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

Konzeptionierung und Entwicklung eines JIRA-Plugins zur Erkennung von Anomalien im Teamverhalten während Sprints

Bachelorarbeit

im Studiengang Informatik

von

Michael Mircea

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Ann-Christin Klünder
Betreuer: M. Sc. Fabian Kortum

Hannover, 02.02.2021

Erklärung der Selbstständigkeit

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

Hannover, den 02.02.2021

_____

Michael Mircea

Zusammenfassung

Projektumgebungen im Bereich der Softwareentwicklung werden heutzutage zunehmend als soziotechnische Systeme betrachtet. In solchen Systemen gibt es Wechselwirkungen zwischen menschlichen und technischen Faktoren, welche anhand soziotechnischer Metriken festgehalten werden können. Diese Faktoren und dessen Wechselwirkungen können entscheidend für den Projekterfolg sein. Allerdings werden mögliche Anomalien in den zugehörigen Metriken während eines Projekts oft erst im Nachhinein erkannt, was sich negativ auf den Projektverlauf auswirken kann.
Vor diesem Hintergrund soll ein Konzept hergeleitet und entwickelt werden, welches frühzeitig auf mögliche Anomalien aufmerksam macht um späteren Problemen im Projekt entgegenzuwirken. Das Konzept umfasst eine Anomalieerkennung, welche auf soziotechnischen Metriken basiert, und ein Benachrichtigungssystem, welches Entwicklerteams schon während Sprints über anomale Verläufe im Teamverhalten benachrichtigt. Grundlage für die Erarbeitung des Konzepts sind Datensätze des *Softwareprojekts*, ein jährliches Gruppenprojekt im Informatik Bachelorstudium an der *Leibniz Universität Hannover*. Das Konzept soll allerdings nicht nur für studentische Projekte, sondern auch für Projekte in der Industrie einsetzbar sein. Eine Evaluation zweier solcher Projekte wird daher auch durchgeführt.
Das Konzept wird für die weit verbreitete Projektmanagement Software *Jira* entwickelt und in das dafür bereits bestehende Plug-in *ProDynamics* integriert. In dieser Arbeit wird insbesondere auf die Erarbeitung des Konzepts, dessen Entwicklung und Evaluierung, sowie einen Ausblick auf potentielle zukünftige Erweiterungen eingegangen.

Abstract

Project environments in the domain of software engineering are increasingly viewed through the lens of socio-technical systems. Such systems have the property of strong interplay between human factors and technical factors, which can be captured through socio-technical metrics. These factors and their interplay can be a deciding factor for project success. However, possible anomalies in the corresponding metrics oftentimes are only identified retrospectively, which can have negative consequences for project progression. In light of these issues a concept shall be derived and developed, which proactively draws attention to potential anomalies in order to counteract possible adverse effects in later stages of the project. The concept includes an anomaly detection, based on socio-technical metrics, as well as a notification mechanism, which notifies developer teams during sprints in cases of anomalous team behaviour. Foundation for the conceptualization will be data sets sourcing from the *Softwareprojekt*, a yearly group project for computer science undergraduate students at the *Leibniz Universität Hannover*. However, the concept shall not only find application in student projects, but industry projects as well. Therefore two such projects are also included during evaluation.
The concept will be developed for the widely used project management software *Jira* and integrated in the pre-existing plug-in *ProDynamics*. This paper will go into detail on the elaboration of the concept, its development and evaluation, as well as a perspective on potential future extensions.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Software wird heutzutage überwiegend im Team entwickelt und ist geprägt von der damit verbundenen Zusammenarbeit. Dennoch wurde historisch der soziale Aspekt in der Softwareentwicklung vernachlässigt, obwohl auch in diesem Feld menschliche Faktoren einen starken Einfluss auf den Projekterfolg haben, so John et al. [3]. Das Feld des Software Engineering entwickelt sich rasant weiter: Anforderungen an Entwicklungszeit und Softwarequalität steigen, während die Toleranz für Fehler fällt. Law et al. [9] beschreiben daher, dass Projekterfolg immer schwerer zu erreichen ist und nennen soziale Faktoren, wie 'knowledge sharing', 'motivation' und 'customer collaboration', als entscheidend.
Der Ansatz des 'soziotechnischen Systems' hat aus diesen Gründen im Feld des Software Engineering steigend an Relevanz zugenommen. Dieser versucht Menschen, Technologien, Prozesse und Organisationen als ganzheitliches System zu betrachten. Emery et al. [1] beschreiben, dass der Erfolg eines Unternehmens nicht davon abhängt, wie es als technisches System mit austauschbaren Individuen funktioniert, sondern wie es als soziotechnisches System arbeitet. Verschiedene Ansätze, mit der Zielsetzung die sozialen und technischen Aspekte in Softwareprojekten zu untersuchen, sind Themen aktueller Forschung (siehe Kapitel 2.5). In dieser Arbeit wird ein weiterer Ansatz verfolgt, welcher versucht ein besseres Verständnis über das Zusammenspiel von soziotechnischen Faktoren in Entwicklerteams zu fördern. Um dies zu erreichen wird ein Algorithmus zur Anomalieerkennung, sowie ein proaktives Benachrichtigungssystem konzeptioniert und entwickelt.

1.1 Problemstellung

Soziale und technische Faktoren sind entscheidend für den Projekterfolg. Allerdings werden Probleme bei diesen meist erst retrospektiv vom Entwicklerteam erkannt [12], zu welchem Zeitpunkt sich die daraus resultierende Problemsituation bereits manifestiert hat. Sinkt die Produktivität, so könnte zum Beispiel festgestellt werden, dass das Team vor ein paar Wochen unübliche Verhaltensweisen hinsichtlich der Kommunikation zwischen Entwicklern aufwies. Solche Abweichungen können während agiler Entwicklung nicht zuverlässig wahrgenommen werden. Es bestehen bereits Werkzeuge, die unter anderem das Erheben, Aufbereiten und Präsentieren der zugehörigen Metriken ermöglichen [8]. Dies kann helfen, um nach dem Auftreten eines Problems Erkenntnisse über Abhängigkeiten zu gewinnen und somit zukünftige Risiken zu verringern. Es besteht allerdings noch kein proaktiver Feedbackmechanismus, welcher das ursprüngliche Problem präventiv beheben kann.

1.2 Lösungsansatz

Um die im vorigen Abschnitt beschriebene Problemstellung zu bewältigen, wird in dieser Bachelorarbeit eine Anomalieerkennung und ein zugehöriges Benachrichtigungssystem konzeptioniert und als *Jira* Plug-in entwickelt. Dieses soll die Projektleitung im Falle von anomalem Verhalten während eines Sprints alarmieren. Dadurch können Schwachstellen und Risiken identifiziert werden, bevor diese zu einer Gefahr für den Projekterfolg werden. Des Weiteren ist eine potentielle Problemquelle leichter zu erkennen, wenn diese aktuell besteht, als sie retrospektiv zu identifizieren.
Der Algorithmus zur Anomalieerkennung arbeitet hierbei mit aggregierten, teambezogenen Daten. Auf Grund der individuellen Arbeitsweise von agilen Entwicklerteams kann allerdings keine weitere Hilfe, in Form von Handlungsempfehlungen oder Ähnlichem, vom Plug-in geleistet werden. Zielsetzung dieser Bachelorarbeit ist es, die Entwickler über Anomalien in den Metriken, dessen Bedeutung, so wie deren Relevanz zu informieren. Das Team muss selbst entscheiden, ob sie diese Anomalien genauer betrachten und auf dessen Basis Handlungsänderungen vornehmen wollen. Die Priorität liegt hierbei in einer ausreichend Vermittlung auffälliger Daten, ohne die Entwickler mit exzessiven Benachrichtigungen zu überlasten.

1.3 Struktur der Arbeit

Diese Arbeit ist wie folgt strukturiert: Nach der Einleitung in Kapitel 1, werden in Kapitel 2 die für spätere Kapitel benötigten Grundlagen präsentiert. Unter anderem findet sich hier eine Erklärung der verwendeten soziotechnischen Konzepte und ein kurzer Einblick in *Jira* als Projektmanagement Tool, sowie in das dafür zu erweiternde Plug-in *ProDynamics* [8]. Der größte Fokus der Arbeit liegt auf Kapitel 3, welches sich mit der Konzeptionierung und Planung der Anomalieerkennung, sowie des zugehörigen Benachrichtigungssystems, befasst. Dessen Entwicklung und Integration in *ProDynamics* wird in Kapitel 4 dargestellt. Nachfolgend wird das Konzept, insbesondere der Erkennungsalgorithmus, in Kapitel 5 evaluiert. Abschließend wird in Kapitel 6 eine Zusammenfassung, sowie ein Ausblick über potentielle, zukünftige Erweiterungen des Systems präsentiert.

Kapitel 2

Grundlagen

In diesem Kapitel werden die wichtigsten Grundlagen und Technologien dargestellt und erklärt, welche für das Verständnis der späteren Konzeptkapitel und anschließenden praktischen Realisierung vorausgesetzt werden.

2.1 Statistische Grundlagen

Im Zentrum der Anomalieerkennung steht die Auswertung von Daten mittels eines statistischen Verfahrens. Dafür müssen einige grundlegende Begriffe der Statistk erläutert und anschließend Informationen zur Erhebung der untersuchten Metriken besprochen werden.

Eine Möglichkeit Daten besser zu verstehen, ist die Datenobjekte anhand von deren Merkmalen und Verteilungen zu beschreiben. Ein Merkmal ist eine Eigenschaft eines Objekts, welches sich entweder von Objekt zu Objekt, oder zu verschiedenen Zeitpunkten unterscheiden kann [18]. Als relevantes Beispiel kann die Motivation als Merkmal betrachtet werden. Die Motivation von Entwickler A kann sich zu der von Entwickler B unterscheiden. Ebenso kann die Motivation von Entwickler A vor Projektstart anders als während des Projekts sein.
Zur statistischen Auswertung müssen Merkmale von Objekten mit Symbolen oder Zahlen assoziiert werden [18]. Unterschiedliche Arten von Merkmalen werden daher in verschiedene Kategorien unterteilt. Je nach Art des vorliegenden Datentyps werden entsprechend verschiedene Typen von Skalen, auch Skalenniveaus genannt, verwendet.

2.1.1 Datentypen und Skalenniveaus

Skalenniveaus dienen der Klassifizierung von Daten. Ein Überblick der vier Skalenniveaus ist in Tabelle 2.1 präsentiert. Des Weiteren kann man die Ska-

Skalenniveau	Beispiel	Zugelassene Operationen	Lagemaß
Nominalskala	Geschlecht, Postleitzahl	$=, \neq$	Modus
Ordinalskala	Dienstgrad, Schulnoten	$=, \neq, >, \geq, <, \leq$	Median
Intervallskala	Temperatur[Grad Celsius], Datum	$=, \neq, >, \geq, <, \leq, +, -$	Arithmetisches Mittel
Verhältnisskala	Temperatur[K], Preis in Euro	$=, \neq, >, \geq, <, \leq, +, -, \times, \div$	Geometrisches Mittel

Tabelle 2.1: Skalenniveaus (in Anlehnung an [18])

lenniveaus in kategorische (Nominal und Ordinal), oder numerische (Intervall und Verhältnis) Skalen unterteilen. Der größte Unterschied zwischen den zwei Typen ist die Zuordnung der Symbole. Numerische Merkmale werden immer durch Zahlen repräsentiert und erlauben deswegen eine größere Anzahl an zugelassenen Operationen, während kategorische Merkmale durch Symbole repräsentiert werden. Selbst wenn diese Symbole Ganzzahlen sein sollten, wie zum Beispiel bei einer Postleitzahl (Nominalskala), kann man mit diesen nicht wie mit Ganzzahlen umgehen [18]. Für diese Arbeit sind auf Grund der vorliegenden Datenmengen insbesondere Ordinalskalen und Intervallskalen relevant. Die Haupteigenschaft von Ordinalskalen ist, dass man zwar eine Ordnung auf diesen definieren, aber keine numerischen Operationen durchführen kann. Ein ‘sehr gut’ ist zum Beispiel besser als ein ‘gut’, aber ‘sehr gut’ mit einem ‘gut’ zu addieren ist im Kontext von Schulnoten nicht definiert. Intervallskalen hingegen erlauben diese Operationen (mit Einschränkungen). Man kann also nicht nur sagen, dass der 5. Februar weiter in der Zukunft liegt als der 1. Februar, sondern auch, dass dieser genau vier Tage weiter in der Zukunft liegt. Des Weiteren unterscheiden sich die zwei Skalen in Bezug auf die Berechnung ihres Lagemaßes. Das Lagemaß beschreibt das mittlere Niveau eines Merkmals [19]. Bei Ordinalskalen muss hierfür der Median verwendet werden, während bei Intervallskalen grundsätzlich das arithmetische Mittel benutzt wird (auch wenn der Median hier ebenfalls zugelassen ist). Das Lagemaß allein ist allerdings nicht ausreichend, um die Verteilung eines Merkmals zu beschreiben. Deswegen wird zusätzlich ein Streuungsmaß verwendet, welches die Streuung um das Zentrum beschreibt. Im Folgenden werden zwei verschiedene Streuungsmaße genauer diskutiert.

2.1.2 Varianz und Standardabweichung

Die Varianz ist das gebräuchlichste Streuungsmaß in der deskriptiven Statistik [19] und beschreibt die Streuung um das Zentrum eines Datensatzes. Die Varianz **v** ergibt sich aus der mittleren quadratischen Abweichung zum arithmetischen Mittel $\overline{x}$:

$$v = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.1}$$

Das hat zur Folge, dass die Einheit der Varianz nicht der ursprünglichen Einheit eines Merkmals, sondern dessen Quadrat entspricht. Wird zum Beispiel Körpergröße mit der Einheit Meter als Merkmal betrachtet, ergibt sich entsprechend daraus für die Einheit der Varianz Quadratmeter. Deswegen bedient man sich einer weiteren Größe, der Standardabweichung σ. Diese errechnet sich aus der positiven Wurzel der Varianz und hat dadurch die gleiche Einheit wie das Ursprungsmerkmal:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2.2}$$

Die Standardabweichung wird bei Normalverteilungen unter anderem auch als Mittel zur Beschreibung von Prozentilen verwendet. Die 68-95-99.7 Regel besagt, dass sich innerhalb von ein, zwei, oder drei Standardabweichungen um das Mittel jeweils 68%, 95%, oder 99,7% der Werte befinden[15]. Man kann also sagen, dass ein Merkmal, welches nicht innerhalb von drei Standardabweichungen um das Mittel liegt, nur mit einer Häufigkeit von 0.3% vorkommen sollte und somit selten ist. Diese Verteilungen sind in Abb. 2.1 visualisiert.

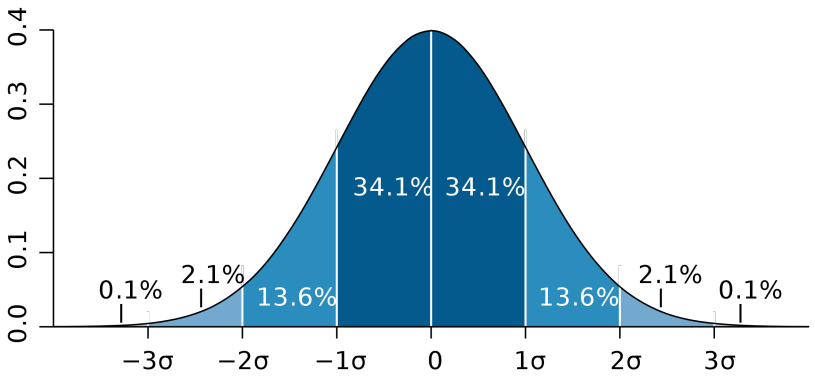

Abbildung 2.1: Normalverteilungen mit Standardabweichungen und entsprechenden Prozentilen[15]

Aus diesen Gründen wird häufig davon ausgegangen, dass in einem normalverteilten Datensatz grundsätzlich alle Werte innerhalb von 3 Standardabweichungen um das Mittel liegen sollten.
Dies findet auch Anwendung in der Identifizierung von Ausreißern. In der Auswertung von Datensätzen werden Objekte häufig als Ausreißer klassifiziert, wenn sie nicht innerhalb von $\overline{x} \pm 3\sigma$ liegen [10].

2.1.3 Mittlere absolute Abweichung vom Median (MAD)

Wird der Median $\overline{x}_{0.5}$ als Lagemaß verwendet, so wird oft als Streuungsmaß die mittlere absolute Abweichung vom Median berechnet [19], auch MAD genannt. Sie ist das Gegenstück der Standardabweichung für den Median und ergibt sich aus dem Mittel der Abweichung zum Median:

$$MAD = \frac{1}{n} \sum_{i=1}^{n} |x_i - \overline{x}_{0.5}| \tag{2.3}$$

2.2 Soziotechnische Metriken

Das Verständnis über die verwendeten Metriken und deren Erhebung ist essentiell für die Konzeptionierung des Anomalieerkennungsalgorithmus. Insgesamt wurden 28 Metriken untersucht [7], welche sich in sieben Kategorien unterteilen lassen. Zur Übersicht sind die Metriken im Folgenden aufgelistet:

1. **Kommunikation:** *FLOW Distance* [16], *FLOW Centralitzation* [6], Einzelgänger Score, Medienkanal Nutzung, Wahrgenommene Intensität.
2. **Meeting:** Dauer, Beteiligung, Anzahl.
3. **Stimmung:** Positiver Affekt, Negativer Affekt, Motivation, Druck.
4. **Teamgeist:** Balance, Teamgeist, Performance, Zufriedenheit, Arbeitsabhängigkeiten.
5. **Probleme:** Konflikte, Probleme
6. **Produktivität:** Fertiggestellte Issues, Fertiggestellte Story Points, Verpasste Issues, Verpasste Story Points, Geplante Issues, Geplante Story Points.
7. **Zufriedenheit:** Kundenzufriedenheit, Teamleiterzufriedenheit

Es handelt sich hierbei um bereits aufbereitete, teambezogene Daten. Der Großteil der dafür verwendeten Rohdaten wird manuell durch Fragebögen an die Teammitglieder erhoben. Allerdings gibt es auch Rohdaten, insbesondere produktivitätsbezogene, welche systemseitig in *Jira* erfasst werden.

2.2.1 Erfassung der Rohdaten durch Fragebögen

Bei der Erfassung der Daten durch Fragebögen ist die Art der Befragung für die spätere statistische Auswertung von großer Bedeutung.
Ein signifikanter Teil der Befragung erfolgt mit Hilfe von Likert-Skalen. Die Likert-Skala ist ein geläufiges, psychometrisches Hilfsmittel zum Erfassen von psychologischen Daten [4]. Sie setzt sich aus mehreren Fragen, sogenannten 'Items', vom Likert-Typ zusammen. Hierbei werden Partizipanten eines Fragebogens dazu aufgefordert, ihre Zustimmung zu einer Aussage einzustufen. Typischerweise werden Fünf-Punkt-Likert-Skalen verwendet, also Likert-Skalen bei denen es fünf Antwortmöglichkeiten gibt. Ein Beispiel einer solchen Likert-Skala, welche sich auf soziotechnische Faktoren bezieht, ist in Abbildung 2.2 zu sehen. Dieser Skalentyp findet auch Anwendung bei der Ermittelung der Stimmungsmetriken. Genauer wird hierfür *PANAS* (Positive and Negativ Affect Schedule) verwendet, welches anhand einer Fünf-Punkt-Likert-Skala bewertet wird. *PANAS* ist ein Fragebogen, bei dem die Befragten 20 (oder in kürzerer Ausführung nur zwölf [17]) verschiedene Gefühlslagen auf einer Fünf-Punkt-Skala einschätzen müssen. *PANAS* wurde als ein zuverlässiges Mittel zur Selbsteinschätzung des positiven und negativen Affekts bewertet [20].
Bei der Erfassung der Rohdaten für die Kommunikationsmetriken werden außerdem noch weitere Skalentypen verwendet. Diese müssen hier allerdings nicht genauer betrachtet werden, da sie für die Analyse der daraus resultierenden Metriken unproblematisch sind.

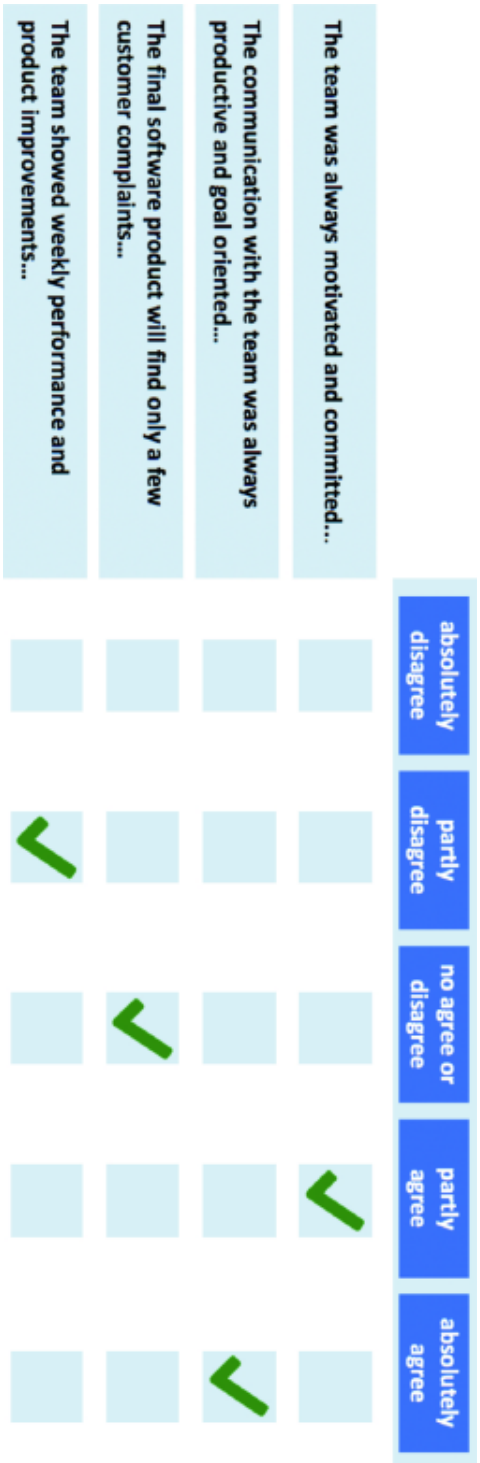

Abbildung 2.2: Beispiel einer Fünf-Punkt-Likert-Skala mit vier Likert-Typen [7]

2.3 Operationen auf Daten aus Likert-Skalen

Mit Hilfe der vorherigen Teilkapitel können nun Entscheidungen getroffen werden, wie mit den vorliegenden Daten umgegangen werden kann. Die Frage, ob Likert-Skalen vom Typ Ordinal oder Intervall sind, ist eine Frage, die seit mehreren Jahrzehnten diskutiert wird [11] und bei der es keinen eindeutigen Konsens gibt. Sollte diese eine Intervallskala sein, so können parametrische Methoden, wie die Standardabweichung, darauf angewandt werden. Ist diese jedoch vom Typ Ordinal, so dürfen ausschließlich nicht-parametrische Methoden, wie z.B. Prozentilberechnung, benutzt werden. Auch wenn Likert-Typen, wie in Abb. 2.2 zu sehen, zunächst eindeutig wie Ordinaldaten wirken, werden diesen trotzdem numerische Werte zugewiesen, meist in einem Bereich von eins bis fünf. Diese eignen sich dann zur Verarbeitung mit parametrischen Methoden, unter Gültigkeit der Annahme, dass die arithmetischen Abstände ihren tatsächlichen Abständen in der Wertigkeit der Antwortmöglichkeiten entsprechen. Damit lägen diese auf einer Intervallskala.

Es liegt eine Vielzahl an wissenschaftlichen Werken vor, die schlussfolgern, dass die Erkenntnisse aus der Analyse von Likert-Skalen nicht davon beeinflusst werden, ob parametrische oder nicht-parametrische Mittel verwendet werden [14][11][13]. Norman [14] schlussfolgert, dass parametrische Statistik auf Daten aus Likert-Skalen angewandt werden kann, auch wenn diese nicht normalverteilt sind, oder nur eine kleine Sample Size vorliegt. In späteren Kapiteln wird deutlich, dass insbesondere letzteres eine große Problematik für die Wahl des Anomalieerkennungsalgorithmus darstellt. Noch genauer schränken Joshi et al [4] ein, dass einzelne Likert-Typen als Ordinaldaten behandelt werden müssen, während Metriken, welche aus mehreren Likert-Typen zusammengesetzt sind, als Intervalldaten behandelt werden können. Somit wird, unterstützt von aktueller Literatur, in dieser Arbeit angenommen, dass Metriken, welche aus Likert-Skalen resultieren, wie Intervallskalen behandelt werden dürfen. Dadurch können diese Metriken einer größeren Auswahl an statistischen Auswertungsmethoden unterzogen werden.

2.4 Grundlagen zu Jira

Jira ist eine weit verbreitete Softwareanwendung zum Projektmanagement, welche überwiegend im agilen Software Engineering Anwendung findet. Laut der Entwicklerfirma *Atlassian* wird *Jira* weltweit von über 65.000 Unternehmen verwendet[1]. *Jira* wird in verschiedenen Varianten angeboten, in dieser Arbeit bezieht sich der der Begriff *Jira* allerdings nur auf die *Jira Software* für *Jira Server*.
Zu den größten Features von *Jira* gehört die Organisation von Projekten, insbesondere das Anlegen und Verwalten von Issues. Weitere Features sind Scrum und Kanban Boards, Roadmaps, Agile Reporting so wie Möglichkeiten zum Bugtracking. Eine Besonderheit von *Jira* ist die Individualisierbarkeit und Erweiterbarkeit für Unternehmen. *Atlassian* stellt ein offizielles Developer Kit, das *Atlassian SDK*, zur Verfügung, um die *Jira* Umgebung mit eigenen oder öffentlichen Plug-ins um weitere Funktionalitäten zu erweitern. Dieses wurde auch im Rahmen dieser Arbeit verwendet.

2.5 Verwandte Arbeiten

Diese Arbeit bezieht sich zu großen Teilen auf Forschung des *Fachgebiet für Software Engineering* der *Leibniz Universität Hannover*. Im Folgenden werden verwandte Arbeiten kurz dargestellt.

2.5.1 Arbeiten zu soziotechnischen Metriken

Die Anomalieerkennung arbeitet auf Grundlage von zusammengesetzten Metriken, von denen einige im Rahmen von wissenschaftlichen Arbeiten erörtert wurden. Schneider et al. [16] definierten 2015 erstmals den Begriff *FLOW Distance*, eine komplexe Kommunikationsmetrik, welche versucht, das Gefühl von Distanz in Entwicklerteams zu quantifizieren. In der Arbeit "*Media, Mood, and Meetings: Related to Project Success?*" [17] untersuchten Schneider et al. die *FLOW Distance* und weitere soziotechnische Metriken. Sie beschrieben die Relevanz dieser Metriken für Entwicklerteams und ermittelten Zusammenhänge zwischen diesen, der Teamstimmung und dem Projekterfolg. Die Autoren konnten diverse Korrelationen nachweisen und schlussfolgerten, dass frühe Indikatoren über Stimmung, Kommunikation und Zusammenarbeit späteren Problemen im Projekt entgegenwirken können.
Weiterführend untersuchten Klünder et al. [6] *FLOW Centralization*, eine Metrik, welche auf der *FLOW Distance* basiert. Die Autoren konnten zeigen, dass diese Metrik eine Wechselwirkung mit anderen sozialen Metriken aufweist. Zum Beispiel konnte eine Korrelation zwischen der *FLOW Centra-*

[1] Stand 2021 https://www.atlassian.com/software/jira/guides/use-cases/who-uses-jira

lization zum Mittelpunkt des Projektzeitraums und den sozialen Konflikten gegen Ende des Projekts gezeigt werden [6]. Die Untersuchung von *FLOW* Metriken als Hilfsmittel für die Kommunikationsanalyse zum Verständnis von Zusammenarbeit im Team ist weiterhin Thema aktueller Forschung [5].

2.5.2 ProDynamics

Das *Jira* Plug-in *ProDynamics* wurde im Rahmen vielzähliger wissenschaftlicher Studien entwickelt und untersucht. Es stellt die datentechnische und funktionale Basis für die Entwicklung dar, welche im Rahmen dieser Arbeit getätigt wird. Zu den Zielen des Plug-ins gehören die Erfassung, Verarbeitung und Visualisierung der soziotechnischen Metriken (wie in Kapitel 2.2), um Entwicklerteams in agilen Software Projekten zu unterstützen. Einen Überblick über diese Funktionalitäten präsentierten Kortum et al. [7] in ihrer Arbeit "*Towards a Better Understanding of Team-Driven Dynamics in Agile Software Projects*", in welcher sie außerdem sehr starke, statistisch signifikante Abhängigkeiten zwischen den Sprintdynamiken feststellen konnten. Den positiven Effekt des Feedbacks an die Entwicklerteams, welches vom *ProDynamics* Plug-in bereitgestellt wird, beschrieben Kortum et al. in einer weiteren Arbeit [8]: Bei einer Untersuchung von 15 Projekten konnten sie mit statistischer Signifikanz feststellen, dass Teams mit Zugang zum Feedback des *ProDynamics* Plug-ins einen eindeutigen Anstieg in der Entwicklungsperformance aufwiesen. Der festgestellte Erfolg des Feedbacks war die Hauptmotivation zur Erstellung eines proaktiven Feedbackmechanismus, mit dem sich diese Bachelorarbeit auseinandersetzt. Die Autoren konnten zeigen, dass Entwickler aus bereitstehenden Informationen Wissen extrahieren und entsprechende Handlungsänderungen aufweisen. Um sicherzustellen, dass die Entwickler wichtige Informationen nicht übersehen, sollen sie über markante Verläufe in den zugehörigen Metriken durch ein Anomaliebenachrichtigungssystem informiert werden.

Kapitel 3

Konzeptionierung und Planung

In diesem Kapitel wird das Vorgehen bei der Erarbeitung des Konzepts vorgestellt. Dafür werden die Überlegungen, welche im Rahmen der Konzeptionierung der Anomalieerkennung und des Benachrichtigungssystem angestellt wurden, erläutert und diskutiert.

3.1 Vision und Grundphilosophie der Arbeit

Hauptziel des zu entwickelnden Plug-ins ist es, einen proaktiven Feedbackmechanismus im Falle von auftretenden Anomalien bereitzustellen. Dieser soll die Entwicklerteams durch Benachrichtigungen auf untypisches Verhalten, basierend auf soziotechnischen Projektdaten, hinweisen. Hierbei sind sowohl negative, als auch positive Anomalien relevant. Sollten negative Verhaltensmuster auffallen, so können diese korrigiert werden bevor sie zu einem Problem führen. Aber auch Benachrichtigungen über positive Anomalien können hilfreich sein. Sollte das Team zum Beispiel neue Maßnahmen einführen, um die Stimmung im Team zu verbessern, kann diese Entscheidung über einen positiv resultierenden Ausschlag in den Stimmungsmetriken bekräftigt werden. Allerdings werden negative Anomalien grundsätzlich als schwerwiegender eingeschätzt, da diese meistens eine größere Handlungsnotwendigkeit für das Team darstellen.
Die akzeptable Menge an Benachrichtigungen an ein Team ist ebenfalls ein zentrales Diskussionsthema der Konzeptionierung. Ein Risiko ist hierbei, dass Entwickler bei einer zu großen Anzahl an Benachrichtigung diese ignorieren oder deaktivieren. Aus diesem Grund wurde priorisiert False Positives zu vermeiden, während eine gewisse Menge an False Negatives akzeptiert werden muss. Dadurch folgt für das Team, dass die zugestellten Benachrichtigungen von hohem Wert sind, da sie nur Informationen über markante, eindeutige Anomalien beinhalten.
Das System kann jedoch keine Verbesserungsvorschläge oder Handlungsempfehlungen an das Team aussprechen. Dies ist zum einen durch die individuelle

Arbeitsweise von Entwicklerteams, und zum anderen durch grundlegende Limitierungen einer solchen Systemerweiterung gegeben.

3.2 Was ist eine Anomalie?

Die zentrale Frage, die geklärt werden muss, ist wann ein Verhalten als Anomalie gilt. Im Allgemeinen ist eine Anomalie in Datensätzen als ein Objekt definiert, welches sich signifikant von den restlichen Objekten unterscheidet. Es ist allerdings schwer objektive Richtlinien dafür zu definieren, wann genau ein Datenpunkt als Anomalie klassifiziert werden soll. Die vorliegenden Datensätze sind 'unlabeled'. Es liegt also kein Richtwert vor, welcher eine eindeutige Klassifizierung der Datenpunkte ermöglicht. Ein mögliches Merkmal für solch einen Richtwert wäre die Kundenzufriedenheit. Diese hängt jedoch von zu vielen Faktoren ab, um daraus Rückschlüsse über Anomalien in einzelnen Metriken ziehen zu können. In solchen Fällen erfolgt die erstmalige Klassifizierung von Datensätzen oft durch die subjektive Einschätzung von Personen. Daher wurde eine Heuristik festgelegt, die das Klassifizieren von Anomalien erleichtern soll: Je länger der Verlauf einer Metrik stabil ist, also geringe Abweichungen aufweist, desto eher sollte eine Abweichung als Anomalie klassifiziert werden. Dabei ist eine Abweichung definiert als eine Differenz zu den bisherigen Werten in positiver oder negativer Richtung. Je stärker diese Abweichung ist, desto markanter ist eine Anomalie. Anomalien, welche als markant eingeschätzt werden, sollen unter allen Umständen vom Algorithmus erkannt werden.

3.3 Exploration der Daten

Bevor sich der Konzeptionierung des Anomalieerkennungsalgorithmus gewidmet werden kann, müssen die vorliegenden Daten verstanden werden. Für die Datenexploration wurde die Anwendung *Weka*[1] genutzt, welche eine Vielzahl an Tools zur Visualisierung und Analyse von Datensätzen bereitstellt. Damit konnte schon früh festgestellt werden, dass eine Erkennung auf individueller Entwicklerebene, auf welcher die Rohdaten vorliegen, nicht zielführend wäre. Eine Analyse auf individueller Ebene würde sicherlich die Anzahl an False Negatives reduzieren, da diese nicht durch Mittlung auf die Teamebene kaschiert werden können. Allerdings stammt ein Großteil der vorliegenden Daten aus Likert-Typen und basiert somit auf einer Ordinalskala, was diese ungeeignet für parametrische Verfahren macht. Dadurch sind wahre Ausreißer schwer, bis gar nicht, zu erkennen. Innerhalb eines entwicklerbezogenen Datensatzes liegen pro Metrik nicht genug Daten vor, um Machine Learning Algorithmen zu trainieren. Sollte man etwa ab der

[1] https://www.cs.waikato.ac.nz/ml/weka/

vierten Projektwoche Anomaliebenachrichtigungen ermöglichen wollen, so liegen zu diesem Zeitpunkt nur drei Datenpunkte vor, was Machine Learning Algorithmen ausschließt. Ebenso sind nicht-parametrische statistische Methoden zur Erkennung von Ausreißern, wie etwa Prozentilberechnung, bei kleinen Datensätzen nicht effektiv. Hierfür müssen umfangreichere Datensätze vorliegen, welche mehrere Monate umfassen. Dies könnte die Gesamtdauer von einigen Softwareprojekten übersteigen, was eine proaktive Benachrichtigung nicht möglich macht.
Eine Möglichkeit mit größeren Datensätzen zu arbeiten, ist es die Entwicklerdaten teamübergreifend zusammenzuführen und als einen einzigen Datensatz zu betrachten. Dies ist allerdings aus mehreren Gründen problematisch. Zum einen können sich Entwickler stark voneinander unterscheiden, was es schwer macht eine Norm, und somit eine entsprechende Abweichung von dieser, zu definieren. Zum anderen können auf Grund des vorliegenden Wertebereichs ([1, 2, 3, 4, 5]) vieler Metriken in einigen Wochen alle möglichen Werte angenommen werden. Das macht eine Klassifizierung von Ausreißern beinahe unmöglich. Diese Schwierigkeit ist in Abb. 3.1 mit *Weka* visualisiert.

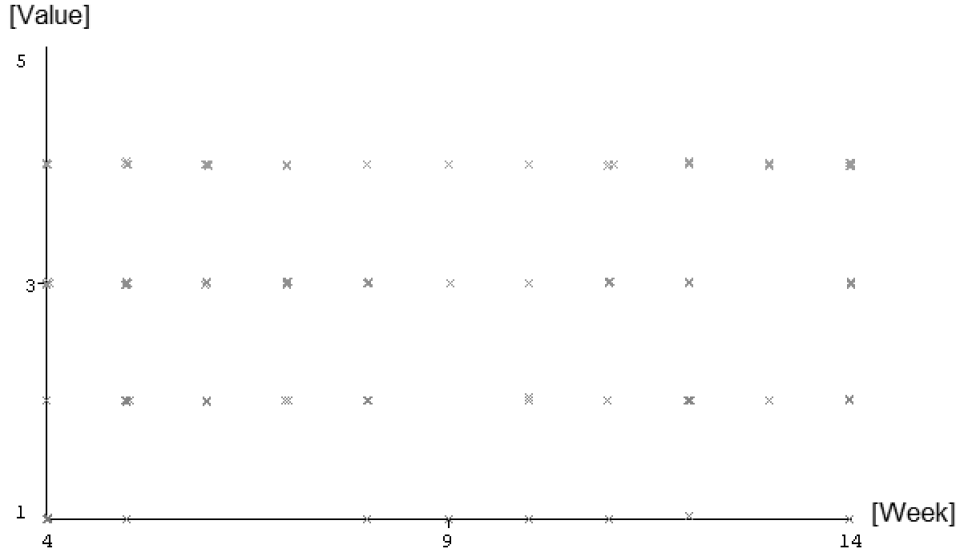

Abbildung 3.1: Entwicklerbezogener Datensatz einer Metrik über ein gesamtes Team (Jitter zur besseren Visualisierung). In den meisten Wochen sind fast alle möglichen Werte verteten.

Aggregierte Metriken auf Teamebene haben im Vergleich dazu einige Vorteile. Zwar verbergen sie teilweise individuelle Ausreißer im Team, allerdings wurden False Negatives in der Vision des Projekts als weniger problematisch eingestuft. Dafür ist eine Anomalie auf Teamebene aussagekräftiger über das Gesamtverhalten und somit relevanter für alle Teammitglieder. Auf Teamebene ist der Datensatz in jedem Fall zu klein, um Machine Learning

Algorithmen darauf zu trainieren. Allerdings können parametrische Verfahren in diesem besonderen Fall darauf angewendet werden (siehe Kapitel 2.3). Letztlich sind die aggregierten Metriken selbst aussagekräftiger als die Rohdaten, was es den Teams erleichtern könnte diese zu interpretieren und auf deren Basis zu handeln.
Basierend auf der Datenexploration und den daraus resultierenden Erkenntnissen zu den Daten wird die Anomalieerkennung auf Basis von wöchentlichen, teambezogenen Daten konzeptioniert. Dabei soll der Algorithmus nur Datensätze einzelner Teams analysieren. Ein Vergleich zwischen verschiedenen Entwicklerteams ist auf Grund der höchst individuellen Arbeitsweise von Teams ausgeschlossen.

3.4 Grundlegender Funktionsmechanismus

Die Konzeptionierung des Anomalieerkennungsalgorithmus startete mit der Wahl des grundlegenden Funktionsmechanismus. Der Ansatz der Outlier Detection diente als Ausgangspunkt für die Konzeptionierung.

3.4.1 Nachteile von Machine Learning

Zu Beginn der Konzeptionierungsphase wurde eine Lösung angestrebt, welche auf Machine Learning basiert. Im Bereich der Outlier Detection sind Machine Learning Algorithmen, wie etwa k-nearest Neighbors oder Neural Networks, in vielen Fällen ein sehr effektives Hilfsmittel [2]. Allerdings zeigte die Datenexploration, dass die vorliegenden Daten, auf Grund der geringen Sample Size und des kleinen Wertebereichs, nicht für solche Algorithmen geeignet sind. Auch eine Erweiterung der Datensätze durch Betrachtung mehrerer Entwickler oder mehrerer Teams wäre nicht zielführend. Daher wird stattdessen ein statistisches, univariates Verfahren als Funktionsmechanismus eingesetzt.

3.4.2 Vorteile von statistischen Verfahren

Es wurde ein simples und effektives Verfahren zur Outlier Detection gewählt, welches auf der Standardabweichung basiert. Eine gängige Praktik, um Ausreißer in einem Datensatz zu identifizieren, ist zu berechnen, ob ein Objekt innerhalb von drei Standardabweichungen um das arithmetische Mittel (oder den Median) liegt [10]. Allerdings hat dieses Verfahren auch inhärente Schwächen. Leys et al. [10] beschreiben, dass sowohl das arithmetische Mittel, als auch die Standardabweichung, selbst zu stark von Ausreißern beeinflusst werden, um diese effektiv zu identifizieren. Die Autoren empfehlen daher die mittlere absolute Abweichung um den Median zu verwenden, da diese nicht von einzelnen Ausreißern beeinflusst werden. Allerdings ist die genannte Problematik bei den vorliegenden Datensätzen zu vernachlässigen,

da nur ein relativ kleiner Wertebereich vorliegt. In den meisten Fällen liegen selbst die größtmöglichen Ausreißer (z.B. '1' oder '5') numerisch sehr nah am Rest des Datensatzes, weswegen die Standardabweichung von solchen nicht stark beeinflusst wird. Außerdem ist eine Beeinflussung des Streuungsmaßes in einigen Fällen sogar vorteilhaft (siehe Kapitel 3.5). Die Art des Lagemaßes wurde je nach Metrik, anhand dessen Effektivität für die Anomalieerkennung, gewählt. Im Falle von kontinuierlichen Daten wurde das arithmetische Mittel genutzt, während bei diskreten Daten der Median genutzt wurde.
Das Verfahren wird außerdem univariat, also pro Metrik, angewandt. Dies eignet sich gut für den vorliegenden Kontext, da man die Teams somit auf Anomalien in den einzelnen Metriken hinweisen kann.

3.5 Entwurf des Erkennungsalgorithmus

Ausgehend von der Grundfunktionalität der Outlier Detection mit Hilfe der Standardabweichung wurden diverse Anpassungen und Erweiterungen vorgenommen, um den Erkennungsalgorithmus zu entwerfen. Dafür wurden zwei besonders datenreiche Projekdatensätze aus dem *Softwareprojekt* genauer betrachtet und für die Konzeptionierung genutzt. In frühen Iterationen klassifizierte der Algorithmus noch deutlich zu viele Werte als Anomalie. Bei der Anpassung wurde daher vor allem Wert darauf gelegt, die Anzahl der klassifizierten Anomalien möglichst weit zu reduzieren, ohne dabei markante Anomalien zu übersehen.

3.5.1 Optimierung des σ-Faktors

Die Anzahl an Standardabweichungen um das Mittel, von jetzt an σ-Faktor genannt, in welchen ein Wert liegen muss um nicht als Anomalie gewertet zu werden, ist der gröbste Faktor zur Sensibilisierung des Algorithmus. Wie bereits erwähnt ist eine Faustregel dafür $\sigma = 3$, allerdings ist dieser Wert auf Grund der vorliegenden Wertebereiche ungeeignet. Mit $m \pm 3\sigma$ würden Anomalien oft unmöglich werden, da dieses Intervall häufig aus dem möglichen Wertebereich ausbricht. Ein Beispiel hierfür ist in Abb. 3.2 präsentiert. Zu sehen ist ein gradueller Anstieg in einer Metrik. Solch ein Verlauf wird als relativ normal eingeschätzt, allerdings gibt es einen verhältnismäßig starken Anstieg in Woche sechs. Es zeigt sich, dass das $\pm 3\sigma$-Intervall schnell aus dem Wertebereich, welcher für diese Metrik zwischen eins und fünf liegt, ausbricht. Die Verwendung von 3σ hat sich auch darüber hinaus als zu ungenau herausgestellt.
Ein σ-Faktor von zwei hat sich hingegen über alle Metriken als geeigneter erwiesen. Das $\pm 2\sigma$-Intervall, ebenfalls in Abb. 3.2 zu sehen, bleibt über den gesamten Verlauf im gültigen Wertebereich. Außerdem klassifiziert es den starken Anstieg in Woche sechs als Anomalie.

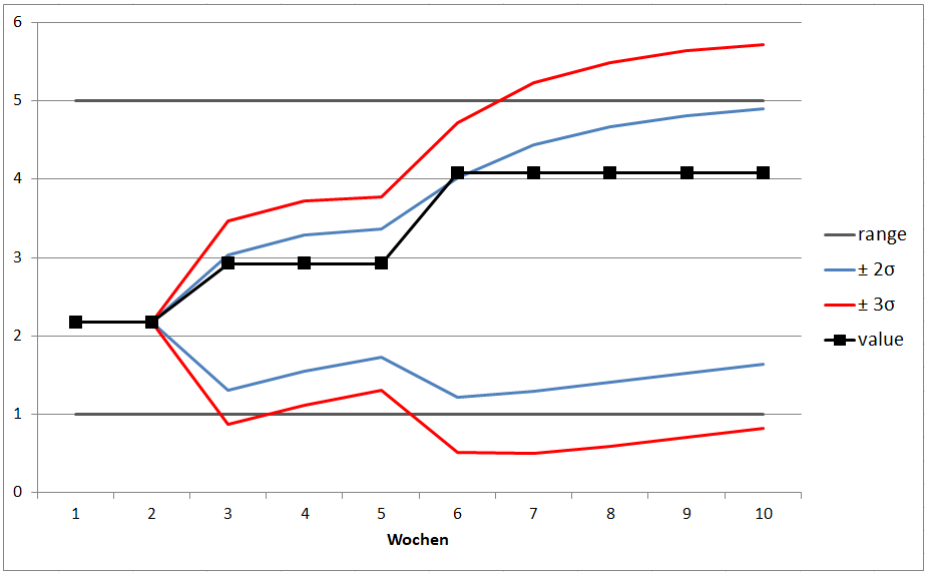

Abbildung 3.2: Verlauf einer Metrik über 10 Wochen. Intervall mit σ-Faktor = 3 bricht aus dem Wertebereich aus.

3.5.2 Positive und negative Gewichtung

Negative Anomalien, also solche, die einen außergewöhnlich schlechten Verlauf in einer Metrik signifizieren, sollen schon bei geringerer Ausprägung erkannt werden, als positive. Je nach Metrik kann eine negative Anomalie entweder durch einen besonders hohen oder tiefen Wert entstehen. Ein repräsentatives Beispiel hierfür sind der positive und negative Affekt. Beim positiven Affekt signalisiert ein besonders hoher Wert eine positive Anomalie, während dies beim negativen Affekt eine negative Anomalie signalisiert. Daher sollen je nach Metrik unterschiedliche σ-Faktoren für Negativ- und Positivgrenze des Normintervalls genutzt werden. Somit ist der Algorithmus weniger sensibel gegenüber positiven Anomalien, aber kann diese dennoch in extremeren Fällen erkennen. Es sind ausschließlich die Metriken von dieser Gewichtung betroffen, die eine kontextuell eindeutige Interpretation erlauben. Die betroffenen Metriken sind in Tabelle 3.1 aufgelistet. Alle nicht aufgelisteten Metriken wurden als neutral bewertet und mit einem Intervall von $\pm 2\sigma$ um das Lagemaß analysiert.

Interpretation (hoher Wert)	Betroffene Metriken	Norm-intervall
Positiv	Positiver Affekt, Kundenzufriedenheit, Teamgeist, Teamleiterzufriedenheit, Motivation, Beteiligung, Performance	$[m - 3\sigma, m + 2\sigma]$
Negativ	Negativer Affekt, FLOW Centrality, FLOW Distance, Einzelgänger Score, Verpasste Issues/Story Points, Probleme, Konflikte	$[m - 2\sigma, m + 3\sigma]$

Tabelle 3.1: Negativ oder positive gewichtete Metriken mit zugehörigen Normintervall (m steht für das jeweilige Lagemaß der Metrik).

3.5.3 Einbeziehen von Anomalien im Streuungsmaß

Basierend auf Voranalysen wurden sowohl vergangene Anomalien, als auch der Datenpunkt der aktuellen Woche (ebenfalls eine potentielle Anomalie) in die Berechnung der Standardabweichung miteinbezogen. Wie bereits erwähnt wird die Standardabweichung von Ausreißern in einem Datensatz beeinflusst. Dies hat allerdings zwei wünschenswerte Effekte für den Erkennungsalgorithmus, welche anhand des Beispiels in Abb. 3.3 genauer erklärt werden können.

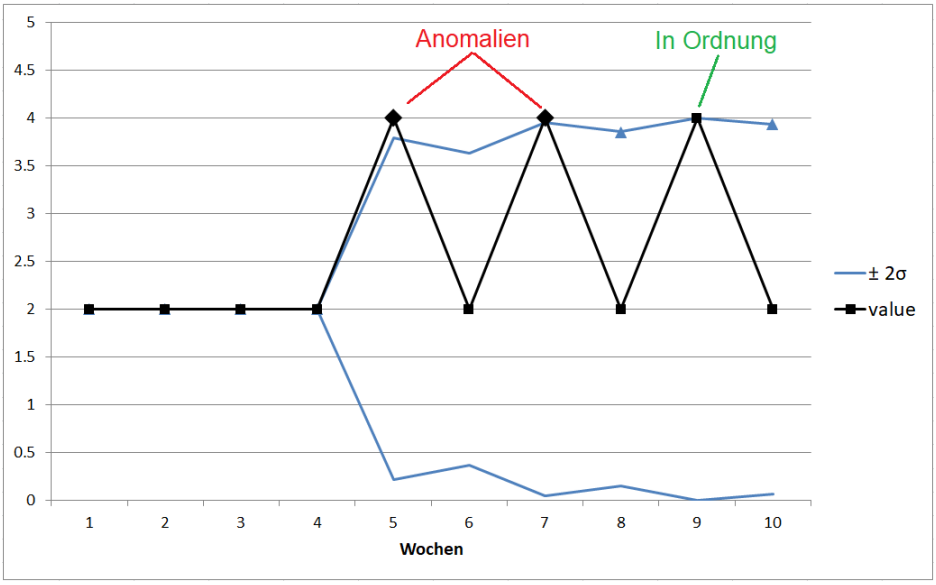

Abbildung 3.3: Plötzlich auftretende, starke Schwankungen. Anomalien Beeinflussen die Standardabweichung, der Median hingegen ist unbeeinflusst.

Zum einen müssen Anomalien aus der Vergangenheit für die Entscheidung über zukünftige Anomalien miteinbezogen werden, um Verhaltensmuster erkennen zu können. Zeigt das Team einen plötzlichen Verhaltenswechsel, wie in Woche fünf, der danach aber konsistent weitergeführt wird, muss man dies als Verhaltensmuster betrachten, welches nicht mehr als anomal gilt. Würden die Anomalien nicht mit in das Streuungsmaß einbezogen werden, so würde das Team fortan in jeder zweiten Woche über eine Anomalie benachrichtigt werden. Stattdessen wird ab der dritten Wiederholung des Musters dieses nicht mehr als Anomalie bewertet. Zum anderen würde ein unbeeinflusstes Streuungsmaß, wie die mittlere absolute Abweichung, immer den ersten Wert, welcher von den bisherigen Werten abweicht, als Anomalie klassifizieren. Die MAD in Woche fünf aus Abb. 3.3 wäre null, da die Mehrheit der bisherigen Werte diese Abweichung hat. Hier wäre also auch die kleinstmögliche Abweichung automatisch eine Anomalie. Die Standardabweichung hingegen würde in diesem Fall mit Einbezug der aktuellen Woche einen Wert von bis zu 3,5 erlauben.

3.5.4 Weitere konzeptionelle Adjustierungen

Weitere Maßnahmen, die vorgenommen wurden, um eine Verringerung der als Anomalie klassifizierten Datenpunkte zu erreichen, ohne markante Anomalien zu übersehen, sind hier aufgelistet.

Eingewöhnungszeit: Die ersten drei Wochen (für welche Datenpunkte vorliegen) werden in keinem Fall als Anomalie gewertet, da zu diesem Zeitpunkt noch keine zuverlässigen Annahmen über das Normalverhalten gemacht werden können. Grundsätzlich ist eine längere Eingewöhnungszeit, und ein somit größerer Datensatz, immer von Vorteil. Allerdings umfassen die studentischen Projekte des *Softwareprojekts*, die hier als Konzeptionierungsgrundlage verwendet wurden, maximal 14 Wochen. Daher führt eine kürzere Eingewöhnungszeit in diesem Fall zu einem deutlich größeren Zeitraum mit proaktivem Feedbackmechanismus.

Ignorieren von Nullwerten: Dies ist nur relevant für Metriken welche ‘0’ als Wert annehmen können. Diverse Gründe können dazu führen, dass Metriken einen Nullwert annehmen. Sollten etwa die geplanten oder fertiggestellten Issues in einer Woche null sein, so ist wahrscheinlich, dass das Team *Jira* in dieser Woche nicht genutzt hat, oder zumindest keine Daten bereitgestellt hat. Dies kann zum Beispiel durch Urlaube oder Weiterbildungen passieren. Solche Werte führen zu einer Verfälschung der Daten. Sie werden daher nicht miteinbezogen und gelten auch nicht für die Eingewöhnungszeit.

Median für diskretisierte Metriken: Einige Metriken liegen zur besseren Interpretierbarkeit des Teams in diskretisierter Form vor. Für diese Metriken hat sich der Median als ein effektiveres Lagemaß für die Anomalieerkennung herausgestellt, als das arithmetische Mittel.

Keine Back-to-Back Anomalien: Für den Fall, dass eine Metrik plötzlich rasant ansteigt, brauchen das Lage- und Streuungsmaß mindestens eine Woche, um sich auf das neue Niveau einzustellen. Daher wird bei zwei aufeinanderfolgenden Ausreißern in derselben Metrik der Zweite nicht mehr als Anomalie gewertet. Das Team wurde nach der ersten Anomalie schon auf den Trend hingewiesen und sollte sich dessen bewusst sein. Eine weitere Benachrichtigung wird daher als überflüssig bewertet. Dieser Sachverhalt kann zum Beispiel oft beim Wechsel von der Explorationsphase in die erste Iteration auftreten. Eine Metrik die von diesen Umständen oft betroffen ist sind die fertiggestellten Story Points. Ein Beispielverlauf dieser ist in Abb. 3.4 dargestellt.

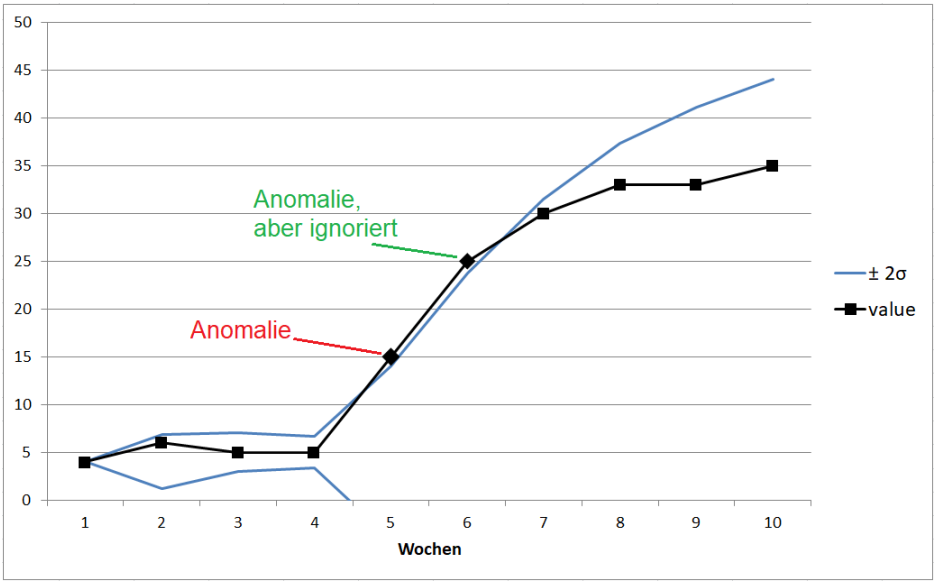

Abbildung 3.4: Bei plötzlichem Anstieg werden zwei aufeinanderfolgende Wochen als Anomalie erkannt. Zweitere wird in diesem Fall ignoriert.

3.5.5 Effektivität der Maßnahmen

Alle in Erwägung gezogenen Anpassungen wurden anhand mehrerer Projektdatensätze überprüft. Es wurde keine Anpassung genannt oder umgesetzt, die eine Kaschierung von markanten Anomalien zur Folge hatte. Die iterative Erhöhung der Genauigkeit, welche aus den Maßnahmen resultierte, ist an einem 14-wöchigen Projektdatensatz eines *Softwareprojekts* in Tabelle 3.2 demonstriert.

Hinzugefügte Maßnahme	Erkannte Anomalien
Keine (Alle Datenpunkte)	298
Normintervall $m \pm 2\sigma$	56
Positive und negative Gewichtung	43
Einbeziehen von Anomalien im Streuungsmaß	19
Eingewöhnungszeit	12
Ignorieren von Nullwerten	9
Median für diskretisierte Metriken	7
Keine Back-to-Back Anomalien	6

Tabelle 3.2: Effektivität der Maßnahmen auf Projektdatensatz

Es ist zu erwähnen, dass die Anomalien gehäuft in einzelnen Wochen auftreten und die Anzahl der erkannten Anomalien nicht der Anzahl der Wochen mit enthaltener Anomalie entsprechen.

3.6 Aufbau des Benachrichtigungssystems

Die Form, in welcher die Teams über Anomalien benachrichtigt werden, war ebenfalls Diskussionselement der Arbeit. Anfangs war eine graphische Komponente zur Benachrichtigung geplant, welche dem Team Anomalien aus der letzten Woche anzeigt. Allerdings stellt solch eine graphische Benutzeroberfläche in diesem Fall keinen großen Wissensgewinn für das Team dar. Sollte das Team sich für den zeitlichen Verlauf einzelner Metriken interessieren, so besteht solch eine Ansicht bereits als Teil des *ProDynamics* Plug-ins.
Außerdem besteht das Risiko, dass die Teammitglieder eine Anomaliebenachrichtung in dieser Form zu leicht übersehen oder ignorieren können. Anomalien traten tendenziell, bemessen an Studentendaten, in Wochen auf, in denen das Team nicht sehr aktiv gearbeitet hat oder *Jira* nicht aktiv genutzt wurde. Daher wäre es auch unwahrscheinlich, dass die Anomaliebenachrichtigung innerhalb von *Jira* wahrgenommen werden würde.
Eine Benachrichtigung über E-Mail an den Projektleiter wird daher als sinnvollere Lösung angesehen. Personen haben bereits etablierte, individuell effektive Benachrichtigungsmechanismen für eingehende E-Mails. Dadurch

sind diese schwerer zu übersehen als etwa systeminterne Benachrichtigungen in *Jira* und ermöglichen somit einen effektiveren Feedbackmechanismus. Sollte sich die Projektleitung von den Benachrichtigungen gestört fühlen, so besteht die Möglichkeit die Anomaliebenachrichtigungen über einen Direktlink innerhalb der E-Mail abzubestellen. Somit ist ein guter Mittelweg zwischen Zuverlässigkeit und Unaufdringlichkeit gewährleistet. Eine E-Mail mit Anomaliebenachrichtigung enthält dabei folgende Komponenten (siehe Abb. 3.5):

1. **Einleitung**: Die Einleitung soll der Projektleitung den Kontext der E-Mail nahelegen, sowie um welches ihrer Projekte es hierbei geht.

2. **Auflistung der Metriken**: Hauptteil der Benachrichtigung ist die Information über die aufgetretenen Anomalien. Diese werden mit ihrer jeweiligen Ausschlagsrichtung aufgelistet.

3. **Information über potentielle Interpretierbarkeit**: Auch wenn keine wirklichen Handlungsempfehlungen ausgesprochen werden können, werden eindeutig interpretierbare Metriken (siehe Tabelle 3.1) zusätzlich kommentiert. Dies soll insbesondere bei negativen Anomalien eine Notwendigkeit zur Evaluation der Anomalie vermitteln.

4. **Ansicht zu zeitlichem Verlauf der Metriken**: Ein Direktlink zur graphischen Ansicht der Metriken im jeweiligen Projekt erleichtert es der Projektleitung, die Anomalien unverzüglich zu evaluieren. Da nicht alle Metriken selbsterklärend sind, wurde diese Ansicht außerdem um detaillierte Tooltips erweitert, welche die Metriken erklären. Dadurch kann die Bedeutung einer Anomalie leichter vom Team interpretiert werden.

5. **Link zum Abbestellen**: Sollte sich der Empfänger von den Benachrichtigungen gestört fühlen, oder möchte diese aus anderen Gründen nicht mehr erhalten, so soll es möglichst einfach sein, diese abzubestellen. Daher wird diese Funktionalität direkt mittels eines Links in der E-Mail angeboten. Der Link deaktiviert nur die Benachrichtigungen für das entsprechende Projekt.

Sehr geehrte(r) ***<Name des Projektleiters>***,

in Ihrem Jira Projekt ***<Name des Projekts>*** sind in der letzten Woche eine oder mehrere Anomalien im Teamverhalten aufgefallen.
Dies muss nicht zwingend etwas Negatives sein, aber als Projektleiter(in) empfiehlt es sich, die Anomalien in den folgenden sozio-technischen Metriken zu evaluieren: 1

- ***<Metrik 1>*** war ***<höher/tiefer>*** als erwartet.
- ***<Metrik 2>*** war ***<höher/tiefer>*** als erwartet. *(Dies ist meistens ein unerwünschter Trend)* 2 3

Den Verlauf dieser Metriken in den letzten Wochen können Sie in folgender Übersicht einsehen: *<Link zur Ansicht des zeitlichen Verlaufs>* 4

Möchten Sie in Zukunft keine Anomaliebenachrichtigungen mehr erhalten, so klicken Sie bitte auf folgenden Link: *<Link zum Abbestellen>* 5

Abbildung 3.5: Anomaliebenachrichtigung in E-Mail Form

3.7 Anforderungen an die Entwicklung

Den letzten Teil der Konzeptionierung stellt die Anforderungsanalyse an den Entwicklungsprozess dar, um das Konzept möglichst effektiv umzusetzen. Das Konzept unterliegt wenig expliziten Anforderungen. Bis auf eine hohe Zuverlässigkeit gab es keine inhärenten Qualitätsaspekte, die bei der Konzeptionierung beachtet werden mussten. Eine Vielzahl an Usability Qualitätsaspekten sind auf Grund der automatisierten Arbeitsweise des Konzepts, oder dem Mangel von Benutzerinteraktionen sowie einer grafischen Benutzeroberfläche, nicht relevant.
Da das Konzept allerdings als Erweiterung eines bestehenden *Jira* Plug-ins umgesetzt wird, entstehen einige implizite Anforderungen an die Entwicklung, um eine einheitliche Integration zu gewährleisten:

- **R1 (Modularität)**: Das Programm soll im Rahmen eines neuen Moduls in die bestehende Programmstruktur integriert werden.
- **R2 (Testbarkeit)**: Die Funktionalität des Moduls muss separat testbar sein.
- **R3 (Zuverlässigkeit)**: Das Modul darf nicht fälschlicherweise oder mehrfach E-Mails an Projektleiter verschicken, auch nicht während des Entwicklungsprozess.
- **R4 (Fehlertoleranz)**: Das Modul muss damit umgehen können, dass in Zukunft Metriken umbenannt, hinzugefügt oder entfernt werden könnten.
- **R5 (Sicherheit)**: Alle REST-Calls müssen verschlüsselt über eine Secure REST Call Methode durchgeführt werden.
- **R6 (Dokumentation)**: Der Code muss gut lesbar, verständlich geschrieben und ausreichend kommentiert sein.
- **R7 (Sprache)**: Alle Texte im Frontend müssen dem *i18n* Internationalisierungsstandard entsprechen.

Kapitel 4

Entwicklung und Integration

Dieses Kapitel stellt vor, wie das Konzept letztendlich entwickelt und als Plug-in in *Jira* integriert wurde. Dafür wird auf die verwendeten Technologien, sowie Maßnahmen und Besonderheiten bei der Entwicklung eingegangen.

4.1 Verwendete Technologien

Es wurde eine Vielzahl von Technologien für die Entwicklung genutzt. Die relevantesten sind im Folgenden aufgelistet:

- **ProDynamics**: Das bestehende *ProDynamics* Plug-in ist unverzichtbar für die Funktionalität des entwickelten Konzepts. Zum einen ist es stark von der Datenerhebung abhängig, welche von diesem getätigt wird. Zum anderen wurden bestehende Klassen des Plug-ins genutzt und erweitert, um die Funktionalitäten zu implementieren. Ebenso greift es auf die zugehörigen Datenbankstrukturen zu. Das Plug-in hat einen modularen Aufbau, teilt also die verschiedenen Funktionalitäten in eigene Module auf. Das Konzept zur Anomalieerkennung und Benachrichtigung wurde daher ebenfalls als neues Modul des Plug-ins umgesetzt, um Anforderung **R1** zu erfüllen.
- **Atlassian Plugin SDK**: Ein notwendiges Software Development Kit zur Entwicklung von *Jira* Plug-ins. Das SDK basiert auf *Java* und dem *Spring* Framework und wird bereits vom *ProDynamics* Plug-in genutzt. Für die Entwicklung wurde aus Kompatibilitätsgründen die Version 6.3.10 genutzt, da *ProDynamics* ebenfalls in dieser Version entwickelt wurde. Das SDK ermöglicht unter anderem das einfache Hinzufügen von Komponenten, die Nutzung von *Jira* Services, sowie Datenbankstrukturierung und Zugriff durch *Active Objects*.

- **Active Objects**: *Active Objects* sind Datenstrukturen, die bei Ausführung des Systems erstellt werden. Sie erleichtern den Zugriff auf die verbundene Datenbank. Dabei werden Klassen, welche auf Datenbankelemente zugreifen müssen, mit den *Active Objects* als Konstruktionsparameter initialisiert. Innerhalb der Klassen können Funktionen dann mit SQL-Queries Datenbankeinträge abrufen.
- **REST**[1]: *REST*, kurz für Representational State Transfer, ist eine Technologie zum 'stateless' Austausch von Web Ressourcen zwischen Client und Server. Stateless bedeutet hierbei, dass alle Informationen, die für den Client notwendig sind, um diese zu verstehen, mitgesendet werden. Der Client muss sich also nicht in einem bestimmten Zustand befinden, um diese korrekt zu interpretieren. Web Ressourcen können in diesem Kontext alle möglichen Informationen sein, welche der Client verstehen kann (z.B. *JSON* Responses, XML Dokumente).
  Das gesamte Plug-in nutzt *REST*-Calls für sämtliche Kommunikation zwischen Server, Client und Datenbank. Dafür erfordern folgende Kommunikationsarten eine kurze Erklärung:
  - **GET**: Fordert Daten vom Server an.
  - **POST**: Fordert den Server dazu auf, Daten zu akzeptieren.
  - **CRUD**: Create, read, update and delete. Ein Sammelterm für die Interaktionen die mit persistenten Daten getätigt werden können.

4.2 Umsetzung und Integration

Dieses Kapitel beschreibt, wie das erarbeitete Konzept effektiv realisiert wurde.

4.2.1 Umsetzung des Algorithmus

Um eine zuverlässige Benachrichtigung an die Projektleiter zu gewährleisten, was auch eine Vermeidung fälschlicher Benachrichtigungen einschließt, durchläuft der Algorithmus über den gesamten Prozess mehrere konditionelle Verzweigungen. So wird die Anomalieerkennung gar nicht erst ausgeführt, wenn der Projektdatensatz nicht mindestens eine Metrik enthält, für welche Datenpunkte von über drei Wochen vorliegen. Des Weiteren kann eine Benachrichtigung nur verschickt werden, wenn sowohl Anomalien der letzten Woche gefunden wurden, als auch E-Mail Benachrichtigungen für das Projekt nicht vom Projektleiter deaktiviert wurden. Der gesamte Prozess ist durch ein Aktivitätsdiagramm in Abb. 4.1 dargestellt. Auf die Beinhaltung der Testfunktion wird in späteren Unterkapiteln näher eingegangen.

[1] https://restfulapi.net/, letzter Zugriff Jan. 2021

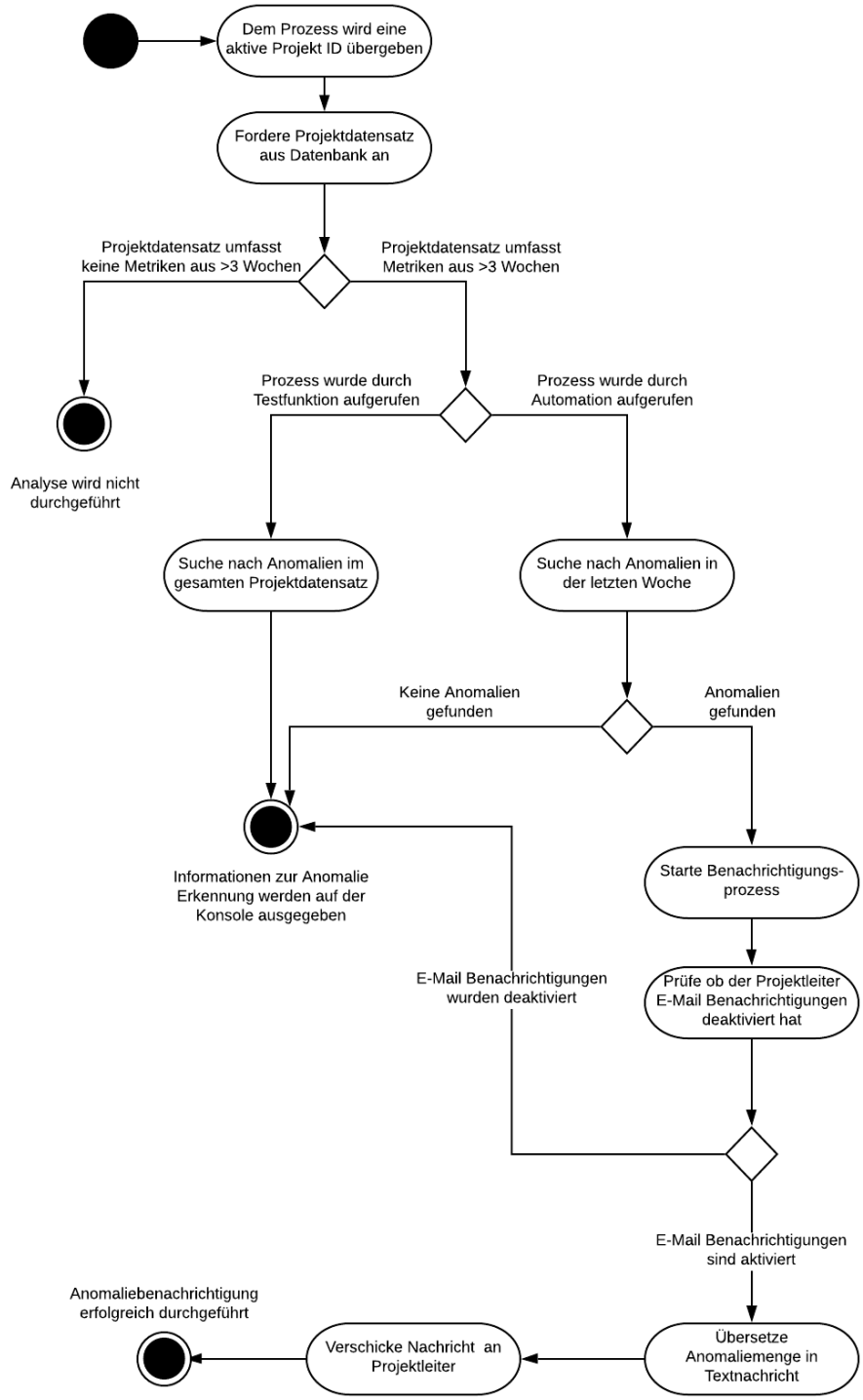

Abbildung 4.1: Aktivitätsdiagramm des Programmablaufs

4.2.2 Iteratives Entwicklungsmodell

Bei der Entwicklungen wurde nach einem iterativen System vorgegangen. Dadurch konnte eine Testbarkeit und funktionale Korrektheit der grundlegenden Teilfunktionalitäten sichergestellt werden, bevor diese von späteren Prozessschritten genutzt werden. Die genannten Systemkomponenten sind in Abb. 4.2 dargestellt und werden in Kapitel 4.2.3 weiter ausgeführt.

1. **Datenstrukturen vorbereiten**: Als erstes wurde sichergestellt, dass die relevanten Projektdaten für den Erkennungsalgorithmus korrekt aus der Datenbank angefordert und in geeignete Datenstrukturen transformiert werden. Dafür wurde der *Anomaly Data Handler* geschrieben, welcher auf die *Analytics Week Rest* Ressource zugreift.

2. **Anomalieerkennung implementieren**: Nachdem die notwendigen Datenstrukturen bereitstanden, konnte die Anomalieerkennung, wie erarbeitet in Kapitel 3, implementiert und getestet werden. Dies geschieht innerhalb des *Anomaly Finder*.

3. **Benachrichtigungssystem**: Für den Benachrichtigungsprozess mussten zuerst die *Leader Project Rest* Ressource und das zugehörige *Active Object* erweitert werden, um die Opt-Out Funktionalität zu ermöglichen. Der *Anomaly Mail Handler* startet dabei den Benachrichtigungsprozess, indem er mit einer Menge an Anomalien und einer Projekt ID von dem *Anomaly Data Handler* aufgerufen wird. Nachfolgend soll dieser die notwendigen Daten des Projektleiters aus der *Leader Project Rest* Ressource anfordern, und zusammen mit den Anomalien an den *Anomaly Mail Builder* übergeben. Dieser übersetzt alle relevanten Daten zu einer lesbaren Nachricht wie beschrieben in Kapitel 3.6. Diese wird an den *Anomaly Mail Handler* zurückgegeben, welcher die Nachricht per E-Mail an den Projektleiter verschickt (solang dieser Benachrichtigungen nicht abbestellt hat).

4. **Einbindung in die Automatisierung**: Damit das Modul so funktioniert, wie es in der Praxis eingesetzt werden soll, musste es schließlich in den wöchentlichen Automatisierungsprozess von *ProDynamics* eingebunden werden.

4.2.3 Integration in Plug-in Struktur

Die Anomalieerkennung und potentielle Benachrichtigungen sollen in wöchentlichem Rhythmus stattfinden. *ProDynamics* besitzt bereits mehrere Tasks, welche zu Wochenbeginn automatisiert ausgeführt werden. Das entwickelte Modul wird daher ebenfalls von dieser Automatisierung angestoßen, um eine korrekte Integration in das Plug-in sicherzustellen. Wie genau die entwickelten Komponenten mit dem *ProDynamics* Plug-in und den *Jira* Services interagieren ist Abb. 4.2 dargestellt. Dafür müssen folgende Komponenten genauer erläutert werden:

- **Anomaly Data Handler:** Wird mit einer Projekt ID aufgerufen, um den gesamten Prozess für ein Projekt zu starten. Ruft mittels *REST*-Call die notwendigen Projektdaten aus der Datenbank ab und transformiert diese in Datenstrukturen, welche für den Algorithmus besser nutzbar sind. Ruft anschließend den *Anomaly Finder* und, wenn nötig, den *Anomaly Mail Handler* auf.
- **Anomaly Finder**: Identifiziert Anomalien in einem Projektdatensatz nach dem erläuterten Vorgehen aus Kapitel 3 und gibt diese zurück.
- **Anomaly Mail Handler & Builder**: Übersetzen die Anomaliemenge in eine verständliche Nachricht und senden diese als E-Mail an den jeweiligen Projektleiter.
- **Leader Project REST Ressource**: Greift auf das *LeaderProject Active Object* zu. Zuständig um die relevanten Daten zum Projektleiter abzurufen. Dies beinhaltet auch das Lesen und potentielle Ändern eines *Opt-Out*-Flags, welches Ausschlag darüber gibt ob E-Mails an den Projektleiter versendet werden sollen oder nicht.
- **Analytics Week REST Ressource**: Greift auf das *AnalyticsWeek Active Object* zu. Liest die wöchentlichen Werte der Metriken eines Projekts aus, welche der *Anomaly Finder* benötigt.
- **E-Mail**: Die E-Mail wird vom E-Mail Server des *Jira* Systems versendet. Sie ist nicht direkter Teil der Programmstruktur, kann aber trotzdem durch POST Anfragen (Opt-Out-Link) mit dieser interagieren.
- **Socio-technical Metrics View**: Eine Komponente des *ProDynamics* Plug-ins, welche eine Ansicht über den zeitlichen Verlauf einzelner Metriken im Projekt gewährleistet. Die Ansicht wurde erweitert, um detailliertere Erklärungen für die Metriken bereitzustellen und kann durch einen Direktlink per E-Mail aufgerufen werden.

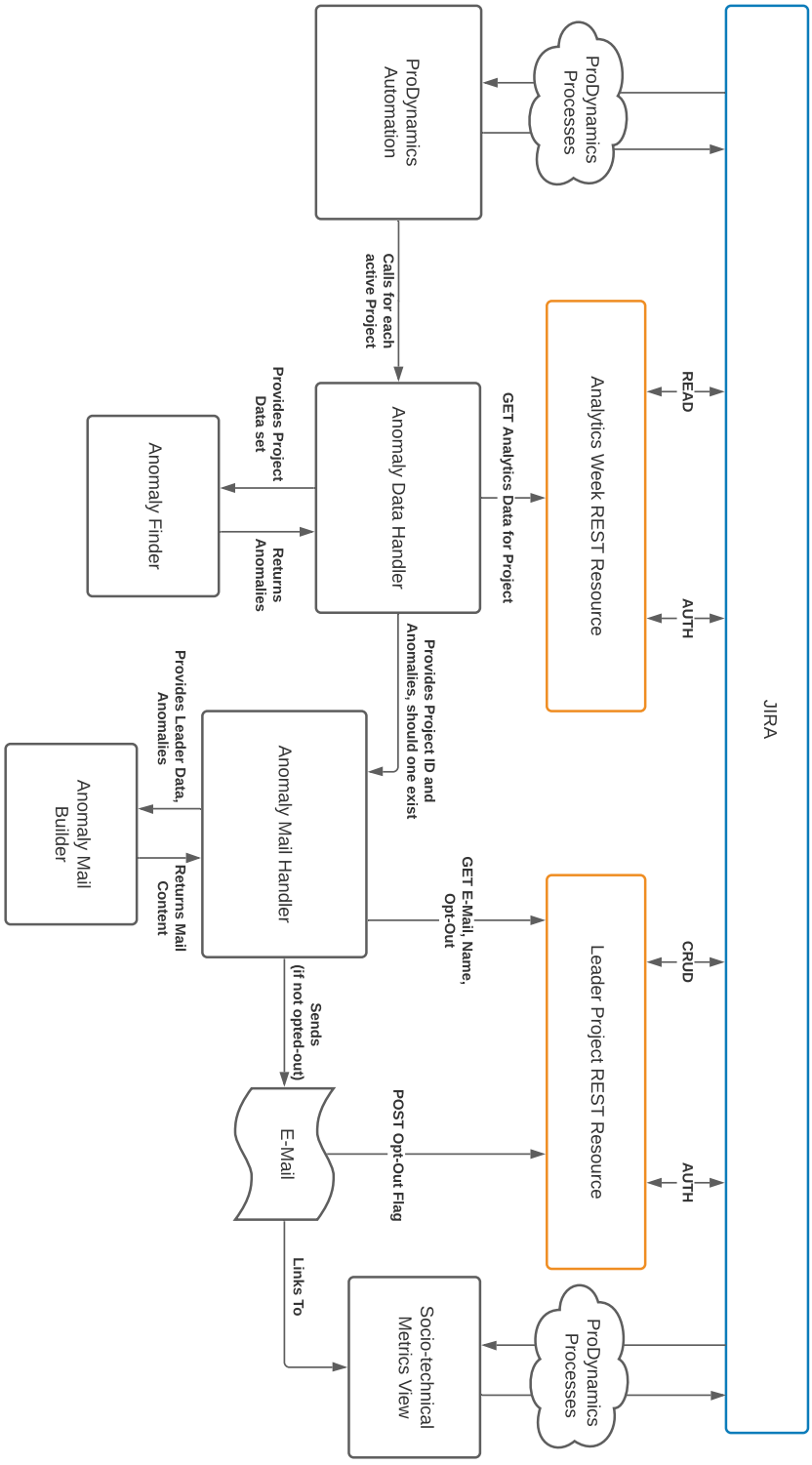

Abbildung 4.2: Überblick der Systemarchitektur

4.3 Besonderheiten während der Entwicklung

Bei dem Entwicklungsprozess gab es einige Besonderheiten, welche hier aufgeführt werden.

4.3.1 Iterative Entwicklung und Testen mit Atlassian SDK

Eine Eigenschaft der Entwicklung für *Jira Server* ist, dass sämtliche Programmfunktionalitäten nur bei laufendem Server ausgeführt und getestet werden können. Dies stellte eine Schwierigkeit für die iterative Entwicklung dar. Die einzelnen Funktionalitäten benötigen Zugriff auf die *Active Objects*, welche erst bei Ausführung des Systems erstellt werden. Daher können diese nicht separat durch Testing Frameworks wie *JUnit* effizient getestet werden. Dennoch musste funktionale Korrektheit der einzelnen Programminkremente sichergestellt werden, bevor die Gesamtfunktionalität fertiggestellt war. Daher wurde das Testen einzelner Programmfragmente durch manuelle Funktionsaufrufe aus dem Frontend durchgeführt.
Diese Funktionalität blieb auch nach Fertigstellung des Moduls noch erhalten, um Anforderung **R2** gerecht zu werden und ist in Abb. 4.1 zu sehen. Dadurch kann weiterhin aus der Administrationsansicht des Plugins die Anomalieerkennung getestet werden, ohne etwa Benachrichtigungen auszulösen. In der Automatisierung wird die Anomalieklassifizierung nur für Datenpunkte aus der letzten Woche durchgeführt. Der Testaufruf hingegen berichtet Anomalien innerhalb des gesamten Projektdatensatzes. Dies ermöglicht eine separate Testbarkeit, und somit auch eine wesentlich leichtere Anpassbarkeit, des Anomalieerkennungsalgorithmus.

4.3.2 Maßnahmen zur Einhaltung der Anforderungen

In Kapitel 3.7 wurden Anforderungen an die Entwicklung gestellt. Diese Anforderungen wurden durch diverse Maßnahmen eingehalten, von denen einige, noch nicht adressierte, im Folgenden genauer erläutert werden:

- **R3 (Zuverlässigkeit):** Abb. 4.1 zeigt, dass mehrere Konditionen erfüllt werden müssen, bevor eine E-Mail verschickt werden kann. Des Weiteren wurden während der Entwicklung sämtliche Projektleiter E-Mail Adressen von vergangenen Projekten gelöscht und durch eine Adresse zum Testen ersetzt. Dadurch konnte sichergestellt werden, dass auch während der Entwicklung nicht fälschlicherweise E-Mails an Personen verschickt wurden.

- **R4 (Fehlertoleranz):** Das Modul muss mit Änderungen der soziotechnischen Metriken umgehen können. Daher werden sämtliche Strings, die als Metriktyp aus der Datenbanktabelle erhalten werden, als Metrik akzeptiert und in ein entsprechendes Datenobjekt transformiert. Da Metriken jedoch trotzdem je nach Typ unterschiedlich vom Algorithmus behandelt werden müssen, werden diesen bei der Umwandlung in ein *Metric* Datenobjekt Attribute zugeordnet. Dies geschieht durch zwei Switch Cases, welche überprüfen, ob die zu erstellende Metrik einer bekannten Metrik entspricht. Der erste Switch Case überprüft, ob die erstellte Metrik eine positive bzw. negative Gewichtung (siehe Kapitel 3.5.2) hat. Der zweite Case überprüft, ob die Metrik eine diskretisierte Metrik ist (relevant für Berechnung des Lagemaßes). Alle nicht bekannten Metriken werden als ungewichtete, kontinuerliche Metriken interpretiert. Dadurch kann auch ohne eine Anpassung des Systems weiterhin eine Funktionalität bei Änderung, Hinzufügen, oder Entfernen von Metriken gewährleistet werden. Sollten diese neue Metriken ebenfalls eine der genannten Eigenschaften besitzen, so können sie in die bestehenden Switch Cases mit aufgenommen werden.
- **R7 (Sprache):** Das *ProDynamics* Plug-in wird momentan von deutsch-, englisch-, und spanischsprachigen Entwicklern genutzt. Bei der Einhaltung des *i18n* Internationalisierungsstandards wurde daher darauf geachtet, alle drei Language Property Dateien für jeden, im Frontend hinzugefügten, Text zu erweitern. Auf Grund mangelnder Spanischkenntnisse wurden jedoch die Texte der entsprechenden spanischen Datei ebenfalls in Englisch verfasst.

Kapitel 5

Evaluation

In diesem Kapitel wird das entwickelte Anomaliebenachrichtigungskonzept evaluiert. Dabei wird zuerst der Erkennungsalgorithmus ausgewertet und anschließend eine Evaluation des Gesamtsystems besprochen.

5.1 Auswertung des Erkennungsalgorithmus

Nachdem der Erkennungsalgorithmus konzeptioniert und entwickelt wurde, muss dieser noch ausgewertet werden. Dafür werden zunächst dessen Stärken und Limitationen anhand von Beispielen aus, zuvor nicht in der Konzeption betrachteten, *Softwareprojekt* Daten präsentiert. Nachfolgend werden die durchschnittlichen Resultate bei Anwendung auf die studentischen Projekte diskutiert. Diese stellen weitergehend eine Vergleichsbasis für die anschließende Evaluation anhand zweier Industrieprojekte dar. Basierend auf den Resultaten werden potentielle Erweiterungen für das System abgeleitet.

5.1.1 Stärken und Limitationen

In diesem Unterkapitel werden die Stärken und Schwächen des Erkennungsalgorithmus anhand von Studentenprojekten qualitativ analysiert und durch Beispiele aus diesen demonstriert.

Resistenz gegenüber False Positives

Durch die eingeführten Maßnahmen zur Erhöhung der Genauigkeit (siehe Kapitel 3.5.4) konnten False Positives weitestgehend eliminiert werden, was in der Vision des Projekts als besonders wichtig eingeschätzt wurde. Dies erfordert, dass das genutzte Verfahren resistent gegenüber natürlichen Schwankungen ist, welche in einem Projekt auftreten können ohne eine Anomaliebenachrichtigung zu rechtfertigen. Abb. 5.1 zeigt die Akzeptanz von anfänglichen Abweichungen (links), so wie natürlichen Schwankungen (rechts) ohne diese als Anomalie zu klassifizieren.

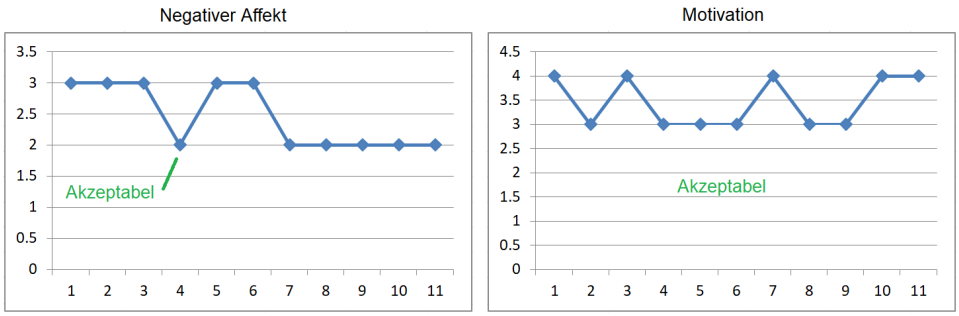

Abbildung 5.1: Resistenz gegenüber False Positives.

Erkennung von markanten Anomalien

Durch Sichtung der Datensätze konnte festgestellt werden, dass der Algorithmus alle als markant eingeschätzten Anomalien korrekt identifiziert. Dies umfasst starke Abweichungen von den bisherigen Datenpunkten, aber auch leichtere Abweichungen, wenn der bisherige Verlauf stabil war. Abb. 5.2 zeigt eine 'offensichtliche' Anomalie nach gewöhnlichem Verlauf (rechts), und eine Anomalie, die durch eine leichtere Abweichung nach höchst stabilem Verlauf zu Stande kommt (links).

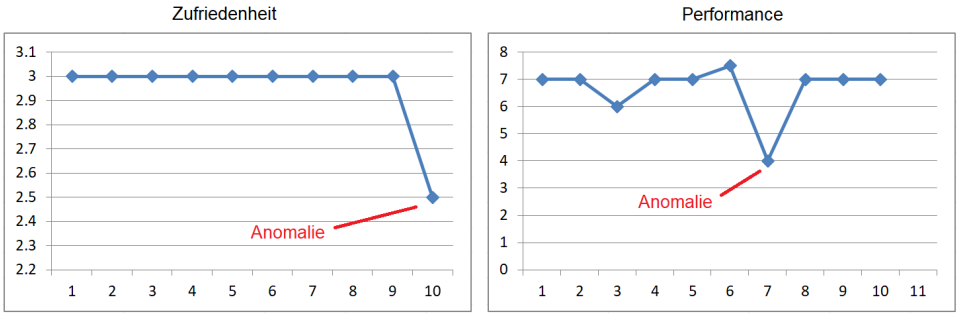

Abbildung 5.2: Markante Anomalien werden erkannt.

Grenzfälle

Für einige Verläufe kann der Erkennungsalgorithmus nur schwer Anomalien finden. Allerdings ist diskutabel, ob in solchen Fällen überhaupt Anomalien identifiziert werden können. Dies umfasst zum Beispiel starke, eskalierende Schwankungen (Abb. 5.3 links), bei denen der Algorithmus nur eine anfängliche Anomalie erkennen kann, bevor die Varianz im Datensatz zu groß wird, um weitere Datenpunkte als Anomalie zu klassifizieren. In solchen Fällen muss sich der Verlauf wieder für längere Zeit stabilisieren, bevor realistische Datenpunkte erneut als Anomalie klassifiziert werden können.

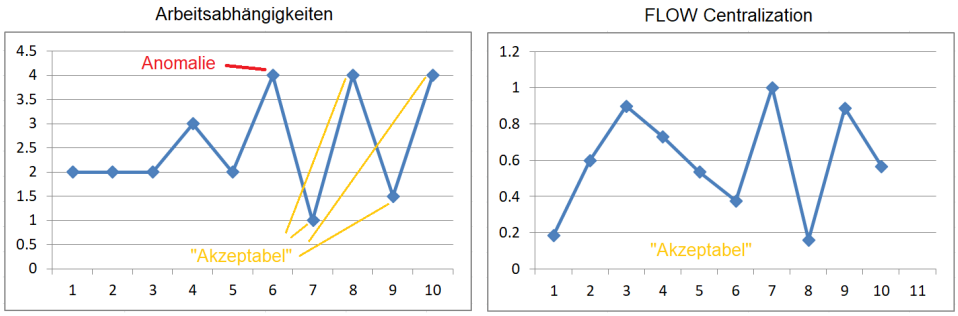

Abbildung 5.3: Einige Verläufe sind Grenzfälle für die Anomalieerkennung.

Ein weiterer Grenzfall ist das anfängliche Auftreten von sehr starker Varianz (Abb. 5.3 rechts) in einer Metrik. In solchen Fällen können spätere Anomalien ebenfalls nur nach langer Stabilität wieder erkannt werden. Allerdings muss womöglich akzeptiert werden, dass bei einer Metrik mit einem solch chaotischen Verlauf allgemein nur schwer ein Datenpunkt als Anomalie klassifiziert werden kann. In solchen Fällen muss starke Varianz als Normverhalten akzeptiert werden.

5.1.2 Quantitative Auswertung

Es liegt kein Datensatz mit eindeutiger Klassifizierung vor, anhand dessen eine quantitative Analyse der Erkennungsgenauigkeit durchgeführt werden kann. Stattdessen erfolgt eine quantitative Auswertung anhand der durchschnittlich erkannten Anomalien innerhalb von Studenten- und Industrieprojekten. Diese werden diskutiert und anschließend verglichen. Generell muss erwähnt werden, dass die Anzahl der erkannten Anomalien stark von der Anzahl der analysierten Metriken abhängt. Je mehr Metriken untersucht werden, desto höher ist die Wahrscheinlichkeit eine Anomalie zu finden.

Quantitative Auswertung anhand von Studentenprojekten

Die größte Datenbasis stellen die Projektdaten aus dem *Softwareprojekt* dar. Dennoch variiert die Qualität der Datensätze durch die unterschiedlich aktive Beteiligung der Studenten stark. Für die quantitative Auswertung wurden deswegen nur Projekdatensätze mit mehr als 180 Datenpunkten genutzt. Bei einer Menge von 28 analysierten Metriken entspricht dies durchschnittlich 6,5 Datenpunkten pro Metrik. Da der Algorithmus eine Eingewöhnungszeit von drei Wochen benötigt, sind noch kleinere Datensätze nur schwer zu bewerten. Insgesamt konnten dadurch zehn Projekte betrachtet werden, von denen zwei als ‘Trainingsdatensätze’ für die Konzeptionierung genutzt wurden und daher nicht in der Auswertung berücksichtigt werden. Die Ergebnisse aller Projektdaten sind in Tabelle 5.1 präsentiert.

Projekt ID	Datenpunkte	Erkannte Anomalien	Wochen mit Anomalie
*1**	*306*	*8*	*3*
*2**	*298*	*10*	*4*
3	295	7	3
4	279	7	2
5	278	4	2
6	278	14	5
7	250	12	3
8	204	3	1
9	194	12	3
10	192	5	2
∅	**246,3**	**8,0**	**2,6**

**(Trainingsdatensätze zur Anschaulichkeit mit aufgelistet, werden allerdings nicht in der Durchschnittsberechnung berücksichtigt)*

Tabelle 5.1: Ergebnisse der Anomalieerkennung in Studentenprojekten

Aus Tabelle 5.1 können folgende Erkenntnisse gewonnen werden:

- Es wurden im Durchschnitt **acht** Anomalien pro Projekt erkannt.
- Bei 28 analysierten Metriken entspricht dies **einer Anomalie in jeder dritten bis vierten Metrik** über die gesamte Projektdauer.
- Im Durchschnitt lagen 246,3 Datenpunkte vor, von denen **162,3 Datenpunkte zu klassifizieren** waren. Dies kommt daher, dass Datenpunkte aus den ersten drei Wochen niemals als Anomalie klassifiziert werden. Bei durchschnittlich 8 Anomalien bedeutet dies, dass **4,9% aller möglichen Datenpunkte als Anomalie klassifiziert** wurden. Dies entspricht pro Metrik ca. einer Anomalie alle 20 Wochen.

Des Weiteren kann festgestellt werden, dass pro Projekt im Durchschnitt 2,6 Wochen eine Anomalie enthielten. Daraus kann geschlossen werden, dass Anomalien eindeutig gehäuft auftreten. Durchschnittlich treten pro anomaler Woche Anomalien in ca. drei verschiedenen Metriken auf. Dies spiegelt die anfängliche Annahme wieder, dass Wechselwirkungen zwischen den soziotechnischen Metriken bestehen, wodurch diese gehäuft zeitnah zueinander auftreten.
Da die Benachrichtigungen auf wöchentlicher Basis stattfinden, bedeutet dies auch, dass ein Team im Schnitt 2,6 Benachrichtigungen über die gesamte Projektdauer von 14 Wochen erhalten würde.

Vergleich mit Industrieprojekten

Durch die Auswertung der Studentenprojekte wurde eine Vergleichsbasis geschaffen, die es erlaubt die Effektivität des Erkennungsalgorithmus bei Nutzung in Industrieprojekten einzuschätzen.

Im Rahmen der Arbeit lagen Projektdatensätze für zwei verschiedene Industrieprojekte vor. Das ist zwar nicht ausreichend, um definitive Aussagen über die Effektivität der Anomalieerkennung zu machen, gibt aber erste Hinweise für zukünftige Evaluationen. Die Projekte sind sowohl von der Datenpunktmenge, als auch von der Zeitspanne (maximal 20 Wochen) deutlich umfangreicher als die Studentenprojekte. Außerdem ist ein anderes Verhalten in einem professionellen Umfeld zu erwarten, als im Kontext einer Universitätsveranstaltung.
Die Ergebnisse des Erkennungsalgorithmus bei Anwendung auf die zwei Industrieprojekte sind in Tabelle 5.2 aufgelistet.

Projekt ID	Datenpunkte	Erkannte Anomalien	Wochen mit Anomalie
11	510	14	7
12	466	19	5
∅	**488**	**16,5**	**6**

Tabelle 5.2: Ergebnisse der Anomalieerkennung in Industrieprojekten

Aus Tabelle 5.2 können folgende weitere Erkenntnisse gewonnen werden:

- Bei Projekt 11 waren **3.3%** der möglichen Datenpunkte eine Anomalie.
- Bei Projekt 12 waren **5.0%** der möglichen Datenpunkte eine Anomalie.
- Im Durchschnitt tritt pro Metrik ca. eine Anomalie alle 24 Wochen auf. Dies ist geringfügig seltener als in den untersuchten Studentenprojekten.
- In Projekt 12 traten Anomalien deutlich gehäufter auf (**3,8 Anomalien pro anomaler Woche**) als in Projekt 11 (**2 Anomalien pro anomaler Woche**)

Alle Werte sind in Anbetracht der größeren Datenpunktmengen sehr ähnlich zu den Studentenprojekten. Sowohl die erkannten Anomalien, als auch die Wochen, welche eine Anomalie enthalten, sind ungefähr doppelt so hoch wie in den Studentenprojekten bei etwa doppelt so großer Datenmenge pro Projekt.

Vergleich mit laufenden *Softwareprojekt* Daten

Sämtliche bisher betrachteten *Softwareprojekt* Daten sind bereits abgeschlossene Projekte der vergangenen zwei Jahre. Die *Softwareprojekte*, die zeitgleich zum Bearbeitungszeitraum dieser Arbeit stattfanden, stellen eine besondere Untersuchungsgrundlage dar. Die Projekte verliefen während der *COVID-19* Pandemie, wodurch die Entwickler unter besonderen Umständen arbeiteten. Diese Projekte sind zum Zeitpunkt dieser Auswertung noch nicht vollständig abgeschlossen, wodurch nur zwei Projekte ausreichend Datenpunkte beinhalten um berücksichtigt zu werden. Die Ergebnisse sind in Tabelle 5.3 aufgeführt.

Projekt ID	Datenpunkte	Erkannte Anomalien	Wochen mit Anomalie
21	193	12	4
22	189	6	2
∅	**191**	**9**	**3**

Tabelle 5.3: Ergebnisse bei laufenden *Softwareprojekten*

Aus Tabelle 5.3 lässt sich, wie nach bisheriger Methodik, ableiten, dass **8.4%** der möglichen Datenpunkte als Anomalie erkannt wurden. Dies ist höher als die bisherig betrachteten Werte, jedoch ist eine Abweichung auf Grund der unvollständigen Projektdatensätze zu erwarten.
Insbesondere fällt jedoch auf, dass vier der zwölf Anomalien in Projekt 21 in Kommunikationsmetriken auftraten. Dies ist proportional deutlich höher als in den bisherig betrachteten Projekten.

5.2 Diskussion der Ergebnisse und potentielle Erweiterungen

Diskussion der quantitativen Auswertung

Die Anwendung des Erkennungsalgorithmus auf die Projektdaten vergangener studentischer Projekte lieferte vielversprechende Ergebnisse. Es gibt keine Referenzwerte die zum Vergleich herangezogen werden können, dennoch wird eine Anomalie alle 20 Wochen pro Metrik nach eigener Einschätzung als plausibel bewertet. Ebenso sind 2,6 Benachrichtigungen über eine Projektdauer von 14 Wochen eine akzeptable Menge. Im Durchschnitt erhält die Projektleitung alle 5,4 Wochen eine E-Mail, was nicht als exzessiv oder störend eingeschätzt wird. Sollte sich dennoch herausstellen, dass die Menge der Benachrichtigungen zu hoch ist, so kann diese durch Anpassungen leicht verringert werden. Beispielsweise besteht die Option, eine Mindestanzahl an Anomalien zu definieren, die als Voraussetzung für eine Benachrichtigung gilt.

Anhand der Ergebnisse bei Anwendung auf zwei Industrieprojekte ist zu erwarten, dass die Anomalieerkennung in Industrieprojekten ähnliche Resultate produziert wie bei Studentenprojekten. Damit wird das System auch für Industrieprojekte mit einer Projektzeitspanne von bis zu 20 Wochen als einsetzbar bewertet.
Letztlich lieferten aktuelle Projektdaten aus der *COVID-19* Pandemie interessante Ergebnisse. Da nur zwei unvollständige Projekte ausgewertet werden konnten, stellen diese keine zuverlässige Vergleichsbasis in Hinsicht auf durchschnittlich erkannte Anomalien dar. Jedoch konnte erkannt werden, dass eines der beiden Projekte überproportional viele Anomalien in Kommunikationsmetriken aufwies. Es könnte sich hier um ein untypisches Team handeln, allerdings sind die Ergebnisse auch durch die ungewöhnlich verteilte Arbeitsweise, die während des Projekts stattfinden musste, zu erklären.

Alle hier angestellten Bewertungen finden unter dem Vorbehalt statt, dass sämtliche diskutierten Werte aus Projekten ohne dem entwickelten Anomaliebenachrichtigungssystem stammen. Sollte der proaktive Feedbackmechanismus seine gewünschte Wirkung erzielen, so sind prinzipiell geringere Häufigkeiten für Anomalien in späteren Wochen eines Projekts zu erwarten.

Potentielle Erweiterung für Langzeitprojekte

Eine genauere Betrachtung von Industrieprojekt 12 zeigte, dass 14 der 19 erkannten Anomalien in produktivitätsbezogenen Metriken auftraten (Issues und Story Points geplant, fertig, oder verpasst). Diese Metriken hatten in diesem Industrieprojekt in der Tat deutlich untypischere Verläufe, als bisher in den Studentenprojekten gesehen. Der Verlauf einer dieser Metriken ist zur Veranschaulichung in Abbildung 5.4 dargestellt. Der Verlauf zeigt stark eskalierende Schwankungen, die sich nach dem Maximum wieder normalisieren. Es ist erkennbar, dass das Normintervall sich wieder verkleinert sobald die Schwankungen vorüber sind, allerdings noch sehr groß ist. Es ist schwer einzuschätzen wie gewöhnlich solch ein Verlauf in der Industrie ist und ob der Erkennungsalgorithmus für die kommenden Wochen durch die Schwankungen zu ungenau geworden ist. Allerdings lässt sich aus dem Verlauf eine Anforderung für Langzeitprojekte vermuten, welche bei den Studentenprojekten nicht zur Geltung kam:
In längeren Projekten besteht die Möglichkeit, dass sich die Verhaltensweisen von Entwicklerteams dauerhaft ändern können. Ein Team kann sich zum Beispiel nach einer längeren chaotischen Anfangsphase einfinden und ihr Verhalten dauerhaft stabilisieren. In diesem Fall sollten die Anfangswerte keinen großen Einfluss auf die Klassifizierung zukünftiger Punkte haben. In Studentenprojekten wurden auch ältere Datenpunkte gleichwertig in die

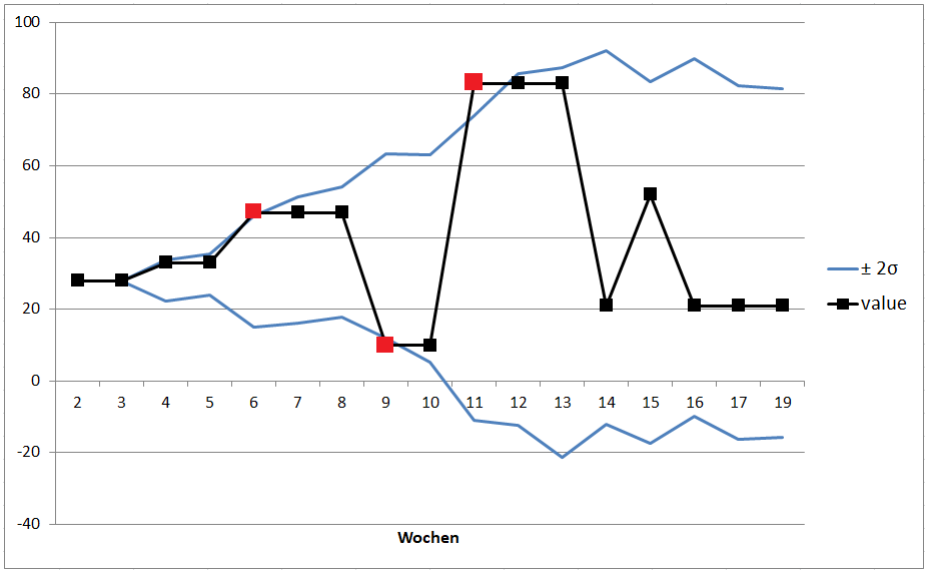

Abbildung 5.4: Verlauf der fertiggestellten Storypoints in einem Industrieprojekt.

Berechnungen miteinbezogen, um von vergangenem Verhalten auf zukünftiges Verhalten schließen zu können. Generell sind in längeren Projekten die neueren Datenpunkte allerdings relevanter für die Betrachtung, da in der Zwischenzeit eine Verhaltensänderung aufgetreten sein könnte.
Eine potenzielle Erweiterung des Systems, um besser für Langzeitprojekte (länger als 20 Wochen) geeignet zu sein, ist daher zeitlich neuere Werte höher zu gewichten. Dies kann entweder dadurch geschehen, dass nur Datenpunkte der letzten Monate betrachtet werden, oder eine kontinuierlich abfallende Gewichtung der Werte in die Vergangenheit stattfindet. Dabei könnte sowohl eine lineare, als auch eine stärkere Gewichtung (quadratisch, exponentiell, etc.) in Frage kommen. Um die Effektivität dieser Erweiterung zu testen stehen allerdings nicht genügend Datensätze von Langzeitprojekten zur Verfügung, weshalb auf die Implementierung dieser verzichtet wurde. Weitere Untersuchungen sind hierfür erforderlich.
Des Weiteren ist für längere Projekte ab einem gewissen Punkt ein Wechsel zu Machine Learning Algorithmen zur Outlier Detection vorstellbar. Sobald genug Datenpunkte vorliegen, können komplexere Algorithmen angewandt werden, um bessere Aussagen über zukünftige Verhaltensanomalien zu treffen. In diesem Fall würde der bisher besprochene Algorithmus nur für die ersten Monate Anwendung finden, um dem Team auch in früheren Projektphasen schon Feedback bereitstellen zu können.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel dient dazu, einen abschließenden Überblick über die Inhalte und die wichtigsten Ergebnisse der Arbeit zu geben.

6.1 Zusammenfassung

Das Verständnis über soziotechnische Faktoren bleibt ein anzustrebendes Ziel zur Erhöhung des Projekterfolgs in der Softwareentwicklung. In dieser Arbeit wurde ein Plug-in Modul für die Projektmanagement Software *Jira* entwickelt, welches diesem Ziel einen Schritt näher kommen soll. Dafür wurde ein proaktives Feedbacksystem konzeptioniert und entwickelt, welches Entwicklerteams, basierend auf soziotechnischen Metriken, auf Anomalien im Teamverhalten hinweist. Dieses System umfasst einen Anomalieerkennungsalgorithmus und einen zugehörigen Mechanismus zum Senden von Anomaliebenachrichtigungen.
Das Feedbacksystem wurde unter Verwendung des *Atlassian SDK* erfolgreich als Modul in das *ProDynamics Jira* Plug-in integriert. Dieses Plug-in ist unter anderem für die Erhebung der Daten, welche vom Anomalieerkennungsalgorithmus analysiert werden, zuständig und somit für die entwickelten Funktionalitäten unverzichtbar.
Auch wenn Machine Learning Algorithmen sich im Feld der Anomalieerkennung bewährt haben, waren diese für die vorliegenden Datensätze nicht geeignet. Der Erkennungsalgorithmus beruht daher auf einem parametrischen, univariaten, statistischen Verfahren und beinhaltet mehrere Adjustierungen, um Anomalien in soziotechnischen Metriken mit höherer Genauigkeit zu erkennen. Bei der Konzeptionierung des Erkennungsalgorithmus wurde hoch priorisiert, die Menge an erkannten Anomalien möglichst gering zu halten, ohne markante Anomalien zu übersehen. Der Algorithmus konnte den anfangs festgelegten Erwartungen gerecht werden.

Dieser produzierte bei der Analyse von Studentenprojekten zufriedenstellende Ergebnisse. Es wurden etwa 5% der möglichen Datenpunkte als Anomalie klassifiziert, ohne dabei markante Anomalien zu übersehen. Der Erkennungsalgorithmus wird zur Anwendung in Projekten im Rahmen des *Softwareprojekts* für funktional bewertet.
Proportional vergleichbare Ergebnisse wies ebenfalls die Analyse von zwei Industrieprojekten auf. Dies gibt erste Anzeichen dafür, dass das System auch für Projekte in professionellen Arbeitsumfeldern Anwendung finden kann. Außerdem konnten sich aus der Evaluierung der Industrieprojekte potentielle Erweiterungen des Systems ableiten lassen. Diese erfordern allerdings zur Untersuchung weitere Datensätze.
Das System benachrichtigt die Projektleiter per E-Mail über Anomalien in soziotechnischen Metriken aus der letzten Woche. Hierbei werden den Projektleitern Informationen zu den Anomalien, sowie ein Überblick der soziotechnischen Metriken bereitgestellt. Dadurch stehen den Entwicklerteams die notwendigen Mittel zur Verfügung, um eine direkte und effektive Evaluierung der Anomalien durchzuführen. Außerdem wird den Projektleitern eine unkomplizierte Möglichkeit angeboten, um die Benachrichtigung zu deaktivieren, sollten sie sich von diesen gestört fühlen. In Verbindung mit der Zielsetzung des Erkennungsalgorithmus, nur die markantesten Ausreißer als Anomalien zu klassifizieren, wurde somit ein unaufdringliches, proaktives Feedbacksystem entwickelt, welches Entwickler befähigt präventive Schritte gegen Leistungsminderung im Team einzuleiten.

6.2 Ausblick

Obwohl der Erkennungsalgorithmus vielversprechende Resultate zeigt, konnten bisher keine Evaluationen anhand von Langzeitprojekten (über 20 Wochen) durchgeführt werden. Für den Algorithmus kommen dabei potentielle Erweiterungen in Frage, die in der Arbeit diskutiert werden. Des Weiteren konnte im Zeitrahmen der Arbeit keine aktive Begleitung an einem *Softwareprojekt* stattfinden. Eine Evaluation basierend auf aktiver Begleitung mit Befragung der Entwickler würde aufschlussreiche Informationen über die Effektivität des entwickelten Feedbacksystems geben.

Literaturverzeichnis

[1] F. E. Emery, E. Thorsrud, and E. Trist. *Industrielt demokrati*. Universitetsforlaget, 1964.

[2] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[3] M. John, F. Maurer, and B. Tessem. Human and social factors of software engineering: Workshop summary. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, July 2005.

[4] A. Joshi, S. Kale, S. Chandel, and D. K. Pal. Likert scale: Explored and explained. *Current Journal of Applied Science and Technology*, pages 396–403, 2015.

[5] J. Klünder. *Analyse der Zusammenarbeit in Softwareprojekten mittels Informationsflüssen und Interaktionen in Meetings*. Logos Verlag Berlin GmbH, 2019.

[6] J. Klünder, K. Schneider, F. Kortum, J. Straube, L. Handke, and S. Kauffeld. Communication in teams-an expression of social conflicts. In *Human-Centered and Error-Resilient Systems Development*, pages 111–129. Springer, 2016.

[7] F. Kortum, O. Karras, J. Klünder, and K. Schneider. Towards a better understanding of team-driven dynamics in agile software projects. In *International Conference on Product-Focused Software Process Improvement*, pages 725–740. Springer, 2019.

[8] F. Kortum, J. Klünder, and K. Schneider. Behavior-driven dynamics in agile development: The effect of fast feedback on teams. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 34–43. IEEE, 2019.

[9] A. Law and R. Charron. Effects of agile practices on social factors. In *Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering*, HSSE '05, page 1–5, New York, NY, USA, 2005. Association for Computing Machinery.

[10] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[11] C. Mircioiu and J. Atkinson. A comparison of parametric and non-parametric methods applied to a likert scale. *Pharmacy*, 5(2):26, 2017.

[12] N. B. Moe, T. Dingsøyr, and T. Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.

[13] J. Murray. Likert data: what to use, parametric or non-parametric? *International Journal of Business and Social Science*, 4(11), 2013.

[14] G. Norman. Likert scales, levels of measurement and the "laws" of statistics. *Advances in health sciences education*, 15(5):625–632, 2010.

[15] P. Page. Beyond statistical significance: Clinical interpretation of rehabilitation research literature. *International journal of sports physical therapy*, 9:726–36, 10 2014.

[16] K. Schneider and O. Liskin. Exploring flow distance in project communication. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 117–118. IEEE, 2015.

[17] K. Schneider, O. Liskin, H. Paulsen, and S. Kauffeld. Media, mood, and meetings: Related to project success? *ACM Trans. Comput. Educ.*, 15(4), Dec. 2015.

[18] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[19] H. Toutenburg and C. Heumann. *Deskriptive Statistik: eine Einführung in Methoden und Anwendungen mit R und SPSS*. Springer-Verlag, 2008.

[20] D. Watson, L. A. Clark, and A. Tellegen. Development and validation of brief measures of positive and negative affect: the panas scales. *Journal of personality and social psychology*, 54(6):1063, 1988.