
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: Februar 2019



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor-Praktikum WS 2018/2019
Fachbereich Informatik

Inhaltsverzeichnis

1. Einleitung	2
2. Qualitätsziele	4
2.1. Korrektheit	4
2.1.1. Beschreibung	4
2.1.2. Maßnahmen	4
2.1.3. Prozessbeschreibung	4
2.2. Modularität und Erweiterbarkeit	6
2.2.1. Beschreibung	6
2.2.2. Maßnahmen	6
2.2.3. Prozessbeschreibung	7
A. Anhang	8
A.1. Pull-Request-Review Checkliste	8
A.2. Pull-Request-Review Protokoll	8
A.3. GitHub Wiki Artikel Checkliste	8

1 Einleitung

Das sogenannte Un-Announced-Meal-Problem (UAM) ist derzeit eine der größten Therapie-schwächen bei der Behandlung von Diabetes Mellitus Typ 1 ¹ und beschreibt Mahlzeiten, welche vom Patienten nicht berücksichtigt wurden und somit den Blutzuckerspiegel unkontrolliert steigen lassen. Denn bei dieser Autoimmunkrankheit produziert der Körper kaum bis gar kein eigenes Insulin. Insulin reguliert den Blutzuckerspiegel und ist für die Energieversorgung des Körpers unentbehrlich. Bei Betroffenen muss die Blutzuckerregulierung dementsprechend manuell durchgeführt werden. Extrem hohe oder niedrige Blutzuckerwerte können akute schwerwiegende und sogar tödliche Folgen haben. Befindet sich der Blutzuckerspiegel regelmäßig und über längere Zeiträume außerhalb der Normalwerten, kann dies auch zu schweren Langzeitschäden führen.

Um in Zukunft ein voll automatisches System mit Messgerät und Insulinpumpe zu ermöglichen, welches die Insulinversorgung des Patienten übernimmt, wird eine Möglichkeit benötigt, Mahlzeiten verlässlich zu erkennen. Die von der Krankheit betroffene Person muss nämlich neben der Grundversorgung an Insulin zusätzlich auch mit einer zu jeder Mahlzeit präzise berechneten Insulindosis versorgt werden. Bisherige Systeme können mit der Problematik von unangekündigten Mahlzeiten (Un-Announced-Meals) nicht umgehen, wenn eine Mahlzeit vom Patienten bei der Planung vergessen oder zu gering eingeschätzt wurde. Eine rechtzeitige und korrekte Insulindosierung ist dann nicht mehr möglich. Stattdessen wird das notwendige fehlende Insulin erst allmählich mit dem steigendem Blutzuckerspiegel verabreicht, was zu spät ist, um hohe Werte zu verhindern. In solchen Situationen kann es zu einer Überkorrektur kommen, was wiederum zu einem lebensbedrohlich tiefen Blutzuckerspiegel führt.

Das Ziel unseres Projekts ist es anhand der steigenden Blutzuckerwerte mit wenigen Messwerten eine Mahlzeit verlässlich zu berechnen. So kann die adäquate Menge an Insulin schon frühzeitig injiziert werden. Dazu bauen wir auf dem Open Source Projekt Nightscout ² auf, welches der Visualisierung von Blutzucker Messwerten dient. Die Messungen werden alle fünf Minuten von einem Sensor unter der Haut durchgeführt. Über die Cloud werden sie in einer Datenbank gespeichert und können auf jedem internetfähigen Gerät abgerufen werden. Zusätzlich werden Insulindosierungen und bekannte Mahlzeiten als Events kenntlich gemacht. Unser Programm bezieht die Messwerte und Events aus der Datenbank und startet einen Algorithmus zur Berechnung von Mahlzeiten. Da es keine bekannte optimale Methode gibt, um Mahlzeiten zu erkennen, entwickeln wir unterschiedliche Ansätze, aus denen gewählt werden kann. Diese Ansätze beruhen auf verschiedenen wissenschaftlichen Publikationen, welche die Auswirkungen von Kohlenhydraten und Insulin auf den Blutzuckerspiegel mathematisch beschreiben. Wir benutzen diese Modelle um von Blutzucker Messwerten und eventuell bekannten Insulin Behandlungen auf Mahlzeiten (also die Zufuhr von Kohlenhydraten) zurück zu schließen. Die berechneten Mahlzeiten werden dann als neue Events in Nightscout eingetragen und je nach Ansatz auch im weiteren Verlauf berücksichtigt.

Es ist nicht sicher, ob sich das UAM-Problem mit dem aktuellen Sachverstand der Wissenschaft zuverlässig lösen lässt. Die Ansätze basieren auf bisherigen Modellierungen des Einflusses

¹ Diabetes Typ 2 fällt nicht unter das UAM-Problem, da es sich dabei lediglich um eine Insulinresistenz handelt. Eine nicht erkannte Mahlzeit stellt also ein geringeres Risiko dar.

² <http://www.nightscout.info>

von Insulin und Kohlenhydraten auf den Blutzuckerverlauf. Unser Projekt dient also als Plausibilitätsprüfung, ob sich das UAM-Problem zurzeit realistisch lösen lässt. Wir werden unsere Algorithmen, wenn sie sich als verlässlich erweisen, der Nightscout Community zur Verfügung stellen.

2 Qualitätsziele

2.1 Korrektheit

2.1.1 Beschreibung

Unser wichtigstes QS-Ziel ist die Korrektheit. Dabei liegt das Hauptaugenmerk neben der Erzeugung von korrekten Daten auch darauf, Fehler und Fehlerquellen korrekt zu erkennen. Wenn die zur Verfügung stehenden Daten zur Berechnung einer Mahlzeit nicht ausreichen oder sonstige Probleme auftreten, muss der Benutzer darüber in Kenntnis gesetzt werden, und es dürfen keine potentiell falschen Ergebnisse produziert werden. Diabetespatienten, die unsere Algorithmen verwenden werden, 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 die die Nutzung von automatischen Coverity Scans ¹ und dem Continuous Integration Tool Travis CI ² erreichen. Travis CI kann kostenlos genutzt werden, da wir an einem Open-Source-Projekt arbeiten, welches auf GitHub ³ veröffentlicht wird.

Travis CI kompiliert die Software und überprüft automatisch alle implementierten Tests 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-Branche. Es sind Pull-Request-Reviews und Status Checks nötig, um auf den Master Branch zu schreiben. Diese Status Checks beinhalten das erfolgreiche Kompilieren des Codes durch Travis CI und den erfolgreichen Durchlauf aller Tests. Erst im Anschluss können Pull-Requests akzeptiert werden. Wenn mindestens eine andere Person den Pull-Request überprüft und akzeptiert 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

¹ <https://scan.coverity.com>

² <https://travis-ci.org>

³ <http://github.com/TUDa-BP-11/opendiabetes-uam-heuristik>

Abschluss des Features und der 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-Request erstellt hat, zum nächstmöglichen Zeitpunkt. Treten dabei Probleme oder Schwierigkeiten auf, 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 wurde eine entsprechende .travis.yml Datei angelegt. Travis CI kompiliert die Software und startet alle implementierten Tests. Schlägt einer der Tests fehl, werden wir per E-Mail informiert. Außerdem vergibt Travis CI Badges, anhand derer in GitHub sichtbar ist, welche Branches in ihrem aktuellen Stand ohne Probleme kompiliert und getestet wurden.

Die Tests bestehen aus statischen Unit-Tests, welche zunächst alle grundlegenden Funktionen testen, wie zum Beispiel das korrekte Verbinden auf eine Nightscout Test Instanz und die volle Funktionalität der API-Schnittstelle. Außerdem werden alle verwendeten mathematischen Modelle - zum Beispiel zur Berechnung von Insulinabbau über Zeit im Blutkreislauf - mit festen Testdaten auf Korrektheit überprüft. Diese Testdaten wurden zuvor unabhängig von der Implementierung der Funktionen berechnet, wir testen die Funktionen also nicht mit Werten, die durch die Funktionen selbst generiert wurden. In einem zweiten Schritt werden - abhängig vom Status und Inhalt der in der Nightscout Test Instanz gespeicherten Daten - dynamisch weitere Tests generiert, die die implementierten Algorithmen auf verschiedene Weisen testen. Dies ist stark von den jeweiligen Algorithmen abhängig, wir testen aber mindestens jede offen verfügbare (public) Methode. 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, schlagen die Tests mit eindeutigen Meldungen fehl und teilen dem Benutzer mit, wie die Fehler zu beheben sind. Wenn auf der Nightscout Test Instanz unzureichende oder falsche Daten gespeichert sind, überprüfen die Tests, ob die Algorithmen korrekt abbrechen und dem Benutzer mitteilen, welche Probleme mit den verfügbaren Daten bestehen. Nicht ausgeführte Tests auf Grund von fehlenden Daten in der Nightscout Test Instanz werden als nicht ausgeführt markiert, lassen den gesamten Test aber nicht fehlschlagen.

Coverity wird nur bei Commits auf den Coverity Branch ausgeführt, da die Nutzung von Coverity beschränkt ist. Dies wird durchgeführt, bevor in den Master Branch gemerged wird. Die Anzahl der Builds pro Woche ist auf 28 mit maximal 4 Tests am Tag beschränkt, wenn weniger als 100.000 Zeilen Code getestet werden. Wurden die maximale Anzahl der Builds pro Tag erreicht, werden weitere Builds an dem entsprechenden Tag abgelehnt. Wenn Travis CI oder Coverity Fehler erzeugen, müssen diese von demjenigen, der gepusht hat, zum nächstmöglichen Zeitpunkt behoben werden.

2.2 Modularität und Erweiterbarkeit

2.2.1 Beschreibung

Unser zweites QS-Ziel ist die Modularität und damit verbunden die Erweiterbarkeit. Unser Ziel ist es, ohne viel Aufwand neue Algorithmen zur Berechnung der Mahlzeiten einpflegen zu können, sowie die Möglichkeit zu bieten neue Daten und Datenquellen einbinden zu können.

Dies ist besonders sinnvoll, da unser Code einer Opensource-Community, unter der AGPLv3 Lizenz, auf GitHub zur Verfügung steht und Mitglieder der Nightscout-Community (einschließlich Herrn Heuschkel als Auftraggeber) den Code wieder verwenden möchten.

So ist es möglich, wenn neue Ansätze für eine bessere Berechnung von Mahlzeiten gefunden wurden, diese einfach zu implementieren und auszuführen, ohne dass das Hauptprogramm geändert werden muss. Genauso können neue Datenquellen, wie zum Beispiel eine MySQL Datenbank, oder ein entferntes Dateisystem, eingefügt werden.

2.2.2 Maßnahmen

Um dies möglich zu machen, werden zunächst GitHub Wiki Artikel zur Verfügung gestellt ⁴. In diesen wird ausführlich erklärt, wie unsere Software zu installieren, verwenden und erweitern ist. Die Projektstruktur ist modular, sodass neue Algorithmen und Datenquellen einfach durch die Implementierung eines Interfaces hinzugefügt werden können. Es werden verschiedene Entwurfsmuster (Design Patterns) verwendet, um die Erweiterbarkeit sicher zu stellen.

Unser Projekt gliedert sich in drei Hauptmodule: Das Hauptprogramm (Modul 1), die Anbindung an Nightscout und der Daten-Parser (Modul 2) und die Repräsentation der Daten (Modul 3). Diese Module können unabhängig voneinander verwendet werden, wobei sie jeweils auf das in der obigen Liste folgende Modul aufbauen. Unser Auftraggeber verwendet zum Beispiel die im dritten Modul definierten Datenklassen auch für andere Projekte (diese Klassen wurden uns vorgegeben), und die Anbindung an Nightscout kann benutzt werden, um beliebige Daten von und zu Nightscout zu übertragen.

Das Hauptprogramm gliedert sich in vier weitere Module: Die Hauptklasse und ihre dazugehörigen Klassen, zum Starten des Programms (Modul 1), die Datenquellen, zum Einlesen und Abspeichern der Daten (Modul 2), und die Algorithmen, zur Verarbeitung der Daten (Modul 3). Häufig benutzte Funktionen und die Implementierungen der verwendeten mathematischen Modelle, welche in verschiedenen Algorithmen benötigt werden, werden in eigene Klassen ausgelagert (Modul 4). Dies ermöglicht es viele Funktionen in verschiedenen Algorithmen wieder zu verwenden und diese unabhängig von den Algorithmen testen zu können.

In einem beispielhaften Programmablauf wird also das Hauptprogramm gestartet und öffnet, je nach Argumenten und Konfiguration, eine Datenquelle, um die dort hinterlegten Daten an einen Algorithmus weiterzugeben. Dieser benutzt dann gegebenenfalls verschiedene der implementierten mathematischen Modelle und gibt am Ende neue, berechnete Daten an das Hauptprogramm zurück.

Wie bereits beim Ziel der Korrektheit beschrieben, werden Pull-Requests auf den Master Branch von mindestens einer anderen Person kontrolliert. Hier wird auch überprüft, ob die oben beschriebene Modularität eingehalten wurde.

⁴ <https://github.com/TUDa-BP-11/opendiabetes-uam-heuristik/wiki>

2.2.3 Prozessbeschreibung

Die Wiki Artikel beschreiben auf englisch, Schritt für Schritt, wie ein neuer Algorithmus implementieren werden 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 (siehe Anhang), ob diese vollständig sind und vollziehen die Schritte auf einem neuen System nach. Dies ist erreicht, wenn der Algorithmus nach der Ausführung der Anleitung lauffähig ist.

Sowohl für die Algorithmen als auch für die Datenquellen verwenden wir das Strategy Pattern⁵, um während der Laufzeit verschiedene Algorithmen und Datenquellen zu laden. Das Datenquellen Interface (DataProvider) definiert abstrakte Methoden zum Laden von Blutzucker Messwerten, Insulin Behandlungen, und bekannten Mahlzeiten jeweils in einem bestimmten Zeitfenster. Das Algorithmus Interface (Algorithm) definiert abstrakte Methoden um bekannte Blutzucker Messwerte, Insulin Behandlungen und Mahlzeiten zur Verfügung zu stellen, eine Methode um die Ausführung des Algorithmus zu starten, und eine Methode um die berechneten Mahlzeiten abzurufen.

Die Nightscout Anbindung beinhaltet einen unabhängig verwendbaren Parser, welcher ebenfalls ein Interface erweitert und mit dem Template Method Pattern das direkte Analysieren von Strings (zum Beispiel die Rückgabewerte der Nightscout API) und die Verwendung von normalen Textdateien als Quelle ermöglicht. Dieser Parser kann die JSON-Repräsentation der Daten von Nightscout übersetzen. Da Nightscout selbst das Hinzufügen von neuen Datentypen erlaubt, ist es auch hier kein Problem einen neuen Parser für noch unbekannte Daten hinzuzufügen.

Zur Repräsentation der Daten im Programm benutzen wir verschiedene vom Auftraggeber vorgegebene Datenklassen, welche auch in anderen Projekten des Auftraggebers verwendet werden. Alle Algorithmen und Datenquellen benutzen ausschließlich diese Klassen, um miteinander zu kommunizieren, und ermöglichen dadurch ebenso unser Programm mit anderen vom Auftraggeber verwendeten Programmen zu kombinieren.

Bei der Kontrolle von Pull-Requests auf den Master Branch achtet die kontrollierende Person darauf, dass die verwendeten Entwurfsmuster korrekt umgesetzt wurden, und zum Beispiel ein Algorithmus nicht direkt Daten an das Hauptprogramm weiter gibt sondern dies nur über das dafür vorgesehene Strategie Interface passiert. Außerdem wird darauf geachtet neue Funktionen so abstrakt wie möglich zu implementieren, um die Wiederverwendung an anderen Stellen zu erleichtern. Dabei wird kontrolliert, ob diese neuen Funktionen sinnvoll in ihrem Modul platziert sind, oder ob eine Implementation in einem anderen Modul die Wiederverwendung an mehr Stellen ermöglichen würde. Die im Anhang enthaltene Checkliste beinhaltet auch die hierfür wichtigen Punkte.

⁵ https://en.wikipedia.org/wiki/Strategy_pattern

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.

A.1 Pull-Request-Review Checkliste

A.2 Pull-Request-Review Protokoll

A.3 GitHub Wiki Artikel Checkliste
