Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN-Fakultät
Fachbereich Informatik I Softwaretechnik

Bachelorarbeit:

Crowdsourcing-Systeme Manipulation und Resistenz

-

am Beispiel der Modellierung eines „Online Crowdsourcing Calendar Tool“

Jens H.-P. Bothur
Stellinger Chaussee 35
22529 Hamburg

jens@bothur.de
Matr.Nr.: 5724523

1. Prüfer:	2. Prüfer:
Dr. Guido Gryczan	Till Aust

Hamburg 18.02.2013

„Nulla culpa instantior est quam, quae gratiam dicere. "

-

„Keine Schuld ist dringender als die, Danke zu sagen."

- Marcus Tullius Cicero, ca. 60 v. Chr.

„Namen sind Schall und Rauch.“

- Johann Wolfgang von Goethe, Faust I

Ich bedanke mich bei allen die mich motiviert haben, die mich inspiriert haben, die mich angetrieben haben, die mir aufgeholfen haben, die mich gelehrt und belehrt haben, die mich kritisiert haben und vor allem bei denen, die mich immer und immer wieder unterstützt und mich nie aufgegeben haben.

Ihr wisst wer ihr seid.

Inhaltsverzeichnis

Kapitel 1 Einleitung **1**

1.1 Motivation 1
1.2 Was ist ein „Online Crowdsourcing Calendar Tool“? 2
1.3 Zielsetzung 3
1.4 Vorstellung der Kapitel 4

Kapitel 2 Crowdsourcing **5**

2.1 Definition Crowdsourcing 5
2.2 Beispiele für Crowdsourcing-Systeme 6
2.3 Eigenschaften von Crowdsourcing-Systemen 10
2.4 Eigenschaften des OCCT. 15

Kapitel 3 Manipulation von Crowdsourcing-Systemen **19**

3.1 Warum überhaupt Manipulation von Crowdsourcing-Systemen? .19
3.2 Arten der Manipulation. 20
3.3 Resistenz gegen Manipulation 21
3.4 Nutzung des Spamming-Scores durch das OCCT 30

Kapitel 4 Implementation des Sicherheits-Frameworks **34**

4.1 Paketstruktur 34
4.2 Klassendiagramme 35
4.3 Wie nutze ich das Sicherheits-Framework?. 40
4.4 Tests & Metriken 41

Kapitel 5 Ergebnis, Auswertung und Ausblick **42**

5.1 Ergebnis 42
5.2 Auswertung 42
5.3 Ausblick 43

Erklärung **44**

Literaturverzeichnis **45**

Anhang

Kapitel 1

Einleitung

In diesem Kapitel möchte ich einen groben Überblick über meine Bachelorarbeit geben. Ich werde zunächst meine eigene und die fachliche Motivation erläutern, wie es zum Thema dieser Bachelorarbeit kam und vor allem welcher Weg mich zum Thema Crowdsourcing führte.
Dies gibt mir die benötigten Grundlagen um meine Zielsetzung zu formulieren. Zuletzt wird es noch eine Übersicht über die einzelnen Kapitel geben.

1.1 Motivation

Die **erste** Idee für diese Arbeit war ausschließlich die Modellierung und Entwicklung des OCCT. Die Motivation sich mit diesem Thema auseinander zu setzen und einen solchen Kalender zu entwickeln, entstand aus einem recht pragmatischen Umstand.
Ich wollte mich im Internet über das aktuelle Angebot an Internet-Übertragungen zu einem bestimmten Thema (e-Sports) informieren. In diesem Bereich gibt es im Internet sehr viele private „Sender“ und Produktionen, aber so etwas wie eine zentrale „Fernsehzeitschrift“ gibt es so gut wie nicht. Ich benötigte nahezu 1 Stunde um mir auch nur ansatzweise einen Überblick über das Angebot zu verschaffen, wobei es außer den aktuellen Zuschauerzahlen keine Qualitätsmerkmale gab, welche Sendung gut ist und welche nicht.
Die Idee hierfür ein web-taugliches Tool zu entwickeln liegt auf der Hand. Allerdings wollte ich nicht einer rein praktischen Programmieraufgabe nachgehen und suchte im Dialog mit Till Aust nach Möglichkeiten die Arbeit um einen theoretischen Teil zu erweitern, welcher meinem wissenschaftlichen Anspruch an eine solche Arbeit genügte.
Es entstand die **zweite** Idee für diese Arbeit, dass das OCCT auf Crowdsourcing basieren sollte und es kristallisierten sich folgende zentrale Fragen heraus: *„Was ist eigentlich Crowdsourcing?“* (siehe Kapitel 2 „Crowdsourcing“ für weitere Erläuterung) und *„Wie verhindert man, dass jemand zwecks Eigenwerbung ein solch offenes Tool missbraucht um seine eigenen Sendungen, Termine, Produkte, etc. unberechtigt positiv zu bewerten?“* (siehe Kapitel 3 „Manipulation von Crowdsourcing-Systemen“).

Schon nach kurzer Recherche stellte sich der Umfang dieser Fragen als enorm heraus. Sowohl eine theoretische Behandlung von Crowdsourcing-Systemen und deren Manipulationsresistenz als auch eine praktische Modellierung und Entwicklung des OCCT schien den Rahmen einer Bachelorarbeit deutlich zu sprengen. Schlussendlich blieb als Thema meiner Arbeit primär der theoretische Teil, welcher sich mit Crowdsourcing-Systemen und deren Manipulationsresistenz beschäftigt. Das OCCT soll nur noch modelliert werden und als Beispiel dienen um Sicherheitsprobleme gegenüber Manipulation von Crowdsourcing-Systemen zu beschreiben und insbesondere wie man solchen Sicherheitsproblemen begegnet.

1.2 Was ist ein „Online Crowdsourcing Calendar Tool“?

Zunächst ist ein „Online Crowdsourcing Calendar Tool“ eine von mir erdachte Anwendung, welche als Beispiel zur Veranschaulichung von Crowdsourcing-Systemen und den dazugehörigen Problemen bezüglich Manipulation und Manipulationsresistenz dient. Konkreter ist das OCCT ein Kalender in dem verschiedene Menschen Termine diverser Art online eintragen, bewerten und finden können.

Man stelle sich einen normalen Papierkalender vor, der im Flur eines Studentenwohnheims hängt. Jeder Student des Wohnheims hat dort die Möglichkeit eine von ihm organisierte Veranstaltung (Party, gemeinsames Kochen, Spieleabend, etc.) einzutragen. Zunächst kann sich so jeder andere über die verschiedenen Ereignisse, die an einem Abend stattfinden informieren.
Aber natürlich bietet dies noch wesentlich mehr Möglichkeiten. Zum einen könnte man sich als Teilnehmer bei den verschiedenen Veranstaltungen eintragen. Dann könnte ein "Beauftragter" immer die aktuell beliebteste Veranstaltung nach oben hängen.
Ebenso könnten Besucher der Veranstaltung hinterher ihre Meinung und Kritik über die Veranstaltung dem Termin hinzufügen, damit sich neue Besucher bei der nächsten Veranstaltung derselben Art über den Erfolg dieser bzw. die Zufriedenheit der Besucher informieren können.

Mittlerweile müsste der Papierkalender im Flur des Studentenwohnheimes schon ziemlich groß sein. Besonders wenn er auf einen Blick mehrere Termine an einem Tag über mehrere Wochen darstellen und dabei noch Rückmeldungen verwalten soll. Weiterhin müsste er noch modular sein, damit Veranstaltungen weiter nach oben gehängt werden können, wenn sich deren Priorität (=Beliebtheit) ändert.
Selbstverständlich liegt der Fokus dieser Arbeit nicht auf dem Basteln eines riesigen Papierkalenders, der als Metapher aber das Prinzip hinter einem OCCT gut verdeutlicht.

Ein besonderer Vorteil eines OCCT gegenüber dem Papierkalender ist, dass nicht nur eine Handvoll Studenten zu diesem Kalender beitragen, sondern denkbar ist, dass tausende von Menschen gemeinsam Termine in diesem Kalender erzeugen, gegenseitig bewerten und kommentieren. Somit könnte dem reinen Konsumenten mit wenigen Klicks eine Übersicht ermöglicht werden, welche Veranstaltungen es zu einem bestimmten Zeitpunkt gibt und vor allem welche hiervon mehr oder weniger beliebt bzw. erfolgreich sind.

Natürlich könnte es ein ziemliches Chaos geben, wenn verschiedenste Veranstaltungen in einem Kalender zusammengefasst werden. Denkbar ist also, dass jede Interessengemeinschaft ihren eigenen Kalender bezüglich ihres gemeinsamen Interesses pflegt.
Ebenso kann man sich leicht ein *TAG* (=Kennzeichen)-System überlegen, in dem jeder Termin einen oder mehrerer TAGs erhält. Der Informationssuchende kann sich dann seine gesuchten Termine nach bestimmten TAGs filtern bzw. anordnen lassen.

1.3 Zielsetzung

Im zentralen Augenmerk dieser Arbeit wird die die Manipulationsresistenz eines *Crowdsourcing-Bewertungssystem* stehen. Ich möchte dem Leser näher bringen was überhaupt Crowdsourcing-Systeme sind und warum es attraktiv ist bzw. worin der Vorteil liegt solche Systeme zu manipulieren.
Ebenso werde ich beschreiben inwiefern man versuchen kann eine Modellierung gegen solche Angriffe robuster zu gestalten bzw. worin die Herausforderung und die Problematik liegt aktiv solche Manipulationen zu erkennen. Dies ist weniger trivial, als es zunächst erscheinen mag, berücksichtigt man die Schwierigkeiten einen einzelnen Benutzer des Systems zu identifizieren bzw. die sehr kleinen Datenmengen, welche von einem solchen Benutzer ins System eingebracht werden.

Das OCCT wird als Beispiel dienen die verschiedenen Aspekte und Eigenschaften von Crowdsourcing-Systemen zu veranschaulichen. Selbst werde ich das OCCT nicht entwickeln, sondern nur im groben modellieren. Hierbei werde ich mich ausschließlich mit der von *M. Fowler* beschriebenen mittleren Schicht einer dreischichtigen, objektorientierten Software befassen. Die sogenannte Funktionale-Komponente (FK oder *Domain*). Fowler schreibt hierzu:

„Domain – Logic that is the real point of the system“ - [Fow02], Seite 20

Betrachtungen der Interaktions-Komponente (IAK oder *Presentation*) bzw. des "Graphical User Interface (GUI)" und der Datenbankanbindung (DB oder *Data Source*) bzw. der "Database" sollen nicht Teil dieser Arbeit sein.

Zuletzt werde ich in dieser Arbeit ein *Sicherheits-Framework* implementieren, welches die modellierten Sicherheitsaspekte des OCCT umsetzt. Dieses Framework wird die Erkenntnisse verschiedener Arbeiten, wie zum Beispiel [Lim10] oder [Muk12], auf eine, nach meinem Wissensstand, neue Art und Weise vereinheitlichen und speziell für das OCCT anpassen. Insbesondere da die vom Framework profitierende Software (=OCCT) nicht mit entwickelt wird, hoffe ich eine Basis zu schaffen, auf der zukünftige Arbeiten aufsetzen können.

Die Modellierung und Implementierung des Frameworks werden nach dem Werkzeug-Material (WAM) Ansatz ausgeführt, welcher von *H. Züllighoven* geprägt wird (vergleiche: [Zül05], Kapitel 1.2, Seiten 4-9). Zusätzlich werde ich an geeigneten Stellen versuchen die Anwendbarkeit des WAM-Ansatzes auf meine Modellierung zu beschreiben und zu bewerten bzw. eventuelle Probleme aufzuzeigen.

1.4 Vorstellung der Kapitel

Es folgt eine kurze Übersicht die einzelnen Kapitel dieser Arbeit.

Kapitel 2. Crowdsourcing. In diesem Kapitel werde ich genauer erläutern was Crowdsourcing ist. Hierzu werde ich einige Beispiele für Crowdsourcing-Systeme geben und deren Funktionsweise beschreiben. Im Anschluss werden die verschiedenen Eigenschaften, welche Crowdsourcing-Systeme klassifizieren, wie z.B. die *Art des Problems*, detailliert erläutert.
Am Ende des Kapitels werde ich die verschiedenen Crowdsourcing-Elemente des OCCT ermitteln und deren geplante Funktionsweise festlegen bzw. modellieren.

Kapitel 3. Manipulation von Crowdsourcing-Systemen. In diesem Kapitel setze ich mich mit der Manipulation von Crowdsourcing-Systemen auseinander. Mit der Motivation die dahinter steht, aber vor allem auch mit den verschiedenen Möglichkeiten solche Manipulationen zu entdecken und ein System dagegen zu schützen. Wir werden verschiedene Arten der Manipulation kennenlernen, wie zum Beispiel *Opinion-Spam* oder *Sybil-Angriff*.
Zuletzt werde ich überlegen, welche dieser Möglichkeiten für das OCCT Anwendung finden sollen und die gewünschten Methoden zur Absicherung gegen Manipulation modellieren.

Kapitel 4. Implementierung des Sicherheits-Frameworks. In diesem Kapitel möchte ich die Implementierungsdetails des Sicherheits-Frameworks für das OCCT erläutern. Ich werde mich zunächst grundlegend der entstandenen Paketstruktur widmen und schließlich jedes Paket einzeln durchleuchten, seine Beziehungen und Klassen erklären.
Darauf folgt eine Erläuterung, wie das Sicherheits-Framework zu nutzen ist bzw. wie es in einem bestehenden Projekt oder einer zukünftigen Entwicklung des OCCT eingebunden werden kann, bevor ich mich zuletzt kurz den Themen Testen, Testabdeckung und Metriken widme.

Kapitel 5: Ergebnis, Auswertung und Ausblick. In diesem letzten Kapitel werde ich zunächst einen kurzen Überblick über die Ergebnisse und Ziele geben, welche ich im Laufe dieser Bachelorarbeit erreicht habe. Ebenso möchte ich diese Ergebnisse, aber auch die Arbeitsweise während der Implementierung auswerten.
Zuletzt möchte ich einen Ausblick auf zukünftige mögliche Arbeiten geben. Welche Problematiken habe ich vernachlässigt, welche Probleme sind mir während der Arbeit an meinem Konzept zur Entdeckung von Review-Spam aufgefallen und wie könnte dieses Konzept weiterentwickelt werden.

Kapitel 2

Crowdsourcing

In diesem Kapitel werde ich genauer erläutern was Crowdsourcing ist. Hierzu werde ich einige Beispiele für Crowdsourcing-Systeme geben und deren Funktionsweise beschreiben. Im Anschluss werden die verschiedenen Eigenschaften, welche Crowdsourcing-Systeme klassifizieren, wie z.B. die *Art des Problems*, detailliert erläutert.
Am Ende des Kapitels werde ich die verschiedenen Crowdsourcing-Elemente des OCCT ermitteln und deren geplante Funktionsweise festlegen bzw. modellieren.

2.1 Definition Crowdsourcing

Im Web und in zahlreicher Literatur findet man die verschiedensten Definition für Crowdsourcing. Befragt man seine Bekannten und Kollegen, erhält man von jedem, der mit dem Begriff überhaupt etwas anfangen kann, eine andere Aussage. Sehr einfach beschreiben es [Doa11] mit:

"..., we view CS [=Crowdsourcing] *as a general-purpose problem-solving method."*

Crowdsourcing ist also eine Problemlösungsmethode, die sich für einen beliebigen Zweck anwenden lässt. Dies ist zwar richtig, aber meines Erachtens dennoch so weit gefasst und unkonkret, dass wir immer noch keine bessere Vorstellung davon haben, was Crowdsourcing nun eigentlich ist. Eine wesentlich präzisere Definition liefern [Est12]:

"Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization or company proposes to a group of individuals of varying knowledge, heterogeneity and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken".

Crowdsourcing bietet also dem *Crowdsourcer* eine seiner Bestimmung unterstellte Möglichkeit einer *Crowd* die freiwillige Mitarbeit an einer Aufgabe bzw. Problemlösung vorzuschlagen bzw. zu ermöglichen. Der Crowdsourcer kann hierbei ein Individuum, eine Institution, eine gemeinnützige Organisation oder ein Unternehmen sein. Die Crowd hingegen ist eine zunächst unbestimmte Menge von Individuen von verschiedener Bildung, Herkunft und Anzahl.

Der „Vorschlag“ bzw. die Kommunikation geschieht hierbei über ein offenes System oder einen öffentlichen Aufruf. Dies ist essentiell, damit die Crowd auch wirklich eine vom Crowdsourcer nicht vorherbestimmte Zusammenstellung hat. Wird der „Vorschlag“ nur einer gewissen vom Crowdsourcer exakt vorherbestimmten Menge zugänglich gemacht, lassen sich evtl. auch einige Methoden von Crowdsourcing anwenden, aber es handelt sich im Prinzip nicht mehr um Crowdsourcing.
Die Aufgabe kann hierbei von verschiedenster Art und Weise sowie Komplexität sein. In der Praxis reicht die Komplexität von Crowdsourcing-Aufgaben von simpelsten, ja schon trivialen Tätigkeiten, wie z.B. dem Finden eines Bootes auf einem Meeresbild, bis hin zu hoch komplexer Anwendung von Wissen und Fähigkeiten, wie z.B. Computercode-Entwicklung für ein Betriebssystem. (siehe spätere Beispiele für weitere Details)
Die Unternehmung, in der die Crowd entweder Arbeit, Geld, Wissen und/oder Erfahrung einbringen soll, geschieht natürlich zu beiderseitigem Vorteil. Der Teilnehmer der Crowd erhält die Befriedigung eines Bedürfnisses, dies kann primär finanzieller Natur sein, aber auch gesellschaftliche Anerkennung, steigendes Selbstbewusstsein oder die Entwicklung eigener Fähigkeiten können ein Anreiz sein. Der Crowdsourcer hingegen kann die Leistungen, welche die Crowd eingebracht hat, nun in Besitz nehmen und zu seinem Vorteil einsetzen.

Ich definiere Crowdsourcing wie folgt:

> *„**Crowdsourcing** beschreibt das Vorgehen eines Auftraggebers, des Crowdsourcers, einen Aufruf zu veröffentlichen, so dass eine beliebige, nicht vorherbestimmte Menge von Individuen, die Crowd, zu ihrem eigenen Vorteil eine Aufgabe bzw. Problemlösung für den Crowdsourcer lösen bzw. erarbeiten kann.*
> *Sowohl die Offenheit der Kommunikation und der Problemstellung als auch die Inbesitznahme der Ergebnisse durch den Crowdsourcer sind hierbei essentiell.“*

2.2 Beispiele für Crowdsourcing-Systeme

Somit soll Crowdsourcing ausreichend definiert sein und wir haben eine grundlegende Vorstellung worum es sich dabei handelt. Es folgen einige Beispiele für Crowdsourcing-Systeme. Diese reichen von vielleicht eher unbekannten Systemen, wie z.B. reCAPTCHA, bis hin zu wahrscheinlich fast jedem bekannten Systemen, wie Wikipedia.
Die Vielfalt von möglichen Crowdsourcing-Systemen wird hier zwar schon recht deutlich, ist aber in keinem Fall auf die genannten Beispiele beschränkt.

reCAPTCHA. Um Zugriff auf eine bestimmte Funktion oder einen bestimmten Inhalt einer Internetseite zu bekommen, muss man immer häufiger zunächst eine dargestellte Ziffern- und Buchstabenfolge eingeben. Dies dient zur Verifizierung, dass die Anfrage an die Internetseite bzw. den Server wirklich von einem Menschen vor einem Computer ausgeführt wird und nicht von einem automatisierten Programm.

Automatisierte Programme haben große Schwierigkeiten Ziffern und Buchstaben in Bildern zu erkennen, vor allem, wenn die Bilder auch noch mit weiteren Linien durchzogen oder die einzelnen Zeichen etwas entstellt sind. Für einen Menschen ist es ein Leichtes aus diesem „Gewirr“ die gesuchten Zeichen zu erkennen, für Computer nicht.

Dies nennt man einen *CAPTCHA* oder „**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part“. Also einen „vollständig automatisierten öffentlichen Turing-Test um Computer und Menschen auseinander zu halten". Im Grunde ist so ein CAPTCHA schon ein Crowdsourcing-System im weiteren Sinne, da die Crowd eine Tätigkeit zu ihrem eigenen Vorteil, dem Zugriff auf Funktion bzw. Inhalt, ausübt. Der Crowdsourcer hat die Sicherheit, dass sein System nicht von *Bots* (= automatisierten Programmen) überschwemmt wird und nur „echte“ Menschen Zugriff erhalten.

Aber hat dies jetzt wirklich etwas mit der Lösung einer Aufgabe zu tun und liefert dieses „Crowdsourcing“ ein Ergebnis? Auch hier könnte man im weiteren Sinne wieder mit „Ja“ antworten, aber spätestens wenn man sich die Erweiterungen, welche reCAPTCHA diesem System hinzufügt, ansieht, wird offensichtlich, dass es sich hier um ein Crowdsourcing-System handelt.

reCAPTCHA erweitert den normalen CAPTCHA um eine zweite Ziffern- und Buchstabenfolge. Der User muss beide eingeben. Was der User nicht weiß ist, dass nur eine Folge zur Verifizierung seiner Menschlichkeit benutzt wird. Die zweite ist eine neue Folge, für die das reCAPTCHA-System selbst noch keine Lösung kennt. Wurde die zweite neue Folge nun oft genug „gelöst“ und eine Lösung wurde mit Abstand am häufigsten eingetippt, so wird diese Lösung als „echte“ Lösung der bisher unbekannten Folge angenommen und gespeichert. So erneuert sich die Datenbank mit reCAPTCHAs von selbst.

Aber viel entscheidender ist, dass auf diese Weise alte Texte digitalisiert werden können, deren Auflösung zu niedrig für normale Texterkennung ist. Die einzelnen Wörter dieser alten Texte werden einfach als unbekannte Folge in reCAPTCHAs verwendet und sobald eine Lösung feststeht, hat man das Wort effektiv digitalisiert. Laut www.recaptcha.net hat reCAPTCHA eine höhere bzw. vergleichbare Genauigkeit wie andere anerkannte Methoden zur manuellen Digitalisierung von alten Texten.

Google nutzt reCAPTCHA zum Beispiel zur Digitalisierung aller 130 Jahrgänge der New York Time. Nach eigenen Aussagen werden pro Tag über 200 Millionen reCAPTCHAs für Google gelöst, was einer Arbeitszeit von über 150.000 Stunden entspricht. (http://www.google.com/recaptcha/learnmore)

Amazon: Produktbewertung durch Kunden. www.amazon.de ist ein Internetportal zum Einkaufen. Während wir uns nicht weiter mit allen, der mittlerweile sehr zahlreichen, Funktionen und Angeboten von Amazon beschäftigen wollen, werde ich mich hier zunächst mit der Produktbewertung und später mit dem Amazon Mechanical Turk, einem Marktplatz für Crowdsourcing-Projekte, auseinander setzen.

Amazon erlaubt es seinen registrierten Benutzern ein Produkt zu bewerten. Der Kunde kann hier eine Bewertung von 1-5 Sterne vergeben und eine beliebig ausführliche Rezension schreiben. Andere Benutzer können diese Bewertungen bzw. Rezensionen dann lesen und sie als hilfreich oder nicht hilfreich einstufen. Die Hilfreichsten werden priorisiert angezeigt. Außerdem kann so für jedes Produkt eine durchschnittliche Bewertung berechnet werden. Denkbar wäre zum Beispiel eine durchschnittliche Bewertung von 3,7 Sternen bei insgesamt 167 Bewertungen, welche natürlich einzeln nur ganzzahlig zwischen 1 und 5 liegen.

Der Vorteil für Amazon liegt ganz klar darin, dass Kunden eher dazu neigen ein Produkt zu kaufen, wenn sie Vergleichswerte haben. Egal wie mutmaßlich diese auch sein mögen. Die Motivation der Nutzer solche Rezensionen zu schreiben liegt neben verschiedenen Amazon-Aktionen, wie z.B. einen 5 € Gutschein für die erste mehr als 50% hilfreiche Rezension für ein Produkt, wohl am ehesten die eigene Meinung Kund zu tun und die Zufriedenheit bzw. Unzufriedenheit über ein Produkt zum Ausdruck zu bringen.

Amazon: Mechanical Turk. Dieses Crowdsourcing-System unterscheidet sich von anderen Crowdsourcing-Systemen, da es nicht direkt eine bestimmte Aufgabenstellung hat, sondern viel mehr eine Plattform bietet, auf der beliebige Anbieter, sogenannten *Requester*, eine Crowdsourcing-Aufgabe stellen können. Diese wird dann von der Crowd, hier *Worker* genannt, erledigt. Meistens handelt es sich hierbei um für Menschen sehr triviale Aufgaben, welche aber für Computer ein größeres Problem darstellen. (vergleiche: https://www.mturk.com/mturk/help?helpPage=overview#what_is)
Ein berühmtes Beispiel (vergleiche [Ols08]) ist die Suche nach dem Informatiker James Nicholas „Jim“ Gray. Dieser verscholl am 28.01.2007 mit seinem Boot vor der Küste Kaliforniens. Seine Familie, Freunde und Kollegen ließen daraufhin auf dem Mechanical Turk bis Mitte Mai 2007 von über 12.000 Workern Satellitenphotos von über 360.000 km² (etwas mehr als die Fläche Deutschlands) des pazifischen Ozeans nach dem Boote Grays durchsuchen. Trotz aller Mühen gilt Gray heute als auf See verstorben.
Bei Amazon Mechanical Turk ist es üblich die Worker für ihre Arbeit zu bezahlen. Im Schnitt werden Löhne zwischen 5 und 10$ pro Stunde erzielt.
Einige Veröffentlichungen bezeichnen den Mechanical Turk als den ersten Schritt zu *Meta-Crowdsourcing-Systemen.* Diese Systeme bzw. System-Ideen sind aber nicht Zentrum dieser Arbeit und deswegen wird auf diese Eigenschaften hier nicht weiter eingegangen.

Google: Autovervollständigung. Eine Autovervollständigung ist ein System, welches einen Benutzer z.B. bei einer Texteingabe unterstützt. Während der Eingabe schlägt das System dem Benutzer bereits vollständige Wörter oder sogar Sätze vor, welche dieser dann mit einer weiteren Taste bestätigen kann und das Wort bzw. den Satz so nicht komplett schreiben muss. Dieser Vorschlag aktualisiert sich mit jedem eingegebenen Buchstaben. (vergleiche: http://support.google.com/websearch/bin/answer.py?hl=en&answer=106230)
Ein Beispiel hierfür findet man bei einer Google-Suche. Der Crowdsourcing Gedanke hierbei ist nicht direkt offensichtlich. Die Google-Autovervollständigung basiert hierbei auf einer Datenbank welche so gut wie alle Suchanfragen, die jemals gestellt wurden, gespeichert hat. Sobald dann die ersten Buchstaben einer neuen Suchanfrage eingegeben wurden, werden dem Benutzer basierend auf alten Suchanfragen mit exakt diesen Buchstaben nach Häufigkeit sortierte Vorschläge unterbreitet. Gibt man z.B. auf www.google.de die Buchstaben „deu“ ein, erhält man die Vorschläge: „deutsche Bahn“, „deutsche Bank“, „deutsche Post“ und „deutsch englisch“.

Den meisten Benutzern von Google ist wahrscheinlich nicht bewusst, dass sie während ihrer Informationssuche gleichzeitig als Crowd für Befüllung und Verbesserung dieser Autovervollständigungs-Datenbank tätig sind. In diesem Crowdsourcing-System gibt es also keinen „Lohn“ oder Motivation für den Benutzer und das Ergebnis entsteht sozusagen als Nebenprodukt zu einer anderen Tätigkeit, der Informationssuche.

Wikipedia. Unter www.wikipedia.org findet man eine wahrscheinlich heutzutage vielen Menschen bekannte Enzyklopädie. In der Wikipedia kann jeder zu einem nahezu beliebigen Thema Einträge vornehmen. Man wird dazu angehalten diese Einträge gemäß den Einträgen eines Lexikons zu machen.
Es wird also in Gemeinschaftsarbeit eine Wissensbasis bzw. ein Lexikon geschaffen, welches, nach eigener Aussage von Wikipedia, in gedruckter Form knappe 700 Bücher füllen würde (der aktuelle Brockhaus hat 30 Bücher, siehe Anhang Seite: I).
Das Modell *„Wikipedia“* hat mittlerweile Schule gemacht und in verschiedensten Zusammenhängen werden nun auf dieser Idee des Crowdsourcing Wissensbasen geschaffen, sogenannte *„Wikis“*.
Die Entlohnung der Benutzer, welche Wikipedia oder andere Wikis mit Daten füllen, ist zum einen ein gemeinnütziger Gedanke, aber es spielen auch Effekte wie soziale Anerkennung und Selbstdarstellung eine große Rolle. Es gibt mittlerweile zu fast jeder größeren Firma Informationen auf Wikipedia und es liegt natürlich im Interesse einer Firma sich auf einer solchen Seite zu präsentieren.
Unter diesem Aspekt ist Wikipedia eine Plattform, welche sich für Missbrauch, Fehlinformation oder sogar Diffamierung gerade zu anbietet. Eines der berühmtesten Beispiele ist Herr John Seigenthaler Sr., welcher fälschlicherweise beschuldigt wurde, er könne Teil der Kennedy Ermordungen gewesen sein (vgl. [Cox11]):

„John Seigenthaler Sr. was the assistant to Attorney General Robert Kennedy in the early 1960s. For a brief time. He was thought to have been directly involved in the Kennedy assassinations of both John, and his brother, Bobby. Nothing was ever proven.“

- Wikipedia, Mai 2005

Diesen falschen Teil der Biographie findet man heute nicht mehr auf Wikipedia und es sind mittlerweile mehrere Sicherheitsmechanismen und -prüfungen installiert, welche genau solchen Missbrauch verhindern sollen. (vergleiche: http://en.wikipedia.org/wiki/Wikipedia:Quality_control)

Auf Missbrauch von Crowdsourcing-Systemen und wie man solchen verhindert, wird genauer in Kapitel 3 eingegangen.

2.3 Eigenschaften von Crowdsourcing-Systemen

Nachdem wir nun eine gute Übersicht haben, was Crowdsourcing-Systeme sind und in welcher Vielfalt sie existieren können, werde ich mich jetzt den Eigenschaften von Crowdsourcing-Systemen zuwenden. [Doa11] nennen 9 allgemeine Eigenschaften mit deren Hilfe man Crowdsourcing-Systeme klassifizieren kann. In diesem Teilkapitel werde ich uns einen Überblick über diese Eigenschaften verschaffen, indem ich die 9 von [Doa11] Genannten kurz anreiße und in Zusammenhang mit unseren Beispielen stelle.

Degree of manual effort. Diese Eigenschaft definiert den *Anteil an manuellem Aufwand.* D.h. wie viel Arbeit auf dem Weg zu einer angestrebten Lösung muss bzw. soll manuell erledigt werden. Dies hängt stark von dem Problem bzw. der zu lösenden Aufgabe ab, aber auch von der Automatisierung des Systems.
reCAPTCHA ist zum Beispiel hoch automatisiert. Die zu digitalisierenden Dokumente werden automatisch eingescannt, in Wörter unterteilt und der Datenbank zur Übersetzung zur Verfügung gestellt. Ebenso werden automatisch erfolgreiche Übersetzungen wieder zusammengefügt. Der manuelle Anteil liegt also nur in der Übersetzung und der späteren Kontrolle ob der digitalisierte Text einen Sinn ergibt und ist damit im Vergleich zur Gesamtaufgabe relativ gering. Dies führt uns zu der Frage, wie der Anteil des manuellen Aufwands zwischen der Crowd und dem Crowdsourcer aufgeteilt wird. Bei reCAPTCHA wird der wesentlich größere Teil des manuellen Aufwands, nämlich die Übersetzung, von der Crowd erledigt, während ein deutlich kleinerer Teil, der der Kontrolle, dem Crowdsourcer zufällt.
Im Beispiel Wikipedia ist der Anteil des manuellen Aufwands wesentlich höher. Hier wird die Erstellung nahezu des gesamten Inhalts sowie dessen Überprüfung auf Richtigkeit von der Crowd manuell ausgeführt. Somit hat hier der manuelle Aufwand einen sehr hohen Anteil an der Gesamtaufgabe. Ebenso liegt aber der Großteil hiervon wieder bei der Crowd und nicht beim Crowdsourcer.

Role of human users. [Doa11] unterscheiden vier Arten von *Rollen der menschlichen Benutzer.* Den *Sklaven*, den *Perspektiven-Provider*, den *Inhalt-Provider* und den *Komponenten-Provider.*
Der Sklave übernimmt normalerweise nur eine triviale Aufgabe und übt diese entweder nur sehr kurz oder in sehr hoher Anzahl aus. Ein Beispiel ist die Suche nach Jim Grays Boot auf Satellitenphotos auf Amazons Mechanical-Turk. Hier musste sich der Benutzer lediglich ein Bild ansehen und Stellen, wo er ein Boot vermutete, mit der Maus anklicken.
Der Perspektiven-Provider stellt seine eigene Perspektive bzw. Meinung zur Verfügung. Hieraus kann dann ein Durchschnitt generiert werden. Ein solcher Durchschnitt ist bei recht einfachen Aufgaben, wie z.B. die Vergabe von Sternen in der Produktbewertung auf Amazon, meistens deutlich repräsentativer, als die Meinung eines Einzelnen, z.B. eines Experten.
Das Schreiben einer kompletten Rezension in der Amazon Produktbewertung fällt hingegen bereits in den Aufgabenbereich des Inhalt-Providers. Die Aufgaben des Inhalt-Providers sind des Öfteren komplexerer Natur. Weitere Beispiele sind das Erstellen von Artikeln auf Wikipedia oder auch das automatisierte Auslesen von Suchanfragen durch Google.

Als letztes bleibt uns der Komponenten-Provider. Crowdsourcing-Systeme, in denen die Crowd die Rolle des Komponenten-Provider annimmt, haben meistens die Erschaffung eines Artefakts (siehe „Art des Problem“ weiter unten) zum Ziel. Dies kann die Opensource-Programmierung eines Betriebssystems oder die Digitalisierung eines alten Textes mittels reCAPTCHA sein. Der Komponenten-Provider erstellt hierbei im Laufe seines Handels, wie der Name schon vermuten lässt, eine oder mehrere Komponenten, welche dann zum Zielartefakt hinzugefügt bzw. zusammengefügt werden.
Natürlich ist ein Benutzer nicht auf eine Rolle beschränkt. Eine Rezension in der Amazon Produktbewertung zu schreiben lässt einen gleichzeitig die Rolle des Perspektiven-Providers und die des Inhalt-Providers annehmen. Ebenso nimmt ein Nutzer beim Lösen eines reCAPTCHAS die Rolle des Sklaven und des Komponenten-Providers an.

Standalone versus piggyback. Mit dieser Eigenschaft ziehen [Doa11] den Unterschied zwischen *selbständigen* Crowdsourcing-Systeme und Systemen, welche auf anderen Systemen aufsetzen, sogenannten *Huckepack*-Systemen. Selbständige Crowdsourcing-Systeme arbeiten „alleine“ und müssen sich der Herausforderung (auf Herausforderungen wird später in diesem Kapitel eingegangen) Benutzer zu rekrutieren stellen. Die Amazon Produktbewertung, alle Aufgaben auf dem Amazon Mechanical Turk und die Erstellung von Artikeln auf Wikipedia sind selbständige Systeme, da sie Benutzer motivieren müssen diese Aufgaben zu erledigen.
Die Google-Autovervollständigung und reCAPTCHA sind hingegen Huckepack-Systeme, da sie auf anderen Tätigkeiten aufsetzen und so automatisch Benutzer erhalten und nicht erst welche motivieren müssen. Die Benutzer sind entweder aus anderer Quelle motiviert die Aufgabe zu erledigen, zum Beispiel im Falle von reCAPTCHA den Zugriff auf eine Internetseite zu bekommen, oder bemerken gar nicht, dass sie die Aufgabe erledigen. Letzteres wäre bei einer Google-Suche der Fall.

Nature of collaboration. Die *Natur der Kollaboration* wird in explizite und implizite Zusammenarbeit der Crowd unterteilt. Bei impliziter Zusammenarbeit erledigt das System die Zusammenarbeit der Crowd. Dieses können sowohl Huckepack-Systeme, wie die Google-Autovervollständigung, als auch selbständige Systeme, wie die Vergabe von Sternen auf der Amazon Produktbewertung, sein. Der Benutzer erledigt eine Tätigkeit, bezieht sich dabei aber nicht auf die Arbeit Anderer bzw. Vorhergegangener.
Explizite Zusammenarbeit findet man hingegen nur bei selbständigen Systemen. Hier arbeiten die Benutzer evtl. abwechselnd zusammen und beziehen sich auf vorherige bzw. andere Ergebnisse oder Arbeitsschritte. Als Beispiel wäre hier wieder das Verfassen oder Verbessern eines Artikels auf Wikipedia zu erwähnen.

Type of target problem. Die *Art des Problems bzw. der Aufgabe* beschreibt, was das Ziel des Crowdsourcing-Systems ist und somit auch die Handlungen der Benutzer. [Doa11] unterscheiden *Bewerten, Teilen, Netzwerken, Artefakte bauen* und *Aufgaben erledigen.*
Beim Bewerten geht es sowohl darum aus möglichst verschiedenen Meinung einen Durchschnitt zu generieren, z.B. Vergabe von Sternen auf der Amazon Produktbewertung, als auch ausführliche

Einzelmeinungen zu sammeln, z.B. eine Rezension in der Amazon Produktbewertung schreiben, und evtl. sogar Diskussionen von Benutzern über ein Thema zu entfachen.
Teilen hingegen beschreibt Systeme auf denen Benutzer „Gegenstände“ teilen können. Dies könnten zum Beispiel Filme aber auch aktuelle Nachrichten, wie auf den Internetplattform www.youtube.com und www.twitter.com, sein.
Netzwerk-Systeme, wie z.B. www.facebook.com, erlauben es Nutzern große soziale Graphen zu erzeugen, in dem die Benutzer mit der Zeit Knoten (Benutzerkonten) und Kanten (wer ist mit wem befreundet, verwandt, etc.) hinzufügen.
Artefakt bauende Systeme lassen ihre Crowd Inhalt oder Komponenten erzeugen. In möglichen weiteren Iterationen können die Benutzer dann diese erstellten Komponenten bzw. den erzeugten Inhalt begutachten und Verbesserungen vornehmen. Wikipedia wäre ein Beispiel für mehrere Iterationen, da der Inhalt von zukünftigen Benutzern immer wieder überarbeitet wird. reCAPTCHA hingegen ist ein Beispiel mit nur einer Iteration. Sollten nach dieser Fehler vorhanden sein, werden diese von einer zweiten Instanz, den Kontrolllesern, behoben.
Als letztes bleiben uns die Aufgaben erledigenden Systeme. Diese sind den Artefakt bauenden System sehr ähnlich, nur dass hierbei kein greifbares Ergebnis entsteht bzw. die zu erledigenden Aufgaben evtl. nicht direkt zum Ergebnis beitragen. Z.B. auf der Suche nach dem verschollenen Informatiker Jim Gray hätte nur eine einzige Tätigkeit die Lösung erbracht und alle anderen wären „umsonst“ gewesen. Ebenso ist das Auffinden eines verschollenen Bootes offensichtlich kein Artefakt.

Als nächstes folgen nun vier Eigenschaften, welche gleichzeitig die nach [Doa11] vier zentralen Herausforderungen an Crowdsourcing-Systeme formulieren.

How to recruit and retain users? Zwei der wesentlichen Aufgaben, die ein Crowdsourcing-System zu lösen hat, sind *Benutzerrekrutierung* und *Benutzererhaltung.*
Die ersten drei der insgesamt fünf Möglichkeiten zur Benutzerrekrutierung, welche [Doa11] unterscheiden, eignen sich eher für selbständige Systeme. Zunächst kann man, sollte man über die entsprechende Autorität verfügen, Benutzer verpflichten am System teilzunehmen. Der Vorstand einer Firma könnte zum Beispiel alle oder einen Großteil seiner Mitarbeiter verpflichten an einem Bewertungssystem teilzunehmen. Dies wirft allerdings indirekt Kosten auf, da die Mitarbeiter in der Zeit, während sie als Crowd fungieren, nicht arbeiten. Dies führt uns direkt zur zweiten Möglichkeit, Benutzer zu bezahlen. Amazons Mechanical-Turk ist ein Paradebeispiel für diese Art der Benutzerrekrutierung.
Drittens kann man um freiwillige Mitarbeit bitten. Dies mag zunächst recht utopisch klingen, aber die Beispiele Wikipedia und Linux zeigen, dass auch ein solcher Ansatz von Erfolg gekrönt sein kann.
Im Fall von Huckepack-Systemen ist die Frage der Benutzerrekrutierung meist einfacher beantwortet. Zum einen kann man Benutzer für einen Service, welcher nicht die Benutzung des Crowdsourcing-Systems ist, bezahlen lassen. Die Bezahlung ist die Teilnahme an dem Crowdsourcing-System. reCAPTCHA benutzt eine solche Methode, indem Benutzer, die eigentlich einen anderen Service in Anspruch nehmen möchten, zunächst mit dem Lösen eines reCAPTCHAs „bezahlen“ müssen. Für diese Methode sei erwähnt, dass eine solche Bezahlung sich möglichst simpel halten sollte, damit ein Benutzer

durch die Bezahlung nicht abgeneigt wird den ursprünglichen Service zu benutzen.
Zum anderen kann man auf das Benutzerverhalten aufsetzen und hieraus seine Daten bzw. Handlungen für das Crowdsourcing-System gewinnen. Die Google-Autovervollständigung benutzt diese Methode, und wie schon früher erwähnt merken die Benutzer bei dieser Methode meistens nicht, dass sie während der Ausführung einer Tätigkeit gleichzeitig für ein Crowdsourcing-System rekrutiert wurden und arbeiten.
Zur Benutzererhaltung gibt es ebenfalls fünf Möglichkeiten. [Doa11] unterscheiden „direkte Belohnung“, „unterhaltsame Erfahrung bzw. benötigter Service“, „Entstehung, Ermessen und Darstellung von Berühmtheit, Ruf und Vertrauen“, „teilhabender Besitz des Ergebnisses“ und „Wettkampf“. Diese Methoden sind relativ selbsterklärend und es wird an dieser Stelle nicht weiter auf sie eingegangen.

What and how can users make contributions? Da die Art und Weise *wie und welche Beiträge Benutzer leisten können* in Crowdsourcing-Systemen eine extreme Vielfalt annehmen kann, wird diese Herausforderung von [Doa11] in vier einfachere Fragen unterteilt.
Wie kognitiv anspruchsvoll ist es einen Beitrag für ein Crowdsourcing-System zu leisten? Die Antwort auf diese Frage beeinflusst natürlich stark die Größe der möglichen Crowd aber auch die Qualität des Ergebnisses. Sind die Aufgaben zu anspruchsvoll, erhält ein System vielleicht nicht genug Beiträge um erfolgreich zu sein. Sind die Aufgaben hingegen zu simpel, könnten nicht qualifizierte Teilnehmer das Ergebnis verschlechtern oder sogar verfälschen.
Der Designer eines Crowdsourcing-System ist hier also gefordert den richtigen Mittelweg zu finden und vielleicht sogar verschiedenen Benutzer verschiedene Aufgaben bzw. Aufgabenteile zuzuweisen. Ein neuer Benutzer bekommt vielleicht zunächst nur simplere Aufgaben. Nachdem er diese „erfolgreich“ bearbeitet hat und damit seine Qualifikation unter Beweis gestellt hat, erhält er komplexere Aufgaben.
Welchen *Grad von Auswirkung hat ein Beitrag?* Dies kann je nach Aufgabe sehr stark variieren. Bei der Implementierung eines Systemkerns bei der Opensource-Entwicklung eines Betriebssystems wird jeder noch so kleine Beitrag eine enorme Auswirkung auf das Gesamtergebnis haben. Das Erstellen einer Schafzeichnung auf Amazons Mechanical-Turk für www.thesheepmarket.com (siehe Anhang Seite: II) hat hingegen einen sehr kleinen Anteil am Gesamtergebnis. Ebenso kann der Einfluss der Beiträge sich über die Zeit verändern. Die ersten Sternevergaben auf Amazons Produktbewertung werden die durchschnittliche Bewertung sehr stark beeinflussen, spätere werden statistisch weniger Einfluss nehmen.
Der Designer eines Crowdsourcing-Systems muss also darauf achten einem einzelnen Benutzer nicht zu viel „Macht“ zu verleihen, aber ebenso sollte ein Benutzer nicht das Gefühl bekommen sein Beitrag sei unnütz. Dieser Punkt muss auch bei der Aufgabenverteilung, sollte man verschiedene Aufgaben haben, berücksichtigt werden. Einige Aufgaben haben per Definition eine höhere Auswirkung auf das Ergebnis. Hier kann eine Nutzerhierarchie wieder von Vorteil sein.
Was sind die Beiträge des Systems? Strebt man eine Problemlösung mit einem Crowdsourcing-System an, ist es im Interesse des Crowdsourcers sowohl die Gesamtarbeit der Crowd als auch die Arbeit des einzelnen Individuums der Crowd möglichst gering zu halten. Denn entweder ist diese Arbeit begrenzt, z.B. durch die Motivation der Crowd, oder mit steigender Arbeit steigen auch die Kosten für den Crowdsourcer, welcher Art diese auch sein mögen. Es empfiehlt sich also die Crowd Aufgaben erledigen

zu lassen, deren Schwierigkeitsgrad für Menschen wesentlich niedriger ist als für einen Computer, wie zum Beispiel das Erkennen eines schlecht lesbaren Wortes. Andersrum sollten Aufgaben, welche für Computer leicht sind und für Menschen schwer oder aufwändig, natürlich auch von Maschinen erledigt werden.

Wie simpel kann das Benutzerinterface sein? Ein einfaches Benutzerinterface erleichtert es dem Benutzer Beiträge zu leisten. Leider ist die Einfachheit eines solchen Interface oft durch die Aufgabe begrenzt. Computercode zu schreiben funktioniert nun mal nicht mit natürlicher Sprache. Einen Artikel auf Wikipedia zu verfassen funktioniert so schon eher, obwohl hier auch wieder die Frage der Formatierung bleibt. Nutzt man hier Knöpfe und Dialoge, wie zum Beispiel in einem Textverarbeitungsprogramm, oder lieber Sonderzeichen bzw. -befehle, z.B. HTML.

Eine ganzzahlige Bewertung zwischen 1 und 5 Sternen ist hingegen so simpel, dass es wahrscheinlich mit einem Mausklick zu erledigen ist.

How to combine user contributions? Die *Zusammenführung von Beiträgen* kann ebenfalls von extrem simpel bis nahezu unmöglich reichen. Zum Beispiel können die verschiedenen Sternvergaben auf Amazons Produktbewertung durch ein einfaches arithmetisches Mittel zu einer Durchschnittsbewertung zusammengeführt werden. Hingegen scheint die Aufgabe geschriebene Nutzer-Rezensionen in derselben Produktbewertung zusammenzufassen schier unmöglich, insbesondere für Computer.

Beim Zusammenführen von Fakten kann es hier auch zu einem *Benutzer-Maschinen-Konflikt* kommen. [Wel08] beschreiben mit diesem Begriff den Konflikt, wenn die von einem Benutzer eingegebenen Daten bzw. Fakten denen von einem Computer bzw. einem Algorithmus ermittelten widersprechen. Nehmen wir an ein, das Internet durchsuchender, Algorithmus von Wikipedia ermittelt die Einwohnerzahl einer Stadt auf 10.000. Später korrigiert ein Benutzer von Wikipedia diese Zahl auf 17.000. Daraufhin findet der Algorithmus weitere Belege für seine zunächst ermittelte Zahl von 10.000. Soll der Algorithmus nun die vom Benutzer eingebrachte Veränderung überschreiben? Eine heutzutage gängige Methode ist es in einem solchen Fall einen höher qualifizierten Benutzer oder einen Administrator von dem Algorithmus informieren zu lassen. Dieser entscheidet dann den Benutzer-Maschinen-Konflikt.

Es ist also ratsam sich schon vor der Entwicklung eines Crowdsourcing-Systems genaue Gedanken zu machen, wie man Beiträge verschiedener Art zusammenführen kann, welche Teile dieser Arbeit man automatisieren kann und was manuell zusammengeführt werden muss, evtl. wieder per Crowdsourcing.

How to evaluate users and contributions? Bevor man Beiträge überhaupt zusammenführt, möchte man evtl. zunächst eine *Beitragsbewertung* durchführen. Ebenso benötigt man eine *Benutzerbewertung*, wenn man eine weiter oben angesprochene Benutzerklassifizierung einsetzen möchte, um nur qualifizierten Benutzern komplexere Aufgaben oder Aufgaben mit hoher Auswirkung zuzusprechen. Diese Bewertungen sind natürlich eng miteinander verbunden. Ein Benutzer mit schlechten Beiträgen erhält eine schlechtere Benutzerbewertung und andersherum.

Diese Bewertungen können sowohl positiv als auch negativ durchgeführt werden. Negative Bewertungen befassen sich eher mit der Suche und Aufdeckung böswilliger Benutzer, welche aktiv versuchen falsche Antworten bzw. schlechte Beiträge ins System einzuschleusen bzw. das System auf diese Weise zu

manipulieren. Mit diesem Thema werde ich mich in Kapitel 3 genauer beschäftigen.
Positive Bewertungen versuchen zwischen legitimen Beiträgen verschiedener Benutzer die besten herauszufiltern bzw. zu priorisieren. Dies kann sowohl automatisch als auch per Crowdsourcing passieren. Zum Beispiel können die Benutzer in der Amazon Produktbewertung die geschriebenen Rezensionen als „hilfreich“ oder nicht „hilfreich einstufen“. Auch auf positive Bewertungsfindung wird in Kapitel 3 eingegangen.

2.4 Eigenschaften des OCCT

In den drei vorherigen Teilkapiteln haben wir gelernt was Crowdsourcing ist, uns einige Beispiel näher angesehen und schließlich die verschiedenen Eigenschaften und Herausforderungen von und an Crowdsourcing-Systeme durchleuchtet. In diesem letzten Teilkapitel von Kapitel 2 widme ich mich nun wieder dem OCCT (siehe Kapitel 1.2 „Was ist ein „Online Crowdsourcing Calendar Tool“?“). Ich werde ermitteln welches die Crowdsourcing-Anteile am OCCT sind, die Crowdsourcing-Eigenschaften des OCCT definieren und Antworten auf die Herausforderungen an das OCCT geben.

Crowdsourcing-Anteile des OCCT. Offensichtlich gibt es am OCCT drei Bereiche, deren Inhalt komplett durch Crowdsourcing erreicht wird. Als erstes wäre die Erstellung von Terminen zu nennen. Diese kann zwar durch eine Administration oder sogar durch automatisierte Algorithmen unterstützt werden, aber die Kernidee des OCCT ist, dass die Benutzer die Termine selbst erstellen und verwalten können.
Als zweites kommt die Bewertung dieser Termine. Hier kommt ausschließlich die Crowd zum Tragen. Die Idee ist, dass nachdem ein Termin bzw. Event vollendet ist, die Benutzer eine Bewertung abgeben können. Diese Bewertung dient weiteren Benutzern zur Orientierung für weitere Termine bzw. Events gleicher Art.
Zuletzt gibt es die TAG-Erstellung für Termine. Hier existiert zwar auch wieder die Möglichkeit, dass zunächst eine Menge von TAGs vom Ersteller des Termins vordefiniert wird, aber jeder Benutzer wird dann die Möglichkeit haben den TAG, welchen er für diesen Termin als passend befindet, auszuwählen oder der Menge von TAGs einen eigenen TAG hinzuzufügen. Wie bereits in Kapitel 1 erwähnt, werden die beliebtesten TAGs pro Termin dann angezeigt.

Crowdsourcing-Eigenschaften des OCCT. Betrachten wir zunächst den Anteil an manuellem Aufwand. Für die Terminerstellung, welche nur von registrierten Benutzern mit einem Benutzerkonto genutzt werden kann, ist geplant, dass diese nahezu komplett manuell zu erledigen ist. Ein Benutzer legt einen Termin für ein Datum mit einem Titel und einer Beschreibung an. Optional ist das Einfügen eines Bildes für den Termin, die Anzeige einer URL und das Erstellen einer Menge von TAGs. Die einzige automatisierte Funktion wird die Möglichkeit sein wöchentliche bzw. sich wiederholende Termine zu erstellen. In unserem Beispiel des Studentenwohnheimes könnte zum Beispiel jeden Dienstagabend ein Pokerturnier stattfinden. Der Ersteller dieses Termins könnte dann dieses Turnier für den ersten Dienstag

eintragen und danach den Termin als wöchentlich deklarieren. Das System sollte dann automatisch jeden Dienstag diesen Termin eintragen. Ebenso sollte die TAG-Menge von Termin zu Termin transferiert werden. Für Wiederholung von Terminen sollten Benutzer eine Auswahl von ihren alten Terminen erhalten. So können die Daten von einem alten Termin direkt auf einen neuen transferiert werden.
Die Bewertung der Termine erfolgt ebenfalls manuell durch die Benutzer (sowohl registrierte als auch unregistrierte). Die Berechnung der durchschnittlichen Bewertung erfolgt komplett automatisch. Hier läge natürlich ein einfaches arithmetisches Mittel nahe, aber ich werde eine etwas komplexere Lösung anstreben, wie weiter unten erklärt wird.
Bis auf den Transfer der TAG-Menge für sich wiederholende Termine soll die Erstellung dieser Menge auch komplett manuell erfolgen. TAGs vergeben bzw. erstellen wird wieder nur registrierten Benutzern möglich sein.
Die Rollen der menschlichen Benutzer des OCCT sind schnell definiert. Ein unregistrierter Benutzer ist sowohl Sklave als auch Perspektiven-Provider. Seine einzige Aufgabe ist die Vergabe von Bewertungen, welche wie bereits erwähnt zwischen 1 und 5 liegen können. Der registrierte Benutzer hingegen nimmt zusätzlich die Rolle des Inhalt-Providers ein. Er ist dafür zuständig Termine anzulegen und TAGs zu vergeben bzw. zu erstellen. Sieht man eine Highlight-Liste der beliebtesten 5 Termine für ein Datum auch als Ergebnis des OCCT an, so könnte man den Terminersteller auch als Komponenten-Provider ansehen.
Das OCCT sollte offensichtlich ein selbständiges System sein. Auch wenn es sich dazu eignet, in eine bestehende Gemeinschaft aufgenommen zu werden, wird es nur sehr schwer bzw. selten möglich sein aus bereits existierenden Daten und anderen Benutzereingaben den Inhalt des OCCT zu generieren.
Ebenso auf der Hand liegen die zwei Arten von Zielproblemen des OCCT. Als erste Art ist Teilen zu nennen. Die Erstellung eines Termins fällt definitiv in diese Kategorie, da es im Sinne des Erstellers ist, die Information über diesen Termin bzw. Event der großen Masse mitzuteilen. Bewerten und, meiner Ansicht nach, auch die Vergabe bzw. Erstellung von TAGs fällt in die Kategorie Bewertung.
Die Natur der Kollaboration sehe ich zum großen Teil als implizit. Auch wenn andere Benutzer ihre Arbeit als Teil der Crowd basierend auf erstellten Terminen erledigen, sehe ich dies noch nicht als explizite Zusammenarbeit, weil die „Ergebnisse“ der Terminersteller nicht verändert bzw. nicht darauf basierend neue Termine erstellt werden.

Herausforderungen an das OCCT. Die Antwort auf die erste Herausforderung, Benutzerrekrutierung und Benutzererhaltung, liegt im Grund nicht in unserem Einflussbereich. Das OCCT ist geplant als ein Tool, welches man weiteren Parteien bzw. Interessengemeinschaften zur Verfügung stellt, um per Crowdsourcing Termine und Events zu einem bestimmten Thema zu koordinieren. Die Motivation Benutzer, welche nicht von vornherein Teil dieser Gruppe oder Gemeinschaft sind, für ein solches System zu gewinnen liegt im Aufgabenbereich genau dieser Gruppe bzw. Gemeinschaft.
Zur Erhaltung können wir abschätzen, dass die Erstellung bereits existenter Termine oder Events durch jene Partei oder Gemeinschaft im OCCT dazu führen könnte die Bekanntschaft und Wichtigkeit des OCCT auf ein bestimmtes Thema zu steigern. So wird der OCCT zu einer unterhaltsamen Erfahrung für den Informationssuchenden evtl. sogar zu einem benötigten Service um überhaupt Termine für ein bestimmtes Thema zu finden. Nutzen genug Informationssuchende das OCCT wird es zu einem Tool zum

Entstehen, Ermessen, und Darstellen von Berühmtheit und Ruf. Ist der Einfluss des OCCT erst mal groß genug in einem bestimmten Themenbereich, so könnte sogar ein Wettkampf zwischen den Terminerstellern entstehen.

Der kognitive Anspruch an die Benutzer des OCCT ist trivial. Mit der einzigen Ausnahme der TAG-Vergabe. Dies könnte in gewissen Themengebieten etwas schwieriger werden, aber selbst wenn dies der Fall sein sollte, ist die grundlegende Funktion des Systems nicht auf erfolgreiche TAG-Zuweisung angewiesen. Die aus Beiträgen – Terminerstellung, Bewertung und TAG-Vergabe – resultierende Auswirkung ist schon wesentlich interessanter. Die Auswirkung der Bewertung wird wie schon angedeutet kein arithmetisches Mittel sein. Hierauf wird später genauer eingegangen. Die TAG-Zuweisung wird recht linear sein, da einfach die beliebtesten TAGs für jeden Termin angezeigt werden. Der Einfluss einer jeden Stimme hängt hier natürlich von der aktuellen Stimmanzahl ab. Der Einfluss einer Terminerstellung ist ähnlich. Sind zu wenige Termine im OCCT vorhanden, könnten vermeintlich „schlechte“ Termine zu hohe Anzeigepriorität haben und somit stark das Gesamtbild des OCCT prägen. Es empfiehlt sich also wieder einige bereits existente Termine durch die Administration einzupflegen.

Einige Beiträge des Systems sind offensichtlich. Die Berechnung der Durchschnittsbewertungen, die Ermittlung der anzuzeigenden TAGs und natürlich die Priorisierung der Termine für dasselbe Datum. Aber unter der Oberfläche hat das System noch deutlich weitere Aufgaben. Das wichtigste ist wohl das Aufspüren von *Opinion-Spammern* und anderen Manipulationsversuchen. Hierauf werde ich in Kapitel 3 eingehen. Ebenso muss das System die Benutzerkonten verwalten und in der Lage sein „tote“ sich wiederholende Termine aufzuspüren. Zum Letzteren könnte aber auch wieder Input der Crowd genutzt werden bzw. dem Ersteller eine Bestätigungspflicht für jeden einzelnen Termin auferlegt werden.

Die Benutzeroberfläche kann sehr einfach gehalten sein. Die Funktionen Bewertung und TAG-Vergabe können komplett mit Mausklicks erledigt werden. Benutzerkonten, Termine und TAGs erstellen sind alle mit einfachen Eingabemasken zu bewältigen. Die letztliche Anzeige der Termine, ob nun in Wochen- oder Monatsübersicht oder in den Highlights des Tages, liegt in der Hand des Oberflächendesigners. Aber auch hier sollte es möglich sein sehr übersichtliche und einfache Interfaces zu gestalten.

Die einzelnen Termine werden offensichtlich nicht zusammengeführt werden. Die vergebenen TAGs werden wie weiter oben beschrieben einfach linear zusammen geführt und die Beliebtesten angezeigt. Ich habe mir überlegt, für einen Termin und natürlich für seine Wiederholungen jeden TAG anzuzeigen, welcher über 10% der aktuellen Stimmen erhalten hat. Dies sorgt dafür, dass ein Termin nicht mit zu vielen TAGs überflutet wird (es werden maximal 9 angezeigt) und sollte ein TAG stark dominieren, vermindert dies gleichzeitig die Gesamtzahl an angezeigten TAGs. Denkbar wäre an der Oberfläche die Größe der Anzeige von TAGs proportional zu ihren erhaltenen Stimmen zu gestalten. Natürlich kann die Zahl von 10% beliebig verändert werden, je nach Belieben und Erfahrung.

Die Zusammenführung der Bewertungen ist schon etwas komplexer. Jede Bewertung erhält ein Gewicht. Dies reicht von 0 bis 10 und hat maximal eine Dezimalstelle. Das Gewicht ist das aktuelle Vertrauenslevel des Benutzers, der diese Bewertung abgegeben hat. Das Vertrauenslevel wird später diskutiert.

Sei $w_i \in \{0{,}1\,;\,0{,}2\,;\,0{,}3\,;\,...9{,}8\,;\,9{,}9\,;\,10\}$ das Gewicht und $r_i \in \{1\,;\,2\,;\,3\,;\,4\,;\,5\}$ die „Punktzahl“ der i-ten Bewertung eines Termins t mit k Bewertungen, so ergibt sich für die ungerundete Durchschnittsbewertung D' für den Termin:

$$D'(t) = \frac{\sum_{i=1}^{k} w_i \cdot r_i}{\sum_{i=1}^{k} w_i}$$

Dieser Wert wird nun noch auf eine Nachkommastelle gerundet und wir erhalten die finale Durchschnittsbewertung D:

$$D(t) = \frac{\lfloor 10 * D'(t) \rfloor}{10}$$

Auf warum, sowohl Vertrauenslevel als auch die durchschnittliche Bewertung, auf eine Nachkommastelle begrenzt sein sollten wird in Kapitel 3.4 eingegangen.
Die Bewertung eines Termins wird bei einem nächsten Termin der gleichen Art wieder angezeigt. Die Bewertung des nächsten Termins erfolgt natürlich unabhängig. Es wäre auch denkbar eine Durchschnittsbewertung aller Termine der gleichen Art zu berechnen. Dies wäre dann allerdings ein arithmetisches Mittel.
Natürlich stellt sich nun die Frage, wie ich das Vertrauenslevel eines Benutzer berechne. Grob gesagt hängt dies von den bisherigen „Leistungen“ und „Verstößen“ des Benutzers im System ab. Genauer wird dies in Kapitel 3 beschrieben, wo es auch um die Abwehr von Manipulationen in diesem System geht. Natürlich können nur registrierte Benutzer ein Vertrauenslevel haben. Bewertungen von unregistrierten Benutzern werden einfach mit einem Gewicht von 1 eingerechnet.

Dies deckt grundlegend die Crowdsourcing-Bereiche und -Eigenschaften des OCCT ab. Ich habe geklärt wo genau Crowdsourcing im OCCT passiert, definiert wer was wann darf, einige Entscheidungshilfen gegeben und wie die verschiedenen Ergebnisse zusammengefasst werden sollen.

Kapitel 3

Manipulation von Crowdsourcing-Systemen

In diesem Kapitel setze ich mich mit der Manipulation von Crowdsourcing-Systemen auseinander. Mit der Motivation die dahinter steht, aber vor allem auch mit den verschiedenen Möglichkeiten solche Manipulationen zu entdecken und ein System dagegen zu schützen. Wir werden verschiedene Arten der Manipulation kennenlernen, wie zum Beispiel *Opinion-Spam* oder *Sybil-Angriff*.
Zuletzt werde ich überlegen welche dieser Möglichkeiten für das OCCT Anwendung finden und die gewünschten Methoden zur Absicherung gegen Manipulation modellieren.

3.1 Warum überhaupt Manipulation von Crowdsourcing-Systemen?

Grundlegend gibt es zwei Motivationen Crowdsourcing-Systeme zu manipulieren. Erstens um die eigene Lage, das eigene bewertbare Objekt, die Sicht auf das eigene bewertbare Objekt oder den eigenen Stellenwert zu verbessern, oder zweitens einen dieser Punkte bei der Konkurrenz zu verschlechtern. Es könnte zwar auch sein, dass jemand versucht aus purer Böswilligkeit ein Crowdsourcing-System zu manipulieren, aber dies werde ich hier ignorieren, da die Manipulation zum eigenen Vorteil bzw. zum Nachteil des Anderen meiner Meinung nach eine deutlich wesentlichere Rolle spielt.
Betrachten wir einige Beispiele: Eine Firma für Betriebssysteme könnte z.B. daran interessiert sein die Crowdsourcing-Entwicklung eines Betriebssystems der Konkurrenz zu behindern und absichtlich probieren schlechte Beiträge einzuschleusen. Eine Person könnte aus persönlichem Groll eine Diffamierung über eine andere Person auf Wikipedia veröffentlichen, ich erinnere an das Beispiel von Mr. Seigenthaler. Am offensichtlichsten ist aber der Versuch Bewertungssysteme zu manipulieren. Hier liegt der Nutzen klar auf der Hand. Man kann das eigene Produkt ungerechtfertigt gut bewerten und damit Kunden zum Kauf verleiten. Oder man kann die Produkte der Konkurrenz ebenso ungerechtfertigt schlecht bewerten oder sogar in einer Rezension diffamieren, um somit Kunden vom Kauf dieser Produkte abzuhalten. Gerade da bis zu 15% zukünftiger Rezensionen von vorhergegangenen beeinflusst werden (vergleiche [Jin08]).
Solche Manipulation ist gefährlich da Kunden sich „betrogen“ fühlen können und so von weiterer Nutzung des Crowdsourcing-Systems absehen könnten. Des Weiteren können solche Versuche nach [Cox11] von sehr schwachen Angreifern ausgeführt werden und sind schwierig zu entdecken. Zwar schreibt [Cox11]:

„To this point, attempts to pollute user-contributed data have been rare,
but this seems unlikely to remain true for long.“

Aber meines Erachtens empfiehlt es sich jetzt schon, sich mit der Sicherheit von Crowdsourcing-Systemen gegen solche Manipulationen zu beschäftigen.
Es gibt aber noch mehr Gründe sich damit zu beschäftigen. Die Anzahl an Veröffentlichungen zu diesem Thema ist in den letzten 5 Jahren drastisch gestiegen, viele beschäftigen sich mit der Produktbewertung auf Amazon. [Lim10] nennen Zahlen von 10-20% gefälschten Bewertungen und Rezensionen auf Amazon. Während 3 Jahre früher [Jin07] noch von 4-5% sprechen, aber eine große Dunkelziffer eingestehen.

3.2 Arten der Manipulation

[Jin08] unterscheiden zunächst zwei **Arten von Manipulation**. *Opinion-Spam* und *Review-Spam*. Opinion-Spam bezieht sich auf falsche Ansichten bzw. Meinungen. D.h. dass man bewusst etwas Falsches im Internet oder einem Crowdsourcing-System veröffentlicht. Dieser Opinion-Spam wirkt allerdings zunächst qualitativ recht hochwertig, da er mit viel Aufwand geschrieben bzw. erstellt wird. Er ist somit am schwierigsten zu entdecken.
Review-Spam hingegen beschreibt die Tätigkeit ein und dieselbe Meinung, egal wie nieder qualitativ sie ist, immer wieder ins System einzubringen. Dies ist deutlich leichter zu entdecken. Review-Spam dient dazu ein Bewertungssystem wie die Amazon Sternvergabe zu manipulieren. Opinion-Spam eignet sich eher für Systeme wie das Schreiben von Rezensionen.
Da das OCCT keine textuell verfassten Bewertungen enthält, sondern nur einfache ganzzahlige Bewertungen zwischen 1 und 5, werde ich mich hier nur mit Review-Spam weitergehend beschäftigen.

[Jin08] unterscheiden 3 **Kategorien von Review-Spam.** Als erstes nennen sie „*untruthful opinions*“, was so viel bedeutet wie „unwahrheitsgemäße Meinung“. Diese Bewertungen verbreiten gezielt Missinformationen um Konsumenten des Bewertungssystems in die Irre zu führen. Es wird zwischen *Hyper-Spam* und *Defaming-Spam* unterschieden. Hyper-Spam beschreibt die Überbewertung eines eigenen Produkts um es besser erscheinen zu lassen, als es eigentlich ist. Defaming-Spam bedeutet hingegen das Gegenteil, nämlich ein Produkt der Konkurrenz schlechter darzustellen als es ist, es zu diffamieren.
Die zweite Kategorie ist *„reviews on brands only“*. Hiermit ist gemeint, dass ein Bewerter ein Produkt auf Grund seines Herstellers bzw. Erstellers bewertet und nicht das Produkt selber. Man stelle sich vor, ein langjähriger Apple-Kunde kauft sich das neuste iPhone und schreibt daraufhin eine Rezension auf Amazon, in der er die allgemeinen Qualitäten von Apple lobt, sich aber nicht auf das eigentliche Produkt bezieht. Solche Rezensionen und Bewertungen werden als Spam erachtet, da sie sich nicht direkt auf das Produkt beziehen.
In die letzte Kategorie werden *„non-reviews“* gesteckt. Diese „nicht-Bewertungen“ sind eigentlich keine Bewertungen im eigentlichen Sinne, sondern beinhalten Werbung, Fragen oder anderen zusammenhangslosen Inhalt.

Es gibt natürlich verschiedene Möglichkeiten eine solche Manipulation zu erzielen. Eine der nennenswertesten Methoden ist der *Sybil-Angriff*. [Dou02] prägte diesen Begriff im Jahr 2002 und beschriebt hiermit das Prinzip, dass sich eine einzelne Person in der elektronischen Welt als mehrere Personen ausgibt. Im Beispiel Amazons Produktbewertung hieße dies, man erstelle sich mehrere Benutzerkonten. Mit diesen Konten schreibt man dann ähnliche Rezensionen, welche grundlegend das gleiche Ziel verfolgen, und gibt dazu entsprechende Bewertungen ab. Man könnte sogar so weit gehen auf die zunächst geschriebenen Bewertungen Bezug zu nehmen, diesen Recht geben und als „hilfreich“ markieren. Man könnte annehmen, dass so ein Angriff, insbesondere wenn er gut geplant ist, die Schwierigkeit der Entdeckung nochmals erheblich steigert. [Muk12] definieren dies als eine *„Reviewer Group“*, sogenannte „Bewerter-Gruppen“, und zeigen sogar, dass es einfacher ist eine solche Gruppe aufzuspüren, als einzelne Review-Spammer. Hierauf gehe ich später genauer ein.

Ist es in einem Bewertungssystem, wie dem OCCT, möglich anonym, d.h. ohne jegliche Anmeldung bzw. Registrierung, Bewertungen vorzunehmen, ist es nahezu unmöglich Review-Spam zu entdecken, da jeder Benutzer beliebig oft abstimmen kann. Dies definiere ich als *Anonymer-Spam***.**

3.3 Resistenz gegen Manipulation

Nach dem wir nun wissen, warum überhaupt Crowdsourcing-Systeme manipuliert werden und wie wir solche Manipulationen klassifizieren, will ich mich mit den Möglichkeiten auseinander setzen, solche Manipulationen aufzudecken.

Viele aktuelle Arbeiten ([Kim06], [Jin07] oder [Li11]) beschäftigen sich damit, lernfähige Algorithmen mit manuell als Review-Spam positiv oder negativ markierten Rezensionen zu trainieren und so andere Rezensionen auf Review-Spam zu überprüfen. Andere Veröffentlichungen([Lim10] und [Muk12]) beschäftigen sich hingegen damit auf arithmetische Art und Weise das Verhalten der Bewerter bzw. Bewerter-Gruppen hingehend auf Review-Spam zu untersuchen. Da wir im OCCT keinen textuellen Anteil an den Bewertungen haben, sondern nur die Vergabe eines „*Ratings“* zwischen 1 und 5, werden wir uns auf die Methoden von [Lim10] und [Muk12] konzentrieren.

Bewerterverhalten. Review-Spam auf Grund des Bewerterverhaltens zu entdecken wurde meines Wissens zuerst von [Lim10] untersucht. In ihrer Veröffentlichung stellen sie Ideen vor, wie man einen komplex zusammengesetzten *Spamming-Score* für einen Bewerter auf arithmetische Art und Weise errechnet. Dieser Spamming-Score wurde auf Grund von Amazons Bewertungsdaten erstellt und bezieht sich zu 75% nur auf die Sternvergabe. [Lim10] kommen zu dem Ergebnis, dass ihre Methode deutlich zuverlässiger ist um Review-Spam zu entdecken, als die von Amazon eingesetze. Wir erinnern uns, Amazon evaluiert Bewertungen durch Markierung als hilfreich bzw. nicht-hilfreich durch die Benutzer.
Im Folgenden werde ich die einzelnen Eigenschaften, welche [Lim10] beschreiben, und die zugehörigen Formeln erläutern, und entsprechend der Nutzung auf mein Problem anpassen.

Der Spamming-Score eines Bewerters setzt sich aus 4 Teilen zusammen. Der *Bewertungs-Spam-Score,* der *Hoch- und Niedrig-Bewertungs-Score,* der *Allgemeinen-Abweichung* und der *Frühen-Abweichung.* Bevor ich nun diese 4 Teile weiter durchleuchte, normalisieren wir zunächst alle Bewertungen auf das Intervall 0-1, damit die Ergebnisse auch für andere Bewertungsskalen als „1 bis 5“ benutzt werden können.
Hierzu werden wir unsere Bewertungen, welche ganzzahlige Werte von 1 bis 5 annehmen können, einfach -1 nehmen und dann durch 4 teilen. So liegen unsere Bewertungen e in der Menge $\{0;0{,}25;0{,}5;0{,}75;1\}$ Weiter sei E_u die Menge der normalisierten Bewertungen eines Benutzers u.

Der **Bewertungs-Spam-Score** basiert auf der Idee, dass Benutzer, welche das gleiche Objekt mehrmals bewerten eher dazu neigen Review-Spammer zu sein. Sollten zudem die meisten Bewertungen nicht besonders voneinander abweichen, so ist es umso wahrscheinlicher, dass es sich um einen Review-Spammer handelt. Zunächst will ich den Begriff der *Objekt-* oder *Produktgruppe* definieren. Eine Objektgruppe b_k ist eine Menge von verschiedenen bewertbaren Objekten, welche ein oder mehrere gemeinsame Attribute aufweisen, z.B. MP3-Player des gleichen Herstellers. Der Bewertungs-Spam-Score BSS eines Benutzers u berechnet sich für eine bestimmte Objektgruppe b_k, für die dieser Benutzer mindestens zwei Bewertungen abgegeben hat, wie folgt:

$$BSS_{b_k}(u) := \frac{s}{Max_{u' \in U}(s')}$$

Hierbei repräsentiert s den *Spammer-Anteil* des Benutzers u, s' den Spammer-Anteil des Benutzers u' und U die Menge aller Benutzer. $Max_{u' \in U} s'$ bezeichnet also offensichtlich den größten Spammer-Anteil bezüglich einer Objektgruppe. Definieren wir nun eine Ähnlichkeitsfunktion sim, welche angibt wie ähnlich sich die einzelnen Bewertungen e von einem Benutzer u sind, wie folgt:

$$sim(E_{uk}) := 1 - Avg_{e_i, e_j \in E_{uo}, i<j} |e_i - e_j|$$

E_{uk} sei hierbei die Menge aller Bewertungen des Benutzers u für die Objektgruppe b_k. Der Spammer-Anteil s berechnet sich nun wie folgt:

$$s := |E_{uk}|^2 \cdot sim(E_{uk})$$

Der gesamt Bewertungs-Spam-Score für einen Benutzer ergibt sich aus seinem schlechtesten Verhalten. Das heißt es wird das Maximum über alle Objektgruppen b_k gebildet. Sei B die Menge aller bewertbaren Objektgruppen so ergibt sich:

$$BSS(u) := Max_{b_k \in B}(BSS_{b_k}(u))$$

Somit habe ich den Bewertungs-Spam-Score für einen Benutzer eindeutig bestimmt und habe ein Maß, welches zwischen 0 und 1 liegt, um zu bestimmen wie hoch die Wahrscheinlichkeit ist, dass dieser Benutzer ein Review-Spammer ist, in Bezug auf die Review-Spammer-Eigenschaft: verschiedene Objekte der gleichen Objektgruppe mehrfach gleich zu bewerten.
Bezüglich des OCCT kann ich an dieser Stelle schon einmal vorweg nehmen, dass eine Objektgruppe aus sich wiederholenden Terminen bestehen wird.
Hat ein Benutzer keine zwei Objekte derselben Objektgruppe bewertet, erhält er per Definition einen Bewertungs-Spam-Score von 0.

Kommen wir zum **Hoch- und Niedrig-Bewertungs-Score**. Nach [Lim10] neigen Review-Spammer dazu in einem recht kurzen Zeitintervall einer gesamten Objektgruppe entweder sehr hohe oder sehr niedrige Bewertungen zu geben. Versuchte ein Review-Spammer zum Beispiel den Ruf der MP3-Player seiner Firma A zu verbessern und den Ruf der MP3-Player der Konkurrenz B zu verschlechtern, wäre ein logisches Verhalten in einer „Sitzung“ Objekten der ersten Gruppe eine 5-Sterne Bewertung zu geben und der zweiten Gruppe eine 1-Sterne Bewertung.
Die Herausforderung zur Berechnung dieses Scores ist natürlich die Länge des gewählten Zeitintervalls. Hierfür werde ich, bezüglich des OCCT, eine Dauer von einer Stunde annehmen. Sollte jemand mehrere vergangene Termine eines sich wiederholenden Termins innerhalb einer Stunde entweder komplett hoch oder komplett niedrig bewerten, so wirkt dies sehr verdächtig.
Als eine hohe Bewertung werde ich ausschließlich 5-Sterne Bewertungen ($e=1$) und als niedrige Bewertungen ausschließlich 1-Sterne Bewertungen ($e=0$) zulassen. Beschreibe nun w ein Zeitfenster und $t(e)$ den Zeitpunkt, an dem eine Bewertung vorgenommen wurde, so können wir die Menge der hohen Bewertungen $E_{uk}^{H}(w)$ in diesem Zeitfenster bezüglich eines Benutzers u und einer Objektgruppe b_k definieren.

$$E_{uk}^{H}(w) := \{ e_j \in E_u \mid o_j \in b_k \wedge t(e_j) \in w \wedge e_j = 1 \}$$

Natürlich sollten hier nur Mengen einer bestimmten Größe berücksichtigt werden. [Lim10] schlagen eine Mindestgröße von 3 vor. Mit Hilfe der Vereinigung dieser Menge an hohen Bewertungen eines Benutzers innerhalb eines Zeitfensters für eine Objektgruppe, über die Zeitfenster und Objektgruppen, können wir nun die Gesamtmenge C_u^H der hohen Bewertungen eines Benutzers beschreiben:

$$C_u^H := \cup_{k,w} \{ E_{uk}^{H} \mid |E_{uk}^{H}| \geq 3 \}$$

Letztlich können wir nun den Hoch-Bewertungs-Score HBS für einen Benutzer u definieren:

$$HBS(u) = \frac{|C_u^H|}{Max_{u' \in U} |C_{u'}^H|}$$

Wir nehmen die Anzahl an Bewertungen in der Gesamtmenge der hohen Bewertungen eines Benutzers und teilen sie durch $Max_{u' \in U} |C_{u'}^H|$, was natürlich hierbei wieder die höchste Anzahl einer Gesamtmenge der hohen Bewertungen über alle Benutzer beschreibt. Hat ein Benutzer niemals 3 hohe Bewertungen für dieselbe Objektgruppe innerhalb des Zeitfensters abgegeben, gilt offensichtlich $C_u^H = \emptyset$ und er erhält einen HBS von 0.

Der Niedrig-Bewertungs-Score folgt nun ganz analog. Wir definieren wieder zuerst die Menge der niedrigen Bewertungen E_{uk}^L eines Benutzers u für eine Objektgruppe b_k innerhalb eines Zeitfensters b_k :

$$E_{uk}^L := \{e_j \in E_u \mid o_j \in b_k \wedge t(e_j) \in w \wedge e_j = 0\}$$

Die Gesamtmenge der niedrigen Bewertungen eines Benutzers u bildet sich wieder über die Vereinigung über Zeitfenster:

$$C_u^L := \cup_{k,w} \{E_{uk}^L(w) \mid |E_{uk}^L(w)| \geq 2\}$$

Wir bemerken den Unterschied, dass für die Gesamtmenge der niedrigen Bewertungen nur eine Mindestgröße von 2 festgelegt wird. Dies liegt daran, dass schon kleinere Mengen an sehr negativen Bewertungen wesentlich auffälliger bzw. verdächtiger wirken, als gleichgroße Mengen sehr positiver Bewertungen. Diese These basiert darauf, dass es auf Amazon wesentlich mehr 5-Sterne Bewertungen gibt als 1-Sterne Bewertungen.
Für den Niedrig-Bewertungs-Score NBS eines Benutzers u ergibt sich also:

$$NBS(u) := \frac{|C_u^L|}{Max_{u' \in U} |C_{u'}^L|}$$

Mit dem Hoch-Bewertungs-Score und dem Niedrig-Bewertungs-Score können wir nun den gesuchten Hoch- und Niedrig-Bewertungs-Score $HNBS$ eines Benutzers u definieren, indem wir beide addieren und durch 2 teilen.

$$HNBS(u) := \frac{1}{2}(HBS(u) + NBS(u))$$

Die nächste Idee für einen Teil des Spamming-Scores ist die Überlegung, dass wenn eine Bewertung für ein bewertbares Objekt abgegeben wird, man bis zu einem gewissen Grad erwarten kann, dass diese in etwa der bisherigen Durchschnittsbewertung für dieses Objekt entspricht. Natürlich kann es immer vorkommen, dass ein Bewerter in seiner Meinung von der Allgemeinheit abweicht, aber bildet man den Durchschnitt über alle Abweichungen aller Bewertungen, die dieser Bewerter abgegeben hat, müsste er

schon eine sehr außergewöhnliche Weltanschauung haben um immer von der allgemeinen Meinung abzuweichen.

Hat also ein Bewerter im Durchschnitt eine sehr hohe Abweichung von der allgemeinen Meinung, liegt die Vermutung nahe, dass dieser Benutzer versucht die Bewertungen zu manipulieren und damit ein Review-Spammer ist. Dies beschreiben [Lim10] mit der **Allgemeinen-Abweichung**. Sei $D(o)$ also die aktuelle Durchschnittsbewertung (vergleiche Kapitel 2.4) normalisiert auf das Intervall [0;1] eines bewertbaren Objekts o, so definiert sich die Abweichung $d(e_o)$ einer Bewertung e_o für das Objekt o einfach als die Differenz von der Durchschnittsbewertung:

$$d(e_o) := e_o - D(o)$$

Das resultierende Vorzeichen spielt hierbei keine Rolle, da wir zur Berechnung der Allgemeinen-Abweichung AA eines Bewerters u den Durchschnitt über den Betrag aller Abweichungen dieses Bewerters bilden:

$$AA(u) := \frac{\sum_{e \in E_u} |d(e)|}{|E_u|}$$

Das letzte Merkmal des Spamming-Scores verfolgt eine ähnliche Idee, wie die der **Frühen-Abweichung**. Hier liegt das Verhalten von Review-Spammern zu Grunde, zu versuchen möglichst früh Einfluss auf eine Bewertung zu nehmen. So hat die Manipulation mehr Einfluss auf die Durchschnittsbewertung, insbesondere wenn wir, wie bereits erwähnt, auf die Tatsache Rücksicht nehmen, dass zukünftige Bewertungen von bisherigen Bewertungen, und damit der aktuellen Durchschnittsbewertung, beeinflusst werden.

Die Abweichung einer Bewertung e sei wieder analog definiert, wie auf der vorherigen Seite. Nun erhält aber jede Bewertung noch ein Gewicht $w(e)$. Dieses Gewicht berechnet sich aus dem Rang $r(e)$ der Bewertung. Der Rang ist die Anzahl der vorhergegangen Bewertungen plus eins. Die erste Bewertung für ein Objekt erhält also den Rang 1, die zweite Bewertung den Rang 2 und so weiter. Es ergibt sich also für w:

$$w(e) := \frac{1}{r(e)^{\alpha}} \quad , \alpha > 1$$

α ist hierbei ein Exponent größer 1 um das Abfallen der Kurve zu beschleunigen. Ich werde hier 1,1 benutzen, einen etwas kleineren Wert, als den von [Lim10] vorgeschlagenen (=1,5), da ich diesen Wert für größere Bewertungsmengen als geeigneter erachte, um die entstehende Gewichtungskurve weniger steil zu gestalten. Um die Frühe-Abweichung FH eines Benutzers u nun zu berechnen addieren wir wieder alle Beträge der Abweichungen der Bewertungen des Benutzers multipliziert mit den jeweiligen Gewichten auf und teilen dies wieder durch die Anzahl der Bewertungen:

$$FH(u) := \frac{\sum_{e \in E_u} (|d(e)| \cdot w(e))}{|E_u|}$$

Nun haben wir alle 4 Teile des **Spamming-Scores** SPS eines Bewerters u zusammen und es folgt:

$$SPS(u) := \frac{1}{2} BSS(u) + \frac{1}{4} HNBS(u) + \frac{1}{8} AA(u) + \frac{1}{8} FH(u)$$

Die Gewichtung der einzelnen Teile resultiert aus empirischen Daten, welche von [Lim10] ermittelt, aber nicht weiter erklärt wurden. Laut [Lim10] empfehle es sich mehr Gewicht auf objektgruppenspezifische Merkmale zu legen als auf objektspezifische. Ebenso seien Abweichungen die schwächsten Indizien für Review-Spam.
Der Spamming-Score liegt, wie alle seine einzelnen Teile, zwischen 0 und 1 und erlaubt es einem Crowdsourcing-System nun alle Benutzer in eine Reihenfolge zu bringen. Benutzer mit einem höheren Spamming-Score haben eine höhere Wahrscheinlichkeit Review-Spammer zu sein und liegen folglich weiter oben in dieser Reihenfolge. Dies erlaubt es dem System z.B. dann die oberen 10-20% an Bewertern als Review-Spammer einzustufen und ihre Bewertungen zu löschen oder Ähnliches. Wie genau wir diesen Spamming-Score für das OCCT benutzen werden, wird im nächsten Teilkapitel 3.4 behandelt.

Review-Spammer-Gruppen. Eine ähnliche, aber dennoch andere Methode zum Entdecken von Review-Spammern propagieren [Muk12]. Ihre Idee ist es nicht die einzelnen Benutzer eines Bewertungs-Systems unter die Lupe zu nehmen, sondern ganze Benutzergruppen, da es wahrscheinlich ist, dass sich zur Manipulation eines bewertbaren Objekts mehrere Benutzer zusammenschließen. Dieses Zusammenschließen dient zur Steigerung der Verschleierung der Manipulation, und es können so Hürden, wie z.B. nur eine erlaubte Bewertung pro Benutzer, umgangen werden. Ob eine Benutzergruppe in Wirklichkeit aus verschiedenen Menschen besteht oder es sich um einen Sybil-Angriff handelt spielt hierbei eigentlich keine Rolle. Die Idee basiert darauf, dass es sehr unwahrscheinlich ist, dass z.B. eine Gruppe aus 5 Benutzern 6 verschiedene Produkte exakt gleich bewertet.
Zuerst müssen für diesen Ansatz Bewerter-Gruppen identifiziert werden. [Muk12] nehmen hier folgende Minima an: Eine Gruppe besteht aus mindestens 2 Personen und muss mindestens 3 Objekte bewertet haben. D.h. dass jeder aus dieser Gruppe jedes der 3 Objekte bewertet haben muss.
Als nächstes werden dann, ähnlich wie im Spamming-Score-Ansatz bestimmte Eigenschaften der Gruppe bzw. der Bewertungen dieser Gruppe untersucht. Aus der Zusammenführung der verschiedenen Eigenschaften resultiert dann wieder ein gesamter Grupen-Spamming-Score, entsprechend der Wahrscheinlichkeit, ob es sich bei der Gruppe um Review-Spammer handelt oder nicht. Wie dieser schlussendlich mit dem Spamming-Score eines einzelnen Benutzers zusammengeführt wird ist im nächsten Teilkapitel zu finden. Eine weitere Besonderheit bei diesem Ansatz ist, dass es, falls eine Gruppe als Review-Spam-Gruppe klassifiziert werden sollte, wahrscheinlich ist, dass alle anderen Gruppen, welche mindestens ein Mitglied der Bewerter-Gruppe beinhalten, auch Review-Spam-Gruppen sind.

[Muk12] gehen zwar von Amazon-Bewertungen aus, welche aus einer Sternvergabe und einer Rezension bestehen, aber dennoch bin ich guter Dinge die Eigenschaften *Gruppen-Zeitfenster, Gruppen-Frühes-Zeitfenster, Gruppen-Abweichung, Gruppen-Größen-Verhältnis, Gruppen-Größe* und *Gruppen-Objekt-Anzahl* auf unsere Bewertungen, welche nur aus einer Zahlenbewertung ähnlich der Sternvergabe bestehen, anzuwenden. Es werden zwar noch weitere individuelle und Gruppen- Eigenschaften genannt, aber diese beziehen sich entweder auf textuellen Inhalt der Rezensionen oder überschneiden sich mit den bereits im Spamming-Score-Ansatz berücksichtigten Eigenschaften.

Das **Gruppen-Zeitfenster** erfasst die Tatsache, dass es wahrscheinlich ist, dass Gruppenmitglieder, welche für die Bewertung von Objekten zusammengearbeitet haben, dies innerhalb eines kurzen Zeitintervalls getan haben.
Sei O_g die Menge der von einer Gruppe g bewerteten Objekte. So wird als erstes das Gruppen-Zeitfenster GZF_o einer Gruppe g bezüglich eines bewertbaren Objekts $o \in O_g$ definiert. Hierzu überlegt man sich ein maximales Zeitfenster τ und gibt dann die zeitliche Differenz zwischen dem Zeitpunkt der frühesten Bewertung der Gruppe für dieses Objekt $F(g,o)$ und dem Zeitpunkt der letzten Bewertung $L(g,o)$ im Verhältnis zu τ an. Zieht man dieses Verhältnis von 1 ab, so erhält man einen Wert zwischen 0 und 1, mit 1 als schlechterem Wert, d.h. eher manipulatives Verhalten. Es sei noch erwähnt, sollte die Zeitdifferenz zwischen $F(g,o)$ und $L(g,o)$ größer sein als τ, so ist GZF_o per Definition gleich 0.

$$GZF_o(g,o) = 0 \quad , wenn \;\; L(g,o) - F(g,o) > \tau$$
$$GZF_o(g,o) = 1 - \frac{L(g,o) - F(g,o)}{\tau} \quad , sonst$$

Das gesamte Gruppen-Zeitfenster GZF ist nun gleich dem maximalen GZF_o über alle bewertbaren Objekte $o \in O_g$. Somit wird das schlechteste bzw. auffälligste Verhalten der Gruppe bewertet.

$$GZF(g) = Max_{o \in O_g}(GZF_o(g,o))$$

Bezüglich des OCCT lege ich zunächst ein Zeitfenster von 12 Stunden ($\tau = 12h$) fest. Dieser muss aber durch hinreichendes Experimentieren bestätigt bzw. modifiziert werden.

Auf ähnliche Weise berechnet, aber ein anderes Verhalten beschreibend, ist das **Gruppen-Frühe-Zeitfenster.** Diese Eigenschaft beschreibt, dass eine Gruppe von Review-Spammern versuchen könnte möglichst früh Einfluss auf eine Bewertung zu nehmen um somit zukünftige Bewertungen zu beeinflussen. Hierzu überlegt man sich wieder ein maximales Frühes-Zeitfenster β. o und $L(g,o)$ seien wie oben definiert und $A(o)$ sei der frühestmögliche Zeitpunkt, zu dem eine Bewertung abgegeben werden kann. $GFZF_o$ berechnet sich dann wie folgt:

$$GFZF_o(g,o)=0 \quad ,wenn \;\; L(g,o)-A(o)>\beta$$
$$GFZF_o(g,o)=1-\frac{L(g,o)-A(o)}{\beta} \quad ,sonst$$

Analog zum Gruppen-Zeitfenster folgt für das gesamte Gruppen-Frühes-Zeitfenster:

$$GFZF(g)=Max_{o\in O_g}(GFZF_o(g,o))$$

Hierfür lege ich für den OCCT ein Zeitfenster von 3 Stunden ($\beta=3h$) fest. Auch dieser Wert müsste wieder durch hinreichendes Experimentieren bestätigt bzw. modifiziert werden.

Den größten „Schaden“ durch Manipulation eines Bewertungssystems erzielt eine Gruppe von Review-Spammern, deren Bewertungen am stärksten von denen anderer normaler Benutzer abweichen. Die **Gruppen-Abweichung** versucht genau dieses Verhalten einzufangen. Hierzu wird wieder die Gruppen-Abweichung GA_o von einer Gruppe g bezüglich eines bewertbaren Objektes $o\in O_g$ definiert. Man bildet zunächst die Differenz zwischen der durchschnittlichen Bewertung der Gruppe für diese Objekt e_{go} und der durchschnittlichen Bewertung aller anderer Bewerter $\bar{e}_{go}$. Den Absolutwert dieser Differenz setzt man dann wieder in Verhältnis zur maximalen Abweichung(=1) und erhält einen Wert zwischen 0 und 1, welcher wieder mit 1 auffälligeres Verhalten beschreibt.

$$GA_o(g,o)=\frac{|e_{go}-\bar{e}_{go}|}{1}=|e_{go}-\bar{e}_{go}|$$

Die gesamte Gruppen-Abweichung GA ist nun wieder das Maximum, gebildet über alle von dieser Gruppe bewerteten Objekte:

$$GA(g)=Max_{o\in O_g}(GA_o(g,o))$$

Ein weiterer Indikator für Review-Spamming ist, wenn viele oder nahezu alle Bewertungen für ein Objekt von einer Gruppe stammen. Dies nennt man das **Gruppen-Größen-Verhältnis** und wird mit $GGV(g)$ beschrieben. Sei $M(o)$ die Menge aller Bewerter für ein Objekt, so berechnet sich das Gruppen-Größen-Verhältnis GGV_o bezüglich eines Objektes einfach linear mit:

$$GGV_o(g,o)=\frac{|g|}{|M(o)|}$$

Da es nicht unwahrscheinlich ist, dass die „Mitglieder“ einer Gruppe, dadurch dass sie gleiche Interessen verfolgen, zufällig die einzigen sind, die ein oder mehrere Produkte bewertet haben, wird hier nicht das Maximum, also das schlimmste Gruppenverhalten, sondern der Durchschnitt für das gesamt Gruppen-Größen-Verhältnis genommen:

$$GGV(g)=\frac{\sum_{o\in O} GGV_o(g,o)}{|O|}$$

Es liegt nahe sich auch über die eigentliche **Gruppen-Größe** Gedanken zu machen. Die Wahrscheinlichkeit, dass die Mitglieder einer Gruppe zufällig die gleichen Produkte bewertet haben ist offensichtlich antiproportional zur Gruppen-Größe. Dies allerdings auf $[0,1]$ zu normalisieren, stellt eine nicht zu unterschätzende Herausforderung dar. Der Einfachheit halber bedienen sich [Muk12] hier wieder der maximalen gefundenen Gruppen-Größe. $G\,G(g)$ für Gruppe $g\in G$ berechnet also wie folgt:

$$G\,G(g)=\frac{|g|}{Max_{g'\in G}(|g'|)}$$

Ebenso offensichtlich ist, dass es unwahrscheinlich ist, dass eine Gruppe bei steigender Objektanzahl zufällig zusammenarbeitet. Andersherum bedeutet dies, dass wenn eine Gruppe bei sehr vielen Objekten zusammengearbeitet hat, sie wahrscheinlich Review-Spam betreibt. Es wird also die **Gruppen-Objekt-Anzahl** als Spamming-Eigenschaft wie folgt definiert. GOA für eine Gruppe $g\in G$ bestimmt sich wieder aus der Anzahl der durch g bewerteten Objekte im Verhältnis zur größten Anzahl bewerteter Objekte irgendeiner Gruppe. Die Menge der von einer Gruppe bewerteten Objekte wird weiterhin mit O_g bezeichnet. Es ergibt sich:

$$GOA(g)=\frac{|O_g|}{Max_{g'\in G}(|O_g'|)}$$

Mit diesen 6 Spamming-Eigenschaften sind wir nun in der Lage den Spamming-Score einer Gruppe zu berechnen. [Muk12] benutzen diese und weitere Eigenschaften und etwas andere Spamming-Modelle zu beschreiben, auf die ich an dieser Stelle nicht weiter eingehen werde, da sie sich zu sehr auf das einzelne Gruppen-Objekt- bzw. Bewerter-Objekt-Verhältnis beziehen und weniger auf den gesamten Spamming-Score eines Bewerters. Folglich definiere ich den gesamten Spamming-Score einer Gruppe $GSPS(g)$ in dem ich jede der 6 Spamming-Eigenschaften für Gruppen gleich gewichte:

$$GSPS(g)=\frac{1}{6}\left(GZF(g)+GFZF(g)+GA(g)+GGV(g)+G\,G(g)+GOA(g)\right)$$

Eventuell wäre hier auch eine objektgruppenlastige Gewichtung wieder sinnvoll. Eine genaue Gewichtung sollte durch hinreichendes Experimentieren bestimmt werden.

Mit diesem Gruppen-Spamming-Score kann zusammen mit dem normalen Spamming-Score eines Bewerters der Gesamt-Spamming-Score global im System berechnet werden. Dies und wie das OCCT sich dies zu Nutze macht, behandle ich im nächsten Teilkapitel.

3.4 Nutzung des Spamming-Scores durch das OCCT

Wir erinnern uns an Kapitel 2.4, in dem ich zur Gewichtung einzelner Bewertungen das Vertrauenslevel eines Benutzers herangezogen habe. Die Frage, wie ein solches Vertrauenslevel berechnet wird, hatte ich auf später verschoben. Diese Frage soll nun unter der Verwendung des Spamming-Scores geklärt werden.

Es sei am Rande erwähnt, dass ich mich nicht tiefer gehend mit der Auffindung von Bewertungs-Gruppen beschäftigen werde, außer diese wie oben beschrieben zu definieren (mindestens 2 Mitglieder; mindestens 3 bewertete Objekte). Dies würde einen *Data-Mining-Exkurs* bedeuten, welcher den Rahmen dieser Arbeit sprengen würde. [Muk12] benutzen hierzu eine Technik namens *„frequent itemset mining“* über die sich der interessierte Leser in [Bor12] hinreichend informieren kann.

Widmen wir uns nun aber dem Vertrauenslevel. Zuerst werde ich aufzeigen, wie sich der Gesamt-Spamming-Score $S(u)$ eines Benutzers u ergibt, der sich aus dem Spamming-Score $SPS(u)$ und dem Gruppen-Spamming-Score $GSPS(u)$ zusammensetzt. Hierzu berechnen wir einen durchschnittlichen Gruppen-Spamming-Score aus allen Gruppen-Spamming-Scores aller Bewerter-Gruppen, von der der Benutzer ein Teil ist. Danach wird der Durchschnitt mit dem normalen Spamming-Score gebildet. Somit erhalten SPS und durchschnittlicher $GSPS$ einen gleichen Anteil an S. Sei G die Menge aller Gruppen von denen u ein Teil ist, so ergibt sich:

$$S(u)=\frac{1}{2}\left(SPS(u)+\frac{\sum_{g\in G} GSPS(g)}{|G|}\right)$$

Eine weitere Möglichkeit wäre nicht den durchschnittlichen Gruppen-Spamming-Score, sondern den schlechtesten zu übernehmen. Somit würde das schlimmste Verhalten von u in einer Gruppe berücksichtigt werden und es wäre nicht möglich schlechtes Verhalten durch Gutes bzw. Normales zu verschleiern. In diesem Fall ergäbe sich:

$$S(u)=\frac{1}{2}\left(SPS(u)+Max_{g\in G}(GSPS(g))\right)$$

Diese Variante greift die zu Beginn des „Review-Spammer-Gruppen“-Subkapitels (S. 26) erwähnte Idee auf, dass sollte ein Benutzer Mitglied in einer Bewerter-Gruppe sein, wahrscheinlich alle seine Gruppen als Bewerter-Gruppen einzustufen sind. Auf den nächsten Seiten wird von der zweiten Variante ausgegangen. Die Implementierung des Sicherheits-Frameworks wird beide Varianten vorsehen (siehe Kapitel 4).

Als nächstes wird ein Schwellwert σ benötigt, welcher angibt wie viel Prozent der Benutzer Review-Spammer sind. Wie in Kapitel 3.1 erwähnt, nennt die Literatur hier Zahlen von 5-20%. Ich werde hier zunächst einen Wert von 10% annehmen, die Implementierung wird aber wieder eine einstellbare Größe verwenden.

Dieser Schwellwert wird überhaupt nur benötigt, da der komplette Spamming-Score eines Benutzers ein relativer Wert ist, welcher immer im Verhältnis zu allen anderen Benutzern interpretiert werden muss. Es ist nicht möglich einfach zu sagen, dass alle Benutzer mit einem Spamming-Score größer als 0,8 $(S(u)>0{,}8)$ als Review-Spammer eingestuft werden. Es ist also nur möglich mit dem Spamming-Score eine Reihenfolge $r(u)$ unter allen Benutzern zu erzeugen, welche die Wahrscheinlichkeit angibt, dass ein Benutzer u ein Review-Spammer ist. In dieser Reihenfolge zieht man dann an der Stelle σ eine Grenze. Alle Benutzer über dieser Grenze werden als Review-Spammer eingestuft (Vertrauenslevel = 0), alle Benutzer unter dieser Grenze als mehr oder weniger vertrauenswürdig (Vertrauenslevel > 0, aber nicht größer als 10). Das Vertrauenslevel $V(u)$ für die vertrauenswürdigen Benutzer wird nach der oben bestimmten Reihenfolge auf dem Intervall $[0;10]$ gleich verteilt und dabei auf eine Nachkommastelle gerundet. Sei U die Menge aller Benutzer und x die Anzahl der Benutzer, welche als Review-Spammer eingestuft wurde, so ergibt sich:

$$V'(u)=0 \quad ,wenn \ \frac{r(u)}{|U|}\leq 0{,}1$$
$$V'(u)=10*\frac{r(u)-x}{|U|-x} \quad ,sonst$$

$r(u)-x$ gibt uns die absolute Position von u innerhalb der Reihenfolge aller vertrauenswürdigen Benutzer. $(r(u)-x)/(|U|-x)$ ist folglich die relative Position im Verhältnis zur Anzahl aller vertrauenswürdigen Benutzer im Intervall $[0;1]$. Wird dies mit 10 multipliziert, erhalten wir unsere Verteilung auf das Intervall $[0;10]$. Dieser Wert ist natürlich noch nicht auf eine Nachkommastelle gerundet. Wir erhalten also abschließen für das Vertrauenslevel V eines Benutzers u:

$$V(u)=\frac{\lfloor V'(u)*10 \rfloor}{10}$$

Somit kann das Vertrauenslevel eines jeden Benutzers im System berechnet werden. Mit dem Vertrauenslevel als Gewicht können schlussendlich auch die durchschnittlichen Bewertungen der bewertbaren Objekte, wie in Kapitel 2.4 beschrieben. Leider ist dies noch nicht ausreichend.

Dem aufmerksamen Leser wird nicht entgangen sein, dass in die Berechnung des Spamming-Scores an einigen Stellen die durchschnittliche Bewertung eines bewertbaren Objektes einfließt, diese durchschnittliche Bewertung aber theoretisch erst NACH Berechnung des Spamming-Scores zur Verfügung steht. Diesem Problem begegne ich mit zwei Lösungen.
Erstens wird der Spamming-Score und folglich auch die durchschnittliche Bewertung nicht nur einmal berechnet, sondern zyklisch immer wieder. Zweitens sollte zur Berechnung eines Spamming-Scores eine durchschnittliche Bewertung noch nicht zu Verfügung stehen, zum Beispiel bei einem neu hinzugefügten Objekt oder bei der aller ersten Berechnung, wird einfach eine durchschnittliche Bewertung von 3 Sternen bzw. 0,5 angenommen.

Die zyklische Neuberechnung von Bewertungen und Spamming-Scores ist erforderlich, damit nach einem Startzustand neue Benutzer und Objekte dem System hinzugefügt werden können. Ebenso könnte ein Review-Spammer erst später erkannt werden (durch sinkendes Vertrauenslevel beeinflusst durch neue Bewertungen) oder ein vertrauenswürdiger Benutzer zunächst fälschlicherweise als Review-Spammer eingestuft worden sein (da evtl. zu Beginn weniger Review-Spammer im System vorhanden waren, als zunächst angenommen bzw. σ zu groß gewählt worden ist).
Es empfiehlt sich also immer nach einem passenden Zeitintervall die Neuberechnung aller Spamming-Scores und durchschnittlichen Bewertungen anzustoßen, im Beispiel einer Internet-Fernsehzeitung wäre zum Beispiel eine tägliche Neuberechnung, etwa um 00:00 Uhr, denkbar.

Ein weiterer nicht zu vernachlässigender Punkt ist, dass ein System von sich gegenseitig beeinflussenden Werten „pendeln“ kann. D.h. den exakten Wert wird man kaum bestimmen können. Wir müssen uns hier also mit den auf eine Nachkommastelle gerundeten Werten zufrieden geben. Benutzt man für die Berechnung der durchschnittlichen Bewertung das auf eine Nachkommastelle gerundete Vertrauenslevel und andersherum so sollte sich das System recht schnell stabilisieren.
Es sollte also möglich sein eine Implementierung zu schaffen, welche eine „Neuberechnung“ der durchschnittlichen Bewertung und des Spamming-Scores so interpretiert, diese beiden Werte im Wechsel so lange zu berechnen, bis sie auf eine Nachkommastelle stabil geworden sind. Eine andere Möglichkeit wäre bei jeder „Neuberechnung“ die Werte im Wechsel 10, 100, 1000, etc. Mal zu berechnen.

Zum Schluss möchte ich mich noch kurz dem „Problem“ widmen, dass ein solcher Schutz gegen Manipulation darauf angewiesen ist, dass die Benutzer des Systems registriert sind. Zunächst verhindert auch eine Registrierung nicht, dass ein Benutzer sich mehrfach registriert und einen Sybil-Angriff startet. Hiervor sollte das System aber durch Berücksichtigung des Gruppen-Spamming-Scores schon geschützt sein. Trotzdem könnte man eine E-Mail-Adresse für die Registrierung erforderlich machen und dann natürlich nur eine Registrierung pro E-Mail-Adresse zulassen. Damit dies von Nutzen ist, müsste noch mindestens eine Filterung der bekanntesten *„Trash-Mail-Provider“* (=Wegwerf-E-Mail-Adressen-Anbieter) durchgeführt werden. Eine umfangreiche Liste solcher Anbieter findet man hier: http://www.email-wegwerf.de/wegwerfemail-liste.html

In Kapitel 2.4 erwähnte ich weiterhin, dass es auch nicht registrierten Benutzern möglich sein soll Bewertungen abzugeben, und dass diese Bewertungen mit einem Gewicht bzw. Vertrauenslevel von 1 berücksichtigt werden. Auch wenn ca. 81% der Benutzer - bei $\sigma = 10\,\%$ - ein höheres Vertrauenslevel als 1 erhalten werden, könnte durch massives Auftreten von Anonymen-Spam die durchschnittlichen Bewertungen deutlich beeinflusst oder sogar komplett verändert werden. Im Schnitt müsste man nur 4,5 anonyme Bewertungen für ein Objekt pro registriertem Benutzer, welcher dieses Objekt bewertet hat, abgeben um die gleiche Auswirkung auf die durchschnittliche Bewertung zu erreichen. Gibt man 40,5 anonyme Bewertungen pro registriertem Benutzer ab, hat man 90% der Durchschnittsbewertung bestimmt und die „Leistung“ der registrierten Benutzer auf 10% reduziert.

Es ist also elementar entweder anonyme Bewertungen zu verbieten oder einen weiteren Kontrollmechanismus einzuführen. Hierfür möchte ich kurz 3 Möglichkeiten vorstellen, aber nicht weiter vertiefen.

Zuerst kann man die IP Adresse eines anonymen Benutzers speichern und derselben IP Adresse in einem gewissen Zeitraum keine weitere Bewertung zu demselben Objekt erlauben. Dies ist in Deutschland aber aus datenschutzrechtlichen Gründen nicht ganz einfach, wie man auf https://www.datenschutzzentrum.de/ip-adressen/ nachlesen kann. Eine Speicherung von länger als 7 Tagen ist nicht zu empfehlen. Außerdem erhöht dies den Speicherbedarf und den Verwaltungsaufwand (wann welche IP gelöscht werden muss) nicht unwesentlich. Ebenso ist es für einen ernstzunehmenden Angreifer ein leichtes seine IP Adresse ständig zu wechseln.

Zweitens könnte man ein entsprechendes Cookie im Browser des Benutzers speichern. Somit wäre immerhin der Speicher- und Verwaltungsaufwand aufgehoben. Nichtsdestotrotz bleiben weiterhin Datenschutzprobleme, und da ein Benutzer die Möglichkeit besitzt das Speichern von Cookies in seinem Browser zu verbieten, wird man auch so einen ernsthaften Angreifer nicht aufhalten können.

Die dritte, und meines Erachtens empfehlenswerteste, Möglichkeit ist die Benutzung eines reCAPTCHA (siehe Kapitel 2.2). Die Nutzung ist umsonst, und automatisierte Programme und Bots werden effektiv behindert, zumindest solange keine maßgeblichen Fortschritte in Bild- und Schrifterkennung gemacht werden. Das manuelle Lösen eines reCAPTCHA dauert laut http://www.google.com/recaptcha/learnmore ca. 10 Sekunden. Es müsste also ein wesentlicher Aufwand erbracht werden um auf manuelle Art und Weise hier eine Manipulation vorzunehmen.

Dies beendet das Kapitel 3 Manipulation von Crowdsourcing-Systemen. Wir haben gesehen warum man überhaupt Crowdsourcing-Systeme manipulieren möchte und welche Arten von Manipulation es gibt. Es wurden ausführlich Methoden zur Entdeckung von Review-Spammern beschrieben und gezeigt, auf welche Art und Weise das OCCT diese Nutzen kann.

Im nächsten Kapitel werde ich mich mit der Modellierung und Implementierung des Sicherheits-Frameworks beschäftigen.

Kapitel 4

Implementierung des Sicherheits-Frameworks

In diesem Kapitel möchte ich die Implementierungsdetails des Sicherheits-Frameworks für das OCCT erläutern. Ich werde mich zunächst grundlegend der entstandenen Paketstruktur widmen und schließlich jedes Paket einzeln durchleuchten, seine Beziehungen und Klassen erklären.
Darauf folgt eine Erläuterung wie das Sicherheits-Framework zu nutzen ist bzw. wie es in einem bestehenden Projekt oder einer zukünftigen Entwicklung des OCCT eingebunden werden kann, bevor ich mich zuletzt kurz den Themen Testen, Testabdeckung und Metriken widme.

4.1 Paketstruktur

Für die Paketstruktur des Sicherheits-Frameworks habe ich mir Folgendes überlegt:

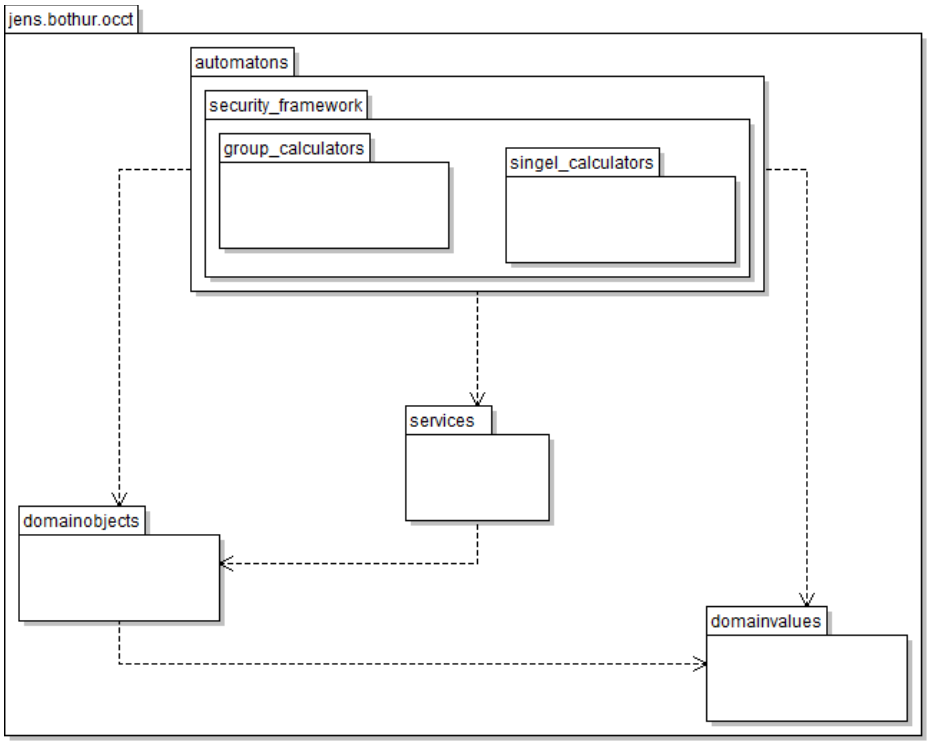


Abbildung 4.1: grundlegende Paketstruktur des OCCT und des Sicherheits-Frameworks

Wie man in *Abbildung 4.1* sieht, sind auch schon einige Aspekte des OCCT eingearbeitet. Diese Teile des OCCT sind vonnöten, da sie vom Sicherheits-Framework benutzt werden. Im Folgenden möchte ich die einzelnen Pakete beschreiben.

Domainvalues. [Zül05] spricht hier von Fachwerten. Eine Fachwertklasse stellt einen Wert dar, welcher eine vordefinierte Wertemenge bildet. „Wochentag“ ist ein Musterbeispiel für einen Fachwert, welcher offensichtlich 7 verschiedene Werte annehmen kann.
Ebenso haben Fachwerte keinen Lebenszyklus. Sie werden weder erstellt noch verbraucht. Sie existieren außerhalb von Raum und Zeit.

Domainobjects oder Materialien sind fast alles das, was Fachwerte nicht sind. Sie repräsentieren ein Objekt mit einem bestimmten Zustand zu einem gewissen Zeitpunkt ihres Lebenszyklus. Sie werden erstellt, verbraucht und evtl. endet ihre Existenz nachdem sie ihren Zweck erfüllt haben.
Nach dem Werkzeug-Material-Ansatz werden diese Materialien bearbeitet – von Werkzeugen – und werden ggf. schließlich ein Teil des erzeugten Ergebnisses.

Services sind der erste Teil in meinem Projekt der Tools bzw. Werkzeuge. Sie dienen hauptsächlich zur Verwaltung von Materialien, aber verändern diese auch bzw. delegieren von außen empfangene Veränderungsaufforderungen an die Materialien.

Automatons bzw. Automaten sind der zweite und wesentliche Teil der Werkzeuge des Security-Frameworks. Ein Automat bearbeitet, wie für ein Werkzeug üblich, die Materialien. Das Besondere ist, dass der Automat hier einer festen Vorgehensweise bzw. einem Algorithmus folgt. Sein Handeln wird kaum oder gar nicht von außen parametrisiert und er führt seine Aufgabe zu festen zeitlichen Punkten aus oder wird durch Ereignisse angestoßen.

4.2 Klassendiagramme

Nachdem wir nun die grundlegende Paketstruktur ergründet haben, werde ich in diesem Teilkapitel tiefer in die einzelnen Pakete hineinschauen. Wir werden sehen, wie sich die einzelnen Klassen untereinander benutzen und ich werde kurz begründen, warum die Klassen in ihrem jeweiligen Paket liegen.

Widmen wir uns zunächst wieder den **domainvalues**. *Abbildung 4.2* zeigt, dass dieses Paket 3 Klassen beinhaltet. Alle diese Klassen haben ausschließlich einen privaten (=nicht öffentlichen) Konstruktor. Alle möglichen Exemplare dieser Klassen werden statisch erzeugt. Exemplare der Klasse *„Review“* werden direkt in statischen öffentlichen Variablen hinterlegt und können so aufgerufen werden. Exemplare der Klassen *„AverageReview“* und *„ConfidenceLevel“* werden in einer Tabelle gespeichert und der Zugriff erfolgt über eine Fabrikmethode (vergleiche [Zül05], Seite 329).

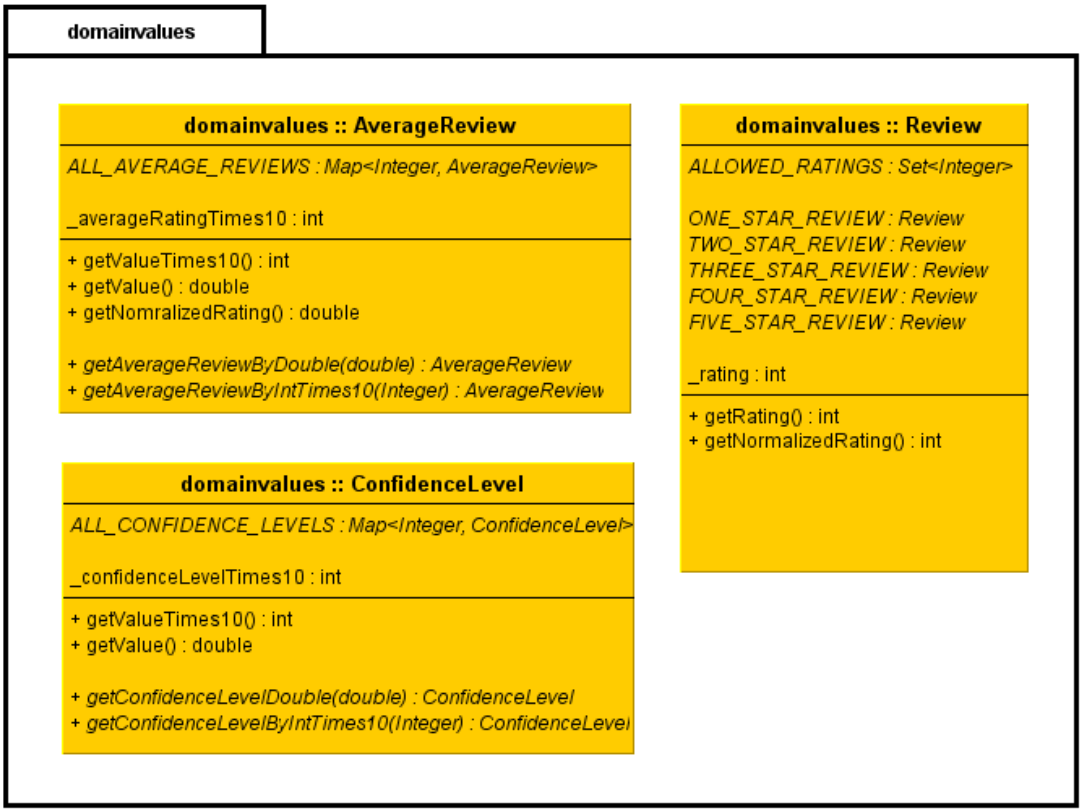


Abbildung 4.2: Klassendiagramm des „domainvalues“-Pakets.

Die Klasse *„Review“* beschreibt fünf mögliche Bewertungen für bewertbare Objekte. Diese reichen ganzzahlig von der 1-Stern- bis zu der 5-Sterne-Bewertung. Sie bietet zusätzlich eine *„Convenience-Methode“* zum Abrufen der normalisierten Bewertung im Intervall $[0;1]$.
Die Klassen *„ConfidenceLevel“* und *„AverageReview“* sind sich recht ähnlich. Beide kapseln einen auf eine Nachkommastelle gerundeten Wert – zwischen 1,0 und 5,0 bzw. 0,0 und 10,0 – und bieten an diesen Wert als `double` oder mit 10 multipliziert als `int` auszugeben.

In *Abbildung 4.3* sehen wir die **domainobjects**. Diese umfassen zunächst die Klasse *„User“,* welche einen Benutzer unseres Systems repräsentiert. Sie kapselt die Id, den Namen und das Vertrauenslevel (=*„ConfidenceLevel“*) des Benutzers.
Als Zweites sehen wir die Klasse *„RateableObject“* bzw. „bewertbares Objekt“. Diese hält neben einer Id auch alle Benutzers (=*„User“*), welche dieses bewertbare Objekt bewertet haben, mit ihren jeweiligen Bewertungen (=*„Review“*) und Bewertungszeitpunkten, die durchschnittliche Bewertung (=*„AverageReview“*) und ein Datum, welches den frühesten Zeitpunkt repräsentiert, an dem eine Bewertung möglich ist. Die Idee ist, dass bei zukünftiger Einbettung des Sicherheits-Frameworks bewertbare Objekte, wie Termine im OCCT, von dieser Klasse erben können und sollen.

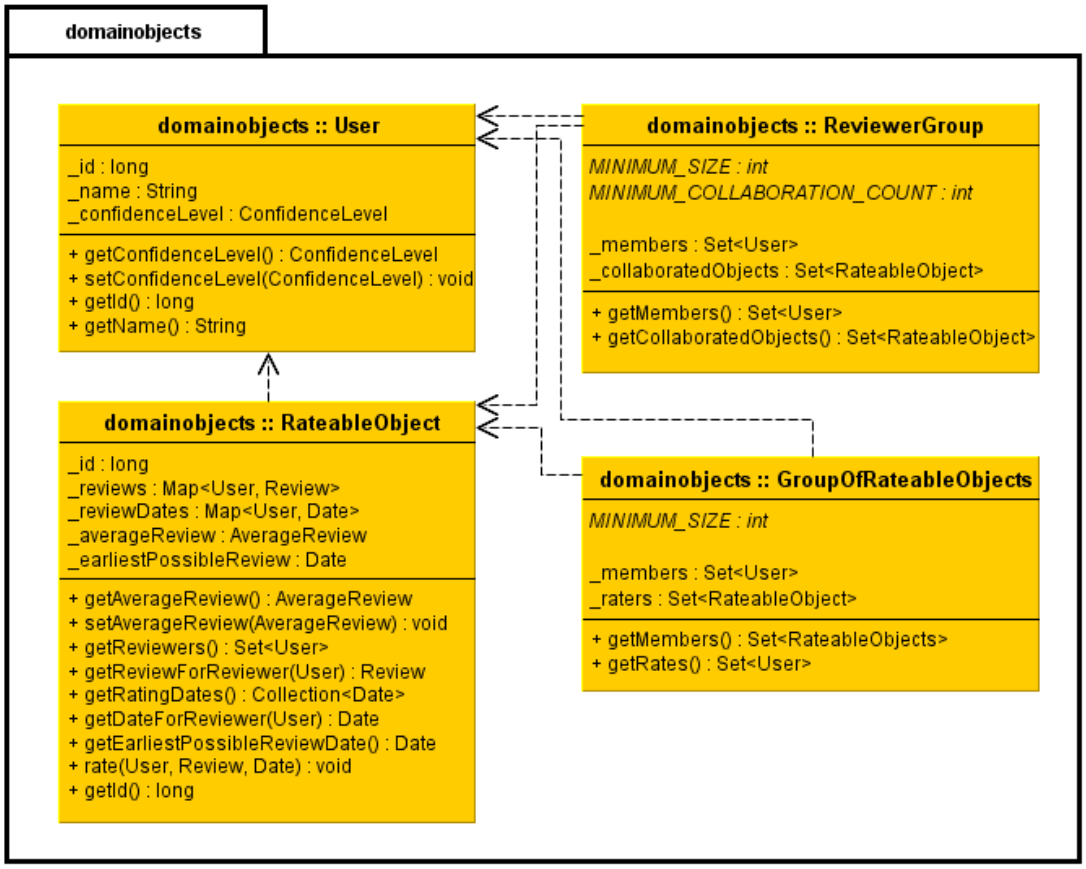


Abbildung 4.3: Klassendiagramm des „domainobjects“-Pakets.

Die Klasse *„ReviewerGroup“* repräsentiert eine, wie in Kapitel 3.3 definiert, Bewerter-Gruppe. Zuletzt haben wir noch die Klasse *„GroupOfRateableObjects“*. Neben der Repräsentation einer, ebenso in Kapitel 3.3 definierten, Objektgruppe werden hier direkt alle Benutzer (=*„User“*) gespeichert, welche mindestens 2 Bewertungen für Objekte (=*„RateableObject“*) der Objektgruppe abgegeben haben.

Kommen wir zum „**services**“-Paket. Dieses Paket enthält 2 Schnittstellen und für Testzwecke 2 Dummy-Implementationen dieser Schnittstellen (hier nicht dargestellt).

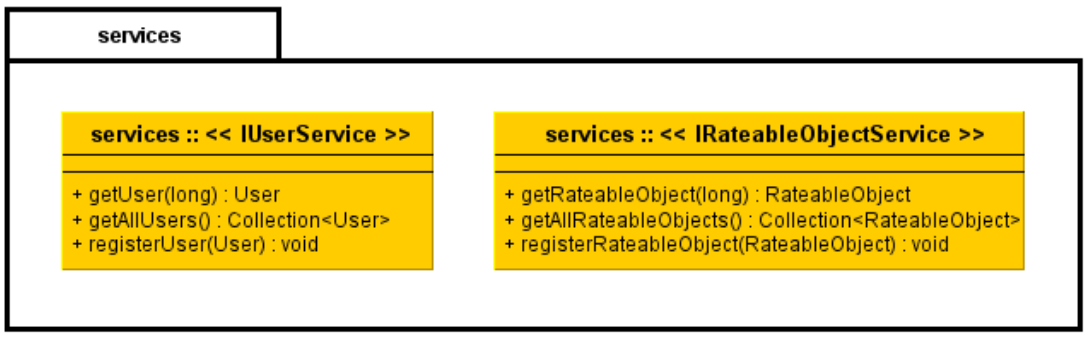


Abbildung 4.4: Klassendiagramm des „services“-Pakets.

Die Schnittstellen beschreiben zwei Werkzeuge zum Verwalten von Benutzern (=„*User*“) und bewertbaren Objekten (=„*RateableObjects*“). Diese Werkzeuge erlauben es das jeweilige Material über seine Id zu finden, eine Sammlung aller dieser Materialien abzurufen und ein neues Material anzulegen.

Dies bringt uns bereits zum „**automatons**“-Paket und damit zum eigentlich Kern des Security-Frameworks:

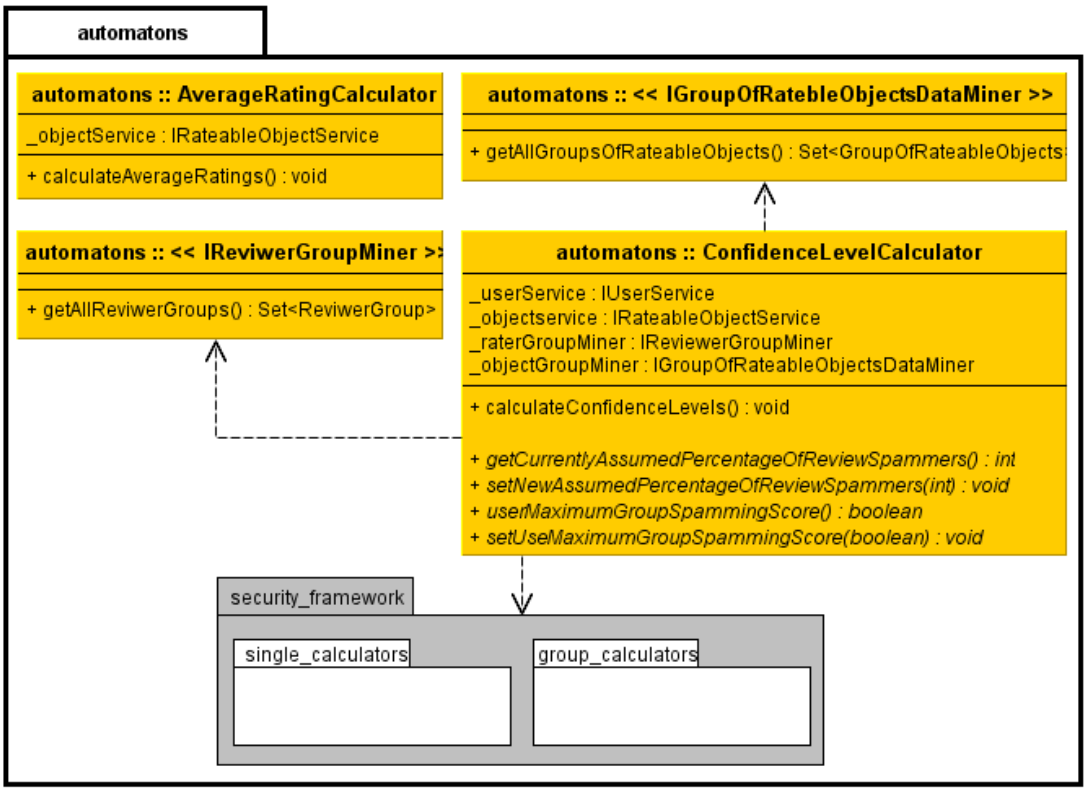


Abbildung 4.5: Klassendiagramm des „automatons“-Pakets.

Wir sehen in *Abbildung 4.5*, dass wir es hier mit 2 Klassen und 2 Schnittstellen zu tun haben. Die Schnittstellen beschreiben zwei Data-Miner, welche als einzige Aufgabe haben die Bewerter-Gruppen (=„*ReviewerGroup*) und Objektgruppen (=„*GroupOfRateableObject*“) zu finden. Wie in Kapitel 3.3 beschrieben, werde ich mich aus Platzgründen in dieser Arbeit nicht weiter mit Data-Mining beschäftigen. Die Klasse „*AverageRatingCalculator*“ hat die Aufgabe alle durchschnittlichen Bewertungen (=„*AverageRating*“) von allen bewertbaren Objekten im System neu zu berechnen. Dazu benötigt dieser Automat einen „*IRateableObjectService*“. Es gibt nur eine sichtbare Methode, welche ohne Parameter aufgerufen wird. Diese Methode berechnet alle durchschnittlichen Bewertungen neu und trägt sie an den jeweiligen bewertbaren Objekten ein.

Der Automat „*ConfidenceLevelCalculator*“ übernimmt ähnlich die Neuberechnung aller Vertrauenslevel (=„*ConfidenceLevel*“) aller Benutzer im System. Da dies wie im Kapitel 3.3 und 3.4 beschrieben eine komplexere Rechnung ist, benötigt dieser Automat beide Arten von weiter oben beschriebenen Services und beide Arten von in diesem Paket beschriebenen Data-Minern. Neben einer parameterlosen Methode zum Anstoßen der Neuberechnung aller Vertrauenslevel, gibt es noch 4 statische Methoden. Diese

Methoden dienen zum Abfragen und Setzen des angenommenen Prozentsatzes von Review-Spammern σ und ob der durchschnittliche oder maximale Gruppen-Spam-Score eines Benutzers berücksichtigt werden soll.

Des Weiteren nutzt dieser Automat insgesamt 10 weitere Kalkulatoren, welche im Unterpaket „**security-framework**“ gekapselt sind. Dieses Unterpaket unterteilt sich in zwei weitere Pakete. Zunächst das „**single_calculators**“-Paket:

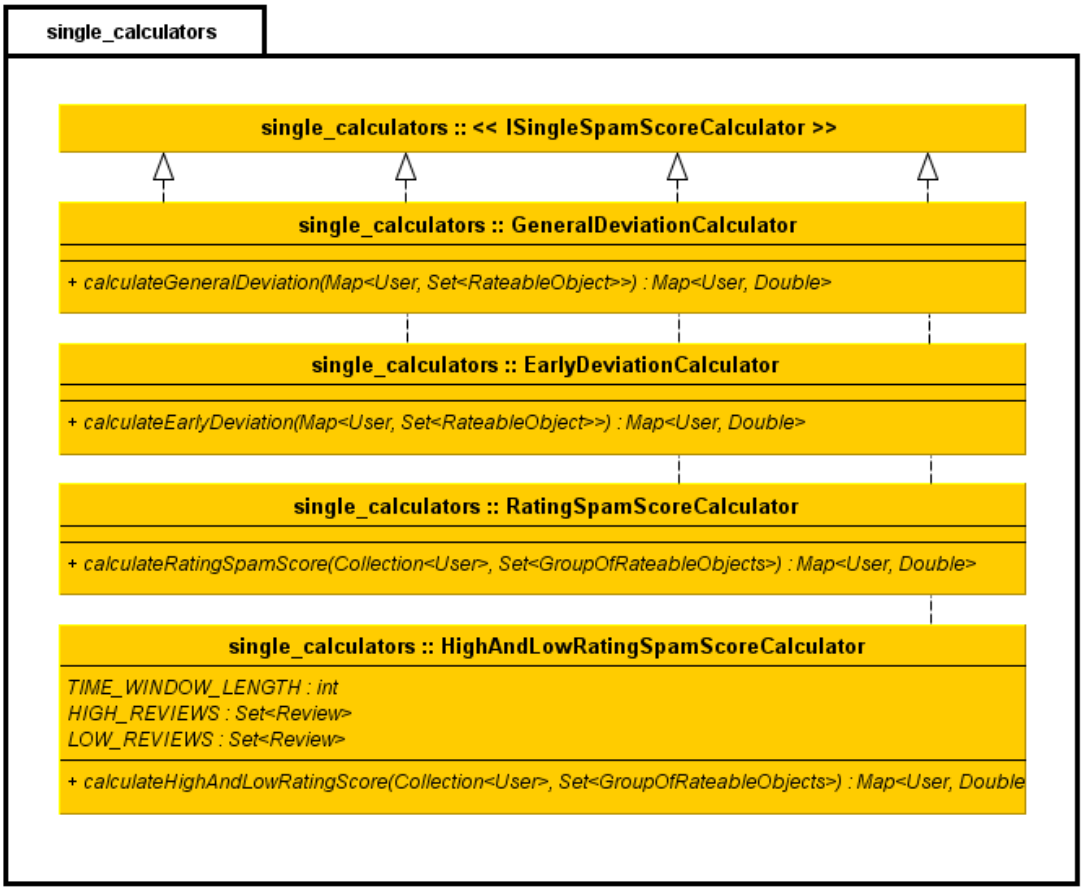


Abbildung 4.6: Klassendiagramm des „single_calculators“-Paket

Alle 4 Automaten in diesem Paket haben eine von außen aufrufbare Methode. Jeder dieser Automaten ist für die Berechnung eines bestimmten Spamming-Score-Anteils zuständig und benutzt hierzu ausschließlich die Materialen.

Ganz analog zeigt sich das „**group_calculators**“-Paket (siehe *Abbildung 4.7*). Es beinhaltet 6 Automaten, welche wieder für die Berechnung genau eines der 6 Gruppen-Spamming-Score-Anteile zuständig sind. Auch sie haben nur eine sichtbare Methode und arbeiten ausschließlich auf den Materialien.

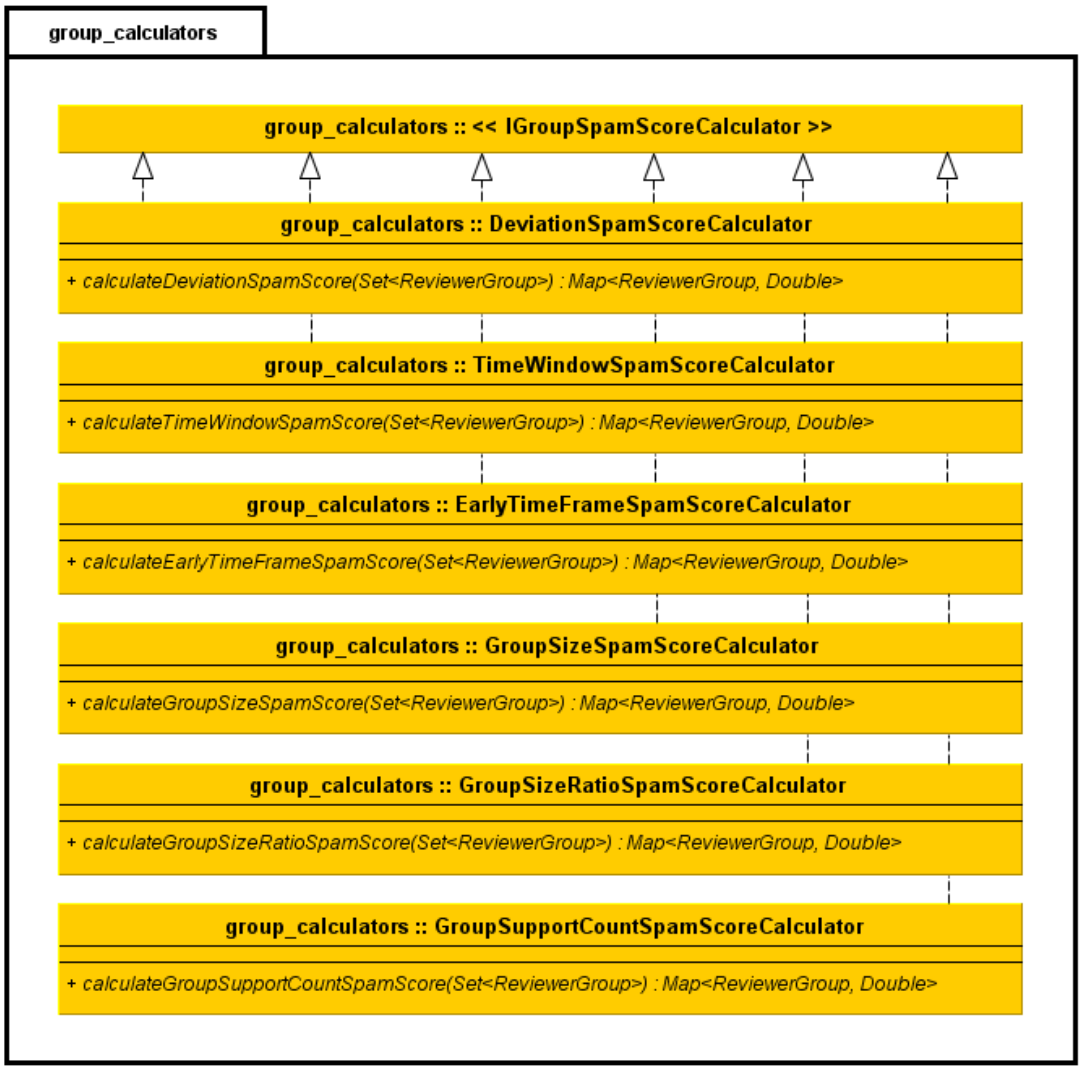


Abbildung 4.7: Klassendiagramm des „group-calculators“-Pakets.

4.3 Wie nutze ich das Sicherheits-Framework?

Die Antwort auf diese Frage ist erstaunlich trivial. Will man das Sicherheits-Framework in der Implementation des OCCT oder eines anderen Crowdsourcing-Bewertungssystems nutzen, muss man zunächst die von mir implementierten Klassen seinem Projekt hinzufügen und die 4 Schnittstellen – 2 Services und 2 Data-Miner – implementieren. Alle bewertbaren Objekte, wie z.B. Termine, sollten von der Klasse *„RateableObjects“* erben und alle Bewerter von der Klasse *„User“*. Alternativ kann man diese beiden Klassen um die für das eigene Projekt benötigten Elemente erweitern.
Sind diese Punkte erfüllt, so muss man nur noch von der zentralen Kontrolleinheit des Systems die Neuberechnung der durchschnittlichen Bewertungen und der Vertrauenslevel anstoßen. Hierzu muss man die parameterlosen Methoden am *„ConfidenceLevelCalculator“* und *„AverageRatingCalculator“* aufrufen. Natürlich muss man sich hierzu jeweils ein Objekt dieser Klasse erstellen und Referenzen auf die entsprechenden Services und Data-Miner übergeben.

4.4 Tests & Metriken

Gehostet habe ich meine Implementation auf dem Quellcode-Repository von Google. Eine lokale Kopie kann man sich unter:

https://code.google.com/p/security-framework-for-the-occt/

herunterladen. Ich habe mich für die Apache 2.0 Lizenz entschieden, da diese meines Erachtens am wenigsten einschränkend auf zukünftige Weiterentwicklungen wirkt.

Insgesamt, d.h. inklusive der Testklassen, umfasst meine Implementation 12 Pakete (jeweils 6 im Programmteil und Testteil) in denen sich 51 Klassen befinden (28 im Programmteil, 23 im Testteil). Diese Klassen werden um 6 Schnittstellen (alle im Programmteil) erweitert. Der Programmteil umfasst knappe 2800 Zeilen Code, welche um ca. 2500 Zeilen Kommentar ergänzt werden. Der Testteil ist etwa halb so groß mit etwas weniger als 1300 Zeilen.

Beim Testen meiner Implementation habe ich mich größtenteils auf Positivtests beschränkt. An einigen Stellen, wo ich es für sinnvoll hielt, habe ich aber auch illegale Parameterübergaben getestet um somit die Robustheit des Frameworks sicherzustellen.
Insgesamt erreiche ich eine Testabdeckung von 97,3%:

Element	Coverage	Covered...	Missed...	Total...
SFOCCT	97,3 %	2.795	78	2.873
src	97,3 %	2.795	78	2.873
jens.bothur.occt.automatons	95,5 %	643	30	673
jens.bothur.occt.automatons.secu	96,9 %	561	18	579
jens.bothur.occt.domainobjects	97,1 %	605	18	623
jens.bothur.occt.automatons.secu	98,3 %	683	12	695
jens.bothur.occt.domainvalues	100,0 %	247	0	247
jens.bothur.occt.services	100,0 %	56	0	56

Abbildung 4.8: Testabdeckung der Implementation, gemessen mit EclEmma (http://www.eclemma.org/)

Dies beendet Kapitel 4. Im nächsten Kapitel werde ich mit der Auswertung meiner Ergebnisse und zukünftigen Problemen fortfahren.

Kapitel 5

Ergebnis, Auswertung und Ausblick

In diesem letzten Kapitel werde ich zunächst einen kurzen Überblick über die Ergebnisse und Ziele geben, welche ich im Laufe dieser Bachelorarbeit erreicht habe. Ebenso möchte ich diese Ergebnisse, aber auch die Arbeitsweise während der Implementierung auswerten.
Zuletzt möchte ich einen Ausblick auf zukünftige mögliche Arbeiten geben. Welche Problematiken habe ich vernachlässigt, welche Probleme sind mir während der Arbeit an meinem Konzept zur Entdeckung von Review-Spam aufgefallen und wie könnte dieses Konzept weiterentwickelt werden.

5.1 Ergebnis

Im Verlauf dieser Bachelorarbeit habe ich zunächst grob die Idee des Online-Crowdsourcing-Calendar-Tool beschrieben. Ich habe mir ein grundlegendes Verständnis zum aktuellen Stand der Forschung über Crowdsourcing-Systeme angeeignet und, anhand dieser Eigenschaften und Herausforderungen von und an Crowdsourcing-Systeme, das OCCT weiter spezifiziert.
Schließlich habe ich mich mit der Manipulation von Crowdsourcing-Systemen beschäftigt. Nach einem Überblick über aktuelle Forschungen zu diesem Thema habe ich mich der Herausforderung gestellt ein Konzept zur Entdeckung von Review-Spam zu entwickeln, welches ausschließlich auf einer Vergabe von numerischen Bewertungen basiert. Ich habe die Ansätze von [Lim10] bezüglich des Spamming-Scores und [Muk12] bezüglich des Gruppen-Spamming-Scores miteinander verbunden und entsprechend meiner Bedürfnisse angepasst.
Dieses Konzept habe ich dann, frei nachdem Werkzeug-Material-Ansatz von [Zül05], in Java implementiert und getestet.

5.2 Auswertung

Zunächst möchte ich erwähnen, dass es für mich sehr einfach war, mein Projekt nach dem Werkzeug-Material-Ansatz zu modellieren und zu implementieren. Ich hatte von vornherein eine gute Vorstellung davon, unter welche Design-Metapher jede Klasse fallen würde, was mir half meine Arbeit zu strukturieren, indem ich zuerst die Material- und dann die Werkzeug-Schicht implementiert habe. Ich hatte nie Probleme meinen Bedürfnissen entsprechend eine Klasse einzuordnen und würde jederzeit wieder nach dieser Leitmetapher entwickeln.

In einer Testumgebung von 100 Benutzern – wovon 10 als Review-Spammer und 90 als normale(=zufällige) Bewerter designt wurden –, 200 bewertbaren Objekten und insgesamt 4000

Bewertungen hat sich das Security-Framework bewährt. In über 200.000 verschiedenen Testumgebungen dieser Art – die Testumgebung wurde von einem Zufallsgenerator erzeugt und ich habe 200.000 verschiedene Seeds ausprobiert – waren immer 7 der 10 Review-Spammer und den ersten 10 Plätzen. D.h. sie haben ein Vertrauenslevel von 0 erhalten. Die Top-3 bestand immer komplett aus Review-Spammern.
Ergänzend kann ich noch sagen, dass in keinem dieser 200.000 Tests das Framework mehr als 2 Durchläufe brauchte um sich zu stabilisieren. D.h. eine abwechselnde Neuberechnung von durchschnittlichen Bewertungen und Vertrauenslevel pendelte zwar zunächst ging aber nie über die zweite Neuberechnung der Vertrauenslevel hinaus.
Ebenso hat keine Berechnung (inklusive der zufälligen Erstellung der Testumgebung) länger als 20ms gedauert. Natürlich ist dies für größere Systeme eine recht schwache Aussage. Dies führt mich zum nächsten Teilkapitel.

5.3 Ausblick

Ich möchte kurz auf Probleme bzw. Schwachstellen meiner Ergebnisse eingehen. Zunächst ist die Tatsache zu nennen, dass die Review-Spammer in meiner Testumgebung exakt nach dem Eigenschaftenprofil designt waren, welches ich ihnen in Kapitel 3.3 unterstelle. [Lim10] und [Muk12] belegen zwar weitgehend, warum diese Annahmen korrekt sind, aber dennoch würden Review-Spammer mit einem deutlich anderen Bewertungs-Profil nicht erkannt werden.
Als sweites ist die viel zu kleine Testumgebung anzusprechen. Ohne einen „echten“ Datensatz hielt ich es allerdings aus oben genannten Grund nicht für sinnvoll eine größere Umgebung künstlich zu designen. Deswegen nehme ich die Ergebnisse meiner Testumgebung eher als grundlegenden Beweis, dass meine Implementation auch das tut, was ich in meinem Konzept zur Entdeckung von Review-Spammern beschreibe.
Als nächsten Schritt würde ich mir wünschen mein entwickeltes Framework auf einen realen Datensatz anzuwenden. Allerdings müsste in so einem Datensatz von Hand markiert werden, welche Bewertungen Spam sind und welche nicht. Für einen realistisch großen Datensatz übersteigt der Aufwand den Rahmen dieser Arbeit um ein Vielfaches. [Lim10] und [Muk12] haben beide mit einem solchen Datensatz, welcher auf der Amazon-Produktbewertung basiert, gearbeitet. Leider habe ich auf meine Kontaktversuche einen solchen markierten Datensatz zu erhalten keine Antwort erhalten. Auch Amazon selbst wollte mir keinen Datensatz zur Verfügung stellen.
Mit einem solchen größeren Datensatz könnte man auch tatsächliche Aussagen darüber treffen, wie lange das System bei Neuberechnung pendelt, und somit ziemlich genaue Abschätzungen treffen wie viel Rechenleistung das Framework tatsächlich benötigt.

Erklärung

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Außerdem erkläre ich, dass ich mit der Einstellung dieser Bachelorarbeit in den Bestand der Bibliotheken der Universität Hamburg einverstanden bin.

Hamburg, den 18.02.2013

Jens H.-P. Bothur

Literaturverzeichnis

Bücher:

[Fow02] **Fowler,** Martin (2002): *„Patterns of Enterprise Application Architecture.“* Amsterdam: Addison-Wesley Longman

[Zül05] **Züllighoven**, Heinz (2005): *„Object-Orientated Construction Handbook: Developing Application- Oriented Software with the Tools & Materials Approach.“* Heidelberg: dpunkt- Verlag

Veröffentlichungen und Foliensätze:

[Bor12] **Borgelt,** Christian (2012): „*Frequent Pattern Mining*“, Intelligent Data Analysis and Graphical Models Research Unit, European Centre for Soft Computing, Seiten: 1-280, online: http://www.borgelt.net/slides/fpm4.pdf

[Cox11] **Cox,** Landon P. (2011): „*Truth in Crowdsourcing,“* aus „IEEE Security and Privacy“, Sept.-Okt. 2011, Seite: 74-76

[Dou02] **Douceur**, John R. (2002), „*The Sybil Attack“* online: http://www.cs.rice.edu/Conferences/IPTPS02/101.pdf

[Doa11] **Doan,** A., **Ramakrishnan,** R., **Halevy,** A. (2011): *„Crowdsourcing Systems on the World-Wide Web,“* aus „Communication of the ACM“, April 2011, Seite: 86ff oder online: http://delivery.acm.org/10.1145/1930000/1924442/p86-doan.pdf?ip=80.171.60.189&acc=OPEN&CFID=107177166&CFTOKEN=35659663&__acm__=1338907271_207303d1a21640fb655c666867747bbd

[Est12] **Estellés-Arolas,** E., **González-Ladrón-de-Guevara,** F. (2012): *„Towards an integrated crowdsourcing definition,“* aus „Journal of Information Science“, XX (X), Seite: 1-14 oder online: http://www.crowdsourcing-blog.org/wp-content/uploads/2012/02/Towards-an-integrated-crowdsourcing-definition-Estell%C3%A9s-Gonz%C3%A1lez.pdf

[Jin08] **Jindal,** N., **Liu,** B. (2008): *„Opinion Spam and Analysis“* aus WSDM, 2008, Seiten: 219-230 oder online: http://www.cs.uic.edu/~liub/FBS/opinion-spam-WSDM-08.pdf

[Jin07] **Jindal**, N., **Liu**, B. (2007): „*Review Spam Detection*“ aus WWW, 2007, oder online: http://www2007.cpsc.ucalgary.ca/posters/poster930.pdf

[Kim06] **Kim**, S., **Pantel,** P., **Chklovski,** T., **Pennacchiotti,** M. (2006): *„Automatically Assessing Review Helpfulness“* aus EMNLP, 2006, Seiten: 423-430 oder online: http://delivery.acm.org/10.1145/1620000/1610135/p423-kim.pdf?ip=80.171.139.138&acc=OPEN&CFID=165216608&CFTOKEN=62103605&__acm__=1348839172_85a8f5ac6d1723ffaa649f6e70c66111

[Li11] **Li**, F., **Huang**, M., **Yang**, Y., **Zhu**, X. (2011): *„Learning to Identify Review Spam“* aus IJCAI, 2011, oder online: http://ijcai.org/papers11/Papers/IJCAI11-414.pdf

[Lim10] **Lim**, E., **Nguyen**, V., **Jindal**, N., **Liu**, B., **Lauw**, H. (2008): *„Detecting Product Review Spammers using Rating Behaviors“* aus CIKM, 2010, Seiten: 939-948 oder online: http://www.cs.uic.edu/~liub/publications/cikm-2010-final-spam.pdf

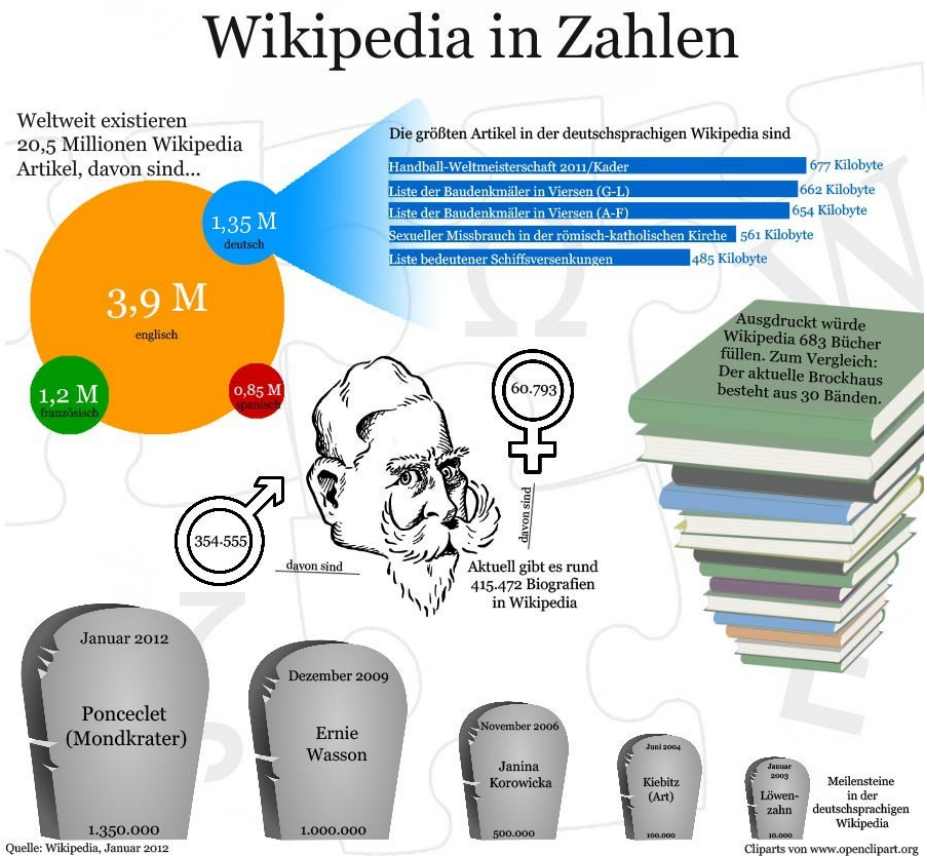
[Muk12] **Mukherjee**, A., **Liu**, B., **Glance**, N. (2012): *„Spotting Fake Reviewer Groups in Consumer Reviews“* aus WWW-2012 oder online: http://www.cs.uic.edu/~liub/publications/WWW-2012-group-spam-camera-final.pdf

[Ols08] **Olson**, Michael (2008): *„The amateur search“* aus SIGMOD Record 37,2 Seite:21-24 oder online: http://www.sigmod.org/publications/sigmod-record/0806/p21.olson.pdf

[Wel08] **Weld**, D.S., **Wu**, F., **Adar**, E., **Amershi**, S., F, J., **Hoffmann**, R., **Patel**, K., **Skinner**, M. (2008): *„Intelligence in Wikipedia.“* aus AAAI, 2008, oder online: http://www.cond.org/iwp.pdf

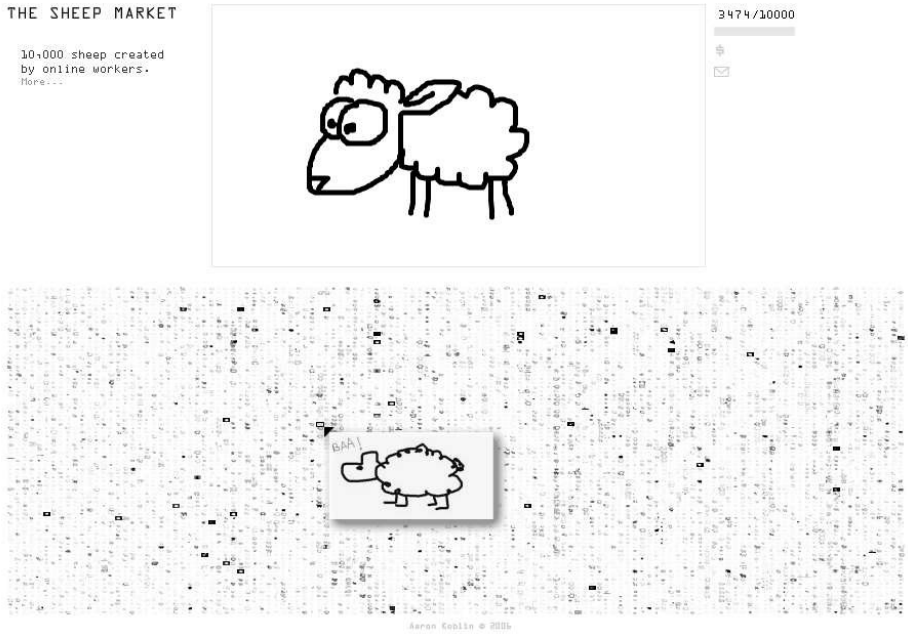
Anhang

Bild "Wikipedia in Zahlen"



Quelle:
http://www.technikjournal.de/cms/upload/bilder/2011_12_Wintersemester/Heute/Wikipedia/Gro776e.jpg

Aaron Koblin zahlte 10.000 Benutzer auf Amazons Mechanical Turk jeweils 0,02$ für die Zeichnung eines nach links blickenden Schafes. Animationen zu Entstehung eines jeden Schafes kann man auf www.thesheepmarket.com sehen.



Screenshot von: www.sheepmarket.com
Zeichnung des Schafs #3474.