ended activities. *AI Magazine*, 34(3):13–26, 2013.

[19] HASKELL B. CURRY. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[20] Lisa Dent, Jesus Boticario, John Mcdermott, Tom Mitchell, and David Zabowski. A personal learning apprentice. pages 96–103. AAAI Press, 1992.

[21] Sidney K. D'Mello and Arthur Graesser. Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. *User Modeling and User-Adapted Interaction*, 20(2):147–187, 2010.

[22] Takács Gábor; Pilászy István; Németh Bottyán; Tikk Domonkos. [acm press the 2008 acm conference - lausanne, switzerland (2008.10.23-2008.10.25)] proceedings of the 2008 acm conference on recommender systems - recsys '08 - matrix factorization and neighbor based algorithms for the netflix prize problem. 2008.

[23] Matt Duckham and Lars Kulik. *A Formal Model of Obfuscation and Negotiation for Location Privacy*, pages 152–170. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[24] Hassan El Moussaoui and Prof. Dr. Klaus Zeppenfeld. *AJAX: Geschichte, Technologie, Zukunft*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[25] Richard M Felder and Linda K Silverman. Learning and teaching styles in engineering education. *Engineering education*, 78(7):674–681, 1988.

[26] David Flanagan. *JavaScript: the definitive guide*. Ö'Reilly Media, Inc.", 2006.

[27] Michael Jendryschik. Einführung in xhtml, css und webdesign. *Standardkonforme, moderne und barrierefreie Websiten erstellen*, 2, 2009.

[28] D. Karaboga and B. Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing*, 8(1):687 – 697, 2008.

[29] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: A bibliography. *SIGIR Forum*, 37(2):18–28, September 2003.

[30] James Kennedy. Particle swarm optimization. In Claude Sammut and GeoffreyI. Webb, editors, *Encyclopedia of Machine Learning*, pages 760–766. Springer US, 2010.

[31] Yong Se Kim, Sungah Kim, Yun Jung Cho, and Sun Hee Park. Adaptive customization of user interface design based on learning styles and behaviors: A case study of a heritage alive learning system. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 555–559. American Society of Mechanical Engineers, 2005.

[32] Kazuaki Kishida. *Property of average precision and its generalization: An examination of evaluation indicator for information retrieval experiments*. National Institute of Informatics Tokyo, Japan, 2005.

[33] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[34] Ashery Mbilinyi, Shinobu Hasegawa, and Akihiro Kashihara. *Design for Adaptive User Interface for Modeling Students' Learning Styles*, pages 168–177. Springer International Publishing, Cham, 2016.

[35] Hiroshi Motoda and Kenichi Yoshida. Machine learning techniques to make computers easier to use. *Artificial Intelligence*, 103(1):295 – 321, 1998.

[36] Netflix. Netflix prize data set. 2009.

[37] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83, 1998.

[38] Ante Odić. *Detecting, Acquiring and Exploiting Contextual Information in Personalized Services*, pages 374–377. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[39] Ante Odić, Marko Tkalčič, Jurij F Tasič, and Andrej Košir. Predicting and detecting the relevant contextual information in a movie-recommender system. *Interacting with Computers*, 25(1):74–90, 2013.

[40] Chihiro Ono, Yasuhiro Takishima, Yoichi Motomura, and Hideki Asoh. *Context-Aware Preference Model Based on a Study of Difference between Real and Supposed Situation Data*, pages 102–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[41] David M Pennock, Eric Horvitz, C Lee Giles, et al. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *AAAI/IAAI*, pages 729–734, 2000.

[42] Eduardo Kessler Piveta and Luiz Carlos Zancanella. Observer pattern using aspect-oriented programming. *Scientific Literature Digital Library*, 2003.

[43] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[44] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[45] J Ross Quinlan. Generating production rules from decision trees. In *IJCAI*, volume 87, pages 304–307. Citeseer, 1987.

[46] Esmat Rashedi, Hossein Nezamabadi-pour, and Saeid Saryazdi. Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232 – 2248, 2009. Special Section on High Order Fuzzy Sets.

[47] Dietmar Ratz. *Automatische Ergebnisveri kation bei globalen Optimierungsproblemen*. PhD thesis, Dissertation, Universit at Karlsruhe, 1992.

[48] Luis Rodriguez-Lujan, Pedro Larrañaga, and Concha Bielza. Frobenius norm regularization for the multivariate von mises distribution. *International Journal of Intelligent Systems*, 32(2):153–176, 2017.

[49] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. Tfmap: Optimizing map for top-n context-aware recommendation. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 155–164, New York, NY, USA, 2012. ACM.

[50] J. H. Su, H. H. Yeh, P. S. Yu, and V. S. Tseng. Music recommendation using content and context information mining. *IEEE Intelligent Systems*, 25(1):16–26, Jan 2010.

[51] Leonie Sugarman. Experiential learning: Experience as the source of learning and development, david a. kolb, prentice-hall international, hemel hempstead, herts.,

1984. no. of pages: xiii+ 256. *Journal of Organizational Behavior*, 8(4):359–360, 1987.

[52] Julian Valentin. Hierarchische Optimierung mit Gradientenverfahren auf Dünngitterfunktionen. Master's thesis, Universität Stuttgart, Germany, 2014.

[53] Z. Yujie and W. Licai. Some challenges for context-aware recommender systems. In *2010 5th International Conference on Computer Science Education*, pages 362–365, Aug 2010.

[54] Yong Zheng. A revisit to the identification of contexts in recommender systems. In *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, IUI Companion '15, pages 133–136, New York, NY, USA, 2015. ACM.

[55] Yong Zheng, Robin Burke, and Bamshad Mobasher. *Differential Context Relaxation for Context-Aware Travel Recommendation*, pages 88–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift