Master's Thesis

Design and Development of a Mobile Computerized Glucose Management System to Support Inpatient Care for Patients with Diabetes

Bernhard Höll, BSc

Institut of Information Systems and Computer Media (IICM),
Graz University of Technology
8010 Graz, Austria

Institute for Biomedicine and Health Sciences (HEALTH),
JOANNEUM Research Forschungsgesellschaft mbH
8010 Graz, Austria

Supervisor: Assoc. Prof. Andreas Holzinger, PhD, MSc, MPh, BEng, CEng, DipEd, MBCS

with support of: Dipl.-Ing. Stephan Spat

Graz, 13.09.2011

This page intentionally left blank

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

Design und Entwicklung eines mobilen computerisierten Glukosemanagementsystems zur Unterstützung der stationären Behandlung von Patienten mit Diabetes

Bernhard Höll, BSc

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz
8010 Graz, Österreich

Institut für Biomedizin und Gesundheitswissenschaften (HEALTH),
JOANNEUM Research Forschungsgesellschaft mbH
8010 Graz, Österreich

Begutachter/Betreuer: Univ.-Doz. Ing. Mag. Mag. Dr. Andreas Holzinger

mit Unterstützung von: Dipl.-Ing. Stephan Spat

Graz, 13.09.2011

This page intentionally left blank

Abstract

This master thesis deals with the design and development process of a mobile Android application to support the inpatient glucose management of patients with diabetes at the University Hospital Graz in order to optimize the current paper based glucose management. An integrated decision support service for insulin dosing should provide additional security and support for clinicians. The master thesis was carried out in the course of the EU-project REACTION at the Joanneum Research institute HEALTH - Institute for Biomedicine and Health Sciences - in Graz. The thesis is generally divided into two parts. The first part deals with an extensive requirements analysis, in order to get an imagination of the design of the application's user interface, as well as to understand clinical workflow patterns. The design phase followed a user-centered design approach, which means that the end-users have been involved in every step of the design process. In the second part, the achievements of the requirements analysis were used to set up the implementation of the inpatient glucose management system. Due to maintainability and expandability it was decided to distinguish between a frontend application for user interactions and a platform independent backend application, which contains the business logic for the decision support, as well as the data storage and interfaces to the hospital information system. The exchange of data between the backend and the frontend is done via encrypted web services to provide data security. This master thesis primarily deals with the development of the frontend apllication and should illustrate collected experiences during the design and the development process. It should demonstrate the requirements and challenges of implementing safety-critical medical software and should show how the end user can be involved in the engineering process.

Keywords

MEDICAL DEVICE SOFTWARE, USER-CENTERED DESIGN, ANDROID

ÖSTAT classification

1138, 1140, 1157, 3927

ACM classification

D.2, H.4.1, H.5.2, J.3

This page intentionally left blank

Kurzfassung

Diese Diplomarbeit präsentiert den Design- und Entwicklungsprozess einer mobilen android-basierten Anwendung zur Unterstützung des stationären Glukosemanagements für Patienten mit Diabetes auf der Universitäts-Klinik in Graz. Eine integrierte Entscheidungshilfe für Insulindosierungen soll in Sicherheit und Unterstützung für das klinische Personal resultieren. Die Masterarbeit entstand im Rahmen des EU-Projekts REACTION am Institut für Biomedizin und Gesundheitswissenschaften (HEALTH) der Joanneum Research in Graz. Die Arbeit besteht grundsätzlich aus 2 Teilen. Der erste Teil beschreibt eine umfassende Anforderungsanalyse um einerseits eine Vorstellung für die Gestaltung der Benutzerschnittstelle zu bekommen und andererseits um klinische Abläufe zu verstehen. Die Designphase folgte einem benutzerzentrierten Ansatz, d.h.: die Endbenutzer wurden in jedem Schritt eingebunden. Im zweiten Teil wurden die erhobenen Anforderungen in eine mobile Anwendung implementiert. Um Wartbarkeit und Erweiterbarkeit zu gewährleisten wurde die Anwendung in ein Frontend, welches die Benutzerschnittstelle darstellt, sowie in ein plattformunabhängiges Backend, welches die Logik für die Entscheidungshilfe und den Datenspeicher beinhaltet, aufgeteilt. Der Austausch der Daten zwischen dem Frontend und dem Backend wird durch verschlüsselte Web Services ermöglicht. Die Diplomarbeit beschäftigt sich in erster Linie mit der Entwicklung der Frontend-Anwendung und soll gesammelte Erfahrungen während der Design- und Entwicklungsphase aufzeigen. Die Arbeit soll die Anforderungen und Schwierigkeiten bei der Entwicklung von sicherheitskritischer medizinischer Software verdeutlichen und soll zeigen wie Endbenutzer in den Entwicklungsprozess eingebunden werden können.

Schlüsselwörter
MEDIZINISCHE SOFTWARE, BENUTZERZENTRIERTES DESIGN, ANDROID

ÖSTAT Klassifikation
1138, 1140, 1157, 3927

ACM Klassifikation
D.2, H.4.1, H.5.2, J.3

This page intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, 13.09.2011

BERNHARD HÖLL

This page intentionally left blank

Acknowledgements

This work was partly funded by the E. C. under the 7th Framework Program in the area of Personal Health Systems under Grant Agreement no. 248590.

This page intentionally left blank

Table of Contents

This page intentionally left blank

List of Abbreviations

ALARP	As Low As Reasonably Practicable
API	Application Programming Interface
BG	Blood Glucose
DVM	Dalvik Virtual Machine
EN	European Norm
GUI	Graphical User Interface
HIS	Hospital Information System
HL7	Health Level 7
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JSF	JavaServer Faces
LIS	Laboratory Information System
MDA	Model Driven Architecture
MDD	Medical Device Directive
NPH	Neutral Protamine Hagedorn
PC	Personal Computer

PIM	Platform Independent Model
POCT	Point Of Care Testing
POM	Project Object Model
PSM	Platform Specific Model
REACTION	Remote Accessibility to Diabetes Management and Therapy in Operational Healthcare Networks
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SOUP	Software of Unknown Provenance
SQL	Structured Query Language
THA	Thinking Aloud
UML	Unified Modeling Language
VBA	Visual Basic for Applications
VO	Value Object
XML	Extensible Markup Language

1. Introduction and Motivation for Research

Computers and modern information technologies have influenced the human lifestyle. Modern technologies should help to make things more comfortable and more safe. Especially in health care, technologies were early used to provide more security to patients. After World War II the first organized forums for engineers working in medicine emerged. The primary focus of healthcare technologies in the 20th century was in areas of medical equipment and technology management. In the last few years, with the turn of the century, the focus of clinical engineering is shifting towards clinical systems integration (Zambuto (2004)). Thus, lots of studies have been carried out that determine improvements of clinical outcomes by using information technologies. One of these studies, of which the results were published in 2009 by Amarasingham, examines whether greater automation of hospital information can be associated with reduced rates of inpatient mortality, complications, costs and length of stay. 167 233 patients were involved in this study. The results show that hospitals with automated notes and records, order entry and clinical decision support had lower mortality rates, lower costs and fewer complications (Amarasingham et al. (2009)).

1.1 Glucose Management at the General Ward

Particularly in the inpatient treatment of diabetes patients, there is potential for improvement. Statistics show that people with diabetes are more likely to be hospitalized and to have longer durations of hospital stay than those without diabetes. Due to the continued worldwide expansion of type 2 diabetes, it is estimated that

22% of all in-hospital days were accounted by people with diabetes and that inpatient care accounted for half of the total US medical expenditures associated with this disease (Moghissi et al. (2009)). Although studies demonstrate that in-hospital hyperglycaemia has been found to be an important marker of poor clinical outcome and mortality among diabetic patients and that aggressive treatment of diabetes and hyperglycaemia results in reduced mortality and morbidity (Clement et al. (2004)). However, the inpatient glycemic control of acute diseased patients with diabetes is often considered secondary in importance. Therefore, patients with diabetes require a well thought-out glucose management during inpatient stays, including continuous glycemic control, reached by blood glucose monitoring and a determination of suitable treatment strategies (Höll et al. (2011a)).
This master thesis deals with the user-centered design and development process of a mobile Android application, including an integrated decision support service, in order to optimize the current paper based glucose management of patients with diabetes at the University Hospital at Graz.

1.2 Objectives of the REACTION Project

In 2010 the project REACTION (Remote Accessibility to Diabetes Management and Therapy in Operational healthcare Networks) founded by the European Union started with the objective to improve treatment quality in both, the long-term management in the outpatient setting as well as the glucose management of acutely ill diabetic patients in the hospital. The project is intended to run for 4 years and the project consortium consists of 16 organisations from 10 different countries, including the two Austrian institutions Joanneum Research and Medical University of Graz.
The project intends to research and develop an intelligent service platform with the aim to provide different clinical applications for monitoring vital signs, context awareness, feedback to the point of care, event and alarm handling as well as integration of clinical workflows, according to patients with diabetes. A planned REACTION platform should consist of production servers for data management, security, application execution and communication. All servers interoperate over web services and are thus completely platform independent. The REACTION platform should connect to sensors and devices, placed at the point of care and to

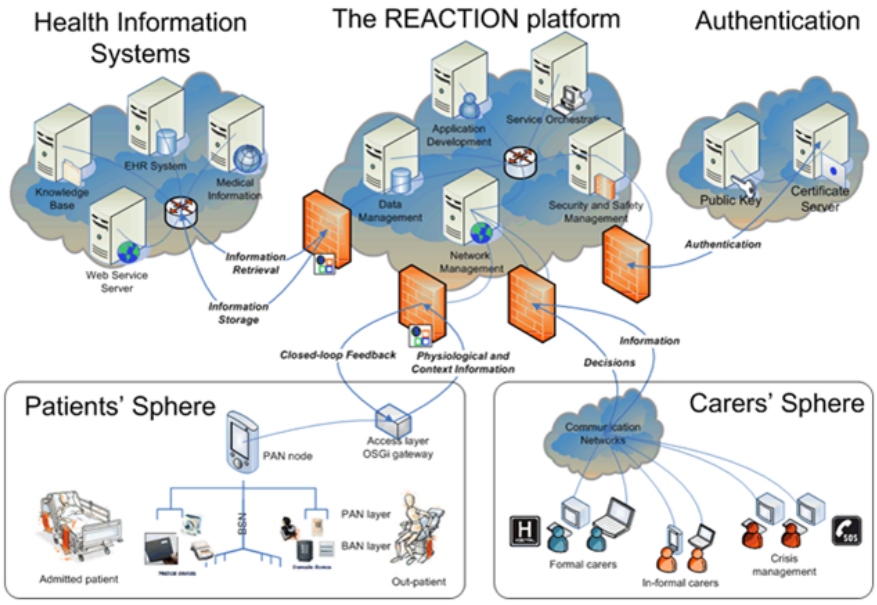

Figure 1.1: Architecture of the REACTION Platform (REACTION (2011))

healthcare professionals as well as emergency and crisis management teams. The platform should also provide an interface to external Health Information Systems and external medical knowledge repositories and security providers. The concept of the REACTION platform is shown in figure 1.1. The REACTION platform should represent the central production environment for the deployment of REACTION applications.

1.3 Structure of Work

Chapter 2 presents some already existing mobile applications, according to the glucose management of diabetics, which were found during a research on the internet. However most of these applications were targeted the outpatient area.
Chapter 3 starts with a short medical introduction related to Diabetes Mellitus, to understand the importance of continuous glucose control of people with diabetes. It also gives a summary about the RABBIT 2 protocol for insulin dosing, which provided the basis for the decision support service. Furthermore, chapter 3 gives an overview of methods as well as technical materials used during the development process. Implementing medical device software requires a high level of quality and security. Therefore standards were established which should help to comply with

the medical device directives. Chapter 3 introduces the challenges of implementing medical device software and points out the most important standards to follow.
Chapter 4 give insights in the accomplished requirement engineering process, where all necessary requirements for the inpatient glucose management system were iteratively collected within two development iterations. It indicates how end users can be intensively involved in the engineering process and deals with the risk management, which accompanies the engineering process. Afterwards chapter 4 presents how the achievements of the requirements analysis were used to set up the implementation of the inpatient glucose management system in a third development iteration. The focus of this master thesis primarily lies on the implementation of the Android-based frontend application, however chapter 4 will also introduce the development of the backend application using Model Driven Architecture (MDA) as well as the data model of the data storage. Chapter 5 presents the current results of the practical implementation. Activity diagrams illustrate the Android application's architecture and the interaction between the application's components and screenshots should visualize actual results of the Android user interface.
Chapter 7 and 8 provide a summary of the topics presented in this master thesis, figures out what could be learned during the development process and finally give an outlook to the next steps of the development.

2. Related Work

Due to the fact that diabetes is widespread and that glucose control for diabetics is indispensable, it is not surprising that intensive research on the internet results that there already exist different software tools, which aim to mange glucose control of diabetics. Especially in the outpatient area there are promising applications for the daily treatment of diabetes. One of them is called Diabeo[1]. Diabeo is a telemedicine solution, where patients can share glucose recordings with their physicians. Therefore a patient uses a mobile self-monitoring application, running on a smartphone, to record his glucose values, nutrition and administered medications. The records are transferred automatically to a secure web portal, where the records can be viewed and analyzed by the patient's physician. Diabeo also provides an insulin calculator for insulin dosing based on a medical protocol established with the physician. Over the web portal the physician can prescribe medication, which is sent to the patient's mobile device. Figure 2.1 shows examples of the mobile interface of Diabeo. Diabeo

[1]http://www.voluntis.com/en/our-solutions/telecare/diabeo.html

Figure 2.1: User interface of the Diabeo tool 1

was already tested in a study, which demonstrates that

> *"individualized insulin dose adjustments combined with telemedicine support significantly improves HbA1c in poorly controlled type 1 diabetic patients"* (Charpentier et al. (2011)).

HbA1c can be seen as an indicator for well controlled diabetes patients. Diabeo relates to the outpatient environment of glucose management. However, there are also software tools, which were established for inpatient glucose management. For example EndoTool [2] manages the glucose control for acutely ill patients in hospital. EndoTool is a desktop application, which has to be installed on the hospital's IT system. EndoTool provides a complex mathematical insulin dosage calculation, which was already verified in a study (Mann et al. (2009)). In addition to costly tools for glucose management in professional use, there are also free or cheep apps available for almost all smartphones. These apps are designed to track the course of somebody's own diabetes disease, including glucose levels, administered medication, food intake and other factors, which have an effect on the health of a patient with diabetes (Preuveneers and Berbers (2008)). Despite the variety of existing tools for the glucose management, no tool could be found which targets on the mobile inpatient environment.

[2] http://www.hospira.com/Products/endotool.aspx

3. Theoretical Background

3.1 Medical Aspects

In order to understand the content and the aim of the thesis some medical background knowledge about Diabetes Mellitus and the inpatient treatment of patients with diabetes is crucial.

3.1.1 Diabetes Mellitus

Diabetes Mellitus is a group of diseases, caused by defects in insulin production, insulin action or both, which leads to high levels of blood glucose, called hyperglycaemia. The most frequent occurring diabetes forms are Diabetes Mellitus Type 1 and Diabetes Mellitus Type 2. Patients with Diabetes Mellitus Type 1 cannot regulate their blood glucose on their own. Therefore they have to take insulin to survive. This form of diabetes is not dependent on lifestyle and can occur at any age. Patients with Diabetes Mellitus Type 2 do not use insulin properly, so the need for insulin rises. At some point, the body is not able to meet the needs of insulin anymore and consequently supplement insulin has to be supplied. Diabetes Mellitus Type 2 is associated with older age and obesity. Especially in patients with Diabetes Mellitus Type 2, the morbidity and mortality compared to healthy patients increased significantly. In Germany, about 5% of the population suffers from diabetes, in which 90% are patients with Diabetes Mellitus Type, 2,5% are patients with Diabetes Mellitus Type 1 and the remaining 5% suffer from other types of diabetes. Patients with diabetes have to control their blood glucose level to avoid hyperglycaemia. The most common but also the most dangerous unwanted effect of an insulin therapy is a too low blood glucose level, called hypoglycaemia.

Hypoglycaemia can be caused by avoidable errors in therapy, such as the absence of food intake, excessive alcohol consumption or incorrect insulin dosage. Occurrences of hyperglycaemia or hypoglycaemia are dangerous and can lead to heart diseases, strokes, hypertension, blindness, kidney diseases, nervous system diseases and amputations (USDepartment (2011)). Treatment measures must be matched to the different individual needs in order to obtain a preferably normal blood glucose control. Principle of treatment is to balance the existing lack of insulin through a flexible replacement therapy. In order to achieve the desired norm-close blood glucose control, the insulin dose has to be coordinated with the feeding behavior and physical activities. However, this is only possible on the basis of regular blood glucose measurements. There is a large variety of available insulin preparations for patients with diabetes mellitus. Generally, insulin can be distinguished between rapid-acting insulin and long-acting insulin. Rapid-acting insulin, also called Prandial or Bolus insulin, is given to patients based on their intake of carbohydrates during scheduled meals and acts fast but only for a short period. On the other hand, long-acting insulin, also called Basal insulin, acts slow but for a longer period. The need for Basal insulin represents about 50% of the total insulin requirements and ensures the continuous supply of the body with insulin, independent of food intake. The Basal insulin is usually administered once a day. In addition to an insulin therapy oral antidiabetics can be used to lower the blood glucose level. Today, there are also established combinations of long-acting and rapid-acting insulin available in order to simplify the insulin treatment. Insulin requirements of a patient vary from day to day. Consequently a same insulin dosage does not mean identical blood glucose profiles. Therefore the insulin dosage depends on the current blood glucose level of the patient (Flasnoecker (1999)). Diabetes is an incurable chronic disease, but patients are able to control the disease, prevent complications and live a normal and vital life through proper care (Chen et al. (2010)).

3.1.2 Protocol for Insulin Dosing (Rabbit 2 Trial)

The correct dosage of insulin is very important in order to avoid hyperglycaemia or hypoglycaemia. At hospital, physicians often define a patient's insulin dosage based on intuitions, which they have collected during years. However, there are some

physicians, which are not as familiar with diabetes and consequently have problems to find correct insulin dosages in some situations. 2007, Umpierrez first published the so-called RABBIT 2 trial (Umpierrez et al. (2007, 2009)). The protocol, which is presented in this trial, is used as the basis for the decision support functionality which should be provided by the inpatient glucose management system. This makes it necessary to describe the basic concept of this protocol.

Aim of the RABBIT 2 trial was to compare the efficacy and safety of a basal-bolus regimen with that of a sliding-scale regular insulin with patients with type 2 diabetes. It was conducted at Grady Memorial Hospital in Atlanta, Georgia and at the Jackson Memorial Hospital in Miami, Florida together with the institutional review board at Emory University and the University of Miami, which approved the study protocol. In this trial 130 nonsurgical insulin-naive patients were randomly assigned to receive either sliding scale regular insulin or a basal-bolus regimen. The goal of the study was to determine differences between treatment groups in the average daily blood glucose concentration, number of episodes of hypoglycaemic and hyperglycaemic events, length of hospital stay and mortality rate. For the patients who were assigned to the basal-bolus regimen, a starting total daily insulin dose was calculated, depending on an up-to-date measured blood glucose value. One half of the daily insulin dose was given as a long acting basal insulin once daily, the other half was subdivided into 3 equal short acting bolus insulin doses, in addition with supplement bolus insulin, calculated per sliding scale protocol, to be administered before meals. If a patient was not able to eat, only a supplement bolus dose, calculated per sliding scale protocol, was administered every 6 hours. The dose of the long acting basal insulin was reduced by 20% after an episode of hypoglycaemia or increased 20% if the mean blood glucose level during the day was higher than 140 mg/dl in the absence of hypoglycaemia. Patients randomized to the sliding scale insulin therapy, received regular insulin, which was calculated per sliding scale protocol, four times a day for glucose concentrations higher than 140 mg/dl. If a patient was not able to eat, he or she received regular insulin every 6 hours. The sliding scale protocol differs between three insulin resistances (sensitive, usual, resistant). If blood glucose measurements of a patient repeatedly resulted in too high values, the patient was increased from the insulin resistance sensitive to usual or usual to resistant. The other way round, the insulin resistance of a patient could also be

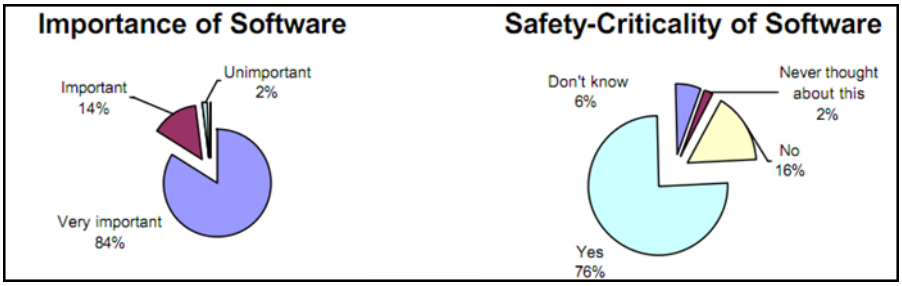

Figure 3.1: Importance and criticality of software in medical environment (Feldmann et al. (2007))

decreased in case of hypoglycaemia. In both treatment regimens the blood glucose was measured before each meal and at bedtime, or every 6 hours in case of missing nutrition.

The results of the study show that a treatment with basal-bolus insulin offers a significant improvement in glycaemic control compared to the sliding scale therapy. The mean blood glucose concentration, measured by patients assigned to the basal-bolus regime results 164 mg/dl, the mean blood glucose measurements at the sliding scale regimen results 188 mg/dl. There were no differences between the two treatment groups according to episodes of hypoglycaemia or length of hospital stay.

3.2 Medical Device Directive (MDD) for Software

Software becomes more and more an integral part of medical devices. To ensure safety and efficacy of a medical device that contains software, it is necessary to concretely know what the software should intend and to prove that the software achieves this effect without causing unacceptable risks (OVE/ON (2007)). In 1996, Wallace and others publicised that 10% of medical product recalls were caused by integrated software. Wallace pointed out that one possibility for this result is the rapid increase of software in medical devices (Wallace and Kuhn (2001)). In 2006 a German survey on medical device recalls indicated that software was the top cause for risk related to construction and design defects of medical device products. The study shows that 21% of medical design failures are caused by software defects (Feldmann et al. (2007)). Figure 3.1 indicates the relevance of software in medical device products. According to Feldmann, 84% of surveyed companies rate software in medical devices as very important and another 14% classifies software as important. The figure also

indicates that software is a safety-critical element of the medical product in 76% of the cases and only 16% of software modules have non safety-critical functionalities.

3.2.1 Conditions of the Medical Device Directives for Software

One of the key issues, a manufacturer of software that is intended for medical use, has to think of is: Does the product come within the scope of the medical device directive? To answer this question it is necessary to know what distinguishes software from medical device software. The Global Harmonization Task Force has proposed the following definition for medical devices:

> *"Medical devices are any instrument, apparatus, implement, machine, appliance, implant, in vitro reagent or calibrator, software, material or other similar or related article, intended by the manufacturer to be used, alone or in combination, for human beings for one or more of the specific purpose(s) of*
>
> - *diagnosis, prevention, monitoring, treatment or alleviation of disease,*
> - *diagnosis, monitoring, treatment, alleviation of or compensation for an injury,*
> - *investigation, replacement, modification, or support of the anatomy or of a physiological process,*
> - *supporting or sustaining life,*
> - *control of conception,*
> - *disinfection of medical devices,*
> - *providing information for medical purposes by means of in vitro examination of specimens derived from the human body,*
>
> *and which does not achieve its primary intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means"* (ISO (2007)).

Medical device software is defined by the IEC 62304 standard as a software system, which was developed to be integrated into a medical device, or for the intended use as an independent medical device. Medical software which comes within the scope of the MDD has to comply with the same rules as medical devices and therefore risk analysis, change requests on requirements, software life cycle management and stringent documentation of all activities have to be performed.

In order to prevent later errors, the development of medical device software is regulated by various standards and laws, which describe recommended software lifecycle models and tools, which should be used by software engineers. The objective of these standards is to define general process steps to produce safe and high quality medical device software.

3.2.2 Relevant Standards for Developing Medical Device Software

Following standards have been considered as relevant for the development of medical software which falls within the scope of MDD (Hall (2010)):

- ISO 13485 standard: defines the requirements for a quality management system for medical devices.
- IEC 62304 standard: has emerged as a global benchmark for management of the software development lifecycle.
- ISO 14971 standard: has traditionally been adopted as the base standard for risk management for medical devices and will also be used for software.
- IEC 62366 and IEC 60601-1-6 standards: provide information about the application of usability engineering for software as medical devices.

3.2.2.1 Software Life Cycle Processes for Medical Device Software (IEC 62304)

IEC 62304 provides a framework of life-cycle processes with tasks and activities, prepared for safe design and maintenance of safety-critical medical device software. The standard defines requirements which developers have to follow in order to fulfill

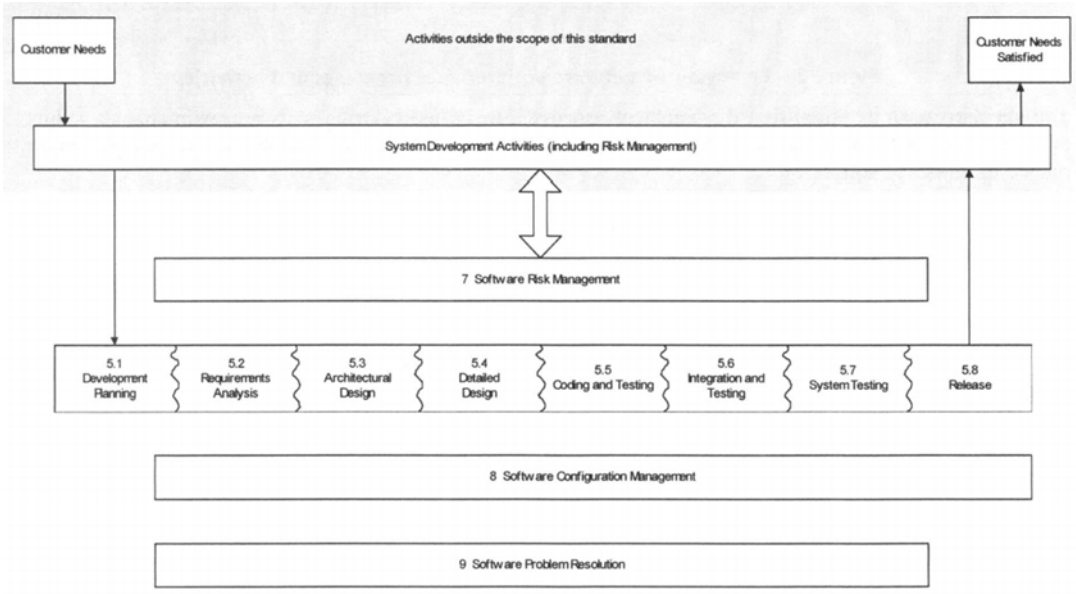

Figure 3.2: Overview of software development process and activities (Box numbers correspond to clauses of IEC 62304) (OVE/ON (2007))

that the implemented software can be later used in medical environment. The standard applies to the development of medical device software, if the software itself is a medical device or if the software is only an embedded part of the medical end device. IEC 62304 requires that software is developed within a quality management and a risk management. According to IEC 62304, before starting the development process, the tended software has to be assigned to a safety class. Safety classes describe the potential impact of a risk to the patient, the user or other persons, which can be caused by the software. The standard differentiates between 3 safety classes:

- Safety class A: No injury or damage to health is possible
- Safety class B: No serious injury is possible
- Safety class C: Serious injury or even death is possible

Figure 3.2 gives an overview of the software development process and its activities, required by the IEC 62304.

Main processes according of the software life cycle process

As a first process phase, IEC 62304 requires to create a software development plan which must include:

- Processes, which are used in the software development.
- Results to be delivered.
- Used software technologies.
- Addressed problems that are discovered in every phase of the life-cycle.
- Traceability between system requirements, software requirements, software system testing and risk control measures that are implemented in the software.

The IEC 62304 standard requires a comprehensive analysis of the system requirements and a documentation of thus requirements. The documentation must include the most important system parameters, such as functional requirements, input and output of the system, interfaces to other systems, requirements for data security, etc.
After analyzing the requirements of the medical device software, the requirements have to be documented in a system architecture, which describes the structure of the software and identifies the software components to be implemented.
Afterwards, the software architecture must be subdivided as long as it is presented by software units in the detailed design phase. These units must then be implemented and verified by unit tests. Finally the implemented software units have to be integrated and validated by integration tests.
Before the software system is ready for release a set of system tests with input values and expected output values have to be defined and successfully executed.

Supporting processes of the software life cycle process

According to IEC 62304 a **software risk management** process has to be performed during the development process. A detailed description of the general requirements of a risk management process is established in the ISO 14971 standard, which is presented in chapter 3.3. Furthermore, a scheme to clearly

identify configuration items and their versions must be established in the **software configuration management** process, which should be reviewed for the project. For used SOUP modules, the title, the manufacturer and the SOUP identification has to be documented. SOUP (Software of Unknown Provenance) is already developed software, which is not intended for use as medical device software. According to IEC 62304, for every problem that was discovered in a software product, a problem report has to be created in the **software problem resolution** process. A problem report must include the type and the criticality of the problem. Afterwards the problem has to be analyzed and if possible the reason for the problem should be identified.

IEC 62304 allows developers lots of flexibility in how they run the individual processes. In addition, also any number of further processes can be executed if deemed as necessary by the developer. However, it is important that each process is well documented in order to ensure traceability over the whole development process.

3.2.2.2 Application of Risk Management to Medical Devices (ISO 14971)

The effective management of risk is crucial to the success of software projects. Risk management is intended to minimize the occurrences of unexpected events and to keep all possible outcomes under tight control. Risk management in software development primarily concerns the development process itself, rather than the end product (Roy (2004)). Especially in the development of medical software products, the use of a well structured risk management is essential. Failures of the developed software can have potentially catastrophic effects that can lead to injury of a patient or even to death. The International standard ISO 14971

> *"specifies a process through which the manufacturer of a medical device can identify hazards associated with a medical device, estimate and evaluate the risks associated with these hazards, control these risks, and monitor the effectiveness of that control"* (ISO (2007)).

The international standard primarily deals with processes for managing risks, related to the patient but also to the operator, to other persons, to equipment and to the environment. ISO 14971 assumes that the concept of risk consists of two components:

- The probability of occurrence of harm.
- The consequences of that harm.

It accepted that the stakeholders understand that the use of a medical device entails some degree of risk and that the acceptability of a risk is influenced by the above mentioned components. Figure 3.3 shows the required risk management process, according to ISO 14971. Therefore the manufacturer of the medical device has to analyze possible risks, evaluate these risks and finally control these risks. Every step of the risk management process has to be well documented. At the beginning of the risk management process, the manufacturer has to perform a **risk analysis**, where he has to describe the intended use and foreseeable misuse of the medical device. Furthermore, the manufacturer has to identify qualitative and quantitative characteristics that could affect the safety of the medical device. At the risk analysis, all possible hazards associated with the medical device in both, normal and fault conditions, have to be identified. Finally, for each identified hazard, the associated risks have to be estimated. If a risk for some reason can not be estimated, the possible consequences have to be listed for use in risk evaluation and risk control. At the **risk evaluation** the manufacturer has to examine the estimated risk for each identified hazardous situation. Then, the defined risk criteria are used to determine whether the estimated risk is acceptable or not. At the risk evaluation the developer also has to decide if a risk reduction is necessary. If the evaluation reveals that a reduction is necessary, the manufacturer has to identify **risk control** measures that reduce the proper risk to an acceptable level. ISO 14971 provides the following three risk control options to reduce a risk:

1. Provision of inherent safety by design
2. Provision of protective measures

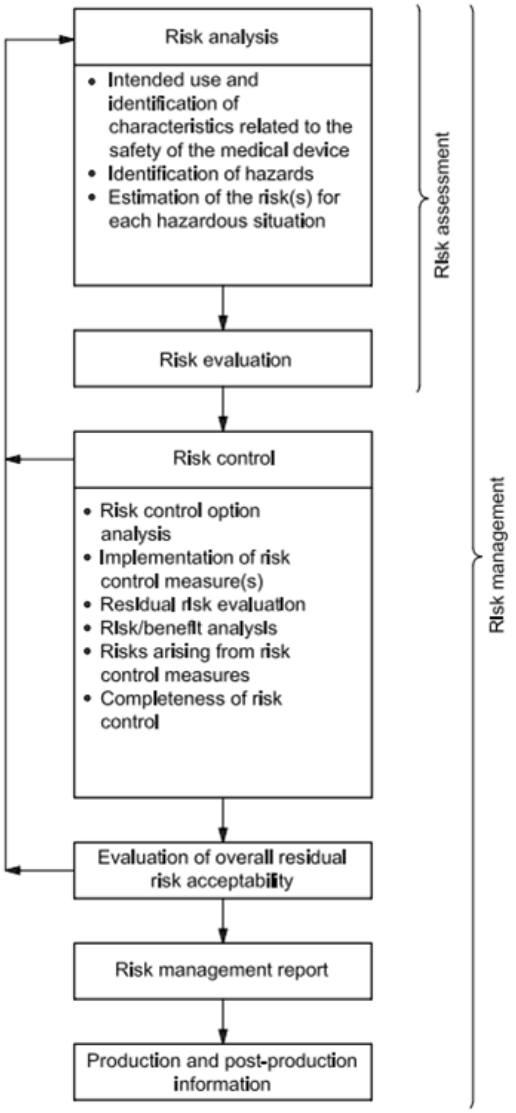

Figure 3.3: Overview of risk management process, according to ISO 14971 (ISO (2007))

3. Provision of safety information.

According to ISO 14971, the manufacturer should try to use the risk control options in the order, listed above. Finally, the implementation of each risk control measure has to be verified and documented. After all risk control measures have been implemented and verified, the manufacturer has to decide again whether all risks lie in the acceptable range. The results of every step of the risk management process have to be recorded in a risk management report. At the end the manufacturer has to monitor production and post-production information that can affect their risk estimates.

3.2.2.3 Quality Management Systems - Requirements for Regulatory Purposes (ISO 13485)

ISO 13485 specifies requirements for a quality management system for the design, development, production, installation and maintenance of medical devices. The standard requires the manufacturer to establish, document, implement and maintain a quality management system, within the development process. According to ISO 13485 the manufacturer has to:

- identify processes required by the quality management system,
- identify sequence and reciprocity of these processes,
- define methods for the effective implementation of these processes,
- provide required resources for the implementation of these processes,
- detect and analyze these processes and
- take measures to achieve planned results.

Furthermore, the manufacturer has to perform a detailed documentation, which has to consist of quality targets, quality policy, used methods and a quality management handbook (CEN (2009)).

3.2.2.4 Application of Usability Engineering to Medical Devices (EN 62366)

Errors, caused by lack of usability in medical devices software give increasing cause for concern. However, many manufacturers of medical device software pay too little attention to the usability of their products. EN 62366 defines requirements for a process to analyze, specify, develop, verify and validate usability aspects of medical devices. According to EN 62366 the manufacturer has to implement, document and follow a usability orientated development process to ensure safety for patients, users and other persons in terms of usability. EN 62366 should guarantee that the final user interface is intuitive and easy to learn or rather to use. Used methods and results, during the usability orientated development process, must be documented. Chapter 3.3.2 and chapter 3.3 provide basics about used techniques to fulfill the user‘s needs and to ensure a high level of usability (CENELEC (2008)).

3.3 User-Centered Design Approach

The development of mobile applications in a medical context provides engineers with a complex task. In addition to the aim of supporting the daily medical routine, usability is an additional important issue, according to clinical safety, to consider when designing the user interfaces and system functionalities. Therfore IEC 62366 (see chapter 3.2.2.4) and IEC 60601-1-6 provide guidlines about the application of usability engineering.
2005, Holzinger indicates that there are five essential usability characteristics that should be part of any software project (Holzinger (2005)):

- **Learnability**: The user should be able to rapidly begin working with the system.
- **Efficiency**: The user should attain a high level of productivity.
- **Memorability**: The user must not forget how to operate the system.

- **Low error rate**: The user should make fewer and easily rectifiable errors while using the system.

- **Satisfaction**: The user should be happy to work with the software.

Non-intuitive usability often leaves the user frustrated and unable to complete simple tasks. The lack of usability in medical devices is dangerous, and can lead to unforeseeable risks to patients. Therefore, the consistent pursuit of a user-centered design is a crucial condition to understand the users, their environment and the context in which the application is used (Hameed (2003); Wu et al. (2007); Holzinger and Errath (2007); Holzinger et al. (2008); Svanaes et al. (2010)). The term "user-centered design" describes a development process that aims to enhance user satisfaction by focusing on direct user feedback in the early stages of the development (Lau (1997)). End-users can influence the engineering process in different ways but the general concept of a user centered design is that users are involved in one way or another (Abras et al. (2001)). One of the most widely used user centered design methods invites users to evaluate existing software (prototypes) in usability trials. With the results, the software can be refactored, according to the needs of the end users. Usability trials can be performed with end users (test methods) and/or without end users (inspection methods). The aim of inspection methods is to identify usability problems and to improve the usability of a design by checking it against established standards. In contrast to inspection usability methods, test methods provide direct information about how people interact with the system. Figure 3.4 shows the comparison of the most widely used usability evaluating techniques. According to Holzinger, the Thinking Aloud (THA) test method is the most valuable usability engineering method. At this method, participants have to continuously think out loud while using the system. The advantage of the THA method is that by verbalizing the end user's thoughts, developers are able to understand how they view the system and can identify the major misconceptions.

	Inspection Methods			Test Methods		
	Heuristic Evaluation	Cognitive Walkthrough	Action Analysis	Thinking Aloud	Field Observation	Questionnaires
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no

Comparison of Usability Evaluation Techniques

Figure 3.4: Usability evaluation techniques according to Holzinger (Holzinger (2005))

3.3.1 Participatory Design

Another user-centered design method is known as "Participatory Design". Participatory Design, which has its origins in Scandinavia, is based on the principle that end users are involved as partners with engineers and consequently have strong influence during the whole design process. Aim of this user centered design method is to ensure that the product will behave to its intended purpose in the intended environment. At the Participatory Design method it is recommended that the team uses prototypes, mockups or paper-based outlines in order to avoid misunderstandings between end-users and developers. An advantage of using a Participatoty Design during the development process is that

> *"a deeper understanding of the psychological, organizational, social and ergonomic factors that affect the use of computer technology emerges from the involvement of the users at every stage of the design and evaluation of the product."* (Abras et al. (2001)).

According to Abras, this approach leads to products that are more efficient, effective and safe and these products require less redesign. Other advantages of the Participatory Design method are that the users develop a sense of ownership for the product and that more creative design solutions to problems can be generated.

Clear disadvantages of the Participatory Design method are that the development process takes more time, requires additional design team members, and consequently is more costly. Furthermore, Participatory Design can lead to products that may be too specific for general use.

3.3.2 Requirement Enineering

The history of software development has shown that requirement engineering is the most critical phase in the software development process. The success of a software system can be measured by the degree to which it meets the purpose for which it was intended. Requirements Engineering in software development is

> *"the process of discovering that purpose by identifying stakeholders and their needs and documenting these in a form that is amenable to analysis, communication and subsequent implementation"* (Nuseibeh and Easterbrook (2000)).

However, there are inherent difficulties in the requirement engineering process. For example, the goals of stakeholders often vary or change during the development process. This makes it necessary to apply requirement engineering practices in every phase of the software development process (Pandey et al. (2010)). The specialized literature describes various techniques that can be used to impose system requirements. The choice of the technique depends on the time and resources available and on the kind of information that is needed. One of these techniques includes the use of questionnaires, surveys, interviews or analysis of existing systems. Other techniques, which are called group elicitation techniques, aim to foster stakeholder agreement while exploiting team dynamics. These techniques include brainstorming or workshops. Also cognitive techniques can be used to impose system requirements. Cognitive techniques include protocol analysis, in which an expert thinks aloud while performing specific tasks, or laddering, in which targeted questions should identify relationships between product characteristics and their benefit. Another elicitation technique is called 'prototyping'. In software engineering, the word 'prototyping' is known as a

concept of software construction within a general strategy for system development. The idea behind prototyping is to use prototypes as another formal document during requirements analysis to provide a better basis for understanding between developers and users. According to Budde and Zullighoven,

> *"a prototype is an operational model of the application system. It implements certain aspects of the future system"* (Budde and Zullighoven (1990)).

Furthermore Budde and Zullighoven describe that prototypes provide a basis for discussion between users, developers and management and help to clarify questions or help to prepare particular decisions. Prototyping can be used in different ways during the requirement engineering process. Budde and Zullighoven distinguish between several aims of prototyping:

- **Exploratory prototyping** is used to determine the requirements and assessment of certain solutions, focusing on the functionality of the system with the aim to get a clear requirement specification.

- **Experimental Prototyping** focuses on the technical implementation of a development goal in order to collect user experiences with the use of these prototypes at a later time.

- **Evolutionary prototyping** describes the continuous development and improvement of prototypes in an iterative software development process.

No matter which prototyping method is chosen, prototyping provides deep insight into the user's needs and can verify system functionalities in early phases of the software development process.

3.3.3 Human Computer Interaction on Mobile Touch Screen Devices

In 2011, 60 % of mobile devices shipped in Western Europe and North America applied a touch screen as user interface (Gartner (2011)). This is not surprising due to the fact that the touch screen technology is gaining sophistication. A touch screen interface is well suited to small mobile devices, because thus there is no need to compromise the display size of the device. However, designing mobile applications, which are using a finger-based touch screen interface, requires a set of usability aspects to consider. In 2009, Anna Haywood and Gemma Boguslawski did a research on the human computer interaction with mobile devices and offered, as a result, best practice guidelines to help designing and evaluating finger-activated touch screen solutions for small mobile devices. The following list summarizes the most important design requirements, according to Haywood and Bogulawski (Haywood and Boguslawski (2009)):

- Screen representation:
  - Never overload the screen with elements.
  - Avoid that fingers occlude important information, so carefully consider the placement of visual feedback (for example: place visual feedback above, not under, a selected item).
  - Image Icons, such as Image buttons, should be labeled with supplement textual clues to avoid confusion.
  - In order to not overload the screen, keep labels and instructions short and simple, but avoid abbreviations if possible.
  - Use familiar icons and color conventions so that the user can associate with them.
  - To enhance visibility, there should be a high contrast between touch elements, text and background colors.
  - Icons should be suitably sized and spaced to avoid accidental selection of nearby icon elements.

- Virtual Keyboard:
  - If a virtual keyboard is necessary offer a clear access to it.
  - Avoid a multi-tap configuration of the keyboard; each character should be placed on one key. Offer an easy way to change between different text input modes.
  - Allow a horizontal view of the keyboard.
- Navigation:
  - Minimize steps to access or perform core functions.
  - Allow a direct navigation to return to the main menu.
  - Ensure consistency throughout the interface.
  - Allow actions to be readily reversible.
  - Keep response times short, otherwise offer information if the system is busy.
  - Design for limited input methods.

Of course these guidelines are very general. In order to ensure best usability of a mobile touch screen application, usability tests with end-users are unavoidable. However, these guidelines can be integrated in early design steps of the application and can therefore avoid time-consuming and costly design refactoring in later development steps.

3.4 Technical Materials

For implementation of both, the frontend and the backend application of the inpatient glucose management system a set of technical materials were used. The most important ones are presented in the following chapters.

3.4.1 Apache Maven

Both, the frontend application and the backend application use Apache Maven 2 as a building tool. This chapter will provide basic information about Apache Maven. As a knowledge source for this chapter the official Maven website [1] was used.

Apache Maven is an open source tool, for building and managing any Java-based project. The primary goal of Apache Maven is to make the build process of a Java application easier so that the developer does not need to know details about underlying mechanisms. Furthermore Maven provides a uniform build system, which means that if someone is familiarized with how one Maven project is built, he automatically knows how all Maven projects are built. Maven adheres strictly to the principles of 'best practices development', so for example, execution and reporting of unit tests are part of the Maven build lifecycle.
Maven projects have a common directory layout. In the project home directory there is the POM.xml (Project Object Model) file, as heart of each Maven project. The POM.xml file includes information about the project's structure, versioning, configuration management, resources, dependencies, etc. In addition to the POM.xml file, the home directory of a Maven project usually consists of a *src*-directory, including all of the project's source material, and a *target*-directory, in which the outputs of the build are placed.
Apache Maven builds a project according to a so-called build lifecycle. The build lifecycle consist of different phases, which are executed sequentially. The most important phases of the lifecycle are listed below:

- **Validate**: Checks if all necessary information is available to build the project.
- **Compile**: Compiles the source code of the project.
- **Test**: Uses a unit testing framework to test the compiled source code
- **Package**: Packages the compiled source code, according to the package type, defined in the POM.xml file.

[1] Apache Maven Project - http://maven.apache.org/ (last access 07/2011)

- **Integration-test**: runs optional integration tests.
- **Verify**: verifies if the generated package is valid.
- **Install**: Installs the generated and verified package into a local Maven repository.
- **Deploy**: Copies the generated package to a remote repository.

Each of the phases can be executed, using *mvn <phase>*, in doing so, all prior phases will be executed first. For example if *mvn compile* is executed the project will be compiled after finishing the validation phase. As already mentioned, the POM.xml file is the basic unit of work in Maven. It is an XML representation of the Maven project. It contains all necessary information about a project and configuration of plugins to be used during the build process. The POM.xml file requires at least a description of the project, including the *modelVersion*, which indicates what version of the object model this POM is using, the *groupId*, which describes an identifier to group different Maven projects, the *artifactId*, which is the name of the Maven project and the *version* of the project. For default, Maven will package the source code of the project to a *.jar*. The package type can be changed by adding the *packaging* tag to the project description. A sample POM.xml file of a project, named 'GluCoManSys', with the package type 'war' can looks like below.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>eu.reaction</groupId>
  <artifactId>GluCoManSys</artifactId>
  <version>1</version>
  <packaging>war</packaging>
</project>
```

If a Maven Project depends on other software modules, such as libraries, these software modules and their location have to be defined in the POM.xml file under the *dependency*-tag. Apache Maven 2 downloads the defined dependencies during the build process and takes care of other libraries, which depends on the defined

one. The example below indicates how to integrate the JUnit library into a Maven project:

```
<project>
  ...
 <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.0</version>
      <type>jar</type>
      <scope>test</scope>
    </dependency>
    ...
  </dependencies>
</project>
```

All dependencies are defined inside the *dependencies*-tag. *groupID*, *artifactID* and *version* identify the desired library. *type* corresponds to the packaging type of the dependency. The *scope* 'test' indicates that the dependency should be only available for the test compilation and execution phases. Maven also provides inheritance between projects. Therefore, a parent project must exist, which has the packaging type 'pom'. Values, such as dependencies, which are defined in the POM.xml file of the parent's project are inherited by its children. In a children's POM.xml file, the parent has to be defined, using the *parent* tag. An example is shown in the code snippet below.

```
<project>
  ...
  <parent>
    <groupId>eu.reaction</groupId>
    <artifactId>GluCoManSys</artifactId>
    <version>1</version>
    <relativePath>../GluCoManSys</relativePath>
  </parent>
  <artifactId>GluCoManSys_childproject</artifactId>
</project>
```

Furthermore a Maven project can have different modules. Modules are projects which should be executed as a group. Therefore these modules have to be defined in a parent's POM.xml file, as shown in the following example.

```
<project>
  ...
  <artifactId>GluCoManSys</artifactId>
  ...
    <modules>
      <module>GluCoManSys_childproject</module>
      <module>GluCoManSys_anotherchildproject</module>
    </modules>
 </project>
```

An advantage of so-called 'aggregator projects' is that in combination with project inheritance the builds of a parent project and all his modules can be controlled through a single POM.xml file of the parent.

3.4.2 The Android Operating System

Android is an open and free software platform for mobile devices, which was developed by The Open Handset Alliance, powered by Google. Android became very popular in the last few years, firstly because the Android‘s source code is completely free and consequently there are no royalty fees and secondarily because Android is highly suitable for expansion and enhancement (Kuzmanovic et al. (2010)). Aim of the software development under Android is not to reinvent the wheel each time. Based on a set of preinstalled applications, new applications can be developed that use parts of other applications.

3.4.2.1 An Introduction to the Android Operating System

Android relies on a Linux-Kernel, which contains of the core system services. Furthermore, the Android Linux Kernel has some improvements in terms of energy consumption and storage management. Both aspects are very important because

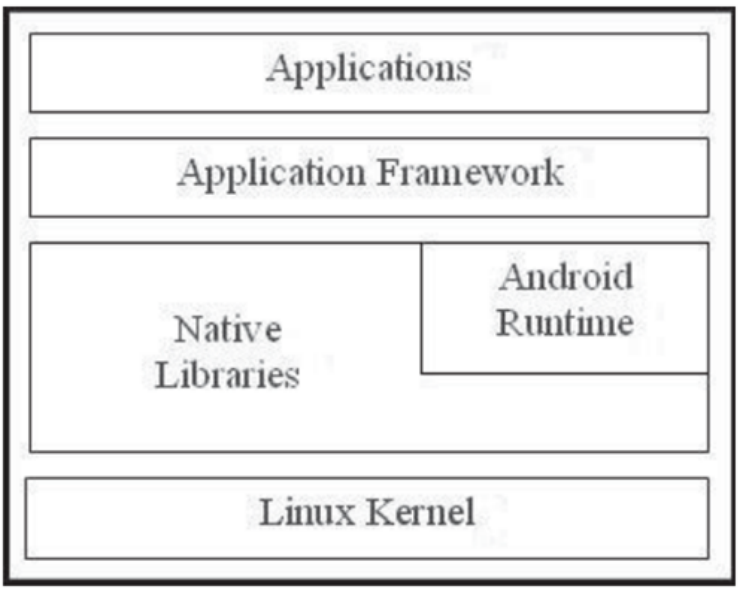

Figure 3.5: General Android system architecture (Kuzmanovic et al. (2010))

storage and energy is limited in mobile devices (Paul and Kundu (2010)). Android consists of a stable runtime environment. Core of the Android runtime environment is the Dalvik Virtual Machine (DVM). Therefore, a program called dx transforms each Android application code into a special DVM compatible byte code, which is finally packaged into an apk-file. An apk-file represents the application package, which can be executed by the DVM at runtime. For each application a separate system process with a DVM is started, which ensures that Android applications do not share storage and that if a system process is killed, only one application is killed too. Furthermore Android contains of a set of native libraries, which are written in C/C++ and are used by various components of the Android system. Android provides system classes in the Android Application Framework that allow Android applications to have access to hardware components. These system classes are completely implemented in Java and can be used by a developer. The top layer of the Android system architecture is the application level, which consists of the Android applications. In the application layer all human-computer interaction and the communication between Android applications take part (Becker and Pant (2010)). Figure 3.5 summarizes the Android system architecture.

3.4.2.2 The Android Sandbox-Principle

Android applications are executed in a so-called sandbox. The Android system with its sandbox principle ensures that an application is running in a dedicated environment and that the application is assigned to an own area in the file system. From the perspective of the operating system an Android Application consists of:

- An own process.
- An own user.
- An own DVM.
- An own area in the file system.
- An own area in the heap memory of the device.

To allow an application access to resources outside the sandbox, permissions have to be explicitly defined in the Android-Manifest of an application. For example, if the application should have access to the internet, the following permission has to be added to the Android Manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

In Android it is also possible that more than one application run in a common sandbox. This can be achieved by using a *sharedUserId*. Therefore all involved applications get a *sharedUserId* defined in the Android-Manifest (Becker and Pant (2010)).

3.4.2.3 Android Components

Android applications consist of essential building blocks, which are called Android components. There are four different types of application components, where each type has a distinct purpose and lifecycle (Becker and Pant (2010)):

- **Activity**: *Activities* are used for representation and management of an application's surface, including the handling of user inputs. An *Activity* should always be implemented to represent exactly one screen. Each element, which is visible on the surface of an *Activity* derived from the Android class View. Views are arranged on the screen by using layouts. An *Activity* can consist of one view element or a set of view elements, which are packed into layouts. More details about views and layouts are provided in chapter 3.4.2.6. Each *Activity* has a specific lifecycle. Therefore an *Activity* can be

  - *active* or *running*, which means that the *Activity* is in the foreground of the screen,
  - *paused*, which means that the *Activity* lost focus but is still visible,
  - *stopped*, which means the *Activity* is obscured by another *Activity*, but still retains all state and member information and
  - *destroyed*, which means that the *Activity* and all its state and member information is destroyed.

  If other applications need memory, *paused* or *stopped* Activities can be destroyed by the system (Android-Developer-Guide (2011)).

- **Service**: An Android Service is responsible for background operations, which do not need any surfaces. Services help to make surfaces faster, according to their responsiveness. Android Services run asynchronously to an *Activity* and report if finished successfully.

- **Content Provider**: The task of an Android Content Provider is to manage structured data sets across application boundaries. A Content Provider can be used for example to import contacts, stored on the device.

- **Broadcast Receiver**: Android often uses system messages to give applications the ability to respond to changes in the system state. Broadcast Receivers receive these system messages. Possible system messages are for example information about low battery state or problems in network connectivity.

3.4.2.4 The Android-Manifest

The Android Manifest is a mandatory xml-file in the root directory of an Android project. The Android Manifest usually contains of the following information (Android-Developer-Guide (2011)):

- The package name of the application
- The version of the application
- The minimal API level, under which the application should run. API level is an integer value that uniquely identifies the version of the Android platform (for example Android 2.2 has the API level 8).
- The Android components (Activities, Service, Content Providers, Broadcast Receivers) of the application
- The permissions of the application
- The libraries, which are used by the application
- Additional information about the application, such a the application name, application icon, etc.

3.4.2.5 Android Resources

Resources are stored by convention in the directory *res* in the root folder of the application. This folder is responsible for handling files, which do not contain any source code. The most important resources, according to Android applications are:

- **Drawables** (*res/drawable*): Include concepts for graphics, which can be drawn to the screen, such as Bitmaps.
- **Layouts** (*res/layout*): Include xml-files, according to the architecture of view elements of an *Activity* or another user interface element.
- **Menus** (*res/menu*): Include xml-files, which define application menus.

- **Values** (*res/values*): Include other resource values, such as the *strings.xml* file, in which textual values of the application are defined.

In order to have access to the resources at runtime, the resources are packaged into an *R.java class* during the compiling process. The resources can be then accessed either within a resource definition (*@ressource-type/resource-name*), or within the Javacode (for example: *getResource().getDrawable(R.drawable.resource-name*). Accesses to resources are always read-only, because the resources are part of the compiled application (Becker and Pant (2010)).
In order to implement applications, which are intended to be used in different countries with different languages, it is important to provide alternative resources, as for example messages. To do this, Android offers an easy way to handle multilingual applications. According to the language settings of the device, Android can use other value directories. So, for example if German should be provided as a second language, a directory *res/value-de* can be added with resources related to the German language. Android always uses the *res/value* directory as a default location. If German is defined as a language on the device, Android will not use the default *res/value* directory, it will use the *res/value-de*, if existing, as a resource directory.

3.4.2.6 Android Layouts and Views

Layouts in Android define rules for the arrangement of surface elements, called Android Views and are usually placed at the resource directory *res/layout*. Alternatively they can also be assembled programmatically in the source code. Android provides a set of different layouts, the most important ones are:

- **LinearLayout**: Linear Layouts arrange view elements linear in horizontal or in vertical direction and scale well.
- **RelativeLayout**: Relative Layouts arrange view elements relative to the margin, do not scale well but have a good performance.
- **TableLayout**: Tabular Layouts arrange view elements in tabular form and

scale well.

- **FrameLayout**: Frame layouts are the most simple layouts and are used only for displaying a single view element.

Each layout has properties, which define their appearance and behavior. The attributes *android:layout_width* and *android:layout_height* are mandatory for all views and layouts. Possible values for these two attributes are *fill_parent*, *wrap_content* or numeric values, declared in pixel (*px*) or density independent pixel (*dp*). *fill_parent* means that the layout or the view takes up as much space as the parent layout or the screen provides. In contrast, *wrap_content* ensures that the layout or view only occupies as much space as its content claims. In order to guarantee that the view or the layout is given an appropriate size on the current screen, *fill_parent*, *wrap_content* or numeric values in density independent pixel should be used (Becker and Pant (2010)). Another important attribute is *android:id*, which can be defined for every view or layout. With this attribute a layout or view gets a unique identifier, which can be used to make the layout or view available in the source code. Layouts can be nested, where child-layouts always inherit the attributes of the parent-layout. According to the frontend application of the inpatient glucose management system, one of the most interesting view elements is the *ListView*. A *ListView* describes a list, which can be scrolled vertically by the user. List items of an Android *ListView* are managed by a list adapter, which is also responsible for formatting list items. To change the appearance of single list items of a *ListView*, Android provides a set of different list adapters, which can be customized by extending the chosen adapter class (Mosemann and Kose (2009)). For the implementation of the frontend application primarily a *SimpleAdapter* was chosen as a list adapter, which maps data to views defined in a layout file. If an *Activity* is used mainly to display a list of data, Android provides a specialized *Activity* class *ListActivity*, which has an implicit *ListView* element as the root of its screen. In addition to the *ListActivity*, which is specialized to show data in a list, Android also provides a *TabActivity*, which is specialized in dealing with tabs. A *TabView* can be used to change between views within the same *Activity* or to change between separate *Activities* (Android-Developer-Guide (2011)).

3.4.2.7 Android Dialogs

The frontend application of the inpatient glucose management system should avoid as much manual user input as possible. However, there is some information, which can not be achieved automatically through the HIS. For these manual inputs the frontend application uses so-called Android Dialogs. A dialog is a window that appears, while the underlying *Activity* loses the focus. Each dialog can get its own layout. Therefore the developer has nearly all possibilities to customize a dialog about his aims. Android offers 4 types of dialogs, which inherit of the base class *Dialog* and can be used by a developer :

- **AlertDialog**: Task of an *AlertDialog* is to prompt a user to perform short tasks, such as selecting an item from a list or entering the username to login.
- **ProgressDialog**: A *ProgressDialog* is and extension of the *AlertDialog* and is responsible for displaying a current progress.
- **DatePickerDialog**: A *DatePickerDialog* allows the user to select the date.
- **TimePickerDialog**: A *TimePickerDialog* provides a dialog to select the time.

Most of the dialogs, used for the implementation of the frontend application, are *AlertDialogs*, because they are fast and on the official Android developer website, *AlertDialogs* are described as the 'suggested dialog type'
(Android-Developer-Guide (2011)).

3.4.2.8 Android Menus

Android generally provides 2 types of menus:

- **Options menu**: Option menus are menus that are activated via the hardware menu key on the device. For each *Activity*, exactly one option menu can exist.

- **Context menu**: Context menus can be activated by clicking (or touching) long on a view element. A context menu can be defined for each view element.

Usually menus in Android are defined as xml-files in the resource folder *res/menu*. The layout definition of the menu can be used by both types, option menus and context menus. For each menu item, which is defined inside a xml layout, the attribute *android:id* must be defined in order to interpret the item at the source code. Additionally a menu can also be defined dynamically at the Java code. This can be helpful if for example menu items or their behavior change during runtime (Becker and Pant (2010)).

3.4.2.9 Automated Testing in Android

Android provides a testing framework, based on JUnit, which offers powerful tools for:

- GUI Tests
- Unit Tests
- Stress Tests

Figure 3.6 summarizes the Android testing framework: Tests in Android are organized into separate Android projects and therefore include own test packages and Android-Manifest files. In an Android-Manifest of a test application the application package, which is the reference to the application that should be tested, has to be defined. An Android test project includes so-called test suites with test classes, where each test class is a container for related test methods. Android allows invoking callback methods in the test code, so it is possible to run through the lifecycle of a component step by step. For Android-independent classes the JUnit *TestCase* class, which does not have any relations to the Android SDK can be used to do unit testing. The Android testing framework also provides classes that create mock system objects in order to isolate tests from the rest of

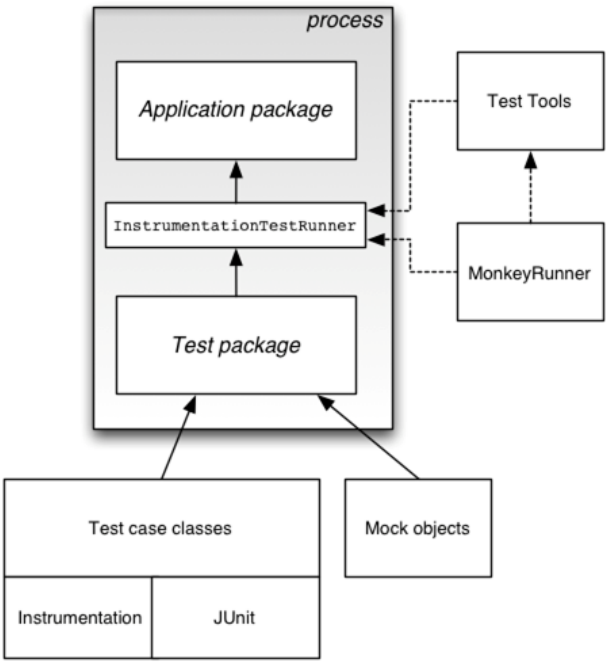

Figure 3.6: Android testing framework (Android-Developer-Guide (2011))

the system. Mock Objects are dummy objects that emulate real code (Mackinnon et al. (2001)). Basis of an Android test is the *InstrumentationTestRunner*, which extends the JUnit test runner framework, loads, sets up, runs and tears down each test case. The *InstrumentationTestRunner* is instrumented in order to control how the test package loads the application and the test cases under test. The *InstrumentationTestRunner* also supports other testing tools, such as *MonkeyRunner*, which is a tool that controls an Android device from outside of Android code and is primarily designed to perform functional-level application tests (Becker and Pant (2010)).

GUI Tests GUI Tests or Activity Tests in Android are JUnit test cases, which start the target application and simulate the surface of the application. GUI Tests should verify that the application prints out the desired output on every input at the surface. GUI testing can be both, testing the whole application's surface or just separate activities. To write GUI tests on Android, it is enough to know the application from the user perspective, it is not necessary to know any source code. The simulation of an application's surface is done by a so-called robot (Becker and Pant (2010)).

Unit Tests Unit Tests should ensure that single classes behave as expected. Therefore for each class to be tested a test class is created, which should validate all visible methods of the target class. In Android a distinction must be made between Android-dependent classes, which use the Android API and Android-Independent classes, which do not refer to the Android library. To implement Unit Tests for Android-independent classes the *junit.framework.TestCase* class can be used to do unit testing. To implement Unit Tests for Android-dependent classes, the system environment, or rather the Android components have to be simulated. For each Android component the Android testing framework provides a base class (Becker and Pant (2010)):

- **ActivityUnitTestCaseClass**: Unit testing of Android *Activities*
- **ProviderTestCase2**: Unit testing of Android Content Providers
- **ServiceTestCase**: Unit testing of Android Services
- **AndroidTestCase**: An *AndroidTestCase* is used whenever only access to a resource of an application or an application context is required. The *AndroidTestCase* can also be used to test a Broadcast Receiver.

Stress Tests Stress Tests are tests, that use an application on the device over a long period of time by random and thus set the application under 'stress'. Stress tests can be performed using *Monkey*, which is a program that runs on the device and generates random streams of user events (Becker and Pant (2010)).

3.4.2.10 Optimization Rules in Android

Mobile devices do not have the same hardware resources as for example a desktop or a server system. Consequently implementing mobile applications effort some rules to keep the memory usage, the battery consumption and the processor load low (Thompson et al. (2009)). Lots of books about Android describe what should be considered by implementing mobile applications in Android, as well as in other mobile operating systems. One of the most important optimization rules is to keep

the source code as slight as possible. Good performance of an application is more important than the appearance of the source code. Therefore, interfaces and not absolutely necessary methods, such as setter and getter methods should be avoided. It is also pointed out to avoid object creation, because object creation needs time and memory (Becker and Pant (2010)). The Android developer website suggests preferring static elements over virtual elements.

> *"If you don't need to access an object's fields, make your method static. Invocations will be about 15%-20% faster"*
> (Android-Developer-Guide (2011)).

Also accesses to the database should be minimized, because SQL requests require lots of memory accesses and memory space for presenting the results (Becker, 2010). Friesen and Smith indicate that integer operations on Android devices are twice as fast as floating-point operations, so floating-point operations should be minimized (Friesen and Smith (2011)).

3.4.3 Model Driven Architecture (MDA)

Developing large software projects often means a large programming effort. Object orientated programming and improved software development processes should help to reduce the programming effort in those software projects. However, software development is still very labour intensive, because most of the steps must be performed manually by programmers. The idea behind Model Driven Architecture is to automate steps in the software development, so that code must not been implemented manually by programmers (Wimmer (2005)). Therefore the focus of the developer shifts away from implementation to modelling. Basis of the Model Driven Architecture concept is thus the step by step transformation of abstract models of software systems to their implementation. Usually the Unified Modelling Language (UML) is used for modelling. In literature, advantages of a software development process using MDA consist of:

- increased productivity,

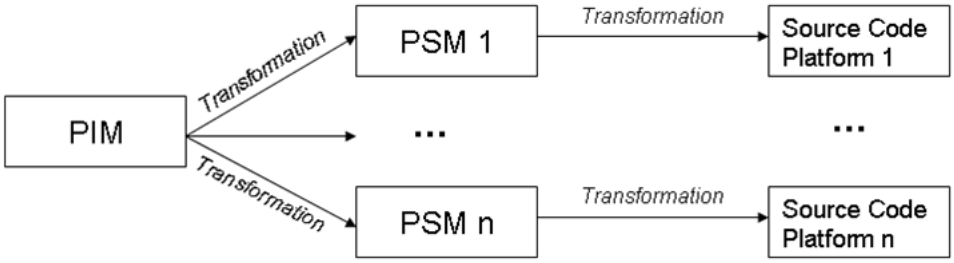

Figure 3.7: Tranformation from PIM to source code Schulz (2005)

- increased portability of applications,
- improvements according to interoperability of components,
- advantages of documentation and maintenance and
- high quality of generated source code (Warmer and Kleppe (2003)).

Important is the distinction between platform-independent (PIM) and platform-specific models (PSM), where a PIM describes the technical aspects of a software system without reference to specific technologies and a PSM consists of additional information in order to enable the transformation into a concrete technology. A PIM has to be transformed to platform specific source code. Therefore, different plugins called cartridges are used, which analyze the PIM and construct a PSM and finally produce platform specific source code. Figure 3.7 provides an illustration of the transformation from a PIM to source code. For the development of the Java-based backend application AndroMDA is used as a MDA tool. AndroMDA is an open source framework and can generate source code from a platform independent UML model which is automatically integrated into the build process. AndoMDA is well documented on the official website (Bohlen et al. (2011)), where most of the facts, described in this chapter were retrieved. AndroMDA provides cartridges to generate code for Hibernate, EJB, Spring, Webservices and Struts. AndroMDA takes an UML model as an input and generates several components which are connected with each other. Figure 3.8 shows various application layers, which are supported by AndroMDA: The **presentation layer** contains of components, which are responsible for user interaction. AndroMDA offers two technologies to build web based presentation layers: Struts and JSF (Java Server Faces). The **business layer** represents the

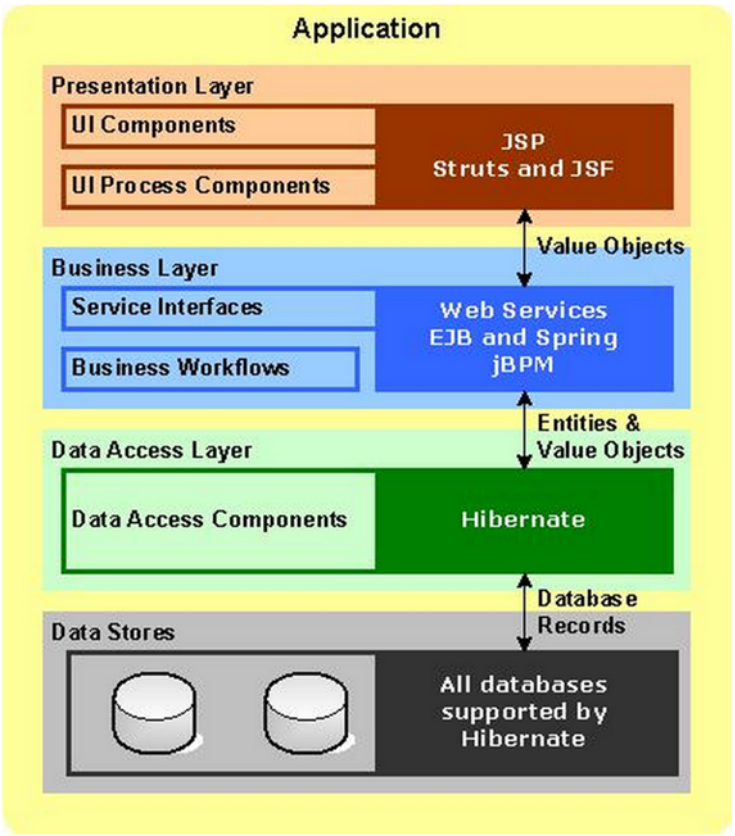

Figure 3.8: Layers supported by AndroMDA (Bohlen et al. (2011))

core functionality of the backend application. The business layer packages all necessary information into so called value objects and transfers these value objects to the presentation layer. The **data access layer** is responsible for accessing and manipulating data. AndroMDA provides Hibernate [2] to generate the data access layer. Hibernate is a solution for java environments to map data from an object model representation to a relational data model representation. Hibernate also provides data query and retrieval facilities to avoid manual data handling. AndrodMDA creates DAOs (data access objects) for the entities, which have been defined in the UML model. The DAOs use Hibernate to convert database records to objects. The **data stores layer** is responsible for data storage of the application. If Hibernate is used to access data, each database which is supported by Hibernate can be chosen.

3.4.4 Unit Testing

For the term 'unit testing' lots of different definitions can be found in literature. For example Koomen and Pol define unit testing as

> *"a test, executed by the developer in a laboratory environment, that should demonstrate that the program meets the requirements set in the design specification"* (Koomen and Pol (1999)).

2006 Runeson wrote:

> *"Unit testing means testing the smallest separate module in the system"* (Runeson (2006)).

Generally, unit testing can be seen as a process of evaluating units of source code in order to determine if they are fit to use. The selected unit test framework, for testing the backend application, is TestNG[3], where NG stands for Next Generation. TestNG is a testing framework, inspired by JUnit, which supports

[2]Hibernate - http://www.hibernate.org/ (last access 07/2011)
[3]TestNG - http://testng.org/doc/index.html (last access 07/2011)

powerful features, such as flexible annotation-based testing, testing groups, parallel testing, etc. TestNG classes don't have to extend particular classes and their methods don't have to follow naming conventions. Each unit test method is flagged by the @Test annotation and the Java asserts function can be used to test calculated values against expected values. Another annotation @BeforeMethod ensures that a defined method is executed before every test and the annotation @AfterMethod can be used to make sure that a defined method is called after each test. TestNG unit tests are usually organized into logical sets, called test suites. Test suites are defined by a XML configuration file. A TestNG configuration file consist of information how the tests are organized and where they are located. A powerful feature of TestNG is its support for test groups. For example a test group can specify a certain part of the system, such as database tests, user interface tests, etc. Each test case can belong to one or more test groups. An advantage of this feature is that test groups can be executed separately at different times or in different places. Another powerful feature of TestNG is the ability to run unit tests simultaneously in several threads (Smart (2008)).

4. Methods

A team consisting of physicians and nursing staff of the Division of Endocrinology and Metabolism at the Medical University of Graz, as well as engineers from Joanneum Research and the Medical University of Graz, was established to impose requirements for the in-hospital glucose management system. Therefore, an iterative requirement engineering process was established, which is indicated in figure 4.1. The requirement engineering process began with a detailed analysis of current clinical workflows, according to the stationary treatment of patients with diabetes. In parallel, relevant publications relating to the ideal in-hospital management of hyperglycaemia including validated glucose control protocols, were identified by the team and discussed with diabetes specialists. The results were described in a status report, based on various patient scenarios, as a starting point for the target analysis. The most important identified requirements were implemented in a first software prototype. The last step of the first iteration of the development process, involved performing real-life usability trials in the Medical University Hospital of Graz. The second design iteration started with the integration of the test results into the requirement set as the basis for the second target analysis. With the revised and extended requirements the full functionality

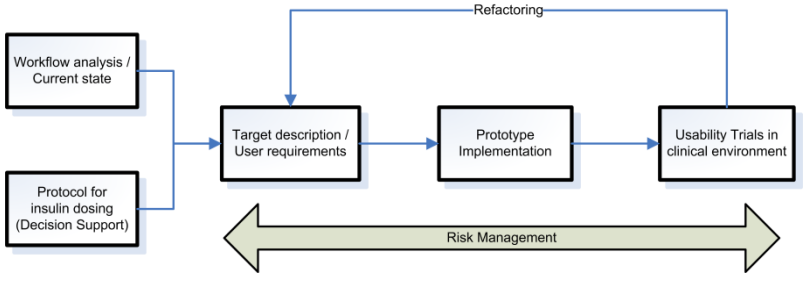

Figure 4.1: Requirement engineering process (Höll et al. (2011b,a))

of the in-hospital glucose management system could be designed and simulated in paper-mockups. Finally the mockups were again evaluated by end-users and modified until the design and functionality could be completely verified.
Right now, the development process is in its third iteration. The aimed prototype as a final product of the third iteration is currently implemented in an Android-based mobile application and should be evaluated in clinical field trials after finishing the implementation. The development process, including every iteration step, was accompanied by continuous interdisciplinary meetings regarding risk identification, evaluation and the setting of appropriate measures to avoid these risks. Emphasis is placed on both technical and medical risks.

4.1 Workflow Analysis/Current State

The treatment of hospitalized patients is primarily based on routine processes. To avoid complications, caused by a too high or too low blood glucose level, patients with diabetes need a time-coordinated workflow management during their insulin therapy. So, as a first part of the development process, lots of time was investigated to identify the common general ward workflow, according to the stationary treatment of patients with diabetes, in detail. The workflow analysis took part at two clinical wards in the Medical University Hospital of Graz (Endocrinology and Cardiology) using comprehensive surveys and different patient scenarios. The 'state of the art' of the inpatient glucose control was analysed with a semi-structured interview. Information from nurses and medical staff was obtained to understand workflow patterns for medical decision making and related problems. Finally, protocols have been developed for the analysis of 50 patients with established or newly diagnosed diabetes mellitus, in order to have an overview of the current situation of the glucose management in these two wards. Especially the following parameters were assessed during the analysis:

- Actual number of glucose measurements.
- Mode of diabetes treatment.

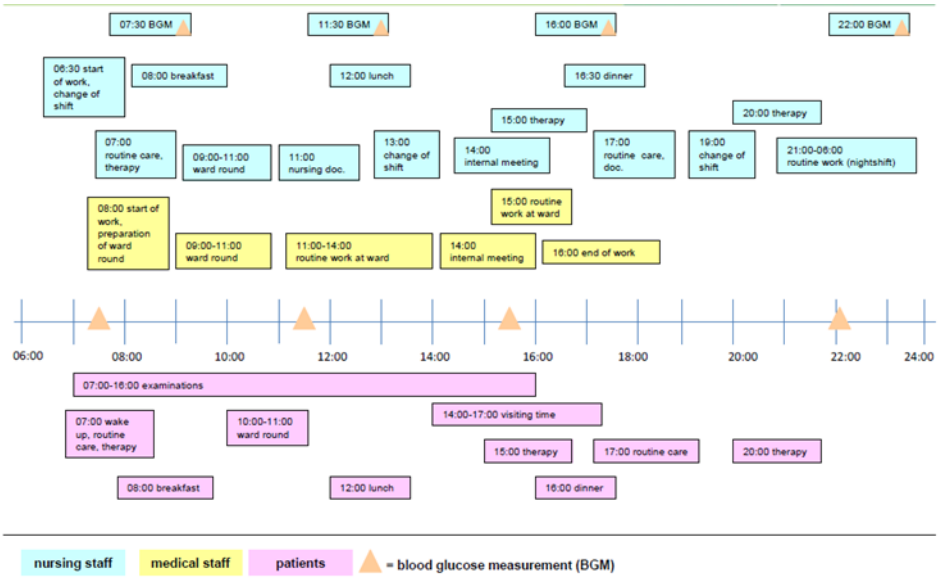

Figure 4.2: Inpatient daily routine

- Use of insulin.
- Algorithms of dose adjustments.
- Overall quality of diabetes control.

The protocols have been submitted to the Ethics Committee of the Medical University Graz, where they have been approved. The achieved daily in-hospital routine is shown in figure 4.2. The analysis of the daily routine revealed that a usual day of a hospitalized patient includes three meals. Before each meal and before bedtime the blood glucose level is measured followed by a possible subsequent insulin administration. After breakfast the physician makes his ward round to order the therapy for the next 24 hours. Although the workflows of all three groups run independently, they have to fit with each other. Out of the daily routine, the workflow is very complex and differs from day to day depending on the patient's health status as well as on planned examinations and potential delays. These circumstances have to be taken into account for safe glycaemic control. The current inpatient workflow related to the glucose management at the Endocrinology and Cardiology wards of the Medical University Hospital of Graz is described in figure 4.3. The whole process of the in-hospital glucose management starts with the admission of a patient with the diagnosis of diabetes mellitus (1).

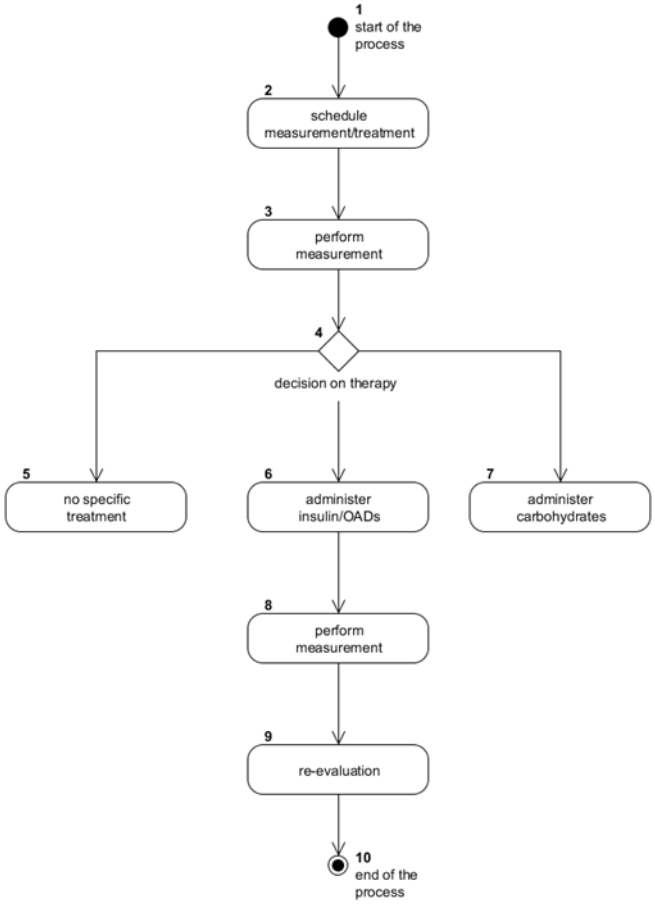

Figure 4.3: Workflow of the inpatient diabetes treatment

The diagnosis of diabetes mellitus and thus for glucose management is based on blood glucose measurements and/or medical history of the patient. Afterwards a physician has to define and to document the treatment and the number of measurement of the patient with diabetes mellitus (2). The determination of the treatment results from a variety of important criteria, such as medical history, actual and general health status, interactions with other medications, associated conditions, etc. As indicated in the daily routine, before meals and before bedtime the blood glucose level is measured (3). The measurement is performed, usually by a nurse using a POCT (Point of Care Testing) device. The measured blood glucose value is manually transferred to a paper chart and automatically transferred by the POCT device to the laboratory information system. Based on the actual blood glucose value, the nutrition (fasting or non-fasting), the actual status of the patient (e.g. severity of illness, estimated insulin resistance, concomitant diseases such as diarrhoea) and scheduled upcoming examinations and treatments, the insulin dose is calculated (5, 6). In case of hypoglycaemia, carbohydrates can be administered to a patient in order to increase the blood glucose level (7). Oral antidiabetic drugs are given at particular predefined time regardless of actual measured glucose values. The dosing is manually transferred to the paper chart by a nurse or a physician. A glucose measurement will be performed again after a certain period of time (usually before the next meal) (8). The glucose value in combination with the chosen treatment and the condition of the patient will be used to evaluate the decisions and to adjust further steps according to the outcome of the treatment (9). The whole process of the inpatient glycaemic control management usually ends with the discharge of the patient (10).

4.2 Protocol for Insulin Dosing (Decision Support)

Only few clinical trials have focused on optimal inpatient glucose and insulin management in the non-critical setting and no definition of adequate glycaemic targets exist. Aim of this development phase was to review the state of the art of

glycaemic management protocols and to find a proper evidence-based protocol to offer a decision support service for insulin dosing, within the inpatient glucose management system. Glycaemic control is based on different protocols, which have different benefits and disadvantages. Research shows that insulin administration should include three components to be effective:

- Basal insulin to achieve a long-term reduction of the blood glucose.
- Nutritional insulin to balance increasing blood glucose values after meals.
- Correctional insulin to provide real-time adjustment of insulin dosing based on the patient's insulin sensitivity.

There are few trials where this concept has been implemented:

- Algorithm of the Rabbit 2 Trial (Umpierrez et al. (2007)).
- Algorithm for the comparison between Detemir plus Aspart versus NPH plus regular insulin (Umpierrez et al. (2009)).
- Algorithm for patients during enteral nutrition therapy (Korytkowski et al. (2009)).

These trials were considered and intensively studied as a starting point. Benefits but also shortcomings were identified. The protocol based on a basal-bolus regimen as provided by the RABBIT 2 trial (see chapter 3.1.2), proved to be most promising for the clinical diabetes experts due to its straightforward advice for insulin dosing, which was shown to be associated with improved outcomes. A significant problem in the analysis of dosage protocols was that important information in these protocols was not described in the necessary depth. For example, there was no clear indication, which blood glucose value (fasting value in the morning or premeal value in the evening) of the last day was used to adjust the daily insulin dose. However, the protocols are supposed to work in an electronic system and therefore all conditions for calculating the insulin dose must be known. The lack of information was discussed in numerous meetings with

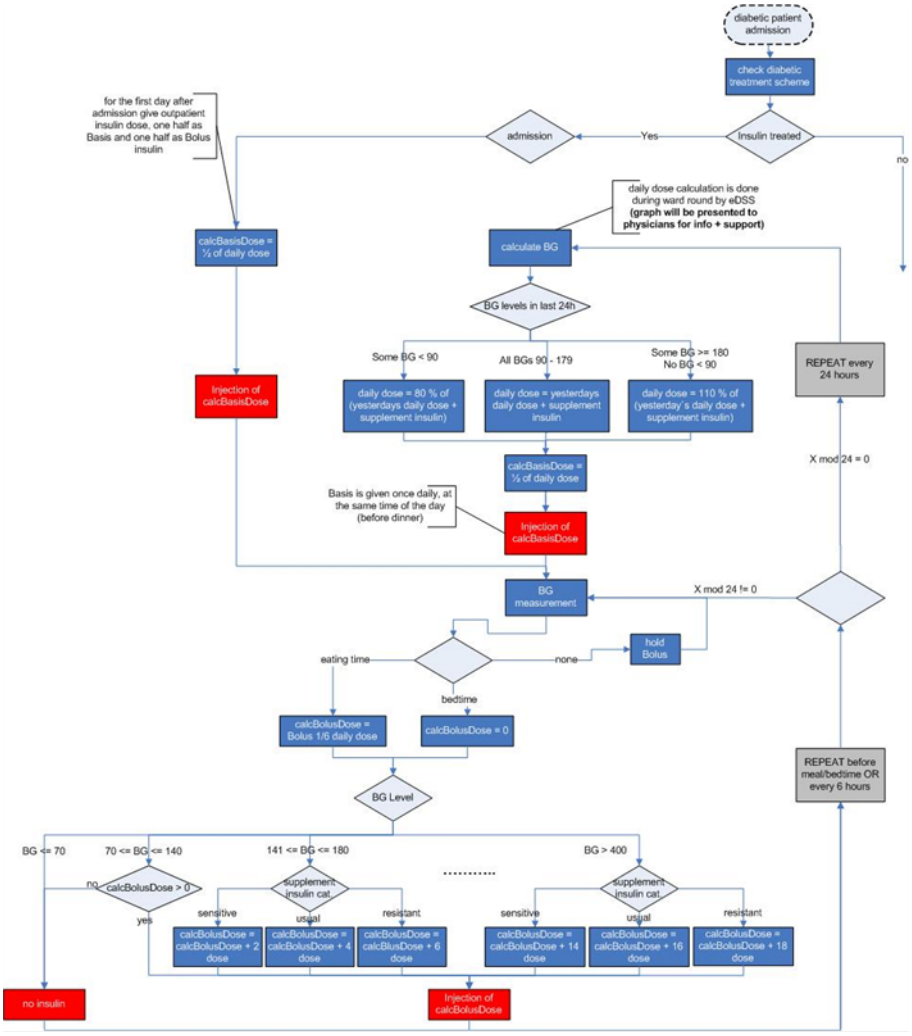

Figure 4.4: Improved RABBIT2 protocol for insulin dosing

diabetes specialists at the Medical University Hospital in Graz and incorporated into the protocol. The meetings resulted in an improved RABBIT 2 algorithm for safe glycaemic control, indicated in figure 4.4. The improved RABBIT 2 protocol is currently evaluated by clinical trials at the Endocrinology and Cardiology wards of the Medical University Hospital of Graz.

4.3 First Iteration - User Motivation

The current state analysis, as well as the analysis and improvement of the RABBIT 2 protocol provided enough knowledge to start a first iteration of the development process. Problems with the current treatment of patients with

diabetes at the analyzed departments at the Medical University Hospital in Graz were identified and possible software solutions were elaborated in the first target analysis. Afterwards the collected basic requirements were implemented in a Microsoft Excel Prototype and tested in usability trials.

4.3.1 Target Analysis of the First Iteration

Aim of the target analysis was to collect first requirements and basic functionalities, according to an inpatient glucose management system. Therefore problems with the current paper-based glucose management were identified, structured, analyzed and documented. For each detected problem a possible software solution was prepared. Table 4.1 shows the detected problems, the workflow phase(s)) in which the problem occured, according to figure 4.3 and an elaborated software solution.

Table 4.1: Results of first target analysis

Problem	**Relation to workflow(s)**	**Solution**
Paper-based access to the data only for one person at the same time.	1-9	Electronic paperless data record with mobile access point (e.g. tablet PC) ensures access for more persons at the same time at different locations.
Difficult traceability of documentation and decision making.	1-9	Usage of centrally managed data repositories to ensure traceability of performed activities.
No data backup in case documents are lost.	1-9	Synchronization of data on server database to ensure backup of data even in case of a system breakdown.

Fixed presentation of data.	1-9	Different modes of visualization to ensure dynamic data presentation.
Suboptimal decisions and treatments, no standardized instructions and decisions.	2,4	Electronic Decision Support system to ensure standardized instructions and decisions (e.g. evidence based medicine, support identification of patients with risk).
Sometimes neglect to perform measurements.	3	Active system-reminder to prevent missed measurements.
Mistakes caused by literal errors or unreadability.	1-9	Automated transfer of measured and relevant data to the patient's record and hospital information system to avoid mistakes by literal errors or unreadability.
Risk of wrong patient identification.	1-9	Automated patient identification to avoid identification mistakes.
Challenge to retrieve archived data.	2,4,9	Electronic archive data repository to make data of former admission available.
Documentation tabular (unclear).	1-9	Visualization of therapy values in charts.
Not an All-in-One workflow.	3-7	Workflow Management of application to approximate an All-in-one workflow (measurement, decision on treatment and insulin administration).

Physician has to command a laboratory analysis of important therapy parameters.	1	Automatic request of important therapy parameters via interface to the hospital information system.
Paper-based documentation is unsanitary.	1-9	Tablet PC disinfectable.
Decision on therapy individually.	4-7	Decision Support or with links to guidelines about decision making.

4.3.2 Microsoft Excel Prototype

The first target analysis helps to get a general imagination of what an inpatient glucose management system could look like and which functionalities could be useful. However, detailed requirements were missing. In order to achieve these requirements a prototype was implemented, based on the ideas and findings of the former development steps. The prototype was implemented in Microsoft Excel 2007 by using Visual Basic for Applications (VBA). Microsoft Excel was chosen, due to the extensive display options of charts and quick and easy visualization of glucose and insulin profiles, as well as optical alarm borders. Furthermore Microsoft Excel is well suited to quickly create prototype solutions. Aim of the Microsoft Excel prototype was to implement step by step the basic functionalities of an inpatient glucose management system. Weekly meetings with nurses and physicians were held in which the design of the user interface, basic functionalities, workflow support and the integration of the decision support service were discussed. For the prototype a Microsoft Access database was used, to store patient and therapy test data.

Figure 4.5 shows a screenshot of the Microsoft Excel prototype, where the following action and information areas can be distinguished:

- Patient data: Patient's name, admission date, insulin resistance, required

Figure 4.5: Miocrosoft Excel prototype

daily insulin dose, borders for hyper-/hypoglycaemia (upper left of the screen)

- Glucose and insulin profiles: Visualization of the glucose and insulin profiles, including glucose values, insulin dosages, insulin on board, target range, borders for hyper-/hypoglycemia, display of food delivery and comment fields (left middle of the screen)

- Basic functionalities: Entering glucose values, creating and selecting a patient, administration of medication, scaling of glucose and insulin profiles (right upper and right middle of the screen)

- Tabular summary of glucose measurement results and administered insulin (bottom of the screen) The prototype also contains a first simulation of the decision support service, which is indicated in figure 4.6.

4.3.3 Testing the Usability of the Microsoft Excel Prototype

In order to implement workflow-supporting software which can be used safely in medical environment, continuous communication to, and feedback from end users

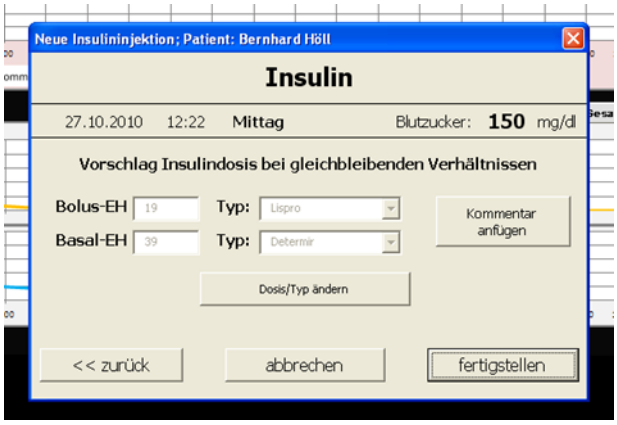

Figure 4.6: Decision support in the Microsoft Excel prototype

are necessary. Therefore, as a last step of the first iteration, usability tests and interviews with clinical end users based on the first prototype of the inpatient glucose management system in clinical environment at the Medical University hospital in Graz were conducted, using three participants:

1. A physician and diabetes specialist with experience of outpatient and inpatient environments.
2. A nurse and head of care of the diabetes ambulance focusing on patients with diabetes type 1.
3. A nurse and head of care of the division of Endocrinology.

Aim of the usability study was to collect the following information and suggestions, arranged to their importance, to the application.

- **Functionality**:
  - Does the system offer all necessary functionalities?
  - Are provided functions unnecessary?
  - Do the provided functions behave correctly?
  - Do the provided functions support the usual workflow?
- **Risks**:
  - What risks can occur while using the application?

  - What preventions can be taken against these risks?

- **Layout**:
  - Is the information clearly arranged?
  - Is the usability intuitive?
  - Do the charts show all necessary information?

During the usability trials, the Thinking Aloud testing method (see chapter 3.3) was used followed by a semi-structured interview. The exact steps were as follows:

- **Introduction**: The investigator gave participants a short description about the project and the aims of the usability trials.
- **Presentation of the prototype**: To familiarize the participants with the prototype, the investigator firstly presented the basic functionalities. Afterwards the participants were invited to perform simple actions on the prototype by themselves.
- **Test Tasks**: Five different tasks with varying levels of difficulty, corresponding to the role of the participants (physician, nurse), had to be performed. The participants were encouraged to vocalize their thoughts (thinking aloud). Aim of these tasks was to highlight weaknesses in the prototype's functionalities, workflow support and usability.
- **Interview**: The investigator asked participants about questions of interest, which had been defined previously and had not been achieved during the test tasks. During the interviews also experiences of the participants, according to mobile devices (smart phones, tablet PCs) were identified. Finally a dummy paperboard mockup, which should simulate a potential mobile device for the inpatient glucose management system, was presented to the user in order to get an imagination of the required size and shape of the desired end-device.

All tests were documented on video, the results were interpreted and suggested improvements were documented as a basis for the second iteration.

4.3.4 Results of the First Iteration

The first iteration, especially the usability trials with the use of the Microsoft Excel prototype provide a variety of new information, according to workflow support, functionalities, displaying patient and therapy information, as well as information to desired potential mobile devices. A selection of the most important results is listed below:

- Each type of insulin, which is available on the market, is identified by a certain color. The application should use these colors in order to provide a good and easy distinction.
- In addition to the graphical representation, the application should also allow the user to view measured blood glucose levels and administered medication in tabular form.
- Only the most important information should be displayed in order to keep clarity. Therefore, only the blood glucose profile, including occupied nutrition and administered medication should be visualized on the patient‘s main screen. Insulin profiles should optionally be available in a submenu.
- Especially during the first days after the admission of a patient it is important to know what therapy the patient had before. Therefore the application has to provide information about the pre-therapy to the user.
- The ordered daily insulin dose must be clearly distinguished from the partial insulin doses, which are administered before meals in order to avoid confusion.
- Beside the most important demographic values to identify a patient, the defined insulin therapy, including current regimen, ordered medication and the insulin resistance has to always be observable.
- Manual user inputs should be minimized.
- The user should be able to add textual comments to every blood glucose measurement in order to provide and document special events, according to a

patient.

- Decision Support for insulin dosage is very desirable. However, the suggested dosage has to be confirmed by an authorized user.
- A reminder for "open tasks" should be provided by the application in order to support clinical personnel to meet all requirements and criteria for decision support.
- A laptop as a mobile device is not desirable, because it is too heavy and has negative effects on the communication between physician and patient, because of its size and shape.
- The desired mobile device should be easy to carry and it should be able to hold the device in one hand.
- The modification or deletion of input values should be possible within a certain period of time (24 hours) under certain circumstances (no influence of subsequent activities).
- In addition to the support of the basal-bolus regimen, also a workflow without decision support should be possible. In this case, the application only serves as a documenting tool.

The findings from the usability trials were finally discussed and reviewed in the team and the improvements were accepted in the requirement specification for the second iteration.

4.4 Second Iteration - Mock-Ups

The user evaluation of the first glucose management prototype and the target analysis of the second iteration resulted in an extensive requirement specification document, which consists of identified functional requirements as well as non-functional requirements related to the inpatient glucose management system. Afterwards the revised user requirements were drawn up in a paper mockup

screenplay consisting of all functionalities, using Visio stencils for Android. Finally, the results were discussed in the team. Aim of the second iteration was to collect all user requirements in order to start the technical specification of the system.

4.4.1 Target Analysis of the Second Iteration

At the second target analysis a detailed requirement specification document was written. As a first step, the results of the first iteration, especially the results of the usability trials were analyzed and discussed and improvements were transferred to functional and non-functional requirements. To provide a better overview of the functional requirements, they were structured into the following 6 main function groups with various use cases.

- **User Management**: The user management is responsible for adding, editing and deactivating the system's users. Only users with administrative roles have access to the user management. In later versions the user management should be synchronized with the user management system of the hospital. Thus, for example users can be retrieved directly form the HIS and manual editing of users will not be necessary any more.
- **Patient Management**: The patient management is responsible for adding and editing the patient profile, displaying and sorting patient lists, searching for patients, and the enrolment of patients for the glucose management system. Data retrieved from the HIS, including personal data and hospital-specific data, will be transferred into the system automatically. Supplement data which is needed by the system has to be entered manually during the enrolment.
- **Drug Management**: The drug management takes care of adding, editing and deleting drugs, including insulin and other medications, which should be available by the system. Only users with administrative roles have access to the drug management. Future versions of the system should interface the register of authorized drugs. Thus, adding, editing or deleting of drugs will not be necessary any more.

- **Glucose Management**: The glucose management is the central part of the system. Data retrieved from the HIS, data entered by the enrolment and also data collected during blood glucose measurements and insulin administrations are visualized in this component. The collected and displayed information is the base for the decision support service for insulin dosing and thus, for the therapy adjustment during the ward round. The main actions performed within the glucose management are 'Blood Glucose Measurement', 'Insulin Administration', and 'Therapy Adjustment'. 'Blood Glucose Measurement' guides the user through the process of retrieving and storing a blood glucose value. Blood glucose values are measured by a POCT (Point of Care Testing) device and are automatically transferred to the LIS and afterwards to the mobile device. So the user does not have to enter the measured blood glucose level manually. Once the blood glucose level is gathered, the system will be able to suggest an insulin dose automatically by using the action 'Insulin Administration', which also requires specifying if a meal is intended. The function 'Therapy Adjustment' will be used to adjust the daily insulin dose and to define the therapy for the next 24 hours. The 'Therapy Adjustment' should be performed at the daily ward round.
- **Open Task Management**: The open task management is responsible to remind users of tasks, which have to be performed in the near future or tasks which have not been done yet.
- **Default Parameter Management**: The default parameter management is responsible for setting default parameters according to patients' therapies.

Figure 4.7 summarizes the identified Use Cases, structured to their proper component.

The use cases diagram shows 5 different actors, which have been identified at the target analysis. The reason for the distinction between different user roles is that physicians have other privileges than nurses. The identified user roles, including their privileges are listed below:

- **Nurse**

Figure 4.7: Identified use cases during the second target analysis

  - Enrol patient for Glucose Management
  - Insulin administration
  - Discontinue Glucose Management
  - Blood glucose measurement
- **Administrative Nurse**
  - Enrol patient for Glucose Management
  - Insulin administration
  - Discontinue Glucose Management
  - Blood glucose measurement
  - Drug Management
- **Physician**
  - Enrol patient for Glucose Management
  - Insulin administration
  - Discontinue Glucose Management
  - Blood glucose measurement
  - Adjust therapy
- **Administrative Physician**
  - Enrol patient for Glucose Management
  - Insulin administration
  - Discontinue Glucose Management
  - Blood glucose measurement
  - Adjust therapy
  - Drug Management
  - Default Parameter Adjustment
- **Administrator**
  - Enrol patient for Glucose Management

  - Insulin administration
  - Discontinue Glucose Management
  - Blood glucose measurement
  - Adjust therapy
  - Drug Management
  - Default Parameter Adjustment
  - User Management

Parallel to the functional requirements, non-functional requirements were identified for the inpatient glucose management system. The identified non-functional requirements consist of:

- Mobile access point in wards of inpatient environment
- Improvement of quality and accessibility of documentation
- Medical and technical safety
- Selection of a mobile device for inpatient glucose control based on given requirements:
  - Lightweight/portable
  - Easy-to hold/handle and ergonomic design
  - Spill an drip resistant (easy disinfection)
  - Inputs via touch screen
  - Wireless communication
  - Ease of operation
- Multi-user availability

4.4.2 Mock-up Story Board

After the specification of functional and non-functional requirements, these requirements had to be implemented into a new prototype. Due to the high

Figure 4.8: Mock-up of the main screen of the glucose management

number of new user requirements, a software prototype seemed to be not suitable, because the implementation would be too time-consuming. Consequently it was decided to design the collected user requirements in a mock-up story board, which should simulate the functionalities. Based on the acquired requirements for a mobile device the Samsung Galaxy Tab was chosen, as a template for the simulated user interface. The Samsung Galaxy Tab uses Android 2.2 as an operating system. So, the mock-ups were designed, using the Android GUI prototyping stencils for Microsoft Visio[1]. The functionalities have been represented in documented paper mock-ups and afterwards discussed in the team and adapted accordingly. As an example, figure 4.8 shows the drawn mock-up of the main screen of the glucose management.

Basic data of the selected patient can be found on the top left side of the mock-up. The basic data consists of the name and the age of the displayed patient. Also the location and the admission date are presented to the user in order to certainly identify the patient. Moreover the basic data consists of the current selected therapy regimen, ordered medication, as well as the insulin resistance of the patient. 'DSS' indicates that the decision support is activated which means that the system automatically gives dosage advice for the next insulin administration.

[1]http://www.artfulbits.com/Android/Stencil.aspx

The bottom left side presents the three main activities of the glucose management, including ‘Blood Glucose Measurement’, ‘Insulin Administration’ and ‘Therapy Adjustment’. The glucose and insulin charts in the middle of the mock-up visually display the most important measurement and insulin administration parameters, which were identified by the target analysis. The user can also view the course of the therapy in tabular form, by touching the ‘Glucose Table’ tab or he can view or adjust therapy parameters, by touching the ‘Therapy’ tab. At the top of the screen, a menu bar allows the user to change to the patient list, to view open tasks and to log out of the glucose management system. By touching the ’View Full Screen’ button the user can view the visualization in full screen mode. With the ‘Edit/Delete Recent Activities’ button, the user can modify or delete already finished activities.

4.4.3 Evaluation of Mock-up Story Board and Results of Second Iteration

In order to evaluate the designed mock-ups, Microsoft Power Point presentations were prepared to simulate use cases of the application. Each slide consisted of exactly one mock-up and additional textual descriptions, which should present the functionality of the presented mock-up and should simulate user-operations. The slides were continuously presented to clinicans at the Medical University hospital in Graz. Afterwards suggestions for improvements, as well as all other comments were documented, analyszed and discussed in the team. The mock-ups were adjusted repeatedly and improvements were incorporated directly into the paper-mockups until they finally met the requirements of the clinicans.

In contrast to the results of the first iteration, where the focus was primarily on identifying new requirements, the second iteration helps to pack the requirements into an already verified design. With the completion of the second iteration the functionality and the design of the application were mostly fixed. Next, the technical requirements had to be considered in order to implement the user requirements into a safe, executable medical device software application.

4.5 Risk Management

The inpatient glucose management system will provide a decision support for insulin dosing and therefore falls within the scope of the medical device directive. Consequently IEC 62304 requires performing a risk management within the development process. A team consisting of physicians and nursing staff at the Medical University of Graz, as well as engineers from Joanneum Research and the Medical University of Graz, was established to perform the risk management.

At first the softare has to be assigned to a safety class, which describes the potential impact of a risk, which can be caused by the software. As described in chapter 3.3, IEC 62304 differentiates between 3 safety classes:

- **Safety class A**: No injury or damage to health is possible
- **Safety class B**: No serious injury is possible
- **Safety class C**: Serious injury or even death is possible

A wrong treatment with insulin can lead to hypoglycaemia or hyperglycaemia which can effect dangerous complications. However, the aim of the inpatient glucose management system is to support medical decisions and physicians finally have to decide by themselves, how to treat a patient. Therefore the application can be assigned to safety class B.

After the assignment to a safety class, possible risks, which can be caused by the application were identified in the risk analysis. Both components of a risk, probability and consequence, were analysed separately for the estimation of a hazard in the risk evaluation. With the estimation of a risk, it had to be decided whether the risk was acceptable or a measure to reduce the risk was necessary.

4.5.1 Risk Analysis

The risk analysis process was started with a description of the intended use and characteristics related to the safety of the medical device. According to these characteristics, the following questions were discussed:

- Does the medical device provide interpretive statements?
- Does the medical device use other medical devices, drugs or medical technology?
- Does the use of the medical device require special training or special skills?
- Can design features, according to the user interface cause errors in the use of the application?
- Is it possible (and if so, how) that the medical device is used intentionally in a false way?
- Does the medical device save data that is critical to patient care?
- Can the medical device affect the mobility of the clinical staff?

Afterwards potential hazards were identified and risks were estimated for hazardous situations. The identification of the hazards and risks were performed in a kind of group brainstorming, within the interdisciplinary team. Emphasis was based on both, technical and medical risks. Finally, the identified risks were structured and documented.

4.5.2 Risk Evaluation

After the risk analysis the risks had to be evaluated. Aim of the risk evaluation was to decide if a risk was acceptable or not acceptable. Therefore, the probability and consequence of each identified risk were analysed separately.
The consequences of each risk were estimated, using the criteria, indicated in table 4.2. The probability of each risk was estimated, using the criteria, presented in

Table 4.2: Criteria for risk consequences

Severity	**Description**
catastrophic (5)	Death of the patient
critical (4)	Permanent disability or life-threatening injury
serious (3)	Injury/disability that requires a knowledgeable medical intervention
low (2)	Temporary injury/disability that requires NO knowledgeable medical intervention
negligible (1)	Inconvenience or temporary discomfort

Table 4.3: Criteria for risk probability (per patient)

Probability	**Description**
frequent (5)	daily
likely (4)	weekly
occasional (3)	monthly
remote (2)	annually
unlikely (1)	every few years

table 4.3:

The estimation of the risk results of:

$$Risk = Probability \times Consequence$$

After the analysis of the probability and consequences of the risks, the risks had to be categorized, according to figure 4.9.

Therefore, a risk can be:

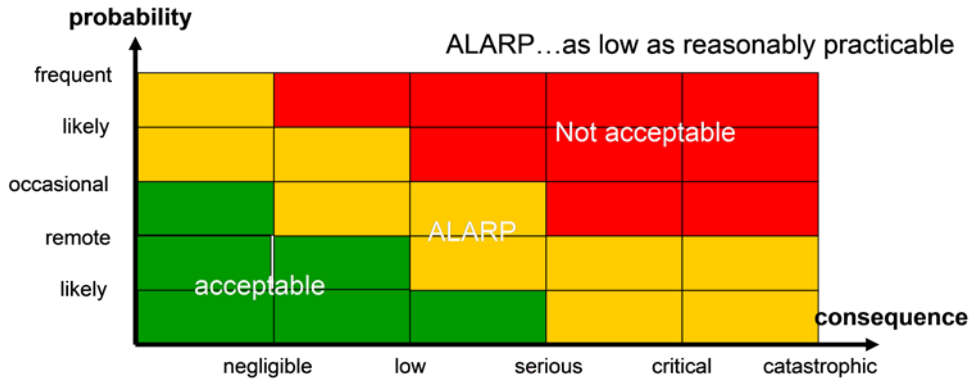

Figure 4.9: Acceptance of risks

- **Acceptable**: The probability that the risk occurs is very low. If the risk still occurs, there are no serious consequences. A measure of risk reduction is not necessary.
- **ALARP** (as low as reasonably practicable): Either the probability that the risk occurs is low, however the appearance of this risk can lead to serious consequences, or the probability that a risk occurs is likely, but does not lead to dangerous situations. Risks, which are assigned to this category, do not necessarily need a risk control.
- **Not acceptable**: A risk which is assigned to the non-accepted category occurs frequently and can lead to serious consequences. Therefore, measures have to be defined to reduce this risk.

4.5.3 Risk Control

For the case, a risk was assigned, during the risk evaluation, to the 'not acceptable' category, ISO 14971, requires defining measures to reduce this risk. Afterwards, the risk has to be newly evaluated, until the risk can be assigned to the 'accepted' or 'ALARP' level. After risk control it is also required to perform a risk analysis again, because identified measures can lead to new risks. For risk control there is a stepwise approach to reduce risk:

1. Inherent safety by design.
2. Protective measures in the medical device itself or in the manufacturing process
3. Information for safety. Consequently, if practicable, the medical device should be designed to be inherently safe. If this is not practicable, then protective measures such as barriers or alarms are appropriate. The least preferred protective measure is a written warning or contra-indication. It is possible that there is no practicable way of reducing the risk to acceptable levels. In this case, a risk/benefit analysis must be carried out to determine whether the benefit of the medical device outweighs the residual risk.

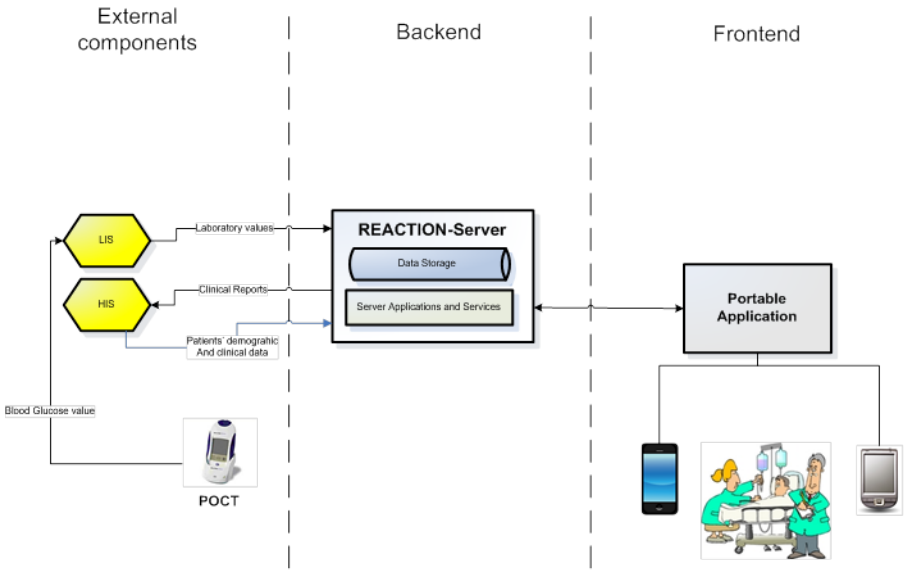

Figure 4.10: Requirement engineering process

4.6 Third Iteration - Practical Implementation

Based on experiences and requirements, which were collected during previous development steps, the technical architecture of the system was built up. Due to maintainability and expandability it was decided to distinguish between an Android-based user interface and a platform independent Java-based backend which contains business logic for the decision support, the reminder functionality, as well as the data storage. The frontend application should present the data, received from the backend, in an appropriate manner to the user and collects new relevant data. The behavior of the frontend application should relate to the clinical workflow in every step, which was identified together with end-users in the design phase. Figure 4.10 indicates the common architecture components of the inpatient glucose management system. The exchange of data between the backend and frontend components is completely done via web services. In order to ensure data security, Fraunhofer Institute, as a project partner, implements a secure Android SOAP (Simple Object Access Protocol) client application, which supports encrypted communication and mutual authentication of both communication endpoints.

The backend application should consist of a configurable interface to query the Hospital Information System (HIS) and Laboratory Information System (LIS) via

HL7 in order to receive actual blood glucose levels (the POCT device automatically sends blood glucose values to the LIS) and demographic values of the patient to avoid manual inputs. HL7 stands for Health Level 7 and is a communication standard for exchanging patient data. At the moment the interfaces to the external components have not been specified more precisely and are therefore not part of this master thesis.

4.6.1 Issue Tracking with JIRA

JIRA is a web-based application, powered by Atlassian[2], to manage issues of any kind within the development process and therefore allows an effective bug-, task- and quality management. During the development process of the frontend, as well as of the backend, JIRA is used as a requirement and task management tool for documenting each implementation step. JIRA acts as preparation for the needs of the IEC 62304 standard, and for ensuring an overview of open and already completed requirements, development tasks and identified bugs. In JIRA, each of these issues can be assigned to an authorized editor who reports after finishing the issue. A provided project summary, including open and resolved issues, helps to have a quick overview of the development progress.

4.6.2 Development of the Backend

The development of the backend is done by engineers of Joanneum Research, as well as by partners of the Computational Medicine Laboratory at the Institute of Computer Science in Greece. Aim of the backend is to provide a data storage for saving necessary patient data and a platform-independent application which is able to tranfer data via web service to a (mobile) user interface. The development of the backend application is not the essential part of this master thesis, however, the next chapters will give a short introduction of how the server database schema, as well as the already implemented web services were generated, using AndroMDA.

[2]http://www.atlassian.com/

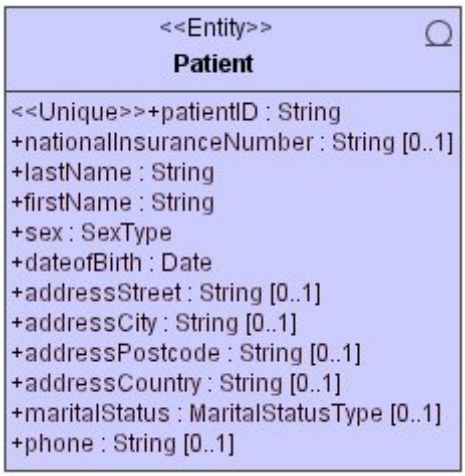

Figure 4.11: UML modelled entity 'Patient'

4.6.2.1 Development of the Backend Using AndroMDA

The schema of the server database was completely created using AndroMDA (see chapter 3.4.3). Let's take a look of how a table can be modelled and afterwards generated automatically by AndroMDA. As an example, the entity 'Patient' with the mandatory attributes name, sex and date of birth, as well as the optional attributes national insurance number, phone number, martial status and the patient's address should be modelled. The sex of a patient can be MALE or FEMALE and the martial status can be UNKNOWN, SINGLE, MARRIED, WIDOWED or DIVORCED. Additionally each patient should have a mandatory unique identifier. Based on these requirements, the class, shown in figure 4.11, can be modelled: To let AndroMDA know that the drawn class is an entity the modelled class must have the stereotype *Entity*. Optional attributes have the multiplicity [0..1], which means that a patient can have one or none of the associated attributes. Each attribute of an entity must have a data type. In the example the data types *String* and *Date* as well as *SexType* and *MartialStatusType* were used. It should be noted that for example the type *String* is not equivalent to the *java.lang.String*. In model driven architecture all elements and their data types are platform independent. The model will be translated to platform dependent code later in the transformation process (see 3.7). Because the sex and the martial status of a patient only accept predefined values, classes for the data types *SexType* and *MartialStatusType* have to be created with the stereotype

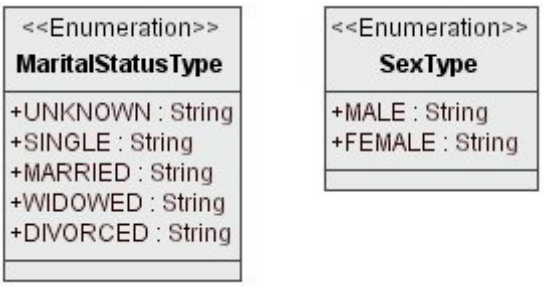

Figure 4.12: Modelling enumerations

Enumeration, shown in figure 4.12.

In order to transform the drawn model to platform dependent code, the maven command *mvn install* can be executed in the source directory of the maven project. In addition to the automatically generation of java classes and source code, AndroMDA can also create tables in a database. Taking the example from above, the maven command *mvn create-schema* will force AndroMDA to create the table *Patient* with the attributes, defined in the UML model, as table columns. If a maven project has more entities, these entities can relate to each other in a different way. AndroMDA allows the common UML class diagram relationships:

- **Associations**: Associations represent relationships between 2 entities. An association can be unidirectional, if the navigability exists only in one direction, or bidirectional, if the association contains navigability in both directions.
- **Aggregation**:

  > *"Aggregation is the part-of relationship"* (Fowler and Scott (1999)).

  and is therefore a special form of the association.
- **Composition**: A composition exists if the existence of an entity depends on another entity.
- **Generalization**: A generalization, according to entities relationships, describes that a child entity is a specialization of a parent entity and therefore the child entity inherits all attributes of the parent entity.

Associations, aggregation and compositions should additionally have a multiplicity at their end to indicate how many entities may participate in the given relationship. Possible types of multiplicities are:

- **0**: 0 entities
- **0..1**: 0 or 1 entity
- **0..***: 0 to an infinite number of entities
- **1**: 1 entity
- **1..***: 1 to an infinite number of entities
- ***: 0 to an infinite number of entities (same as 0..*)

Now, let's take a look of how to model web services. Web services are necessary to transfer data from the business layer to the presentation layer (see figure 3.8). One possibility is to package the necessary information into so called 'value objects'. A value object is characterized that its identity is determined by the values of which it is built (Kuebeck (2009)). As an example a web service *loadDrugs*, which should return a list of available drugs is explained in here. The model of this web service is shown in figure 4.13. First the value objects, which are colored grey, have to be defined. The value objects must have the stereotype *ValueObject* and will later hold the data we want to transfer via the web services. The model includes three value objects:

- *LoadDrugsInVO*: Is the value object the web service will get as a parameter.
- *LoadDrugsOutVO*: Is the value object the web service will return to the client.
- *DrugVO*: *LoadDrugsOutVO* is associated with the value object *DrugVO*, which holds the actual attributes.

The dependency from the entity *Drug* to the value object *DrugVO* will generate helper code in *DrugDaoBase.java* and in *DrugDaoImpl.java* to support the translation between the two components.

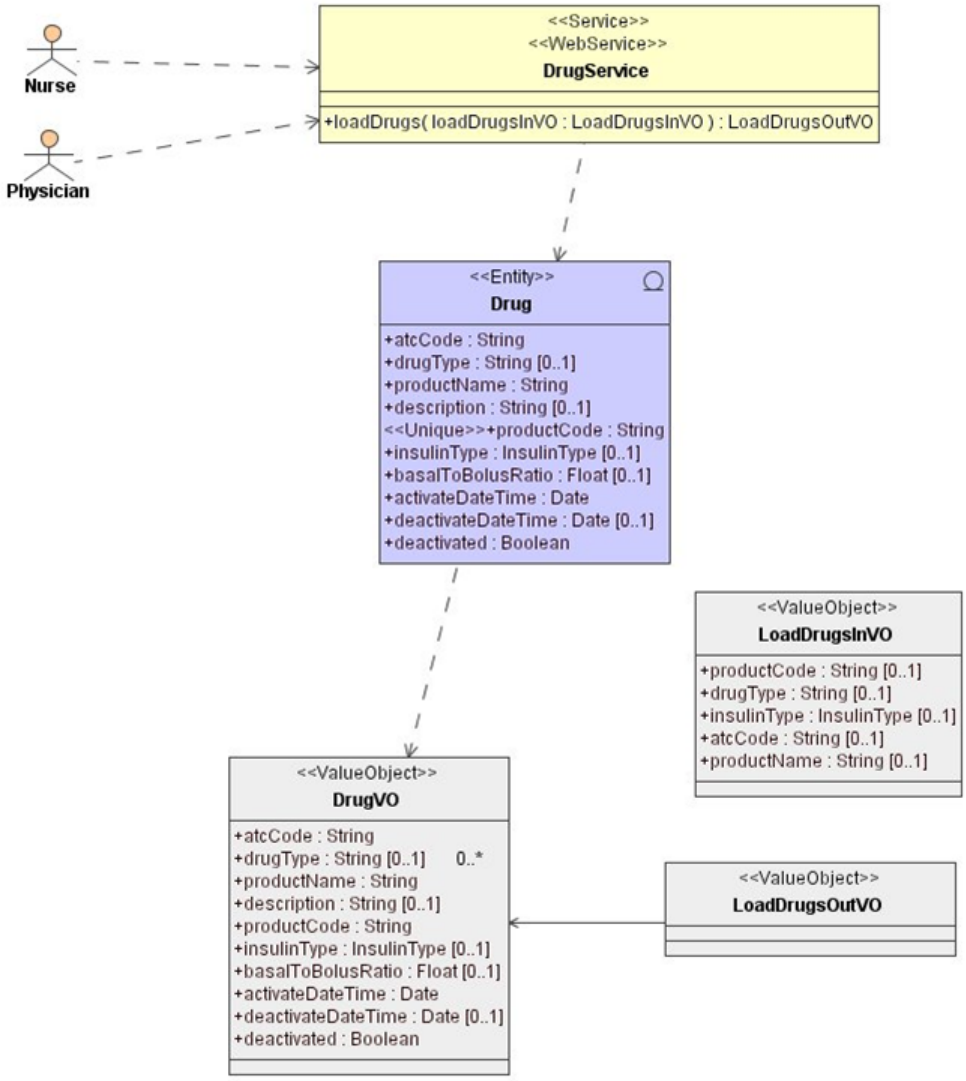

Figure 4.13: UML modelled web service 'loadDrugs'

Next the web service has to be defined. Therefore the *DrugService*, which is colored yellow, with the method *loadDrugs*, and the stereotypes *Service* and *WebService* has to be modeled. The method *loadDrugs* gets the value object *LoadDrugsInVO* as a parameter and returns the value object *LoadDrugsOutVO*. The dependency from *DrugService* to the entity *Drug* ensures that *DrugService* later has access to the *DrugDao*. That‘s it!
Because DrugService is a web service, the value objects are mapped to an xml-structure. What happens exactly by building the project is very complex. In short, the web service cartridge of AndroMDA generates the xsd-file (XML Schema definition) and the wsdl-file (web service description language), which refers to the schema. The generated wsdl-file includes a SOAP (Simple Object Access Protocol) binding, so that the web service can be called by a SOAP message later by the client. Afterwards the java classes, which implement the web service, are generated from the wsdl-file (contract first). The generated output is packaged into a deployable war-file.

4.6.2.2 Testing the Backend, Using TestNG

Standard EN 62304 requires continuous unit testing during the development process. Fort testing the backend functionality, TestNG (see chapter 3.4.4) in combination with Maven 2 (see chapter 3.4.1) was used. The execution of tests is part of the Maven build lifecycle. To tell a Maven application about the usage of TestNG, a dependency has to be integrated into the POM.xml file of the application. The following example indicates how TestNG 5.9 can be integrated into a Maven project.

```
<dependency>
 <groupId>org.testng</groupId>
 <artifactId>testng</artifactId>
 <version>5.9</version>
 <scope>test</scope>
 <classifier>jdk15</classifier>
</dependency>
```

Maven automatically recognizes TestNG test cases and the scope test ensures that the TestNG framework is only available during the Maven test lifecycle phase.

The backend was generally evaluated by unit tests, which can be divided in 2 major categories:

- Domain tests
- Service tests

Domain tests are the unit tests that evaluate data persistence functionality. More analytically, this has to do with entity functionality that implements data persistence transactions (i.e. Create, Read, Update and Delete) and transformation procedures that are required. The functionalities of the entities are tested in certain test cases in order to verify the proper operation and the expected results. Service tests are those tests, which should verify the accessibility of the implemented web services. Therefore, simple test cases at the backend send dummy requests to each web service and parse the status code of their response. Instrumentation tests at the frontend (see chapter 4.6.9) check the correct functionality of the implemented web services in more detail.

4.6.3 Results of the Backend

The implementation of the backend is currently in progress. However, the database scheme and most of the web services are already completed and are available to the Frontend. The set of web services, provided by the current Backend consists of:

- **Enrolment Service**: Service, which performs the enrolment procedure for the patients inside the glucose management system. The service provides methods to start, stop and update the enrolment for a specific patient.
- **Facility Service**: Service, which provides location-dependent information, such as available rooms at a specific ward.

- **Measurement Service**: Service for managing measurement records, including loading measurements, updating measurements and adding new measurements.
- **Drug Service**: Service, which provides medication-dependent information, such as available insulin or tablets.
- **Medication Service**: Service for managing medication records, including loading medications, updating medications and adding new medications.
- **Proposed Medication Service**: Service for managing decision support dependent issues, such as returning calculated insulin dosage suggestions.
- **Patient Service**: Service for finding and returning patients on a specific location.
- **Therapy Service**: Service for setting and loading therapy adjustments.
- **User Service**: Service, which is responsible for the user management, including finding, adding, editing and deleting users of the system.

Accessing the web services from the Frontend is illustrated in chapter 4.6.8.

As already mentioned in chapter 4.6.2.1, the database scheme was completely created by modeling entities. Afterwards AndroMDA generated the database schema out of the modelled entities. Figure 4.14 shows the first part of the entities of the completed data model. The already known entity *Patient* can have none, or one entity *PatientLocation*, which refers to the current location at the hospital, such as the facility, the ward, the room and the bed, related to the patient. The entity *PatientLocation* additionally has a mandatory attribute *startDateTime*, which is the exact time, the patient was assigned to the location, and an optional attribute *endDateTime*, which is the time, the patient leaves the location. A patient can have any numbers of visits (entity *Visit*). Each visit has an admission date and gets a unique admission number. The mandatory boolean attribute *currentVisit* defines if a visit of a patient is the current one. So if a patient has more than one visit, only one *currentVisit* can be true. If a patient is discharged, the attribute *dischargeDateTime* defines the exact time of discharge and the

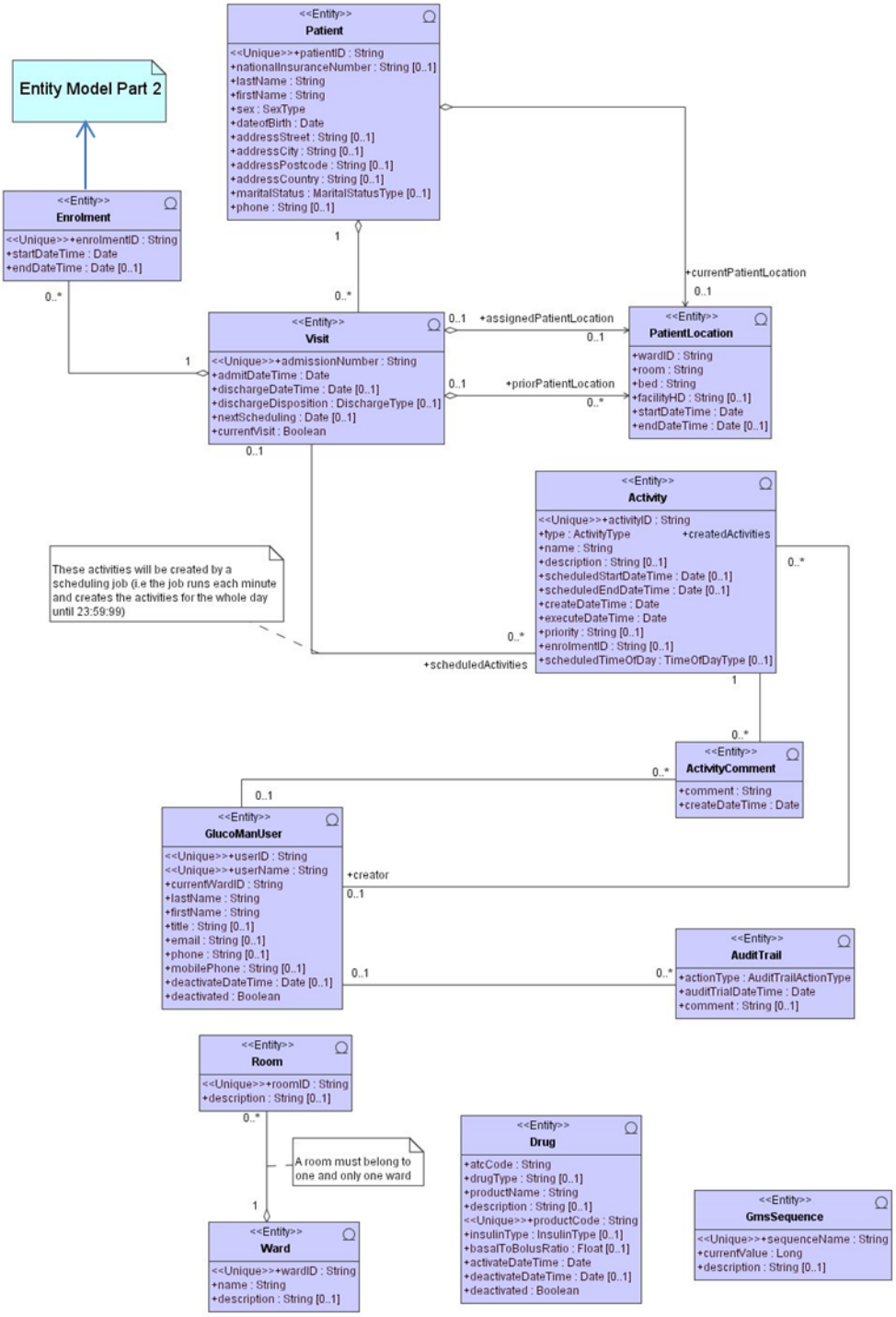

Figure 4.14: Part 1 of the data model

attribute *dischargeDisposition* defines the reason for the discharge. The application should support the clinical workflow, according to the glucose management of patients. This includes helping the medical personnel to remember important activities, such as blood glucose measurements. Therefore the attribute *nextSchedulin* defines at what time the next activity of a patient is pending. The entity *Activity* represents the activities, according to the glucose management. At the moment there are 3 different types of activities:

- Therapy adjustment
- Blood glucose measurement
- Medication (Insulin Administration)

Each activity has a name and a description, as well as the time of execution and creation. The algorithm for suggesting the insulin dose requires the distinction between morning-, midday-, evening- and bedtime activities. This is defined through the attribute *scheduledTimeOfDay*. An activity can also have a scheduled start and end date and information about the priority of the activity. An activity is assigned to the enrolment of a patient via the foreign key *enrolmentID*. Furthermore, each activity can have any number of comments (entity *ActiviyComment*), which consist of a comment to the activity and the exact time, the comment was created. Only if a patient has a current visit the patient can be enrolled for glucose management. Therefore the entity *Enrolment* has a mandatory attribute *startDateTime*, which defines at what time a patient was enrolled, as well as an optional attribute *endDateTime*, which defines at what time the enrolment of a patient was stopped. Furthermore every enrolment has the unique identifier *enrolmentID*. The second part of the entity model will give more detailed information about the enrolment and the associated entities.
An activity can only be performed by a user (entity *GlucoManUser*). Each user has a unique user name and user id to login to the glucose management system. Beside general information about the user, such as name and email, each user is assigned to a ward. If the *boolean* attribute *deactivated* is set to false, a user is not allowed to login. In this case, the attribute *deactivateDateTime* specifies the exact

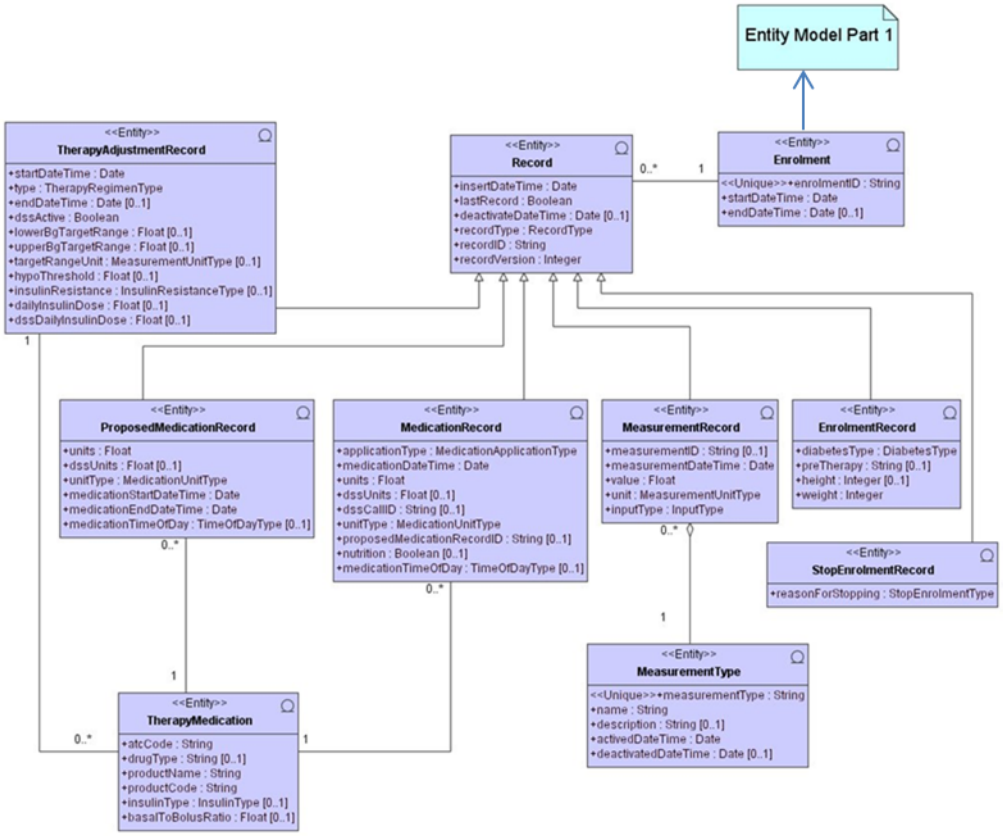

Figure 4.15: Part 2 of the entity model

time the user has been deactivated. The entity *AuditTrail* ensures that each creation and deactivation of a user is documented with the exact time and an optional comment. The entities *Room* and *Ward* include all available wards and rooms. It must be noted that a room must belong to one and only to one ward. Finally, the entity *Drug* represents all drugs, which are available within the glucose management system. A drug consists of the mandatory attributes *atcCode*, which corresponds to the Anatomical Therpeutical Chemical classification code to categorize drugs, *productName*, which is the standardized name of the drug, *productCode*, which is a unique code, every drug must have, *activateDateTime*, which is the time the drug was applied in the system and the *boolean* attribute *deactivated*, which specifies whether a drug is deactivated or not. An optional attribute *drugType* defines the type of the drug. In case of a mixed insulin, the mixing ration of basal and bolus insulin is defined in the attribute *basalToBolusRatio*.

Figure 4.15 shows the second part of the entity model. After a patient was

enrolled, any number of records can be assigned to the enrolment. There are six different types of records. Each type is represented in an own entity and inherits from the parent entity *Record*.

- *EnrolmentRecord*: Is the record of the enrolment itself. The enrolment record consists of 4 attributes, which can be defined by the user during or after the enrolment process. The attribute *diabetesType* specifies, whether the patient is assigned to diabetes type 1, diabetes type 2 or to other rarer diabetes types. The attribute *preTherapy* defines what diabetes therapy the patient got before the current enrolment. The attributes *weight*, needed by the decision support, and *height* are self-explanatory.

- *StopEnrolmentRecord*: This entity documents stopping every enrolment of a patient. The enrolment of a patient can be stopped manually or automatically and efforts specifying a reason for stopping the enrolment.

- *MeasurementRecord*: This entity documents all measurements to the enrolment of a patient. Every measurement record has an entity *MeasurementType*. This entity specifies the type of the measurement. The entity *MeasurementRecord* itself specifies the value, the time, the unit and the input type of the measurement.

- *ProposedMedicationRecord*: Often, the amount of administered insulin won't match with the suggested amount, which was calculated by the decision support. The entity *ProposedMedicationRecord* documents all suggestions calculated by the decision support.

- *MedicationRecord*: The entity *MedicationRecord* documents administered drugs by specifiying the administered units, the unit type, the type of administration, the time of the day and the exact time of the medication. Furthermore a medication record consists of the information if a meal is intended, as well as the suggested amount of insulin, which is calculated by the decision support. Every medication record has an entity *therapyMedication*, which contains of detailed information to administered medication, such as the drug type or the product code.

- *TherapyAdjustmentRecord*: After the enrolment, each patient gets an individual therapy. The current therapy is described through the entity *TherapyAdjustmentRecord*. A therapy of a patient consists of the regimen type (Basal-Bolus or non-supported regimen), any number of *TherapyMedication* entities, a blood glucose target range, the current daily insulin dose, the insulin resistance of the patient and other parameters. The *boolean* attribute *dssActive* describes the status of the decision support, which is true in case the basal-bolus regimen is selected, a daily insulin dose is defined and at least one basal and one bolus insulin is assigned to the patient.

4.6.4 Development of the Frontend

The Android frontend application of the inpatient glucose management system is currently under development. However, this chapter will give insights of the methods, related to the previous implementation work of the Frontend. The frontend application represents the user interface of the inpatient glucose management system. It is responsible for data recording, data representation and data visualization. The recorded data is sent via web service to the backend, where it is processed and finally stored to the server database. The other way round the backend application sends required data via web service to the frontend application, where the data is presented in an appropriate manner to the user. Based on the findings gained during the first and the second iteration, the Samsung Galaxy Tab, shown in figure 4.16, was chosen as a preferred frontend-device. The Galaxy tab, which uses Android 2.2 (see chapter 3.4.2) as an operating system, fits in any coat pocket and is lightweight, compared to other tablets, such as the Apple iPad, which is an important indicator for its intended medical use.

For developing the frontend application each use case, which was identified during the second iteration, was analyzed and the workflow of the use case was drawn in a conceptional activity diagram. Finally each of the activity diagrams was implemented and tested after finishing the implementation.

Figure 4.16: Samsung Galaxy Tab (Niederschmid (2010))

4.6.4.1 Setting up and Building the Frontend, Using Maven for Android

The frontend application uses Apache Maven 2 (see chapter 3.4.1) as a building tool. The Maven-Android Plugin allows an Android application to be built, deployed, tested and released automatically with Apache Maven outside an integrated development environment (IDE). Also the dependency management is organized through the Maven-Android plugin. Before an Android application can be deployed on a device or an emulator, it has to be transformed to a Dalvik compatible byte code and packaged into an *apk*-file, which is managed by the Maven-Android plugin too. The easiest way to set up a Maven-Android project is to use Maven Android Archetypes, which allow creating Android applications with the desired modules and ready POM.xml files with only one command. For the frontend application of the inpatient glucose management system *android-with-test-archetype* was used, which creates a multi-module project. The precise command used to set up the frontend application looks like the following:

```
mvn archetype:generate \
  -DarchetypeArtifactId=android-with-test \
  -DarchetypeGroupId=de.akquinet.android.archetypes \
  -DarchetypeVersion=1.0.5 \
  -DgroupId=eu.reaction.android \
  -DartifactId=GluCoManSysFrontend \
```

The command creates a Maven project, including the following components:

- A parent project *GluCoManSysFrontend* with the package type *pom*.
- A module containing a dummy application to be customized, named *application*, with the package type *apk*.
- A module containing a dummy test application to be customized, named *application-it* with the package type *apk*.

The modules *application* and *application-it* are separate Android applications, and thus can be built and deployed separately. The maven command *mvn install* at the parent project directory will built and deploy both applications on a running emulator or connected device and will automatically execute the instrumentation tests, which are placed in the *application-it* project (GitHub (2011)). The packaging type *apk* in an application's POM.xml file leads to a customized build lifecycle of Maven with the following additional phases in the default lifecycle (see chapter 3.4.1):

- **generate-sources**: Packages the Android specific resources, such as the AndroidManifest.xml, using the Android Asset Packaging Tool (aapt).
- **process-classes**: converts all classes into Davlik compatible byte code, using the dx-tool.
- **package**: packages the Davlik compatible byte code into an apk-file.
- **pre-integration-test**: Deploys all related apk-files to the device or emulator.
- **integration-test**: Executes available instrumentation tests against the deployed application.

The Maven Android Plugin provides a set of additional goals, which execute specific lifecycle phases. The most important ones are:

- **android:apk**: Builds the apk-file of the application.

- **android:deploy**: Builds the apk-file of the application and deploys it on the emulator or device.
- **android:instrument**: runs the instrumentation tests on the device or emulator.
- **android:undeploy**: removes the apk-file of the current application from the device or emulator.

In order to build the Android application the maven command *mvn install android:deploy* in the parent directory *GluCoManSysFrontend/application* is executed.
In order to run the test cases the maven command *mvn install android:deploy* in the parent directory *GluCoManSysFrontend/application-it* is executed.

4.6.5 Design of the Android User Interface

As presented in chapter 3.4.2.3, Android activities are responsible for interacting with the user. Activities are the most used components of the frontend application. The surface of each Android *Activity* is defined by an underlaying layout.xml file. The usage of the frontend application has to fit with the identified clinical workflow. To provide a user-friendly interface, the layouts were mostly reconstructed from the already validated mock up storyboards (see chapter 4.4.2). In addition, the general usability aspects, according to mobile touch screen devices (see chapter 3.3.3), were observed.
Because Android runs on a variety of devices that offer different screen sizes and densities, great attention was paid to the scaling of view elements. Consequently the heights and widths of views and layouts were specified using *fill_parent*, *wrap_content* or, if necessary, numeric values in density independent pixel in order to guarantee that the view or the layout is given an appropriate size on the current screen. Also text sizes were specified using scale-independent pixels (sp) to scale the size of TextViews related to the device‘s density.
The use of long-click events (a view has been clicked and held), has been omitted beside a few exceptions. The reason for this is that long-click events are not

intuitive and not trained users could have problems with finding functionalities, which are hidden behind such long-click events.

4.6.6 Visualizing Therapy Values Using aiCharts

One of the main requirements, identified during previous development steps is to visualize the course of therapy in charts. Therefore, the frontend application of the inpatient glucose management system uses aiCharts, which is a chart library designed and optimized for Android. aiCharts is powered by ArtfulBits[3] and supports a wide range of different chart types, which can be combined in one chart area. To integrate aiCharts in an *Activity*'s layout the view element *ChartView* must be used. Each ChartView element can have several numbers of *ChartArea* elements with a several number of chart series, where each series must belong to a specific chart type. According to the frontend application the chart types *Line* for displaying the course of measured blood glucose values and the chart type *Point*, for displaying medications, nutrition, comments and day borders were used. aiCharts provides good possibilities to customize the appearance of chart series, such as drawables for markers, customized labels, etc. One of the greatest advantages of aiCharts is that charts can be made user interactive, so for example the touch of a *ChartPoint* can be captured. According to the frontend application the touch of a *ChartPoint*, which is represented by a comment, forces a dialog to open, which presents the comment to the user. This can be done by using an *OnTouchListener* of the view element *ChartView*. The following code snippet illustrates an *OnTouchListener* of a chart, named *chartViewBG*:

```
final ChartView chartViewBG = (ChartView)findViewById(R.id.fullScreenChart);
chartViewBG.setHitTestEnabled( true );
chartViewBG.setOnTouchListener( new OnTouchListener()  {
  public boolean onTouch( View v, MotionEvent event ) {
    switch ( event.getAction() ) {
      case MotionEvent.ACTION_DOWN: {
        int x = (int) event.getX();
        int y = (int) event.getY();
```

[3]aiCharts - http://www.artfulbits.com/products/android/aicharts.aspx

```
        List<Object> hitObjects = chartViewBG.hitTest( x, y );
        for ( Object object : hitObjects ) {
          if ( object instanceof ChartPoint ) {
            ChartPoint myPoint = (ChartPoint) object;
            Toast.makeText(ShowChart.this, myPoint.toString(),
                Toast.LENGTH_SHORT).show();
          }
        }
      break;
      }
    }
    return true;
  }
});
```

If a touch event appears, the exact x and y coordinates are fetched. Afterwards, the code tries whether these coordinates match with a *ChartPoint*. At the end a Toast-Message is presented to the user, showing a string representation of the touched chart point.

4.6.7 Data Recording via Android Dialogs

The frontend application of the inpatient glucose management system should avoid as much manual user input as possible. Due to this requirement the backend should later provide interfaces to the hospital information system to receive available patient data and measured blood glucose values. However, there is some information, which could not be achieved automatically. For these manual inputs the frontend application uses Android Dialogs (see chapter 3.4.2.7). A dialog is a window that appears, while the underlying *Activity* loses the focus. Each dialog can get its own layout. Due to terms of maintainability, cutomized dialogs, which extend the *AlertDialog* class, provided by the Android SDK, were implemented. Each customized dialog is responsible for recording a required data type. So, for example the customized dialog WheelPickerDialog is loaded if a numeric value is required. The user can set the value by sliding over the wheel, shown in figure 4.17: Figure 4.18 illustrates another customized dialog example. ListSelectorDialog is

Figure 4.17: Customized AlertDialog WheelPickerDialog

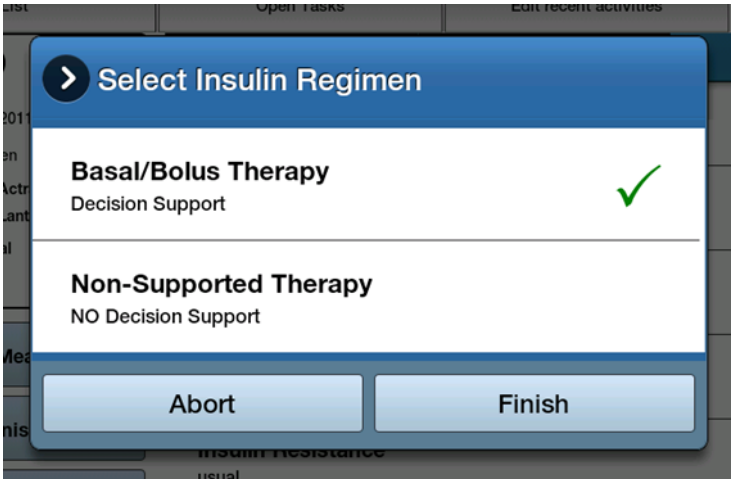

Figure 4.18: Customized AlertDialog ListSelectorDialog

used if the user should select between predefined values, provided in a ListView.

4.6.8 Accessing Backend Web Services

The exchange of data between the backend and frontend components is completely done via web services, using SOAP, in order to provide a platform independent server logic and maintainability, according to the backend application. SOAP (Simple Object Access Protocol) is a standarized XML based protocol for exchanging data (Cerami (2002)). Let's take a look how web service can be accessed. The following SOAP request relates to the example, shown in figure 4.13. A sample SOAP message to load a drug with the product name 'Actrapid' looks as follows:

```
1. <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:med="http://medication.service.glucosemanagement.prototype.reaction.eu/"
        xmlns:med1="http://medication.vo.glucosemanagement.prototype.reaction.eu/">
2.  <soapenv:Header/>
3.  <soapenv:Body>
4.   <med:loadDrugs>
5.    <med:loadDrugsInVO>
6.     <med1:productName>Actrapid</med1:productName>
7.    </med:loadDrugsInVO>
8.   </med:loadDrugs>
9.  </soapenv:Body>
10. </soapenv:Envelope>
```

Line 4 illustrates the web service method, which was defined in the *DrugService* component in figure 4.13. The method *loadDrugs* must get the parameter *loadDrugsInVO* from the type *LoadDrugsInVO*, which corresponds to the modeled value object. The value object *LoadDrugsInVO* has 5 optional attributes, including *productName*, which can be optionally added as parameters inside the *med:loadDrugsInVO* tag. If no attribute would be specified, all available drugs would be returned by the web service.
If the server database contains of a drug with the product name 'Actrapid', the web service will return the following SOAP response:

```
1. <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2.  <soap:Body>
3.   <ns2:loadDrugsResponse
          xmlns="http://medication.vo.glucosemanagement.prototype.reaction.eu/"
          xmlns:ns2="http://medication.service.glucosemanagement.prototype.reaction.eu/"
          xmlns:ns3="http://exception.glucosemanagement.prototype.reaction.eu/"
          xmlns:ns4="http://therapy.vo.glucosemanagement.prototype.reaction.eu/">
4.    <ns2:LoadDrugsOutVO>
5.     <drugVOs>
6.      <atcCode>A10AB01</atcCode>
7.      <drugType>insulin</drugType>
8.      <productName>Actrapid</productName>
9.      <description>Actrapid insulin</description>
10.      <productCode>0000000001</productCode>
11.      <insulinType>BOLUS</insulinType>
```

```
12.       <activateDateTime>2011-02-17T00:00:00+01:00</activateDateTime>
13.       <deactivated>false</deactivated>
14.      </drugVOs>
15.     </ns2:LoadDrugsOutVO>
16.    </ns2:loadDrugsResponse>
17.   </soap:Body>
18. </soap:Envelope>
```

Line 4 to 15 contain the value object *LoadDrugsOutVO* with the defined attributes related to figure 4.13. If one more drug with the product name 'Actrapid' would exist, another *LoadDrugsOutVO* value object would be returned by the web service.

Because patient data must be handled in strict confidence, Fraunhofer Institute, as a project partner, implements a secure Android SOAP client application, which supports encrypted communication and mutual authentication of both communication endpoints. The Android Frontend application uses this secure SOAP client for sending every web service request and receiving every web service response. Therefore a web service handler was implemented, for accessing the secure SOAP client application, displaying a ProgressDialog, and starting a thread, which calls the proper web service method in the background and reports after finishing. The following code snippet should illustrate the functionality of the web service handler:

```
private static SOAPServiceConnection mConn;
public static ProgressDialog progressDialog;
public static final int LOAD_DRUGS = 0;
public static final int ERROR = 99;
...
final Handler handler = new Handler()    {
  public void handleMessage( Message msg )      {
    progressDialog.dismiss();
    if (msg.what == WebServiceHandler.LOAD_DRUGS) {
      ... //Web service was successfully executed
    } else if (msg.what == WebServiceHandler.ERROR) {
      ... //Error appeared during the request
    }
  }
```

```
};
public void initializeWebService( Activity context, Integer method )    {
  currentActivity = context;
  progressDialog = new ProgressDialog( currentActivity );
  progressDialog.setMessage( currentActivity.getString( R.string.ts_loading ) );
  progressDialog.show();
  myMethod = method;
  mConn = new SOAPServiceConnection( currentActivity, this );
  try   {
    mConn.connect();
  }
    catch ( SecurityException e )   {
    ... //Error while connectiong to SOAP client application
  }
}
public void onServiceUp()    {
  Thread checkUpdate = new Thread()    {
    public void run()    {
      Looper.prepare();
      if ( myMethod == WebServiceHandler.LOAD_DRUGS )    {
        WebServicePatientManagement ws = new WebServicePatientManagement();
          if ( ws.loadDrugs( mConn ) == true )
            handler.sendEmptyMessage( WebServiceHandler.LOAD_DRUGS );
          else
            handler.sendEmptyMessage( WebServiceHandler.ERROR );
      } else if {
        ...  //Other web service methods can be placed in here
      }
      mConn.disconnect();
    }
  };
  checkUpdate.start();
}
```

Connecting to the SOAP client is done within the method *initializeWebService(Activity context, Integer method)*, which gets the current Android *Activity* and the web service method to call as parameters. The method starts a progress dialog to let the user know, that the application is busy. The method *onServiceUp()* is called after the connection to the SOAP client was

established successfully. It runs a thread, which is calling the proper web service method and is reporting about the web service result. Afterwards the *ProgressDialog* is closed and depending on which web service method was required and the result of the web service, further steps can be executed.

The web service call itself can be done, using the code snippet below:

```
public boolean loadDrugs(final SOAPServiceConnection mConn) {
  ISOAPService srvc = mConn.getService();
  ISOAPMessage soapMsg = new ISOAPMessage( SOAPConstants.SOAPVERSION.SOAP1_1 );
  soapMsg.getDocumentElement().setPrefix( "soapenv" );
  soapMsg.getBody().setPrefix( "soapenv" );
  Element body = soapMsg.getBody();
  ... //Build soap message for loading drug
  ISOAPMessage resp = null;
  try {
    resp = srvc.send( <service endpoint>, null, soapMsg );
  } catch ( RemoteException e ) {
    return false;
  }
  ... //Parse web service response
  return true;
}
```

The SOAP message is built up using the ISOAPMessage class, provided by the secure SOAP client and is finally sent asynchronously to the proper web service endpoint, which waits for a web service response, using an ISOAPService object.

4.6.9 Testing the Frontend

IEC 62304 requires testing all critical components of medical device software. Because the frontend application of the inpatient glucose management system is only responsible for user interaction and data visualization and doesn't provide any critical business logic, as for example functionality, related to the decision support for insulin dosing, it is sufficient to test the correct behavior of the application, according to the user interface. Consequently GUI tests (see chapter

3.4.2.9) are performed for each implemented use case. Therefore a separate Android test project with the following *AndroidManifest* was implemented:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="eu.reaction.android.test"
  android:versionCode="1"
  android:versionName="1.0-SNAPSHOT">
  <application
  android:icon="@drawable/icon"
  android:label="@string/app_name">
    <uses-library android:name="android.test.runner" />
  </application>
  <uses-permission android:name="idservice.permission.ACCESS"/>
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.RUN_INSTRUMENTATION" />
  <instrumentation android:targetPackage="eu.reaction.android"
   android:name="eu.reaction.android.test.GluCoManSysTestSuite" />
</manifest>
```

The *AndroidManifest* of the test application includes the permission *android.permission.RUN_INSTRUMENTATION*, as well as the element *instrumentation*, which specifies the target application package to test (*eu.reaction.android*) and the instrumentation test runner *eu.reaction.android.test.GluCoManSysTestSuite*, which extends the Android *InstrumentationTestRunner* class and manages the test suite. The test application contains of test classes, where each test class represents a group of test. Each test class can be activated or deactivated in a *TestConfig.properties* file in the application's *asset* directory. The *GluCoManSysTestSuite* loads the *TestConfig.properties*, parses its content and fill the test suite with the test classes that should be executed by the *InstrumentationTestRunner*, if the proper test class is set to true in the properties file. This is done in the overridden method *getAllTests()*. Each test class extends the *ActivityInstrumentationTestCase2* class, which provides testing of Android Activities and offers a method *setUp()*, which initialize the environment before each test runs as well as a method *tearDown()*, which is called after each test has finished and makes sure that the environment is cleaned up before moving to the next test. The simulation of an application's

surface is done by an external library, called *robotium*. *Robotium* is a free testing tool, which simulates touching, scrolling, clicking and other actions, belonging to view elements of Android Activities. In order to check the correct data representation of the frontend application a separate web service was implemented to fill the server database with test data. This ensures that each test class gets a known database initialization and can check if the expected data is presented correctly on the application's surface. The following code snippet should show the base structure of a GUI test case class. The example shows the simplified test class *DisplayCurrentlyEnrolledPatients*, with only one test method *testAmountOfDisplayedPatients()*. The test case should check the correct amount of presented patients in the list.

```
public class DisplayCurrentlyEnroledPatients extends
    ActivityInstrumentationTestCase2<StartScreen> {
  private Solo mySolo;
  public DisplayCurrentlyEnroledPatients() {
    super(StartScreen.class);
  }
  @Override
  protected void setUp() throws Exception {
    super.setUp();
    WebServiceTestHandler ws = new WebServiceTestHandler();
    ws.initializeTestSetup(getActivity(),
    WebServiceTestHandler.DISPLAY_PATIENTS_GMS_TESTCASE); mySolo = new
      Solo(getInstrumentation(), getActivity());
    mySolo.clickOnButton(getActivity().getString(R.string.bt_login));
  }
  public void testAmountOfDisplayedPatients() {
    assertEquals(4, mySolo.getCurrentListViews().get(0).getCount());
  }
  @Override
  protected void tearDown() {
    super.tearDown();
  }
}
```

Because the test class should test an Android *Activity*, *DisplayCurrentlyEnroledPatients* extends the *ActivityInstrumentationTestCase2* class. The parameter *StartScreen* represents the name of the *Activity* under test. In the constructor the super constructor is called to tell the environment what Android *Activity* should be tested. In this case it is the *StartScreen Activity*, because the test should start there. The *setUp()* method initializes the environment before each test. It calls the web service to initialize the server database. At the backend an implemented test case handler, which performs the initialization, is executed, depending on the parameter, sent by the web service request. Each test case class has its own test case handler in the backend to ensure that the frontend test application knows what data is expected. Finally robotium is used to click on the login button of the *StartScreen Activity* to start the Patient Management of the frontend application. After the *setUp()* method the test method *testAmountOfDisplayedPatients()* is executed. The test case checks if the expected amount fits with the amount of currently displayed list items. After the test finishes or an *assert*-failure occurs the *tearDown()* method is called to clean up the environment.

5. Results

Although the frontend application of the inpatient glucose management system is currently at the development stage, most of the identified use cases are already implemented and positive feedback from clinical end users, who have already seen current results, approve that we are on the right way.

Figure 5.1 presents a screenshot of the already implemented main screen with the visualization of the most important measurement and insulin administration parameters of the mobile inpatient glucose management application. In addition, the figure shows the main functionalities of the application:

- **Blood Glucose Measurement**: Enables users to add blood glucose measurements to the system
- **Insulin Administration**: suggests an insulin dose, calculated using the identified protocol (see chapter 4.2) and finally enables users to add an insulin administration to the system.
- **Therapy adjustment**: enables users to adjust the daily insulin dose.
- **Patient List**: Presents all currently enrolled patients at the user's ward.
- **Open Tasks**: reminds users of the system to perform recommended tasks.
- **Logout**: Logs the user out of the glucose management system.
- **Glucose Table**: presents the therapy values of the patient in tabular form.
- **Therapy**: Enables users to adjust therapy parameters

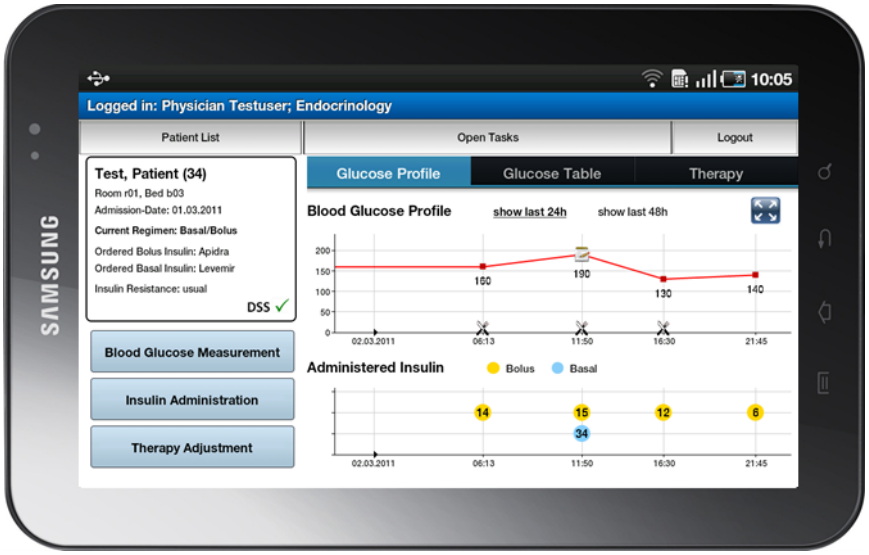

Figure 5.1: Main screen of the glucose management of virtual patient 'Bernhard Höll'

- **Full Screen** (Image Button): presents the visualization of the therapy values in scrollable and zoomable full screen mode.

At the moment the Android application has the following, entirely or almost completed features:

- Displaying (enrolled) patients at ward, including filter/sorting functionality.
- Enrolment of patient for Glucose Management System.
- Stop/Update Enrloment of enrolled patient.
- Initialization of Basal/Bolus therapy.
- Manual adjustment of Basal/Bolus therapy.
- Manual adjustment of non-supported therapy.
- Visualization of therapy values of patient in charts and in tables.
- Daily therapy adjustment (daily dose adjustment).
- (Manually) Adding a BG measurement to a patient.
- Adding an insulin administration to a patient, who is assigned to the basal-bolus regimen.

- Providing full screen mode to view therapy values, including scrolling and zooming functionality.

The following chapters should exemplarily present the current results of the frontend application. Selected use cases and their implementation, are illustrated in activity diagrams and screenshots finally demonstrate the actual results of the implemented user interface. It should be noted that all patients, shown by the screenshots, are just virtual patients and do not have any known diabetes history.

5.1 Displaying (enrolled) Patients at Ward, Including Filter/Sorting Functionality

User Case

Every user is assigned to one ward. After the login the user can see all patients of the suitable ward, who are already enrolled. Optionally the user can also view a list of all patients of the ward. Already enrolled patients are highlighted green in this list. In both lists, the patients are ordered by their room and bed number for default. Optionally the user can sort the patient lists by their names. Furthermore, it is possible to filter the patient lists by room number. Therefore the user has to select rooms, available at the ward to be filtered.
Results

According to figure 5.2, the application is started with the *StartScreen Activity*, which contains a login button. At the moment there is no User Management implemented, so this is just a dummy login where a virtual physician at the ward 'Endocrinology' is automatically logged in. To display the patients at ward the web service *PatientsService* using the method *findPatients* with the unique identifier of the ward (*wardID*), the user is assigned to, is called. The web service method returns a list of the current patients to the proper ward. The *WebServiceHandler* starts the *TabActivity PatientManagement*, which manages two child tabs, *PatientsWithGM* and *PatientsAtWard*. After the login, the *ListActivity PatientsWithGM* is presented to the user, which contains a list where

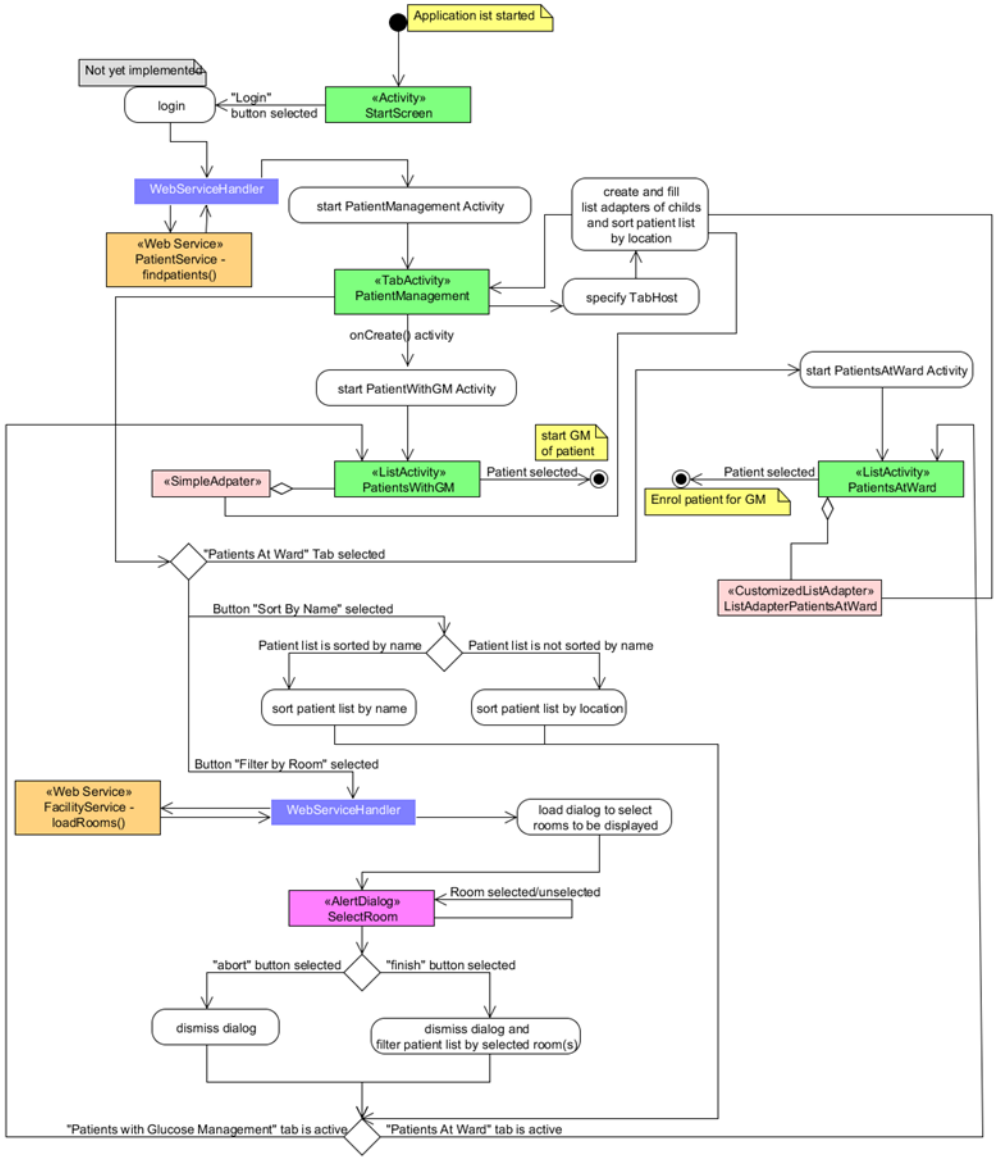

Figure 5.2: Android implementation - Displaying (enrolled) patients at ward, including filter/sorting functionality

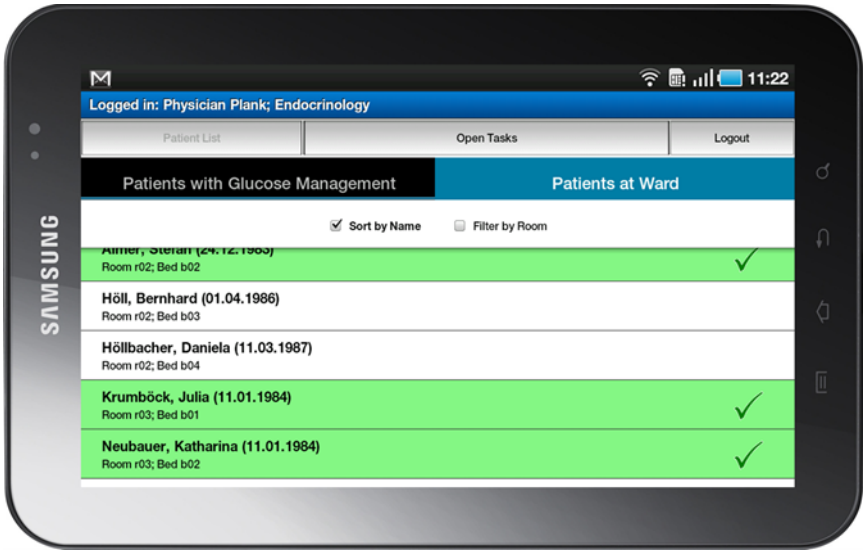

Figure 5.3: Screenshot - Patients at ward, sorted by name

the patients at ward, which are currently enrolled are presented. The other tab includes the *ListActivity PatientsAtWard*, where all patients at ward are displayed. To mark currently enrolled patients a *CustomizedListAdapter ListAdapterPatientsAtWard* was implemented, which checks for each list item if the associated patient is enrolled. Beside the tab navigation, *PatientManagement* owns a 'Sort By Name' and a 'Filter by Room' button. The button 'Sort By Name' changes the sorting of the items of the proper list adapter, firstly by their second name and secondly by their first name. The button 'Filter By Room' calls the method *loadRooms* of the web service *FacilityService* to get available rooms at ward. These rooms are presented to the user, using a standard Android *AlertDialog*. Finally the list items of both list adapters are filtered by the selected rooms. Figure 5.3 presents a screenshot of the frontend application, presenting all patients at the ward 'Endocrinology' in a scrollable list, sorted by their name.

5.2 Enrolment of Patient for Glucose Management System

User Case

To perform an enrolment, a currently not enrolled patient has to be selected in the

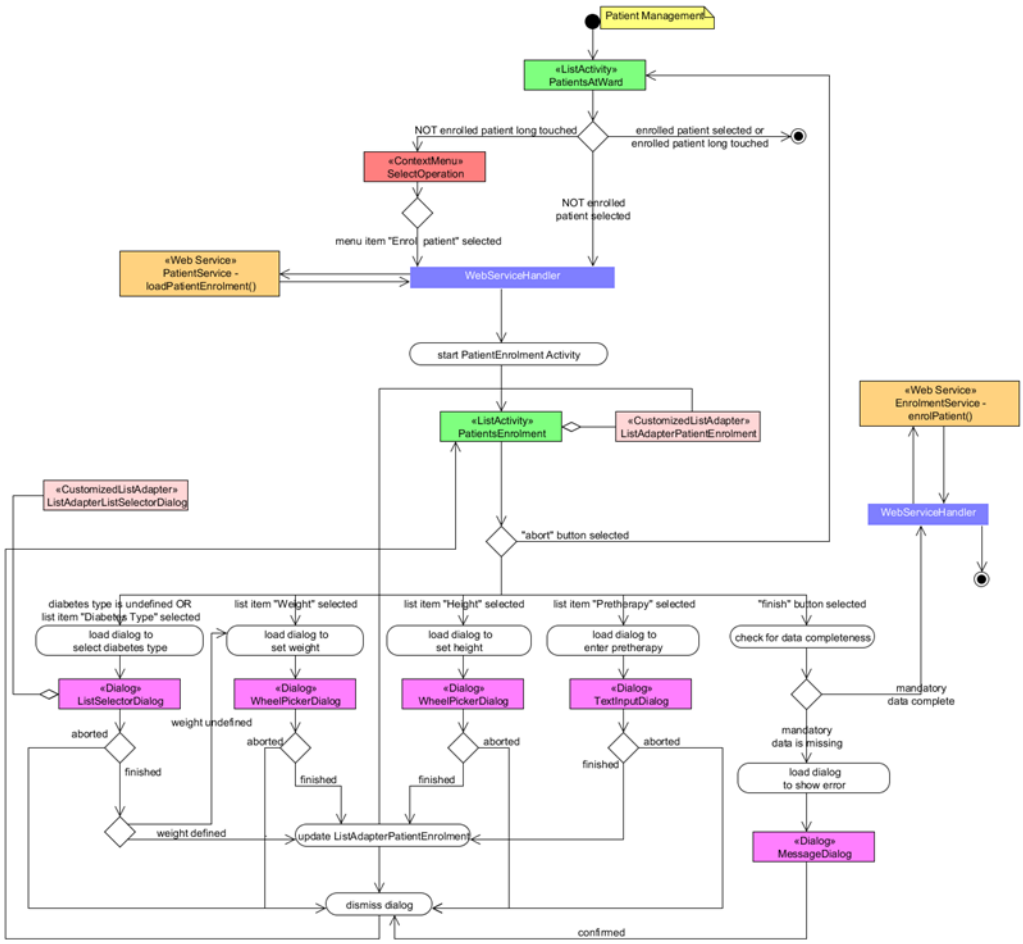

Figure 5.4: Android implementation - Enrolment of patient for Glucose Management System

list, which displays all patients at ward. Afterwards the user has to define the diabetes type and the weight as mandatory enrolment parameters. Optionally the user should be able to set the patient's height and to specify the pretherapy of the patient. Undefined mandatory parameters are marked with a red exclamation point; optional parameters are marked with an orange cross. Already defined parameters should be marked with a green tick. Without defining the mandatory parameters a patient cannot be enrolled.

Results

According to figure 5.4, there are two options to enrol a patient for glucose management. Therefore a patient can be enrolled by touching or long-touching on a list item which contains of a currently not enrolled patient. For the long-touching option, the context menu *SelectOperation* is used. A

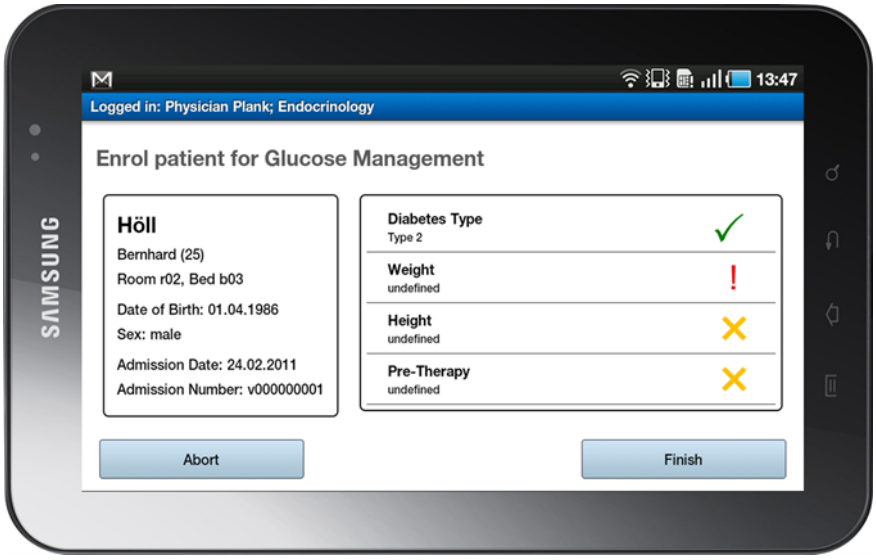

Figure 5.5: Screenshot - Activity to perform enrolment

ContextMenuListener of the *ListView* waits for a long-touch event to build up the context menu. The context menu is defined dynamically in the Java code, because the menu items differ depending on the enrolment status of the long-touched patient. If the selected patient is currently not enrolled the context menu only contains of 1 list item to enrol the patient for Glucose Management. If the selected patient is currently enrolled the user can select between starting the Glucose Management, stopping the enrolment or updating the enrolment in the context menu. According to the aim to avoid long-touch (or long-click) events (see chapter 4.6.5), in this particular use case it was desired by the end users to use a long-touch event. According to the enrolment of a patient most of the work is done by the *ListActivity PatientEnrolment*, which presents some details of the patient, such as the name, age, admission date, etc. which were received through the method *loadPatientEnrolment* of the web service *PatientService*. The received details of the patient were represented by Android *TextViews*. Additionally *PatientEnrolment* includes a *ListView*, to handle enrolment parameters. The customized list adapter *ListAdapterPatientEnrolment* manages the list items and their formatting. *PatientEnrolment* automatically requires the user to define the mandatory parameters by showing the proper customized dialogs. If all mandatory data is defined, the method *enrolPatient* of the web service *EnrolmentService* is requested to finish the enrolment. Figure 5.5 shows the surface of the implemented *Activity PatientEnrolment*.

5.3 Initialization of Basal-Bolus Therapy

User Case

After finishing the enrolment the user is asked to select between the basal-bolus regimen and the non-supported therapy (the patient is assigned to the non-supported therapy by default). After selecting the basal-bolus regimen the user is required to order exactly one basal and one bolus insulin for the proper patient. Afterwards the user is required to order an initial daily insulin dose. Only after these two mandatory parameters have been defined, the user is assigned to the basal-bolus regimen. The initialization of the basal-bolus regimen must not be performed after the enrolment. It can also be performed at any time thereafter. A symbol in the patient details indicates that the decision support service is enabled.
Results

After the enrolment the *GMMainscreen Activity* is started which shows the *ListActivity NonSupportedRegimenTherapySettings* in the background and the dialog to select the therapy regimen in the foreground, using a *ListSelectorDialog*, where the non-supported regimen is marked by default. If the regimen is changed to the basal-bolus therapy the content of the current tab is changed to the *ListActivity BasalBolusRegimenTherapySettings*, which is presented in the background. In the foreground the dialog to order basal and bolus insulin is shown (*SelectBasalBolusInsulinDialog*), which only accepts the ordering of exactly one bolus and one basal insulin. Afterwards the dialog is dismissed and the dialog to order the initial daily insulin dose is displayed (*DailyInsulinDoseDialog*), where the user can manually set the initial daily dose or optionally use the decision support, which calculates the initial daily dose based on the identified protocol (see chapter 4.2), using the *CalcDailyInsulinDoseDialog*. After finishing the therapy initialization the list adapter *ListAdapterTherapySetting* and the patient details are updated and the web service *TherapyService* stores the changes to the backend. Before the therapy initialization is started, the current therapy settings are backed up, in order to restore them if the initialization is aborted or errors during the initialization occur.

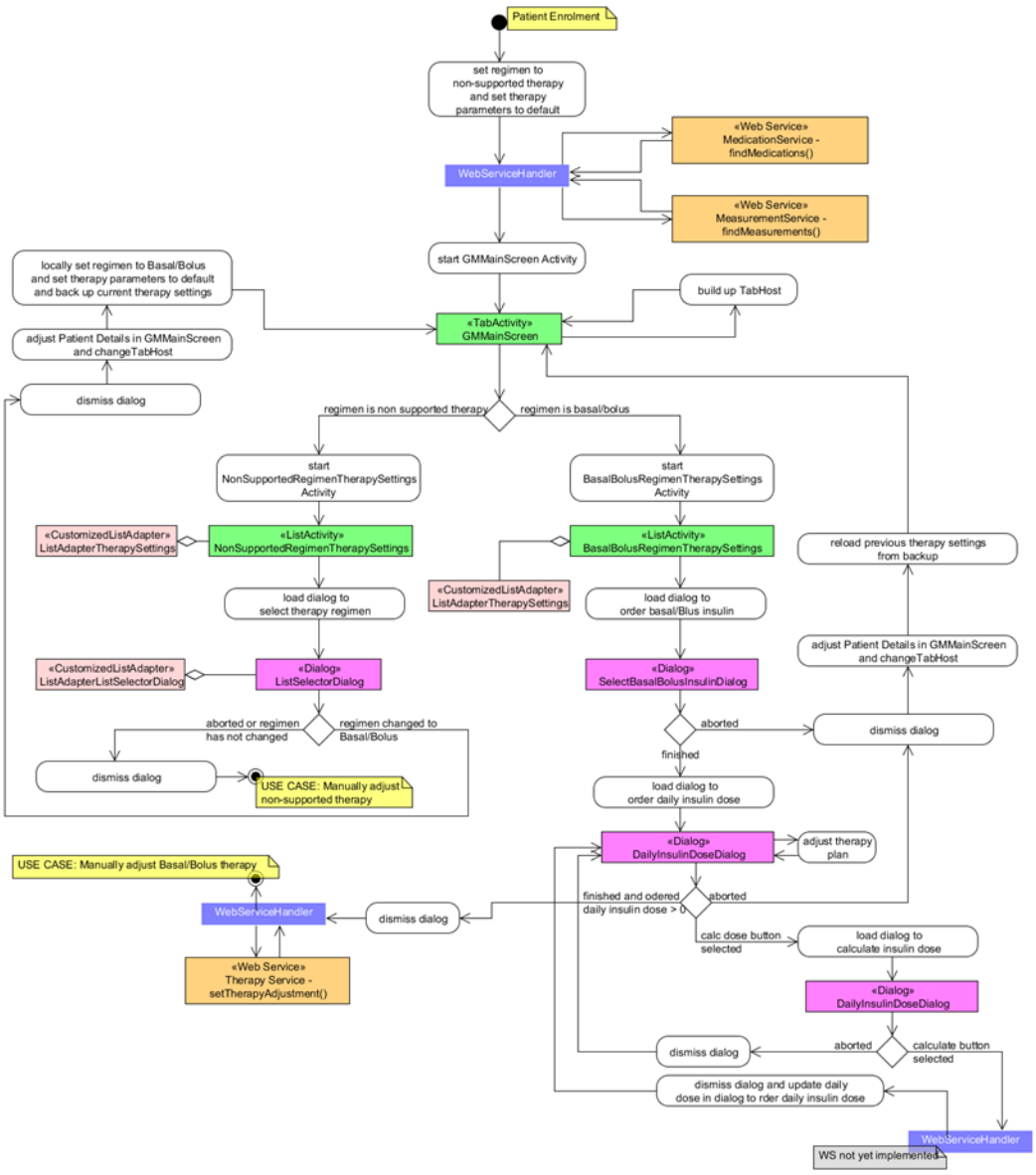

Figure 5.6: Android implementation - Initialization of Basal-Bolus therapy

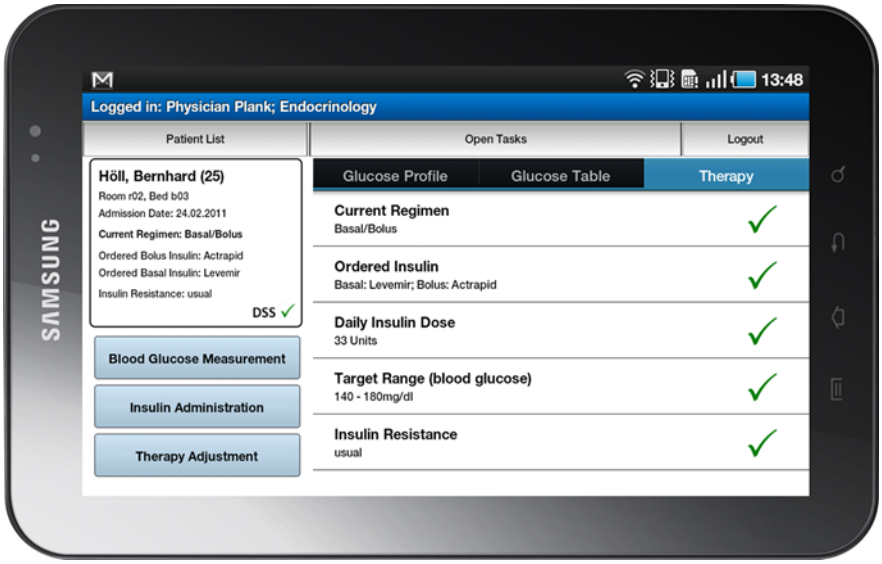

Figure 5.7: Screenshot - Finished basal-bolus therapy initialization

Figure 5.7 illustrates the finished basal-bolus therapy initialization after the enrolment of the virtual patient 'Höll'.

5.4 Adding an Insulin Administration to a Patient, who is Assigned to the Basal-Bolus Regimen

User Case

An insulin administartion of a patient, who is assigned to the basal-bolus regimen requires that a blood glucose measurement has been performed within the last 30 minutes. Otherwise the user should be asked to perform a blood glucose measurement first. In case an actual blood glucose value is available the user should get a suggestion, according to the basal and bolus insulin dosage. The user should be able to adjust both the bolus dose and the basal dose. Finally the user has to approve the specified insulin administration and a message is presented which prompts the user to perform the approved insulin administration.

Results

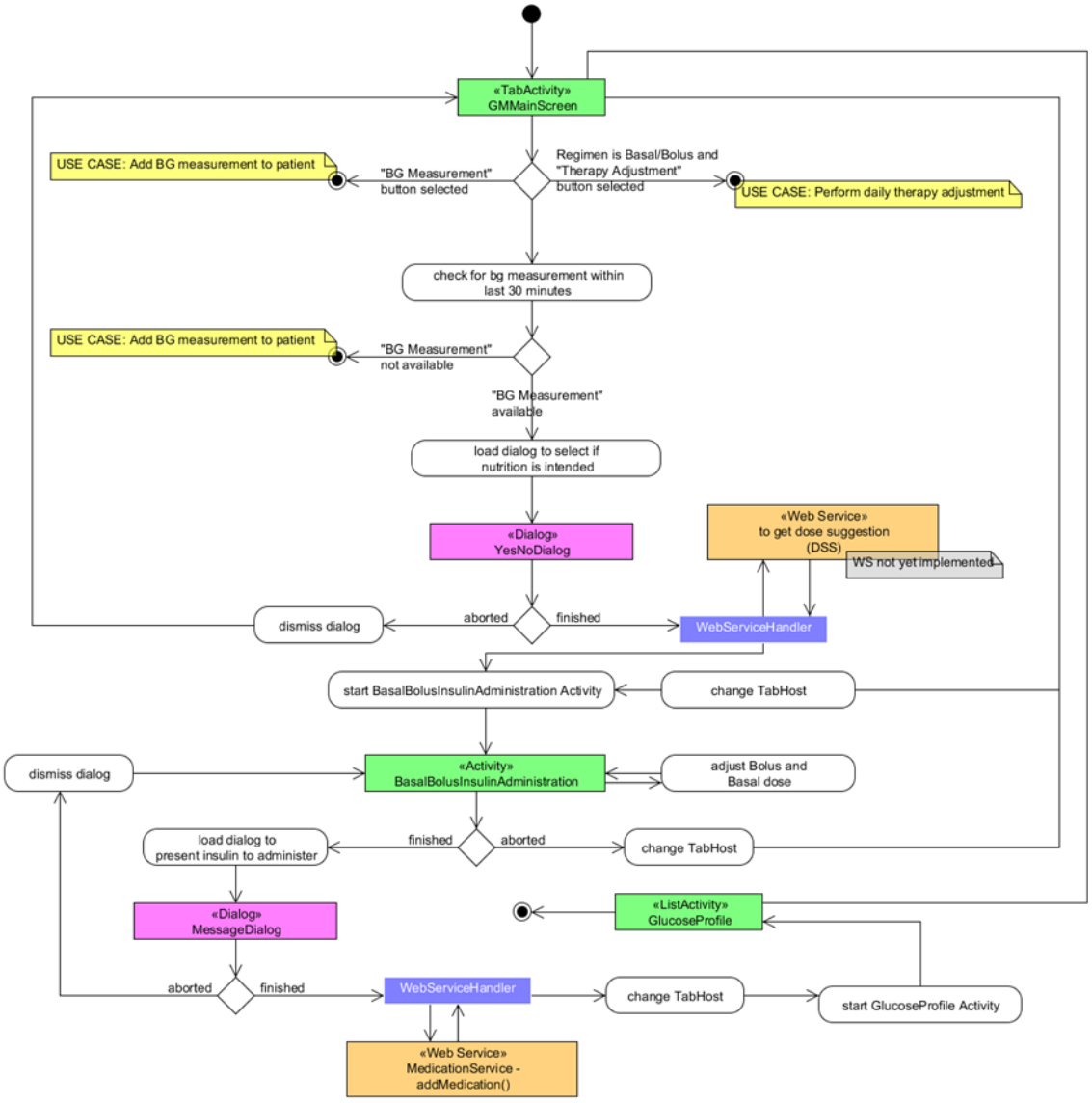

Figure 5.8: Android Implementation - Adding an insulin administration to a patient, who is assigned to the basal-bolus regimen

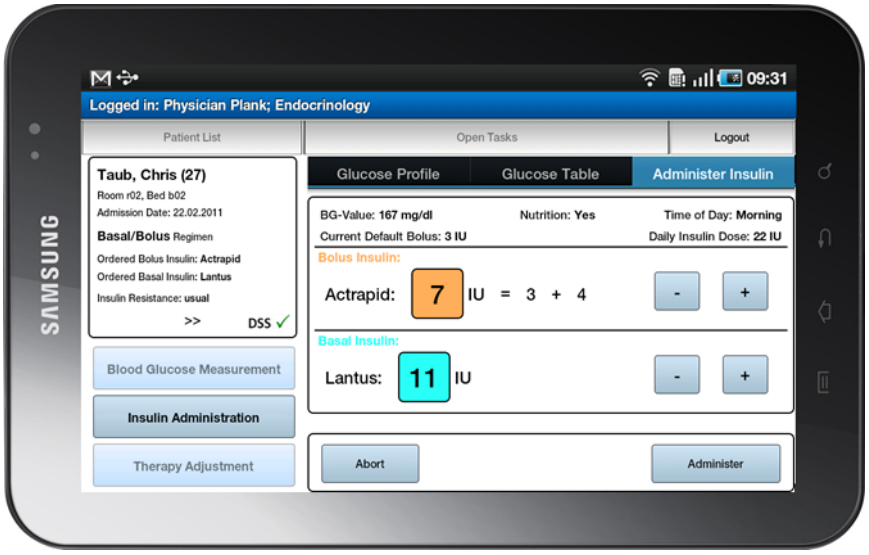

Figure 5.9: Screenshot - Insulin Administration with insulin suggestion

According to figure 5.8 the insulin administration is started by touching the button 'Insulin Administration'. Afterwards the application checks if a blood glucose value is available within the last 30 minutes. If no actual blood glucose value is available the user is prompted to perform a blood glucose measurement. Otherwise a dialog is presented to the user, to specify if a meal is intended using a *YesNoDialog*. Based on the actual blood glucose value and the information about the meal, a webservice request is executed to get a suggestion about the bolus and basal insulin dosage. At the moment this web service does not exist, so the suggestion is simulated by the *WebServiceHandler*, which afterwards changes the content of the 'Therapy' tab to the *BasalBolusInsulinAdministration Activity*. *BasalBolusInsulinAdministartion* contains of some information about the associated blood glucose measurement, as well as the current daily insulin dose and presents the calculated suggestion about the basal and bolus insulin dosages to the user. The user can adjust and has to approve the dosages. During performing an insulin administration adjustment, the user can switch between the tab navigation to view the patient's course of therapy. Finally a *MessageDialog* displays a summary of the defined insulin administration to the user.

Figure 5.9 illustrates the *BasalBolusInsulinAdministartion Activity* with the calculated suggestions about the basal and bolus insulin dosage.

6. Summary and Lessons Learned

This master thesis illustrated the design and development process of a safety-critical mobile Android application in order to support the current paper based glucose management of patients with diabetes at the University Hospital in Graz. Because an integrated decision support service for insulin dosing can influence physicians and nurses in their medical decisions, the mobile glucose management system falls within the scope of the medical device directive and therefore a set of additional requirements have to be considered during the development process.

One important requirement in the context of medical device software is the usability of the end product in order to avoid critical situations that can harm patients or user. Therefore we followed a user-centered design process, in which end-users have been involved in every step of the design phase. Our experiences through the first and the second iteration, presented in this master thesis, show that engineers and clinicians have very different points of view concerning software. While engineers often focus on gathering as much and as complex functionality as possible, clinicians mostly prefer software which offers only the required base functionality. For clinicians a well sophisticated user interface, which is tailored to current workflow patterns, is most desired. A problem, which we encountered during collecting first base requirements, was that even end-users do not exactly know what functions should be provided by the targeted software. Consequently we used prototypes as triggers in each iteration step to give clinicians a preliminary idea of how an inpatient glucose management system could look like. After presenting the prototypes in an evaluation phase, end-users were able to give clear ideas of their requirements for a mobile glucose management system.

The whole development process was accompanied by a risk management. Especially in the risk identification and risk evaluation it is important to have an interdisciplinary team in order to gather all possible risks, including medical risks, software risks, hardware risks, misuse risks, etc.
Based on experiences and requirements, which were collected during the requirement analysis, we decided to use an Android application as a user interface for the mobile glucose management system. As a frontend device the Samsung Galaxy Tab was chosen. Android was primarily designed for private use, which has continuously provided us with new challenges. The Android framework provides a wide range of available libraries but for example we missed sufficient security components that allow secure transfer of patient data via web services. Consequently Fraunhofer Institute, as a project partner is currently implementing a secure Android SOAP client which supports encrypted communication and mutual authentication.
Another risk criterion is that the device on which the software runs should only be utilized for the intended use. Otherwise, for example changed device settings, can lead to undesirable effects. At the Android Market, applications are available, which can lock the access to installed applications on the device, so that targeted applications can only be accessed by using a pin code.
However, while overcoming the various limitations on available libraries, Android offers powerful testing tools, which allow producing high level validation and verification tests for applications, which is one of the most important conditions for medical device software.

7. Future Work

At the moment the protocol which is responsible for the application's decision support for insulin dosing is verified in a clinical trial at the department of Endocrinology and Cardiology at the Medical University Hospital in Graz. Therefore the protocols were submitted to the Ethics Commission, where they were approved. In summer 2012 the inpatient glucose management system should be verified in clinical trials too. Beside the implementation of the outstanding use cases, the completed software must meet all requirements of the medical device directives, in order to pass the request to the Ethics Commission. Therefore a quality management is currently built up which should prescribe the software development process more accurately. Furthermore the quality management should specify how to deal with change requests, if one or more requirements have changed.

In November 2011 further usability tests with at least 10 participants, including physicians and nurses, are planned.

According to traceability, another crucial requirement is to implement a secure User Management. Therefore each user has to login with a username and a password. It is planned that each user must have an own user certificate, stored on the device in order to login.

The user interface was designed to support the current clinical workflow at the analyzed ward. However, each ward in each hospital usually follows its own workflow. Therefore, great attention will be placed in the maintainability of the user interface. For example, it is planned to implement a configurable workflow engine component for handling application sequences based on workflow patterns.

Testing all of the application‘s functionality is one of the base requirement,

according to the medical device directives. This includes a detailed documentation of the test cases or rather of the test results. At the moment the documentation is done manually, in future versions it is planned to automatically produce a reporting document, by running the instrumentation tests.

It's still a long way, that the mobile inpatient glucose management system becomes a medical device which is steadily used in the inpatient environment. However, positive feedback of nurses and physicians make us confident.

List of Figures

List of Tables

References

Abras, C., D. Maloney-Krichmar, and J. Preece [2001]. *User-Centered Design*, pages 763–768. unknown.

Amarasingham, R., L. Plantinga, M. Diener-West, D.J. Gaskin, and N.R. Powe [2009]. *Clinical Information Technologies and Inpatient Outcomes: A Multiple Hospital Study. Archives of internal medicine*, 169(2), pages 108–114. doi:10.1001/archinternmed.2008.520. `http://www.ncbi.nlm.nih.gov/pubmed/19171805`.

Android-Developer-Guide [2011]. *The Developer's Guide*. `http://developer.android.com/guide/index.html`. Last access 07/2011.

Becker, A. and M. Pant [2010]. *Android 2: Grundlagen und Programmierung*. Dpunkt Verlag. ISBN 3898646777. `http://books.google.com/books/about/Android_2.html?hl=de&id=aUkSRQAACAAJ`.

Bohlen, M., C. Brandon, and W. Zoons [2011]. *AndroMDA*. `http://www.andromda.org/`. Last access 07/2011.

Budde, R. and H. Zullighoven [1990]. *Prototyping revisited*. In *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 418–427. ISBN 0-8186-2041-2. doi:10.1109/CMPEUR.1990.113653. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=113653`.

CEN [2009]. *ISO 13485 Medical Devices: Quality management systems: Requirements for regulatory purposes*.

CENELEC [2008]. *EN 62366 Medical devices: Application of usability engineering to medical device*.

Cerami, E. [2002]. *Web services essentials*. O'Reilly. ISBN 9780596002244. http://books.google.com/books?id=j-YOMIoLXWYC.

Charpentier, G., P. Benhamou, D. Dardari, A. Clergeot, S. Franc, P. Schaepelynck-Belicar, B. Catargi, V. Melki, L. Chaillous, A. Farret, J. Bosson, and A. Penfornis [2011]. *The Diabeo Software Enabling Individualized Insulin Dose Adjustments Combined With Telemedicine Support Improves HbA1c in Poorly Controlled Type 1 Diabetic Patients: A 6-month, randomized, open-label, parallel-group, multicenter trial. Diabetes Care*, 34(3), pages 533–539. doi:10.2337/dc10-1259. http://care.diabetesjournals.org/content/34/3/533.

Chen, J., S. Su, and C. Chang [2010]. *Diabetes care decision support system*. In *2010 2nd International Conference on Industrial and Information Systems*, pages 323–326. ISBN 978-1-4244-7860-6. doi:10.1109/INDUSIS.2010.5565846. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5565846.

Clement, S., S.S. Braithwaite, M.F. Magee, A. Ahmann, E.P. Smith, R.G. Schafer, and I.B. Hirsch [2004]. *Management of diabetes and hyperglycemia in hospitals. Diabetes Care*, 27(2), pages 553–591. http://www.ncbi.nlm.nih.gov/pubmed/14747243.

Feldmann, R.L, F. Shull, C. Denger, M. Host, and C. Lindholm [2007]. *A Survey of Software Engineering Techniques in Medical Device Development*. In *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, pages 46–54. ISBN 978-0-7695-3081-9. doi:10.1109/HCMDSS-MDPnP.2007.4. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4438163.

Flasnoecker, M. [1999]. *Thiemes Innere Medizin*. Thieme, Stuttgard. ISBN 3131123613.

Fowler, M. and K. Scott [1999]. *UML Distilled: A Brief Guide to the Standard*

Object Modeling Language (2nd Edition). Addison-Wesley Professional. ISBN 0-201-65783-X.

Friesen, J. and D. Smith [2011]. *Android Recipes: A Problem-Solution Approach*. Apress Series, Apress. ISBN 9781430234135. http://books.google.com/books?id=mJdDp9GuhSQC.

Gartner [2011]. *Gartner Outlines 10 Mobile Technologies to Watch in 2010 and 2011*. http://www.gartner.com/it/page.jsp?id=1328113. Last access 05/2011.

GitHub [2011]. *android-archetypes*. https://github.com/akquinet/android-archetypes. Last access 07/2011.

Hall, K. [2010]. *Developing Medical Device Software to IEC 62304*. http://www.emdt.co.uk/article/developing-medical-device-software-iso-62304. Last access 07/2011.

Hameed, K. [2003]. *The application of mobile computing and technology to health care services*. *Telemat. Inf.*, 20, pages 99–106. ISSN 0736-5853. doi:10.1016/S0736-5853(02)00018-7. http://portal.acm.org/citation.cfm?id=766754.766755.

Haywood, A. and G. Boguslawski [2009]. *I Love My iPhone ... But There Are Certain Things That Niggle Me*. In *Human-Computer Interaction. New Trends*, volume 5610, pages 421–430. Springer Berlin / Heidelberg. http://dx.doi.org/10.1007/978-3-642-02574-7_47.

Höll, B., S. Spat, J. Plank, L. Schaupp, K. Neubauer, P. Beck, F. Chiarugi, V. Kontogiannis, T.R. Pieber, and A. Holzinger [2011a]. *Design of a mobile, safety-critical in-patient Glucose Management System*. In *Proceedings of MIE 2011*, page unknown. ISBN unknown. doi:unknown. unknown.

Höll, B., S. Spat, J. Plank, L. Schaupp, K. Neubauer, P. Beck, T.R. Pieber, and A. Holzinger [2011b]. *Design einer mobilen Anwendung für das stationäre Glukosemanagement*. In *Proceedings of EHealth 2011*, page unknown. ISBN unknown. doi:unknown. unknown.

Holzinger, A. [2005]. *Usability Engineering for Software Developers. Communications of the ACM*, 48(1), pages 71–74.

Holzinger, A. and M. Errath [2007]. *Mobile computer Web-application design in medicine: some research based guidelines. Univers. Access Inf. Soc.*, 6, pages 31–41. ISSN 1615-5289. doi:10.1007/s10209-007-0074-z. http://portal.acm.org/citation.cfm?id=1283708.1283718.

Holzinger, A., M. Hoeller, M. Bloice, and B. Urlesberger [2008]. *Typical Problems with developing mobile applications for health care: Some lessons learned from developing user-centered mobile applications in a hospital environment*, pages 235–240. IEEE.

ISO [2007]. *ISO 14971 Medical Devices: Application of risk management to medical devices.*

Koomen, T. and M. Pol [1999]. *Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing.* Addison-Wesley Professional. ISBN 0201596245. http://www.amazon.com/Test-Process-Improvement-step-step/dp/0201596245.

Korytkowski, M.T., R.J. Salata, G.L. Koerbel, F. Selzer, E. Karslioglu, A.M. Idriss, K. Lee, A.J. Moser, and F.G.S. Toledo [2009]. *Insulin therapy and glycemic control in hospitalized patients with diabetes during enteral nutrition therapy: a randomized controlled clinical trial. Diabetes Care.*, 32, pages 594–596. doi:10.2337/dc08-1436. http://care.diabetesjournals.org/content/32/4/594.full.pdf.

Kuebeck, S. [2009]. *Software-sanierung: Weiterentwicklung, Testen und Refatoring bestehender Software.* mitp-Verlag. ISBN 9783826650727. http://books.google.com/books?id=TjsIEq6U_f4C.

Kuzmanovic, N., T. Maruna, M. Savic, G. Miljkovic, and D. Isailovic [2010]. *Google's android as an application environment for DTV decoder system.* In *Consumer Electronics (ISCE)*, pages 1–5. ISBN 0-7803-4184-8. doi:10.1109/ISCE.2010.5523724. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5522728.

Lau, T. [1997]. *Toward a user-centered web design: lessons learned from user feedback*. In *Proceedings of the Proffessional Communication Conference 1997*, pages 149–153. ISBN 0-7803-4184-8. doi:10.1109/IPCC.1997.637042. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=637042.

Mackinnon, T., S. Freeman, and P. Craig [2001]. *Endo-Testing : Unit Testing with Mock Objects*. *Most*, page 287301. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.3214&rep=rep1&type=pdf.

Mann, E.A., J.A. Jones, S.E. Wolf, and C.E. Wade [2009]. *Computer Decision Support Software (EndoTool) Safely Improves Glycemic Control in the Burn Intensive Care Unit*. https://ccc.amedd.army.mil/conferences/2009/posters/CT5.pdf.

Moghissi, E.S., M.T. Korytkowski, M. Dinardo, D. Einhorn, R. Hellman, I.B. Hirsch, S.E. Inzucchi, F. Ismail-Beigi, M.S. Kirkman, and G.E. Umpierrez [2009]. *American Association of Clinical Endocrinologists and American Diabetes Association Consensus Statement on Inpatient Glycemic Control*. *Diabetes Care*, 32(6), pages 1119–1130. doi:10.2337/dc09-9029. http://care.diabetesjournals.org/content/32/6/1119.full.pdf.

Mosemann, H. and M. Kose [2009]. *Android: Anwendungen fr das Handy-Betriebssystem erfolgreich programmieren*. Carl Hanser Verlag. ISBN 3446417281. http://www.amazon.de/Android-Anwendungen-Handy-Betriebssystem-erfolgreich-programmieren/dp/3446417281.

Niederschmid, B. [2010]. *Samsung Galaxy Tab GT-P1000: Backblech mit Potenzial*. http://www.androidapptests.com/samsung-galaxy-tab-gt-p1000-backblech-mit-potenzial.html. Last access 07/2011.

Nuseibeh, B. and S. Easterbrook [2000]. *Requirements engineering: a roadmap*. In *Proceedings of the Conference on The Future of Software Engineering*, pages 35–46. ISBN 1-58113-253-0. doi:10.1145/336512.336523. http://delivery.acm.org/10.1145/340000/336523/p35-nuseibeh.pdf?

key1=336523&key2=8726584031&coll=DL&dl=ACM&ip=129.27.12.54&CFID=19851822&CFTOKEN=96166632.

OVE/ON [2007]. *EN 62304 Medizingeraete-Software: Software-Lebenszyklus-Prozesse.*

Pandey, D., U. Suman, and A.K. Ramani [2010]. *An Effective Requirement Engineering Process Model for Software Development and Requirements Management.* In *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, pages 287–291. ISBN 978-1-4244-8093-7. doi:10.1109/ARTCom.2010.24. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05656776.

Paul, K. and T.K. Kundu [2010]. *Android on Mobile Devices: An Energy Perspective. Computer and Information Technology, International Conference on*, 0, pages 2421–2426. doi:http://doi.ieeecomputersociety.org/10.1109/CIT.2010.416.

Preuveneers, D. and Y. Berbers [2008]. *Mobile phones assisting with health self-care: a diabetes case study.* In *Proceedings of the 10th Conference on Human-Computer Interaction with Mobile Devices and Services, Mobile HCI 2008,*, pages 177–186. ACM International Conference Proceeding Series. doi:http://doi.acm.org/10.1145/1409240.1409260. https://lirias.kuleuven.be/handle/123456789/229978.

REACTION [2011]. *This is the REACTION project.* http://www.reactionproject.eu. Last access 05/2011.

Roy, G.G. [2004]. *A risk management framework for software engineering practice.* In *Proceedings of the 2004 Australian Software Engineering Conference*, page 60. ISBN 0-7695-2089-8. doi:10.1109/ASWEC.2004.1290458. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1290458.

Runeson, P. [2006]. *A Survey of Unit Testing Practices. IEEE Softw.*, 23, pages 22–29. ISSN 0740-7459. doi:10.1109/MS.2006.91. http://portal.acm.org/citation.cfm?id=1159169.1159387.

Schulz, D. [2005]. *MDA-Frameworks: AndroMDA.*

Smart, J.F. [2008]. *Java Power Tools*. O'Reilly Media. ISBN 0596527934. http://www.amazon.com/Java-Power-Tools-Ferguson-Smart/dp/0596527934.

Svanaes, D., O.A. Alsos, and Y. Dahl [2010]. *Usability testing of mobile ICT for clinical settings: Methodological and practical challenges. . International Journal of Medical Informatics*, 79, pages 24–34. http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6T7S-4TF07W3-2-1&_cdi=5066&_user=464374&_pii=S138650560800110X&_origin=gateway&_coverDate=04%2F30%2F2010&_sk=999209995&view=c&wchp=dGLbVlz-zSkzk&md5=fa12d75864675b17ff43dd9124a6f54c&ie=/sdarticle.pdf.

Thompson, C., J. White, B. Dougherty, and C.D. Schmidt [2009]. *Optimizing Mobile Application Performance with Model Driven Engineering*. In *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, pages 36–46. Springer-Verlag. ISBN 978-3-642-10264-6. doi:http://dx.doi.org/10.1007/978-3-642-10265-3_4. http://portal.acm.org/citation.cfm?id=1694300.

Umpierrez, G.E., T. Hor, D. Smiley, A. Temponi, D. Umpierrez, M. Ceron, C. Munoz, C. Newton, L. Peng, and D. Baldwin [2009]. *Comparison of inpatient insulin regimens with detemir plus aspart versus neutral protamine hagedorn plus regular in medical patients with type 2 diabetes. The Journal of clinical endocrinology and metabolism*, 94(2), pages 564–569. doi:10.1210/jc.2008-1441. http://jcem.endojournals.org/cgi/reprint/94/2/564.

Umpierrez, G.E., D. Smiley, A. Zismann, L.M. Prieto, A. Palacio, M. Ceron, A. Puig, and R.Mejia [2007]. *Randomized Study of Basal-Bolus Insulin Therapy in the Inpatient Management of Patients With Type 2 Diabetes. Diabetes Care*, 30(9), pages 2181–2186. doi:10.2337/dc07-0295.Clinical. http://care.diabetesjournals.org/content/30/9/2181.full.pdf.

USDepartment [2011]. *National Diabetes Statistics, 2011*. http://diabetes.niddk.nih.gov/dm/pubs/statistics/. Last access 05/2011.

Wallace, D.R. and D.R. Kuhn [2001]. *Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data*. In *ACS/ IEEE International Conference*

on Computer Systems and Applications, pages 301–311.
http://csrc.nist.gov/groups/SNS/acts/documents/final-rqse.pdf.

Warmer, J. and A. Kleppe [2003]. *The Object Constraint Language: Getting your models ready for MDA*. Addison-Wesley Professional. ISBN 0321179366.

Wimmer, M. [2005]. *Model Driven Architecture in der Praxis: Evaluierung aktueller Entwicklungswerkzeuge und Fallstudie*. Master's Thesis, TU Wien, Austria.
http://www.big.tuwien.ac.at/system/theses/65/papers.pdf?1298559218.

Wu, J., S. Wang, and L. Lin [2007]. *Mobile computing acceptance factors in the healthcare industry: a structural equation model. International Journal of Medical Informatics*, 76, pages 66–77.
http://www.ncbi.nlm.nih.gov/pubmed/16901749.

Zambuto, R.P. [2004]. *Clinical engineers in the 21st century. Engineering in Medicine and Biology Magazine*, 23(3), pages 37–41. ISSN 0739-5175.
doi:10.1109/MEMB.2004.1317980.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1317980.