Open Diabetes UAM Heuristik Algorithmen

Qualitätssicherungsdokument

Gruppe 11:	Aino Schwarte <aino.schwarte@stud.tu-darmstadt.de> Anna Mees <anna.mees@stud.tu-darmstadt.de> Jan Paul Petto <janpaul.petto@stud.tu-darmstadt.de> Paul Wolfart <paul.wolfart@stud.tu-darmstadt.de> Tom Großmann <tom.grossmann@stud.tu-darmstadt.de>
Teamleiter:	Benedikt Schneider <schneider-benedikt@gmx.net>
Auftraggeber:	M.Sc. Jens Heuschkel <heuschkel@tk.tu-darmstadt.de> Telecooperation Smart Urban Networks
Abgabedatum:	01.12.2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Bachelor-Praktikum WS 2018/2019
Fachbereich Informatik

Inhaltsverzeichnis

1 Einleitung

Diabetes Mellitus Typ 1 ist eine Autoimmunerkrankung, bei der zu wenig oder gar kein körpereigenes Insulin produziert werden kann. Insulin ist notwendig, um den Blutzuckerspiegel zu regulieren. Durch Nahrungsmittel aufgenommene Kohlenhydrate werden während der Verdauung zu Glukose aufgespalten. Diese gelangen über die Darmwand in die Blutlaufbahn, wodurch der Blutzuckerspiegel steigt. Bei gesunden Menschen produziert die Bauchspeicheldrüse daraufhin die notwendige Menge an Insulin, um die Körperzellen anzuregen, den Zucker aufzunehmen. Dadurch sinkt der Blutzuckerspiegel wieder.

Leidet eine Person an Diabetes Mellitus Typ 1, ist folglich ihre Blutzuckerregulierung gestört und die Person muss die Regulierung selbst übernehmen. Andernfalls ist die Energieversorgung des Körpers gefährdet. Extrem hohe oder niedrige Blutzuckerwerte können akute schwerwiegende körperliche Folgen haben. Diese reichen von abgeschwächten Muskelreflexen, Übelkeit, bis zu Koma oder sogar zum Tod. Befindet sich der Blutzuckerspiegel regelmäßig und über längere Zeiträume außerhalb der Normalwerten, kann es zu schlimmen Langzeitschäden wie Nierenversagen kommen.

Diabetespatienten müssen zusätzlich zur Grundversorgung an Insulin für jede Mahlzeit die korrekte Insulindosis berechnen und im Laufe des Tages regelmäßig ihren Blutzuckerspiegel überprüfen. Dabei kann eine in dem Körper implantierte Insulinspritze helfen, indem sie die Grundversorgung übernimmt. Man spricht hierbei von einem Open-Loop-Verfahren. Die Community Nightscout mit rund 20.000 Mitgliedern hat diesen Ansatz mit dem Open Source Projekt Nightscout erweitert. Diesen kann man mit dem Open Artificial Pancreas System Projekt (OpenAPS) erweitern und sein eigenes Closed-Loop System bauen. OpenAPS ist ebenfalls eine Community, die das Projekt ständig weiterentwickelt. Durch einen Sensor im Körper werden regelmäßig Blutzuckerwerte gemessen, in einer (mongo) Datenbank gespeichert und direkt in einem eigenen Webinterface von NIghtscout visualisiert. Dies ermöglicht eine Fernkontrolle der Insulindosen und ein Warnsystem für gefährliche Werte. So können zum Beispiel Eltern die Blutzuckerwerte ihrer Kinder auf einer Smartwatch kontrollieren, wenn diese bei Freunden oder anderweitig unterwegs sind. Leider erfordert dieses Verfahren immer noch einen großen Aufwand und setzt Erfahrung und Fachwissen bezüglich Diabetes voraus.

Bei Closed-Loop-Systemen wird die die Insulinzufuhr automatisch an die gemessenen Blutzuckerwerte angepasst. Diese haben allerdings eine große Schwachstelle, nämlich unangekündigte Mahlzeiten (Un-Announced-Meals, kurz: UAM). Da das System nicht weiß, wie weit der Blutzucker steigen wird, wird am Anfang des Anstiegs nur sehr wenig Insulin gespritzt, als ob es sich um eine harmlose Schwankung handeln würde. Wenn der Blutzucker aber weiter steigt, können gefährlich hohe Werte nur noch mit einer großen Dosis Insulin verhindert werden. Anschließend kann der Blutzuckerspiegel lebensgefährlich tief sinken.

Um solche extremen Schwankungen zu verhindern, müssen Mahlzeiten rechtzeitig erkannt werden. Genau hier setzt unser Projekt an. Unser Ziel ist es anhand der Steigung des Blutzuckerwertes schneller und genauer erkennen zu können, dass diese durch eine Mahlzeit bedingt ist, um somit rechtzeitig die entsprechende Dosis Insulin hinzufügen zu können. Dabei bauen wir auf die bereits geleistete Arbeit der Nightscout Community auf und wollen ihr unseren Algorithmus anschließend zur Verfügung stellen.

2 Qualitätsziele

2.1 Korrektheit

2.1.1 Beschreibung

Unser wichtigstes QS-Ziel ist die Korrektheit. Dies fällt unter ISO/IEC 9126 zur Sicherstellung der Softwarequalität, unter den Punkt Funktionalität mit dem Unterpunkt Richtigkeit.

Diabetespatienten, die später unsere Software verwenden, vertrauen darauf, dass unser Ansatz korrekte Werte zurück gibt und Mahlzeiten richtig erkannt werden. Da ein zu hoher oder zu niedriger Insulinwert, wie bereits erläutert, schwere körperliche Langzeitfolgen haben oder sogar akut lebensbedrohlich sein kann, ist es offensichtlich, weshalb hier keine Fehler unterlaufen dürfen.

2.1.2 Maßnahmen

Wir wollen die Korrektheit durch verschiedene Maßnahmen erreichen. Darunter fallen unter anderem die Nutzung von regelmäßigen Coverity Scans und Travis CI. Travis CI eignet sich hier besonders, da wir an einem Open-Source-Projekt arbeiten, welches auf GitHub veröffentlicht wird. [1].

Travis kompiliert die Software und überprüft automatisch alle Tests, die wir implementieren, und meldet zurück, ob diese erfolgreich abgeschlossen wurden.

Coverity Scans analysieren den Code auf Race Conditions und Speicherlecks, bei denen zwar Arbeitsspeicher belegt, allerdings weder genutzt, noch frei gegeben wird.

Wir verwenden mehrere Branches für zu entwickelnde Features und einen Master-Branch, auf dem immer eine lauffähige Version liegt. Auf dem Master Branch kann nicht direkt gepusht werden, nur über die Feature-Branches. Es sind Pull-Request-Reviews und Status Checks nötig, um auf den Master-Branch zu schreiben. Diese Status Checks beinhalten das erfolgreiche compilen des Codes durch Travis-CI und den erflogreichen Durchlauf aller Tests. Erst im Anschluss können Pull-Requests akzeptiert werden. Wenn mindestens eine andere Person den Pull-Request überprüft und aktzeptiert hat, wird der Branch gemerged. Git führt einen Verlauf, wer den Pull-Request-Review freigegeben hat. Es wird somit immer protokolliert, wer Korrektur gelesen hat. Dieses Protokoll wird im Anhang zur Verfügung gestellt.

2.1.3 Prozessbeschreibung

In GitHub gibt es einen Master-Branch. Entwickelt wird ausschließlich in sogenannten Feature Branches. Für jedes Feature bzw. jede Feature-Gruppe wird ein eigener Branch erstellt. Nach Abschluss des Features und nach Abschluss von Iterationszyklen wird ein Pull-Request erstellt. Mindestens eine andere Person im Projekt kontrolliert die Änderungen im Pull Request, bevor dieser akzeptiert wird. Werden Mängel oder Probleme entdeckt, löst diese die Person, die den Pull

[1] `http://github.com/TUDa-BP-11/opendiabetes-uam-heuristik`

Request erstellt hat. Treten dabei Probleme oder Schwierigkeiten auf, dann wird die Hilfe der Teammitglieder in Anspruch genommen. Danach werden die Änderungen erneut von mindestens einer anderen Person kontrolliert, bis keine Mängel mehr gefunden werden und der Branch in den Master Branch gemerged werden kann.

Travis CI wird automatisch bei jedem Git-Commit ausgeführt. Dafür haben wir eine entsprechende .travis.yml Datei angelegt, damit bei jedem Commit Travis durchlaufen wird. Travis CI testet, ob das neue Commit noch compilebar ist und prüft definierte Tests. Schlägt einer der Tests fehl, werden wir per Mail informiert. Außerdem vergibt Travis CI Badges, anhand derer man in GitHub sehen kann, welche Tests korrekt durchlaufen.

Coverity lassen wir nur bei Commits auf den Coverity Branch testen, da die Nutzung von Coverity beschränkt ist. Dies führen wir durch, bevor wir auf den Master-Branch comitten. Die Anzahl der Builds pro Woche ist auf 28 mit maximal 4 Tests am Tag beschränkt, wenn man weniger als 100.000 Zeilen Code hat. Mit steigender Codelänge darf man den Code immer seltener testen lassen, bei einer Länge von mehr als 1.000.000 Zeilen Code kann dieser nur noch einmal am Tag getestet werden. Hat man die maximale Anzahl der Builds pro Tag erreicht, werden weitere Builds an dem entsprechenden Tag abgelehnt. Wenn Travis oder Coverity Fehler aufwirft, müssen diese von demjenigen, der gepusht hat, verbessert werden.

Unit Tests werden von einem anderen Teammitglied geschrieben, damit eine neutrale Sicht auf den Code gewährleistet ist und man nicht mit der erwarteten Funktion des Features aus Entwicklersicht an die Tests geht. Gefundene Fehler werden von dem verbessert, der den Code geschrieben hat. Sollte man seinen Fehler nicht selber beheben können, wird natürlich die Hilfe der anderen Gruppenmitglieder in Anspruch genommen.

Die Tests bestehen aus statischen Unit-Tests, welche sich automatisch auf eine Test-Instanz von Nightscout verbinden und alle verfügbaren Features überprüfen. Abhängig vom Status und Inhalt der in Nightscout gespeicherten Daten werden dynamisch weitere Tests generiert, die den implementierten Algorithmus auf verschiedene Weisen testen. Dafür verwenden wir die in JUnit-5 integrierte dynamische Test-Generation. Auf diese Weise können wir Tests mit verschiedenen Beispieldaten problemlos mehrmals ausführen und auch die Möglichkeit anbieten, das Programm mit eigenen Testdaten zu überprüfen. Sollte die Test-Instanz von Nightscout einmal nicht verfügbar sein, oder unzureichende oder falsche Daten beinhalten, müssen die Tests mit eindeutigen Meldungen fehlschlagen und dem Benutzer mitteilen, wie die Fehler zu beheben sind. Nicht ausgeführte Tests auf Grund von fehlenden Daten in der Test-Instanz von Nightscout werden als nicht ausgeführt markiert, lassen den gesamten Test aber nicht fehlschlagen.

2.2 Erweiterbarkeit

2.2.1 Beschreibung

Unser zweites QS-Ziel ist die Erweiterbarkeit. Dies fällt unter ISO/IEC 9126 zur Reduzierung des Aufwands die Software zu ändern. Unser Ziel ist es, ohne viel Aufwand neue Algorithmen zur Berechnung der Mahlzeiten einpflegen zu können, genauso wie neu Möglichkeiten Daten und Datenquellen einbinden zu können.

Dies ist besonders sinnvoll, da unser Code einer Opensource-Community, unter der Lizenz AGPLv3, auf GitHub zur Verfügung steht.

So ist es möglich, wenn neue Ansätze für eine bessere oder andere Berechnung von Mahlzeiten gefunden wurden, diese einfach zu implementieren und auszuführen, ohne dass das Hauptprogramm in irgendeiner Art und Weise geändert werden muss. Genauso können neue Datenquellen, wie zum Beispiel eine XML-Repräsentation der Daten, eingefügt werden.

2.2.2 Maßnahmen

Um dies möglich zu machen stellen wir zunächst Wiki-Artikel zur Verfügung. In diesen erklären wir ausführlich, wie unsere Software zu verwenden und zu erweitern ist. Unsere Projektstruktur ist modular, sodass neue Algorithmen und Datenquellen einfach durch die Implementierung eines Interfaces hinzugefügt werden können.

2.2.3 Prozessbeschreibung

Der Wiki-Artikel beschreibt auf englisch, Schritt für Schritt, wie man einen neuen Algorithmus implementieren kann. Dies wird exemplarisch an einem unserer Algorithmen gezeigt. Die Artikel werden von zwei Projektmitgliedern geschrieben. Mindestens zwei andere Mitglieder lesen diese Korrektur und überprüfen anhand einer Checkliste, ob diese vollständig sind. Dies ist erreicht, wenn der Algorithmus nach der Ausführung der Anleitung lauffähig ist.

Die Checkliste wird unter anderem folgende Punkte erhalten: Rechtschreibung, korrekter Satzbau,

Die Algorithmen (Algorithm interface) benutzen zwei interne Datenklassen, `VaultEntry` und `Profile`, welche alle nötigen Daten repräsentieren. Neue Algorithmen arbeiten also mit diesen vorgegebenen Klassen, welche von einer Datenquelle (Dataprovider interface) befüllt werden. Um Daten in Nightscout-JSON-Repräsentation zu verwenden wird bereits ein Parser zur Verfügung gestellt. Dieser erweitert ebenfalls ein Interface welches genauso für andere Datentypen verwendet werden kann und das Umwandeln von Strings oder regulären Dateien mit ASCII-Inhalt bereit stellt.

A Anhang

(Am Ende des Projekts nachzureichen)

Beleg für durchgeführte Maßnahmen, bzw. falls nicht durchgeführt eine Begründung wieso die Durchführung nicht möglich oder nicht erfolgt ist.
Weitere Anforderungen sind den Unterlagen und der Vorlesung zur Projektbegleitung zu entnehmen.