NTNU
Norwegian University of Science and Technology

Usability of Commercial mHealth Toolkits From a Developer Perspective

An Empirical Evaluation of Google Fit, Apple HealthKit and Samsung Digital Health

Nemanja Aksic
Petter Astrup
Erik Gunnar Jansen

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Babak Farshchian, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Preface

This study is a master thesis that aims to evaluate the leading commercial mHealth Toolkits. The associated subject code is TDT4900. The thesis is conducted in the last semester of the master degree program in Computer Science at the Norwegian University of Technology and Science(NTNU), the spring semester of 2016. The thesis is written for the Department of Computer and Information Science. The idea that lead to the master thesis came from discussions on the subject with our supervisor Babak Farshchian. It is assumed that the readers of this thesis have a technical background.

Trondheim, 2016-06-10

Nemanja Aksic   Petter Astrup   Erik Gunnar Jansen

Acknowledgment

Firstly, we would like to thank our supervisor, Babak A. Farshchian, Researcher and Research Manager at SINTEF and Adjunct Associate Professor at the Norwegian University of Science and Technology for his help and feedback in our research process. Secondly we would like to thank all the students that participated in our observation study and lastly we would like to thank all the developers that contributed to the study by answering the questionnaire we posted online.

P.A., N.A. and E.G.J.

(Your initials)

Abstract

Self-monitoring and sharing of data to enhance personal fitness and health has rapidly increased in popularity, and as a result, there has emerged a market for delivering tools supporting development and usage of technology that can contribute to this emerging trend. This thesis aims to describe and evaluate the leading development platforms in this field from the perspective of a developer developing mobile health applications. The conducted research consisted of a case study where data was generated from online documents, online questionnaires, and from the development of example applications using different mobile health platforms. In addition, we performed an observation study with students where we gave the students concrete tasks with the examined mHealth Toolkit platforms, with corresponding interviews and questionnaires. The three platforms Apple HealthKit, Google Fit, and Samsung Digital Health(SDH) were found relevant, examined and then compared, to find strengths and weaknesses of each platform with regards to the stated issues. The analysis was used to evaluate the productivity, usability and added creativity to the development process for developers by using the named mHealth Toolkits.

With the thesis, we found that when designing an mHealth Toolkit to be utilized by developers when developing mobile health applications and services, there are several concerns that need to be addressed to increase the efficiency and productivity of the developer. From the document analysis, we found that the Toolkits differ in technical architecture, data models and application development process, where the most important finding was that Google Fit cannot be used for Health purposes. The example system revealed that mHealth applications can be developed using both Apple HealthKit and SDH, using significantly less time and lines of code with HealthKit. From the observation study, we found that both Apple HealthKit and SDH simplified the overall development process. In addition, SDH's API proved to have a more intuitive naming and a more descriptive documentation than HealthKit. HealthKit's higher level of implementation was however more intuitive. The questionnaire with experienced developers revealed that Apple HealthKit scored highest among the three with developers commenting that no toolkit is stable/reliable enough to be used in production. Despite this, the evaluated Toolkits show potential and if the mHealth Toolkit providers address the identified concerns, they might be viable solutions in the foreseeable future.

Keywords: mHealth Toolkits, Samsung Digital Health, Apple HealthKit, Google Fit, developer productivity, developer efficiency, mobile health application

Contents

Chapter 1

Introduction

The following chapter will *introduce the problem description and motivation for doing the research* as well as the research questions formulated to drive the research in this study. The scope and contribution of the thesis and a thesis outline will also be presented.

1.1 Problem Description and Motivation

The computing power of smartphones and the increased significance of cloud storage and sharing has drastically changed the opportunities within mobile healthcare, and as a result, a need for innovative hardware devices and development Toolkits/platforms to handle interaction with these devices has emerged. The Toolkits/platforms provide the developers with a lot of functionality, where the most important is means to store and access a user's health data in a centralized data store.

In the context of health and mobile devices, *mHealth* is an expression that surfaces. mHealth is an abbreviation for mobile health and is used to describe mobile devices or applications that affect health-related aspects. In [45], professor John Orzechowski describes mHealth "*as the access, provision and/or delivery of healthcare interactions—anywhere, anytime—facilitated by mobile and/or wireless technologies*".

The incorporation of mHealth into the healthcare sector is becoming continuously more common. According to a market report published by MarketsandMarkets (a market research and consulting firm, ranked #2 in the world in premium market research studies published annually), that the mHealth market will increase by 33,4% and reach USD 59.15 Billion by 2020[39]. In addition, Buttarelli[2015] states that "*mHealth offers a wealth of new opportunities, in terms of better and more responsive healthcare for individuals, better disease prevention and lower healthcare costs for welfare systems and greater opportunities for businesses*"[23]. PwC[2012] has conducted research regarding mHealth using a survey and interviews where the participants were managers of healthcare institutions and information technology companies. A part of the research consisted of asking the participants to name the "*Top drivers for patients to consider beginning to use or increasing use of mHealth applications/services*"[48]. The top three drivers given were: increased efficiency for healthcare providers, reduction of healthcare costs and greater user control of own health. These drivers are also among the motivations for developing and employing a Toolkit for healthcare enhancement.

In the mHealth market today there is a range of applications that track and access fitness data. A search for "*fitness*" and "*health*" in Google Play Store and Apple App Store resulted in over 100 applications for each search term in each of the stores. The way these applications are developed, and how the data they generate and access are handled varies. The main idea behind a mHealth Toolkit is to abstract away functionality common to the mHealth applications and provide the developers with tools to make it easier to create mHealth applications. Our definition of a mHealth Toolkit is, based on our preliminary study, a set of tools in the form of libraries and APIs that assist in the creation of an mHealth application. The tools include APIs to store and access data, manage data permissions in addition to the libraries, used to access sensor data from the mobile device. The terms mHealth Toolkit(s) and Toolkit(s) will be used interchangeably in the remaining chapters.

Little to no significant research has been conducted on mHealth Toolkits and how they affect the developer. Our motivation for this study is to explore the leading mHealth Toolkits from the developer perspective and investigate how they can add value to the development process of an mHealth application and increase the developer productivity. It is important to evaluate the leading mHealth Toolkits because the evaluation might give insight to platform providers in how they can improve the experience of using mHealth Toolkits for the developer, assess their approach when it comes to technical architecture and thereby simplify development of mHealth applications. The findings from this thesis might also be generalizable to other types of platforms and Toolkits. According to Kay et al.[2011] in an article from World Health Organization(WHO), a list of the most significant barriers to mHealth implementation, includes themes such as; knowledge and technical expertise[36]. If the mHealth Toolkits can provide an abstraction of low-level implementation and provide tools to the developer that simplifies the process of building mHealth applications, these barriers can be passed.

The Toolkits were selected based on criteria described in section 2.1.2 and were evaluated and investigated by conducting a literature study, developing an example system, conducting an observation study with students, and sending questionnaires to experienced mHealth Toolkit developers. These data generation methods allowed us to examine how the selected mHealth Toolkits differ with regards to technical architecture, data models, and application development process and how they add value to the development process of mHealth applications.

1.2 Research Questions

With the goal of exploring the leading mHealth Toolkits from the developer perspective and investigate how they can add value to the development process of an mHealth application, we have formulated the following research questions:

1. RQ1: What separates the technical architecture, data models and application development process of the leading mHealth Toolkits and what aspects affect productivity for a developer?
    - This research question is formulated to investigate the differences of the leading mHealth Toolkits. We will investigate their technical architecture, data models and application development process in order to determine

what aspects that affect the productivity of developers.

2. RQ2: How do new adopters perceive the mHealth Toolkits in terms of documentation and development process when creating a fundamental mHealth application?

    - This research question is formulated to investigate how new adopters perceive the mHealth Toolkits. If the mHealth Toolkits are to be used and accepted within the developer community, they should be easy to learn and use. This research question is asked to investigate the ease to learn and use a mHealth Toolkit, by having students without prior experience with the mHealth Toolkit in question develop a fundamental mHealth application in a predefined environment. We define a fundamental mHealth application as an application that can store and retrieve Health data.

3. RQ3: What is the view on mHealth Toolkits seen from the experienced developers and new adopters based on their experience?

    - This research question is formulated to investigate what the view developers with different experience have on the mHealth Toolkits. In order to improve, mHealth Toolkits must absorb developer experiences and adapt accordingly. This research question concerns experienced and newly adopting developer's perspective on the use of and experience with the mHealth Toolkits.

4. RQ4: What mHealth Toolkit is best equipped to be utilized in development of an mHealth service in terms of developer productivity?

    - This research question is formulated to investigate how mHealth Toolkits can be used in an mHealth service. mHealth toolkits are built independently and they follow no common practice or industry standards. This research question is asked in order to ascertain which mHealth Toolkit is best equipped to be utilized to create a mHealth service if one were to be utilized.

1.3 Scope and Contributions

This thesis will focus on evaluating the leading mHealth Toolkits from the developer perspective and investigate how the mHealth Toolkits affect developer efficiency and productivity when developing an mHealth application. We have selected this scope because it gives us the ability to provide a thorough evaluation from the developer perspective.

The main contribution of this research will be a theory based on the analysis obtained by investigating the described case, namely developers using mHealth Toolkits for the creation of mHealth applications and how these mHealth Toolkits can provide support and enforce productivity for a developer when developing mHealth applications. In other words, the theory will provide insight into how the productivity of a developer is affected with the use of the chosen mHealth Toolkits.

1.4 Structure of the Report

The master thesis is divided into 10 chapters. In addition to the introduction chapter, there are 9 chapters, which will be presented below.

Chapter 2 Background and previous work

In this chapter, we will describe the background and previous work related to the thesis. The concept of Toolkits will be described and elaborated, both in form the of Software Toolkits as well as the more specified mHealth Toolkits. The process in our preliminary study of selecting relevant mHealth Toolkits to evaluate will also be discussed. Lastly, the previous work related to our thesis will be presented.

Chapter 3 Methodology

In this chapter, we will introduce the methodology used in the research. The conceptual framework will be described, the chosen research strategy will be presented and elaborated, the data generation methods will be presented, and finally the data analysis process will be presented and elaborated.

Chapter 4 mHealth Toolkits

In this chapter, the chosen mHealth Toolkits will be presented. Each mHealth Toolkit will be presented with the same structure. First, the technical architecture of the mHealth Toolkits will be presented, then their data structure and storage, an actor-network of the different actors in the environment of the mHealth Toolkit will be provided, and finally the application development process of the Toolkits will be described.

Chapter 5 Development of Example system with mHealth Toolkits

In this chapter, the process of developing an example system will be described. The different parts of the system will be presented and described in detail. The metrics we chose to track during the development process will then be presented.

Chapter 6 mHealth Toolkits Observation Study, Interviews and Questionnaires with Developers

In this chapter, the observation study, interviews and questionnaires conducted with students are described. Details of the observation study are described first before the interviews and questionnaires are described.

Chapter 7 Questionnaire with Experienced mHealth Toolkit Developers

In this chapter, the questionnaire we conducted with experienced mHealth Toolkit developers will be described. The process of creating and distributing the questionnaire will be described.

Chapter 8 Summary of Findings

In this chapter, the summary of findings from our research will be presented. It will include findings from all the data generation methods.

Chapter 9 Discussion

In this chapter, our findings will be discussed with regards to the background, previous work, and the stated research questions.

Chapter 10 Conclusions and Future Work

In this chapter, we will draw conclusions and propose future work within the field of development with mHealth Toolkits based on our research findings.

Chapter 2

Background and Previous Work

In this chapter, the background and previous work related to our research will be described. In the background section, the factors and findings from our autumn project that influenced our initial choices in the thesis research process will be described. The autumn project was conducted in the subject *TDT4501 Specialization Project* in the fall of 2015. The master thesis may be seen as a continuation of this project, where the project acted as a preliminary research study. After the background section, the previous work relevant to our thesis will be presented

2.1 Background

This section will describe the process leading to the aspects focused on in our master thesis research. The findings from the aforementioned specialization project established a baseline for our research by providing a definition of an mHealth Toolkit, a selection of relevant mHealth Toolkits, as well as generated data for each of the selected ones. The generated data included details concerning technical architecture, data structure, data storage, application development process and actor networks. We choose to structure this information into a separate section, namely *Chapter 4: mHealth Toolkits*.

2.1.1 Software Toolkits

PCMag defines a Software Toolkit as "*A single utility program, a set of software routines or a complete integrated set of software utilities that are used to develop and maintain applications and databases*"[46], while Dictionary.com describes a toolkit as "*a set of tools designed to be used together or for a particular purpose*"[27]. Hence, Software Toolkits are not general purpose, but rather a coordinated set of instruments geared to solving a problem. Software Toolkits are important because they abstract away low-level implementation, make simple things achievable through a few lines of code, enables more rapid prototyping[31] and enables development of products at a lower cost[16]. It also increases developer productivity since the focus is moved to being creative rather than handling low-level implementations. In addition, software Toolkits standardize the way applications within a given field are developed and enables reusage of components.

2.1.2 mHealth Toolkits

Based on our preliminary study, our definition of an mHealth Toolkit is a set of tools in the form of libraries and APIs that assist developers when creating a mHealth application. These APIs and libraries provide functionality for data storage and access, granting data type permissions, and access to sensor data from a mobile device.

Our preliminary study, which involved discovering and evaluating mHealth Toolkits potentially suitable, resulted in a small set of Toolkits deemed relevant. The discovery consisted of a range of composed keyword searches performed in Google Search and Google Scholar, where the sources were published articles and blog posts, scientific papers, plus the Toolkits' documentation. The criteria we based our choice on were how many developers that used them, the documentation quality, the tools, the amount of partnerships within the health segment, and how much resources the manufacturer of the Toolkit had. The evaluation based on the given criteria resulted in the following being chosen for further assessment: Apple HealthKit, Google Fit, and Samsung Digital Health(SDH). All are actors with a substantial market share that provide a vast range of technology and a platform with extensive documentation. The results from the evaluation for each of the chosen Toolkits will be elaborated in Chapter 4.

2.2 Previous Work

In this section, previous work relevant to our thesis will be discussed. First, papers related to how Software Toolkits affect developer efficiency are presented and then papers regarding the usability of Toolkits/APIs for developers and how different aspects of the Toolkits/APIs affects the developer are presented.

Von Hippel and Katz[2002] discuss the importance of a Toolkit, and in the conclusion of the paper they write: "*We conclude by proposing, as we did at the start of this article, that toolkits for user innovation will eventually be adopted by many manufacturers facing heterogeneous customer demand*"[53]. The paper gives reason to believe that Toolkits are important to increase user innovation. The idea behind using Toolkits in software development is to abstract away difficult tasks for the application developer. Software Toolkits include abstractions of functionalities that are time-consuming to implement and are common elements across applications, e.g. low-level implementations of a remote server connection or access to a device sensor. Greenberg[2007] states that a Toolkit should: "*Remove low-level implementation burdens common to all groupware platforms (e.g. simplified access to communications, data sharing, concurrency control, session management)*"[31]. In addition, he states that: "*GroupKit considerably simplified groupware development e.g., using GroupKit we reimplemented GroupSketch and GroupDraw in a few hours using very little code. Other simple groupware tools were similarly rapid to build: a brainstorming tool in 74 lines, a graphical concept map editor in 213 lines, a file-sharing system in 51 lines, and a text-chat system in 80 lines of code*". These findings indicate that Toolkits make developers more productive in terms of lines of code and time used to complete given programming tasks.

Clarke[2004] describes how the usability of software tools affects developer efficiency. It discusses the user-centered design approach, where the objective is to "*make sure that you understand the characteristics of users and how those characteristics impact the way they expect the API to work*"[24]. Further, the paper discusses the psychological term affordance in relationships to APIs and states that "*APIs expose affordances. Every API has a set of actions that it can perform. Therefore, usability problems can exist in an*

API that are related to users not perceiving the affordances the API supports". The paper concludes by stating the need to "*understand the characteristics of users and how those characteristics impact the way they expect the API to work*". The discussion and findings in this paper give reason to believe that in order to create a programming tool, such as a Toolkit, the key is to understand the programmer using the tool. The belief that you need to understand the developer in order to make a good programming tool is underlined in Henning[2007], where the author states that "*APIs should be designed from the perspective of the caller*"[33].

Scheller and Kühn[2012] evaluate a specific API, namely a file zipping API and want to identify influencing usability factors for the two most common concepts of APIs: classes and methods. They do so by conducting a study with 20 programmers and 2 different API variants and evaluate how differences between the APIs influence usability when instantiating classes and calling methods. They found that "*A high number of classes has a negative impact on performance when searching for a class and gives a negative impression to the programmers*"[51] and that "*calling methods with more parameters also takes more time*". Regarding the naming of the API's classes, they write that: "*If a class or method name didn't meet the programmer's expectations, he/she would need much more time finding it*". Additionally, they state that "*the study results indicate that the experience of a programmer has no significant influence on performance*". The study concerns classes and methods and they purpose for future work that "*further studies will need to be conducted to get a broader understanding of usability aspects for different API design decisions, like annotations, class inheritance, interfaces and XML configuration*". The findings from Scheller and Kühn[2012] gives reason to believe that the classes and methods of programming tool such as a Toolkit or an API need to be structured in a way that is easy to understand and learn for the developer.

In Bloch[2006], the importance of good documentation is emphasized. Here, one key is that "*Documentation matters. No matter how good an API, it won't get used without good documentation*"[21]. In Robillard[2009] the importance of good documentation is investigated through a survey and interviews with developers. The survey's core consisted of a three-part, open-ended question on the obstacles developers faced learning

APIs. Regarding the interviews, the author writes: “*I again chose an open-ended, loosely structured style of qualitative interview, which consisted of asking participants to summarize their work with the API and explain the obstacles they faced*” [49]. The study gives insight into how developers feel about APIs, for example, one of the answers received through the survey was: "*I don’t understand the design intents behind the API, the overall architecture, why certain functions are designed as such*". Based on the findings, the paper concludes that "*A major result of the survey is that resources topped the list of obstacles to learning APIs. This is a good reminder that efforts to improve the usability of an API’s structure need to be complemented by efforts to improve the resources available to learn them*". With that statement the author wants to convey that in addition to a good documentation, supplementary information such as code examples must be present in order to encourage an efficient learning process for developers using the software tool.

In this section, we have presented previous work that is related to our thesis. First, papers related to how Software Toolkits affect developer efficiency were presented. The previous work presented in that paragraph is related to RQ1. It is related to RQ1 because the papers suggests that a Toolkit can improve developer efficiency.
Secondly, the papers regarding the usability of Toolkits/APIs for developers are related to RQ2 and RQ3. They are related to RQ2 because both the papers and RQ2 assess how the user perceive Toolkits/APIs. RQ2 aims to investigate how developers grasp a Toolkit, and the papers state that it is important to design Toolkits/API with the user in focus and always have in mind how the user will grasp the API/toolkit. The papers are related to RQ3 because both the RQ and the papers concern the views of developers.
Lastly, papers describing how the different aspects of the Toolkits/APIs, including classes, methods and documentation, affect the developer were presented. These aspects concern all of our RQ’s.

Chapter 3

Methodology

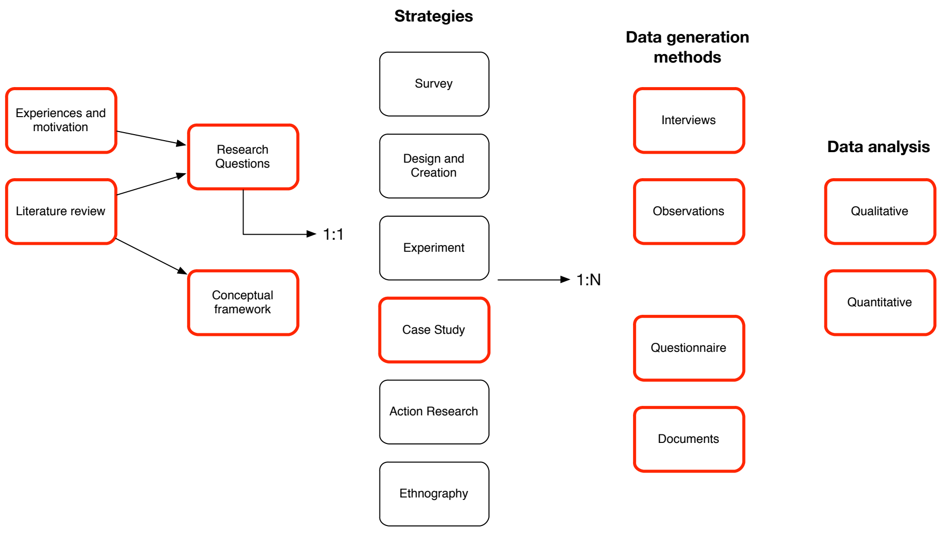


Figure 3.1: Research Process [44, p. 33]

In this section, the research process will be described. A graph depicting an overview of the process can be viewed in Figure 3.1, where the relevant components are highlighted with a thick red border. This process is based on the process described in "Researching Information Systems and Computing"[44]. Each of these components, such as research strategy, data generation methods and data analysis will be described in further detail in this section.

3.1 Conceptual Framework

In an attempt to perform this study we have detailed a conceptual framework, which follows the definition as described by Miles and Huberman[1994]: “*A conceptual framework explains, either graphically or in narrative form [both are much preferred], the main things to be studied – the key factors, constructs or variables – and the presumed relationships among them*”[41, p. 18].

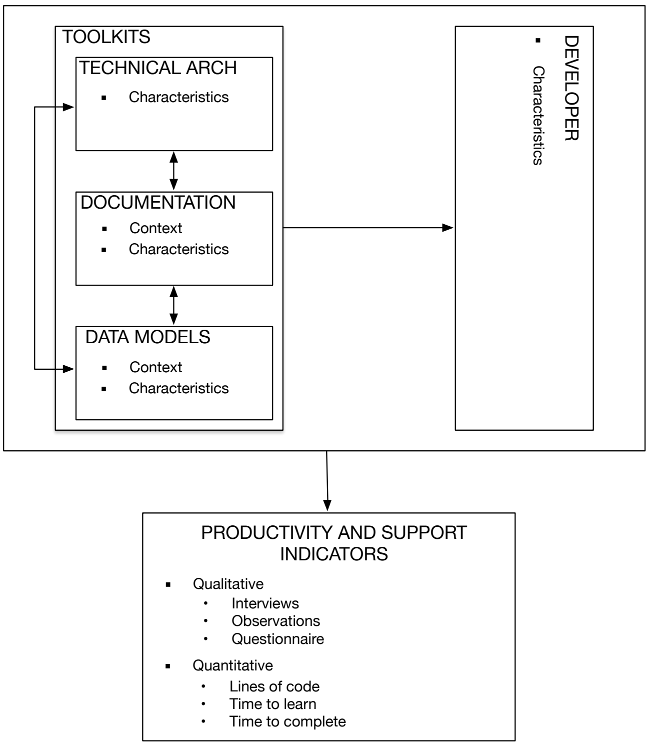


Figure 3.2: Visual description of conceptual framework

The conducted study empirically evaluates the leading mHealth Toolkits in terms of technical architecture, data models and documentation. Another aspect of the study will be to look at how developers are affected with regards to productivity and support. Figure 3.2 is a visual representation of the conceptual framework, which describes the aspects to be examined and their interrelations.

Within the "*TOOLKIT*" box, there are three aspects we will be focusing our attention on: "*TECHNICAL ARCH*", "*DOCUMENTATION*" and "*DATA MODELS*" respectfully. "*TECHNICAL ARCH*" represents the technical architecture of the mHealth Toolkit to be studied, namely the "*characteristics*", which represent distinct aspects of the Toolkits from a technical point of view. The "*DOCUMENTATION*" box is self explanatory as it represents Toolkits' documentation. The "*characteristics*", represent the structure and usability of the documentation, will be evaluated in multiple "*context's*"(i.e. settings). The final box "*DATA MODELS*", focuses on the Toolkit choices of storing data and its "*characteristics*". The context property in this case will represent in what setting it will be evaluated, e.g. whether health or fitness data is stored.

The aspects described above will all be assessed in relationship to a "*DEVELOPER*" and each other; an arrow represents a relation. A developer has certain "*characteristics*" in terms of experience, which are considered in the study. Productivity and support indicators represent the variables within our study, which will be described in further detail in the following chapters.

3.2 Research Strategy

The motivation behind this study is to explore and investigate how mHealth Toolkits support and affect the productivity of application developers. To gain insight into these aspects, we adopted a case study as the research strategy. The main reason for adopting a case study is that we wished to examine mHealth Toolkits within a simulated context through observations, supported by additional data sources related to the Toolkits' effect on developer productivity. Since we wanted to describe and understand these aspect, our case study was determined to be a descriptive one, because it "leads to a rich, detailed analysis of a particular phenomenon and its context"[44, p.143]. In our case the phenomenon is developer efficiency and productivity, the context is development of mobile health applications, and its units of analysis are the Toolkits: Apple HealthKit, Google Fit and SDH. In accordance with the main focus of the case study, namely the current status of mHealth toolkits, the study was mainly short-term and contemporary; but it can also be seen as a historical study, due to the fact that a majority of the literature found on the subject is based on earlier events.

3.3 Data Generation Methods

In compliance with the norm for case studies, it was important for us to have multiple sources of data, both quantitative and qualitative, in order to obtain an adequate basis for the results to be validated against other sources. Quantitative data was vital to provide quantifiable measures of developer productivity and support, while qualitative data was valuable to be able to abstract themes and patterns that we deemed important for our research questions. With this requirement in mind, it was highly suitable with a case study strategy encouraging the use of multiple data generation methods. The previous study described in section 2.1.2 resulted in three Toolkits found relevant(Apple HealthKit, Google Fit, SDH), and for each of these three was data generated by employing the following methods: document analysis, observations of participant development, questionnaires, interviews, and mHealth example system development. The methodology of each of these methods will now be described.

Method	**RQ1**	**RQ2**	**RQ3**	**RQ4**
Documents & Analysis of online publications,regarding Google Fit, HealthKit and SDH:				
Observations:		Observations of students programming in predefined environment; 7 students with Apple HealthKit and 9 with Samsung Digital Health(SDH).		
Questionnaires:			Numerical answers from students post-observations, numerical and textual answers from experienced developers.	
Interviews:		Transcriptions of interviews with students after their first programming experience with given Toolkit (HealthKit and SDH)		
Example System Development:	Development of system with mobile applications that utilizes the main functionality of HealthKit and SDH			

Table 3.1: Relation between the generated data and research questions it provides answers to

3.3.1 Documents

The aforementioned Toolkits were focused on in the generation of data from documents their documentation, technical class structure and hierarchy(addressing RQ1). The data was obtained by the same process as previously described in the previous paragraph, except that a different set of keywords in the searches was used. The list of keywords included: "*Google Fit*", "*Apple HealthKit*", "*Samsung Digital Health*", "*mHealth*", "*developer*", "*development*", "*application*", "*healthcare*", "*service*" , "*partnerships*", "*productivity*", "*usability*". An important note is that this process resulted in Google Fit being excluded from the data generation with the remaining methods(due to reasons described in section 4.1 and elaborated in section 5.2 and 9.1.1).

During the data generation and analysis, there were two concerns, the first being the evaluation of the validity of found documents with regards to their purpose and process, and the second being the amount of relevant documents available. An attempt to increase the validity of the found documents involved putting more weight on scientific publication than other sources, and using those with more citations and having a lower inclusion criteria of five citations(listed by Google Scholar) for all documents published before 2014. The amount of available relevant documents, and especially scientific publications, was estimated to be low, and it was hence decided to put more weight on the other sources in compliance with the method triangulation technique[44, p. 37].

3.3.2 Observations

In order to investigate the elements affecting how new adopters perceive the development of a fundamental application, it was decided to conduct observations(addressing RQ2). Participants for the observations were selected according to non-probabilistic convenience sampling and were hence chosen mainly due to their availability and willingness to be involved[40]. It involved recruiting participants among fellow computer science students both for development with Apple HealthKit and SDH. The requirements for participation were a minimum of two years of experience with computer science from a university, no previous exposure to or experience with the designated Toolkit, and experience with the corresponding programming language and the integrated development environment(IDE).

The observations were overt, i.e. the participants were aware that they were being observed. The observations were conducted by watching the interactions of the participants solving tasks in the designated development environment. The researchers' degree of participation during observation was passive participation, meaning that "*activities are observed in the setting but without participation in activities*"[35]. The participants were also asked to complete a questionnaire and an interview following the end of the observation.

The main source of data collected during the observations were video recordings captured by a screen recording software on the environment computer. These video recordings were to be used as a basis for quantitative and qualitative analysis of the participants' actions. In addition, the observers took field notes of general observations during each observation.

Due to the time spent conducting each observation and the length of the study, it was possible that these time constraints would prevent us from attaining a sufficient sample size and relevance. Nevertheless, the time constraint was overshadowed by the problem with recruiting participants that met the requirements, which seemed to have a greater impact in our case. The attained sample set affects both the findings from quantitative and qualitative data analysis.

The required sample size for qualitative data analysis is highly related to factors such as the type and the goal of the study, and Marshall[1996] states that "*an appropriate sample size for a qualitative study is one that adequately answers the research question*"[40]. As mentioned previously, the qualitative data generated from the observations only served as a supplement to data generated from the interviews, and the required sample size was hence equal to the required sample size for the interviews. The difficulty in attaining a relevant sample set should also be mentioned, which we overcame through the use of a non-probabilistic convenience selection approach and the fact that the population of this case study is considered to be relatively homogenous makes the issue negligible.

3.3.3 Questionnaires

Self-administered questionnaires were used as a method for generating additional data in relation to developers' thoughts, with the purpose of being applied to both developers recently exposed to an experience using a Toolkit as well as developers who have passed this stage and gained valuable experiences(addressing RQ3). In our study, two variants of questionnaires were composed: one to be completed along with the observations, and the other aimed at developers with established experience with the Toolkit(s) in question. The first questionnaire was answered post-development by the participants in the observations to supplement the other data generated from the observations, while the second was used in an attempt to reach out to developers with programming experience with the Toolkits.

Both questionnaires included the same quantitative questions with a clearly defined scale for the answers, but since the experienced developers did not participate in an interview, they were also asked to answer qualitative questions to make it possible for them to provide additional thoughts.

As with the observations, the analysis of quantitative data generated from the questionnaires was impacted by the sample size, and the number of questionnaires related to the observations was limited by the pool of participants and we knew that the questionnaire aimed at experienced developers was susceptible to receiving few responses. In an attempt to increase the number of received responses the questionnaire was sent out via all communication channels expected to have a chance at reaching experienced developers.

3.3.4 Interviews

Interviews were selected as the main source of qualitative data to gain a deeper insight into aspects affecting the development process, the developer productivity, and the comparable differences between the Toolkits, based on the views of developers after obtaining unprecedented experience(addressing RQ2 and RQ3). The interviews were determined to be semi-structured with the advantage that they "*consist of several key questions that help to define the areas to be explored, but also allows the interviewer or* *interviewee to diverge in order to pursue an idea or response in more detail*"[28]. Audiotaping was used during the interview to record the conversation to be used in data analysis.

An essential challenge when conducting interviews is related to the interviewer's performance during the process, including the need to act similarly in all interviews and asking clarifying follow-up questions to the interviewee's initial answers. It was crucial to meet both of these needs across interviews with all participants developing with the same Toolkit and across the set of Toolkits examined. The significance of this challenge was increased because the interviews were to be conducted by two different interviewers, which could result in inconsistencies. Actions taken to prevent such inconsistencies were having the interviewers plan and discuss the interview process together, and then collaboratively create a predefined protocol with guidelines to be followed by each interviewer.

Mentioned earlier in this section, is the sample size for generating qualitative data an issue both for observations and the interviews. Regarding semi-structured interviews, which was the structured chosen, there seems to be some dispute in the literature about the required sample size. For example, there's Morse[2000] that states that "*If, when using semistructured interviews, one obtains a small amount of data per interview question (i.e., relatively shallow data), then to obtain the richness of data required for qualitative analysis, one needs a large number of participants (at least 30 to 60)*"[42], indicating a relatively large sample size required in our case. On the other hand, there's Guest et al.[2006] stating that "*they found that saturation occurred within the first twelve interviews, although basic elements for meta themes were present as early as six interviews*"[32]. Further, they state that "*If the goal is to describe a shared perception, belief, or behavior among a relatively homogeneous group, then a sample of twelve will likely be sufficient as it was in our study*". It's important to note that even though Morse[2000] suggests a size of at least 30, it is only in the case of semi-structured interviews where a small amount of data per question is obtained, and in an attempt to obtain richer data several follow-up questions were asked to clarify and elaborate on answers.

When generating qualitative data, being highly relative and subjective by nature, it is important to validate the results to strengthen the claim of validity. Our attempt at strengthening this claim involved a triangulation of the data that was related to the interviews and observations, where the data from interviews was checked against data from the screen recording notes and observation field notes. The themes identified in the transcription analysis were checked for occurrences in the supplementary data, and the occurrences of themes were checked across interviews, rather than basing the findings on a single statement or individual.

3.3.5 Example System Development

A data generation method used that is not described in "*Researching Information Systems and Computing*"[44] is what we have defined as example system development. The example system development consisted of developing a system with mobile applications, developed by us, to simulate a simulated mobile health service to be employed by patients and healthcare personnel in prevention and treatment of a given chronic disease. The rationale behind the development was to obtain more in-depth and descriptive data from a developer's point of view, regarding the technical structure, documentation, and the interconnection of the developed application within a simulated ecosystem scenario(addressing RQ1, RQ2, RQ3 and RQ4). The system was aimed to be as realistic as possible, while at the same time avoiding aspects that were not relevant to our research questions. Only the basic functionality and architecture of the service were to be simulated, and other non-functional requirements such as security and privacy were disregarded as they fell out of the scope of our research.

The generated quantitative data from the development included development time and lines of code, in addition there were taken notes of problems and challenges experienced with each mHealth Toolkit, which was qualitative data.

It was challenging to develop a system from a realistic scenario that could be generalized, and at same time limit the required resources by disregarding certain concerns, to not have the quantitative data being affected by the development of irrelevant functionality.

3.4 Data Analysis

The case study included a wide range of methods for generating data, both qualitative and quantitative data, where the qualitative data was given principal emphasis as "*it's the main type of data generated by case studies*"[44, p.266]. All five of the abovementioned methods generated qualitative data, while quantitative data was generated from observations, questionnaires and example system development. This section describes the overall process of analysing each of the two data types, by identifying common and distinct elements across the various generation methods.

3.4.1 Qualitative Data Analysis

Qualitative data analysis was performed to evaluate the research in light of the stated research questions. All of the qualitative generated data, except for the video and audio recordings from the observations, was textual. It was therefore decided to perform a general approach to analysing the data, with minor variations depending on the method that generated it. The approach consisted of manually exploring the textual data inductively using a form of content analysis to generate themes and explanations based on the relevance to the research questions. The criteria for and degree of relevance of the textual data was determined by the interpreters, and was hence susceptible to subjectivity in spite of attempts made to be unbiased by staying close to the actual data.

Data from documents was analysed to provide technical details for each of the chosen Toolkits, interview data was chosen to be the main source for evaluating unprecedented development experience, the questionnaire data was the main source of established experience, and the remaining data was used as a supplement for evaluating both unprecedented and established experience.

Documents

In the analysis of data generated from documents, documents were regarded as containers of data to be analysed without the concern of the production and exchange of them. A qualitative technique that consisted of an examination of themes covered in a selection of documents was used. The process consisted of manually labeling data into relevant themes and recognizing similarities and differences between them. This process was conducted in an iterative fashion over a period of about one month where each iteration consisted of re-evaluation and re-analysis of the updated data collection before the final collection of relevant themes was established.

Interviews

For the analysis of interview transcripts, a manual approach was applied, based on the "*Five stages of data analysis*" in Pope et al.[2000] framework approach for analysing qualitative data[47]. The transcripts were first browsed and notes of first impressions were taken. Then the transcripts were "*coded*", which involved going through them more meticulously to label significant sentences and sections. A part was deemed significant if it was relevant to the stated research questions, if the interviewee explicitly stated that it was important, or if it had previously been read about something similar in a publication. The common characteristic among these reasons was the fact that we as the interpreters considered them to be important. After the initial coding, the distinctive occurrences of the codes were counted across the set of interviewees(i.e. there could only be one occurrence per participant).

The set of codes were first grouped into three groups related to the interview questions(Toolkit usefulness, positive experience, negative experience). Then the process of eliminating codes was performed by merging similar codes and removing codes that were regarded as unimportant by the criteria to only have one occurrence and not belonging to any other the codes. To provide relevance to the research questions, the initial groups were regrouped into more descriptive and specialized ones, where each group formed a theme that was given a generalized name. Each themes' set of codes was checked for distinct occurrences across participants to give the themes an occurrence value.

Observations, Questionnaires & Example Development

Before performing textual analysis, the video recordings from the observations were watched to produce textual data from the video footage. General notes of each participant's actions during development were written down in the same manner as the written field notes, which together formed the basis for supplementing and checking the validity of the data analysis of interview data. Additionally, any non-relevant questionnaire answers received from experienced developers were discarded.

Textual data generated from observations, questionnaires and example development, was analysed using the same approach. The textual data consisted of notes in the form of sentences, and these notes were reviewed to recognize relevant themes and their occurrences. The themes produced by the interview data analysis were used as the main set of relevant themes.

3.4.2 Quantitative Data Analysis

Observations

The video recordings were also watched to perform quantitative analysis. The quantitative data analysis consisted of measuring each participant's task completion time, the number of tasks completed, and the development time spent IN available documentation versus in IDE. Statistical analysis was performed on the measures in an attempt to describe the relationship between the number of tasks completed and the ratio between time spent in documentation and IDE, by calculating the correlation between the two variables.

Questionnaires

The "D-T Scale"[55] was used for all questions with quantitative answers. The scale is a continuous scale ranging from Terrible(equivalent to value 1) to Delighted(equivalent to value 7). The main reason that the scale was deemed suitable was that "The measure has reasonable reliability, converges with other rating scales and free-response measures"[55]. The data analysis consisted of calculating the arithmetic mean of the answer's equivalent values, per question for each Toolkit in question, to quantify the developer's thoughts and experience.

Example Development

The quantitative data regarding the amount of code lines and the time needed to develop a given functionality or application was measured for development with each of the given Toolkits. The measures were used to compare the developers' development of a more complete application with each toolkit.

Chapter 4

Analysis of mHealth Toolkits Documents

In this chapter, we will present the analasys of mHealth Toolkit documents we had conducted. Based on our preliminary study, our definition of a mHealth Toolkit is a set of tools in the form of libraries and APIs assisting developers in the creation of a mHealth application. These APIs and libraries provide functionality for data storage and access, managing data type permissions, and access to sensor data from a mobile device. As mentioned in section 2.1.2, we found that the market leading mHealth Toolkits are Apple HealthKit, Google Fit and SDH. For each one, the architecture, data structure and storage, and application development process will be elaborated. In addition, actor networks for the mHealth Toolkits will be presented to provide an overview of the ecosystem of actors related to the mHealth Toolkit in question. In the end, the findings will be presented. The findings from this chapter will contribute to answer RQ1.

4.1 Google Fit

In this section, we will describe the Google Fit platform. The focus will be on describing its technical architecture, data models, and application development process. According to its documentation, Google Fit aims to be "*An open platform that lets users control* *their fitness data, developers build smarter apps, and manufacturers focus on creating amazing devices*" and introduce "*New APIs to make building fitness apps and devices easier*". In other words, Google aims to make fitness data more available both to the developers and end users of Google Fit. Google Fit has a development platform and a user platform, consisting of applications that can be used to access and track fitness data.

The development platform consists of a set of APIs and a SDK for mobile development. The APIs support storage, retrieval, and manipulation of data, and are accessible through both the SDK and REST API. The SDK has additional functionalities including Bluetooth data exchange, access to sensor data, and tracking of user activities. Several third-party applications, e.g. Nike Running and Strava, are integrated with Google Fit to utilize its activity tracking functionality. Integration with Google Fit enables third-party applications to interchange and synchronize data via Google's server network.

4.1.1 Technical Architecture

In this section, we will look at the technical architecture of Google Fit, specified in the deployment diagram in Figure 4.1. Artifacts and components crucial to the Toolkit will be described in detail from a technical perspective, focusing on their properties, roles, as well as underlying structures.

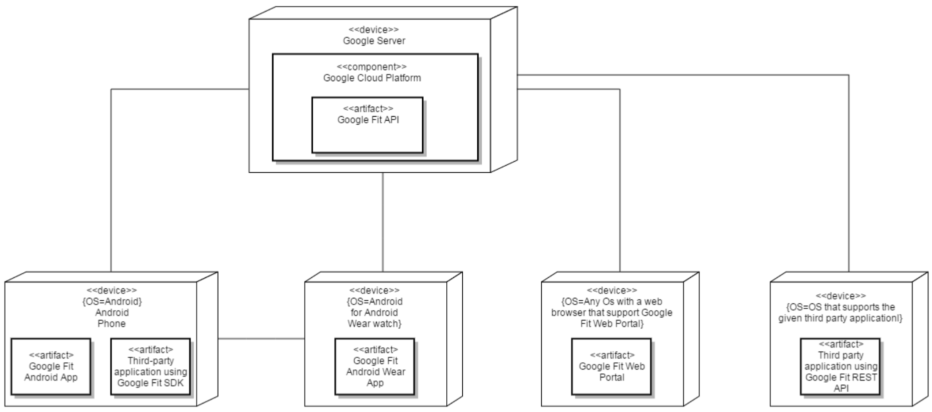


Figure 4.1: Deployment diagram for the Google Fit platform

In Figure 4.1, is a diagram depicting the physical deployment of software components(artifacts) on hardware components(nodes) for a third-party application with Google Fit integration. The boxes represent nodes and the rectangles within boxes the artifacts that are deployed on the given node. As seen in the diagram, is all communication coordinated through the Google Fit API, which is deployed on the Google Cloud Platform and acts as a mediator between the data storage and applications that connect to the API.

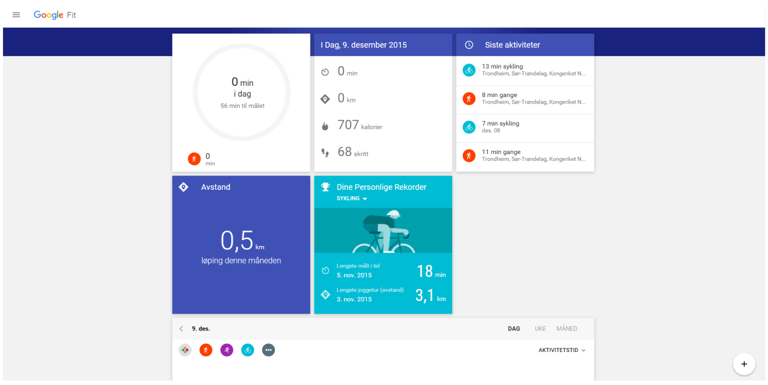


Figure 4.2: Google Fit Web portal

As a part of Google Fit, user applications are provided; a web portal where the users can log in and access their historical fitness data, and applications for Android phones and Android Wear that make it possible to track fitness data utilizing device sensors. The web portal supports devices that have a browser that supports the Google Fit web portal. Figure 4.2 shows a view of the deployment of the web portal in the Chrome browser. The application for Android devices requires Android version 2.3 and above[30]. Figure 4.3 shows an example of deployment on OnePlus One with Android 5.1.1. The application for Android Wear can also be deployed on a smartwatch that supports Android Wear. If a user has the Android Fit application installed on his/her device, the application will automatically track fitness data based on the data stream received from the sen-

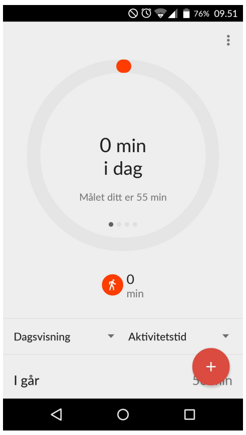


Figure 4.3: Google Fit Android Dashboard App

sors on the device. The application will automatically determine what kind of activity the user is performing and store it in Google Fit. The application also provides the user with a visualization of historical fitness data. Additionally, both the web portal and the applications for Android and Android Wear will present the user's goals based on their progression.

The Google Fit SDK consists of a set of APIs that are accessible after installing it. The *Sensors API* provides sensor data from Android devices and wearables connected to the devices. The *Recording API* provides automated storage of fitness data using subscriptions. The History API provides access to the user's fitness history and functionality to manipulate it. The *Sessions API* provides the option to store fitness data with additional session data. The Bluetooth Low Energy API enables access to Bluetooth low energy devices that are connected to a smartphone and allows them to store data in Google Fit. The *Config API* provides additional settings for Google Fit such as the configuration of custom data types.

The REST API provided by Google Fit is a resource used to access fitness data from the Google Fit fitness store, and it is available to all platforms. The REST API provides functionality for accessing and manipulating datasets and sources, and creation and manipulation of sessions. To connect to the REST API, a developer would have to send a request to Google for authorization to obtain an access token for the information the user has stored in Google Fit. When the developer obtains the access token, the application can make HTTP requests to the REST API to access the information.

4.1.2 Data Structure and Storage

This section will describe the data structure and data storage mechanisms of Google Fit. First, the data structure will be described, then its data types and storage mechanisms.

In the data model, some data types are predefined. The predefined data types are called public data types and include simple data types(See [29] for a complete overview). There are also aggregate public data types that are predefined and consists of aggregates of basic data types. In addition to the public data types Google has created custom data types can be customized by defining data type attributes. With the custom data types, there is little restriction to the types of data that can be stored in Google Fit. Google states in their documentation [29] that a custom data type is only accessible from the application that defined it. If another application were to access the data it would have to request a shareable data type from Google. A shareable data type has the same functionality as a custom data type, but other applications can access the data as well. If another application wants to write to a shareable data type, the developer has to get permission from the creator of the shareable data type.

Google Fit is used for storing and tracking of fitness data. As mentioned in [34], is Google's main focus fitness and nutrition. This statement is backed by the guidelines section of the '*Google Developer Documentation*' for Google Fit. In the section '*Responsible use of Google Fit*' of the documentation[30], Google states the following: "*Do not use Google Fit APIs for non-fitness purposes, such as storing medical or biometric data, selling data, or using data for advertising*"[30]. This statement leads to the conclusion that Google's intention with Google Fit is to store fitness data rather than health data. Storage of health data has previously been attempted by Google with the '*Google Health*' platform[9]. Google Health was depreciated in 2011[9] because "*Google Health isn't having the broad impact that we hoped it would*"[9]. With Google Health, Google wanted to target people who would benefit from a digital health record. The failure of Google Health might be the reason why Google decided not to address healthcare with Google Fit.

Data that applications write to Google Fit is stored in Google's cloud storage. Consequently, the data tracked by a device connected to Google Fit will be accessible to all other devices connected to Google Fit. For example, if a user uses two different smartphones, the data will be synchronized to both devices. The solution increases the data accessibility for developers. Rather than connecting to the application on the device, the developers can access the available APIs. While this form of data storage is great for fitness applications, it might be an issue for applications handling health data. If Google were to change their targeting strategy to include the health segment, it needs to address aspects concerning the platform's data storage and data security.

4.1.3 Actor Network

This section will provide an overview of the different actors that are a part of the Google Fit ecosystem. The overview is represented as an actor-network diagram.

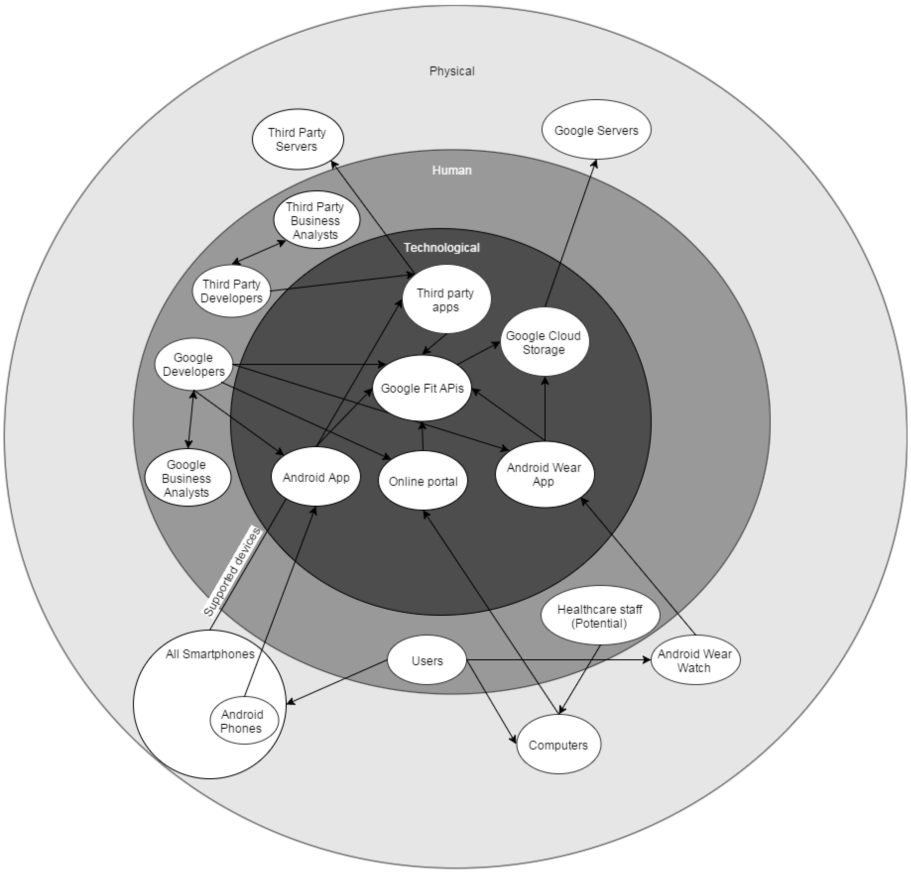


Figure 4.4: Actor-network diagram for the Google Fit platform

Figure 4.4 shows all of the main actors of the Google Fit ecosystem. The Google Fit API is the core of the ecosystem. It contains access points for retrieving and storing data for all users of the system and they connect the various components of the system together. An example scenario is a user accessing the Google Fit Application on his/her smartphone to review his historical step count for the last month. The user acts on the smartphone that acts on the app. The application acts on the Google Fit APIs through an internet connection to retrieve the historical step count. The API for step count then acts on Google Cloud Storage to retrieve the correct data which propagates the act to Google's servers where the actual data is located.

Table 4.1: Description of Actor-network in Figure 4.4.

Actor	Description	Acts on
Technical		
Google Fit Mobile Application	Application that can be used on Android devices which automatically tracks the activity of the user. It can also be used to register fitness data manually and display statistics.	Google Fit APIs
Google Fit Web Portal	Online portal where users can access their fitness history and change their fitness goals.	Google Fit APIs
Google Fit Android Wear Applications	App for smartwatch where you can display your tracked activity. It also makes it possible to track heart rates when a sensor is available on the smart watch.	Google Fit APIs or Android App through Bluetooth low energy
Google Fit APIs	Online access points for retrieving and storing data.	Google Cloud Storage

Continued Description of Actor-network		
Actor	Description	Acts on
Google Cloud Storage	Google's system for accessing cloud storage. Has connection to all of the servers belonging to Google.	Google Servers
Third-party Applications	Different applications that are using the Google Fit APIs to store data, retrieve data or both.	Google Fit APIs, Third-party servers
Human		
Third-party Developers	People using the Google Fit to develop third-party applications for the Google Fit platform.	Third-party App, Google Fit APIs, Third-party Business Analysts
Google Developers	Developers hired by Google in order to improve the API and apps offered by Google	Android App, Online Portal, Android Wear App, Google Fit APIs,Google Business Analysts
Users	People using the applications that are part of the Google Fit Platform	All Smartphones, computers, Android Wear Watch
Healthcare Personnel	Healthcare personnel who potentially can access data stored in Google Fit.	Computers

Continued Description of Actor-network		
Actor	**Description**	**Acts on**
Google Business Analysts	Google's business people, who analyze the business marked and communicate with the developers regarding what should and should not be made.	Google Developers
Third-party Business Analysts	Third-party business people, who analyze the business marked and communicate with the developers regarding what should and should not be made.	Third-party Developers
Physical		
Android Phones	Phones that run the Android operating system.	Android App
Computers	Computers and other devices that can be used in order to access the Google Fit Web Portal.	Google Fit Web Portal
Android Wear Watches	Smartwatches that are running Android and support Android Wear.	Android Wear App
Google servers	Google's server park .	
Third-Party Servers	Servers of third-party applications .	
End of Table		

4.1.4 Application Development Process

In this section, we will describe how developers can use the developer tools to access the functionality of Google Fit. First, the initial setup process will be described, then an example of ways to connect to the framework. The description and examples were collected from the Google Fit documentation[4] and Google's *Github* repository[2].

There are two different use cases for a developer using Google Fit. One is the use of the Android SDK and the other one is the use of Google Fit REST API. To connect to the REST API, a rest service is needed in the application to make requests. For development, Android Studio IDE is required, and Google Services must be included in the application project. For an application to make requests through the REST API or the SDK, the application needs to be authorized with *OAuth 2.0*. When creating a new application, the developer needs to request a *ClientId* for the application, which is used for authorization and to provide appropriate access to the API requests. When the Clientid is obtained, the implementation can start. In order to make calls to the API, one needs to build a *GoogleApiClient*, where one specifies which of the Google Fit APIs that is to be accessed. We performed the connection to Google Fit process in order to investigate the connection process further. The findings will be presented in section 4.4.

The most important access point of the Google Fit APIs is the History API, used to read and write data. The implementation of a connection to the History API includes a call to the function displayed in Listing 4.1. It takes the *mClient*, which is the result of the setup process and a *readRequest* that specifies what kind of data the developer wants to retrieve from the History API, as parameters. The developer can specify in the readRequest what kind of data he/she would like to receive. To get the retrieve the correct data, the developer needs to make an accurate readRequest. The readRequest is similar to an SQL query where the developers can specify different parameters for the readRequest by for example calling the *setTimeRange* function that can be used to retrieve data within a specific time period. The actual call to the API, displayed in the Listing 4.1, is an asynchronous call where the answer from the API call invokes the *onResult* function:

```java
Fitness.HistoryApi.readData(mClient, readRequest).setResultCallback(
    new ResultCallback<DataReadResult>() {
        @Override
        public void onResult(DataReadResult dataReadResult) {
            //do something with the result
        }
    }
);
```

Listing 4.1: Call to the Google Fit History API

The process of connecting to the other APIs in the SDK is similar to connecting to the History API. For the Sensor API, you pass a *dataSourceRequest* with the *findDataSources* function. For the Recording API, you call the subscribe function with the data type you want to monitor through the Recording API. For the Sessions API a created session for a given fitness activity is passed with the functions *startSession* and *stopSession.* For the Bluetooth Low Energy API, you call the *startBleScan* to scan for devices and *claimBleDevice* and *unclaimBleDevice* to claim and unclaim devices.

As mentioned previously, a developer can create customized data types with the Config API, and an example is shown in Listing 4.2 on the next page. In the example, a data type for blood glucose levels is created. First, you create a *DataTypeCreateRequest* that can be set with the name and the fields of the custom data type in question. Then the *createCustomDataType* function of the Config API is called, which is asynchronous. The response from the API invokes the result callback function. If the creation succeeds, the *DataType* will be returned in the callback. After configuration of the custom data type, the History API can be used to perform data insertion and retrieval as previously described. Despite the fact that it is possible to create custom data types, Google is targeting fitness data with Google Fit and will therefore not allow applications to create custom data types that include health data[29]. The functionality for health data types exists, but Google will not allow a publication of such an application because it is not in compliance with their terms[30].

```java
private void buildCustomDataType() {
    DataTypeCreateRequest request = new DataTypeCreateRequest.Builder()
        .setName("com.example.petter.fit.blood")
        .addField("Time", Field.FORMAT_STRING)
        .addField("Value", Field.FORMAT_INT32)
        .build();

    PendingResult<DataTypeResult> pendingResult = Fitness.ConfigApi.
     createCustomDataType(mClient, request);
    pendingResult.setResultCallback(
        new ResultCallback<DataTypeResult>() {

            @Override
            public void onResult(DataTypeResult dataTypeResult) {
                DataType customType = dataTypeResult.getDataType();
                insertCustomData(customType);
            }
        }
    );
}
```

Listing 4.2: Code for building a custom data type in Google Fit

4.2 Apple Health

A large amount of fitness and health applications currently exist in the Apple App-Store, which provides the users of the iOS platform with tools to get in shape and to stay healthy. At the Worldwide Developer Conference(WWDC) in 2014, along with the launch of iOS 8 (operating system for Apple mobile devices), Apple introduced HealthKit[19]. HealthKit's core functionality consists of allowing applications that provide fitness and health services to share their data with the "*Health*" application and each other.

By introducing an own Health app, all of a user's health data can be stored in a centralized and secure location, and it provides users with the choice of determining what data to share with a particular application. HealthKit enables users to combine data from several applications, while at the same time allowing hardware manufacturers to create companion applications for their products(e.g. blood glucose measurement devices and heart-rate monitors) and provide this data to both users and developers. Ryan Faas, a reporter for CITEworld writes: "*Like Samsung, Apple is building a platform rather than a complete health solution. Much as with its new home automation platform, Apple isn't trying to build an overarching Apple solution so much as its providing developers and hardware makers with a way for their new and existing products to exchange data*"[14]. What this means is that with HealthKit, Apple provides a platform for others to build on top of and showcase their products through the AppStore. In the sections below we will go into detail on HealthKit in terms of technical architecture, data structure and its application development process.

4.2.1 Technical Architecture

In this section, we will look at the architecture of a system that implements HealthKit, specified in Figure 4.5. Artifacts and components crucial to the Toolkit will be described in detail from a technical perspective, focusing on their properties, roles, as well as underlying structures that they conform to. A deployment diagram for an application, implementing the Toolkit, consisting of several artifacts. The boxes represent physical devices that consist of different components and artifacts relevant to the application. The most important of the boxes is the lower middle one, representing the iPhone de-

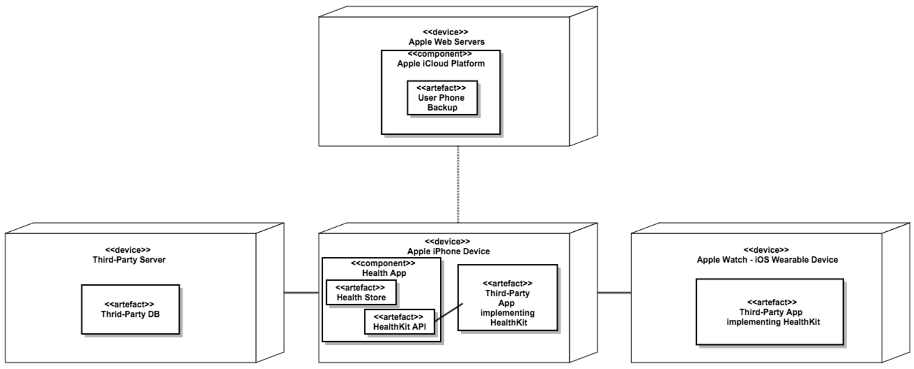


Figure 4.5: Diagram of physical deployment of artifacts on nodes for the Apple HealthKit

vice with its deployed artifacts. All artifacts directly related to the Toolkit are deployed on the iPhone because HealthKit and its functionalities are a part of the iOS SDK. The box on the left portraits a third-party server that may or may not be involved(depending on whether the developer chooses to implement his/her protocol for cloud storage of data).

Health App

In this section we will look into one of the HealthKit's artifacts, the Health App, often referred to as just Health. Health is an application that comes bundled with version iOS 8 and iOS 9, and its functionality within HealthKit is to work as a "command center" for a user's fitness and health data. In other words, data collected from Health App and other third-party applications that integrate HealthKit is administered by the application(Health App is shown in Figure 4.6). Health gives the user the option to either permit or reject third-party access to the "Health Store" that acts as a vault containing the user's personal health data, as described by Apple[20].

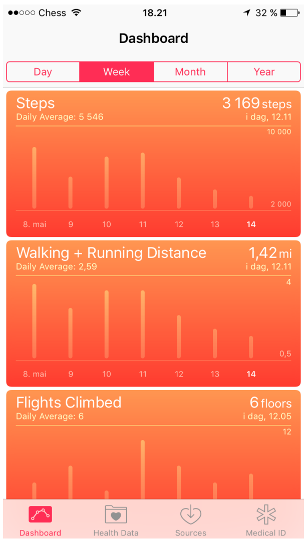
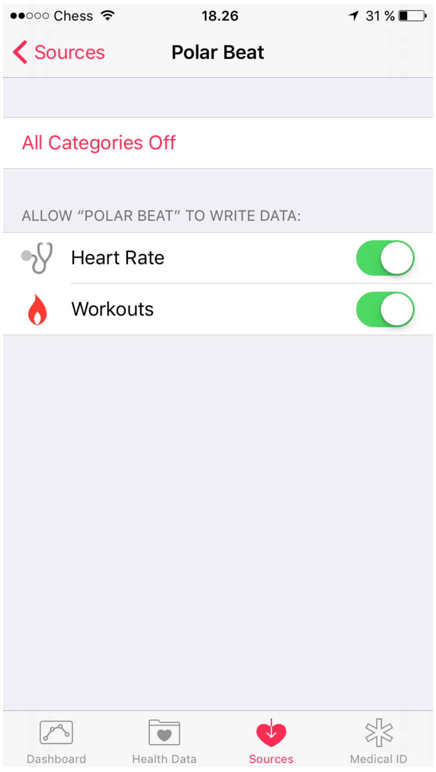


Figure 4.6: Apple Health App Dashboard


●●○○○ Chess 18.26 31 %

Sources — Polar Beat

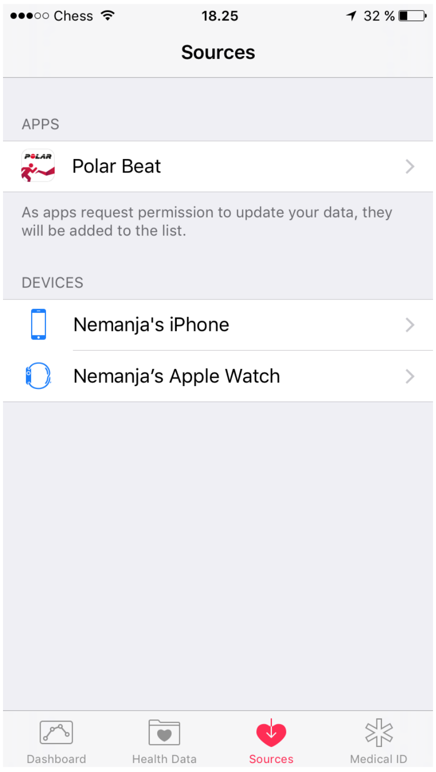
All Categories Off

ALLOW "POLAR BEAT" TO WRITE DATA:

Heart Rate

Workouts

(a) Permitted Data Types for an app


●●●○○ Chess 18.25 32 %

Sources

APPS

Polar Beat

As apps request permission to update your data, they will be added to the list.

DEVICES

Nemanja's iPhone

Nemanja's Apple Watch

(b) Data sources in HealthKit

Figure 4.7: Examples from HealthApp

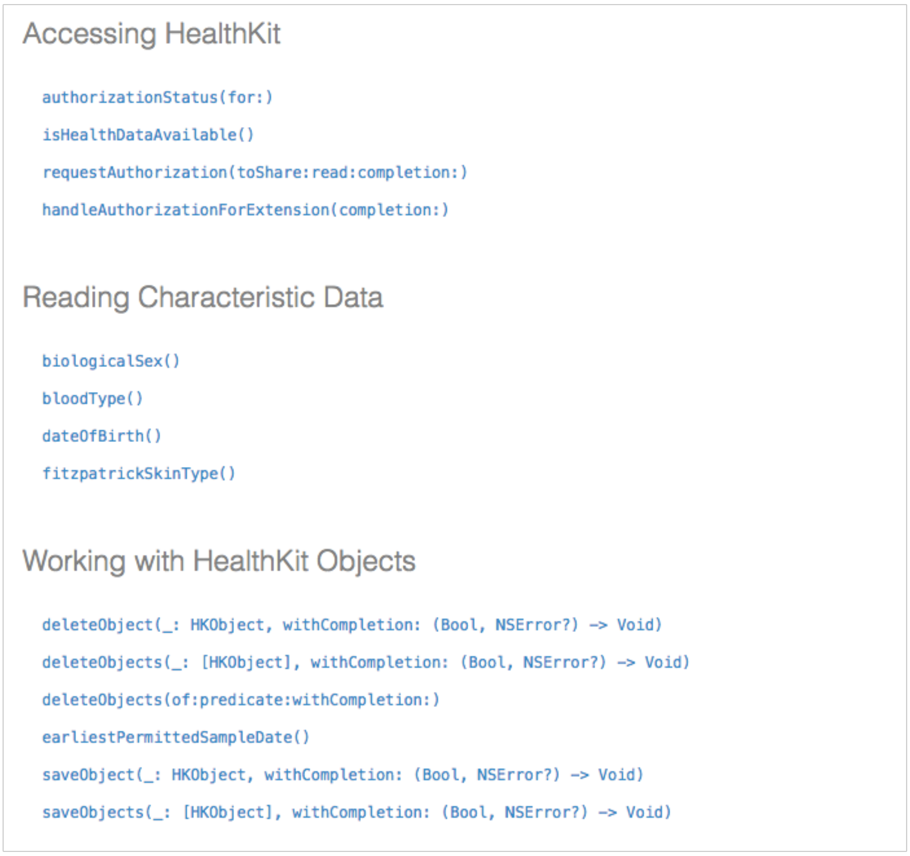
Figure 4.7a shows the screen where the user is asked
to grant an application permission to certain data. Also, the application presents a visualization of data gathered by all of the user's applications. Figure 4.7b shows the data sources in the form of applications and devices.

In addition to displaying data and administering permissions, the Health app also provides a user with the opportunity to enter data manually, as well as to export it, e.g. when switching to a new phone. Exporting the data results in a zipped file containing the user's raw data information in the XML format.

HealthKit API

HealthKit is a platform that lets a developer store and retrieve fitness and health data in/from a device. This section describes the technical aspect of the platform, aimed at providing developers with a more in-depth overview of the HealthKit API. The actual API is embedded within iOS since the launch of HealthKit(in version 8). The API is accessed by importing the HealthKit package, and it is represented in the form of functions of a class within the HealthKit package called HKHealthStore. Functions are analogue to API reference points that access the "*vault*" in the Health app. Functions to request access, store and retrieve data are available to developers, in addition to functions regarding accessibility of HealthKit (as it is only available on iPhones).

The categories of functions can be seen in Figure 4.8 below as "*Accessing HealthKit*" , "*Reading Characteristics Data*", "*Working with HealthKit Objects*", "*Working with Workouts*" and "*Querying HealthKit Data*". The functions belonging to each category can be accessed through Apple's Integrated Development Environment (IDE), Xcode, when creating applications using HealthKit. The application development process will be explained in further detail in section 4.2.4. The API is restricted to the iOS SDK, i.e. the platform only targets users who have an Apple iPhone (and with the recent release of watchOS 2, Apple iWatch), restricting the overall user base drastically.



Accessing HealthKit

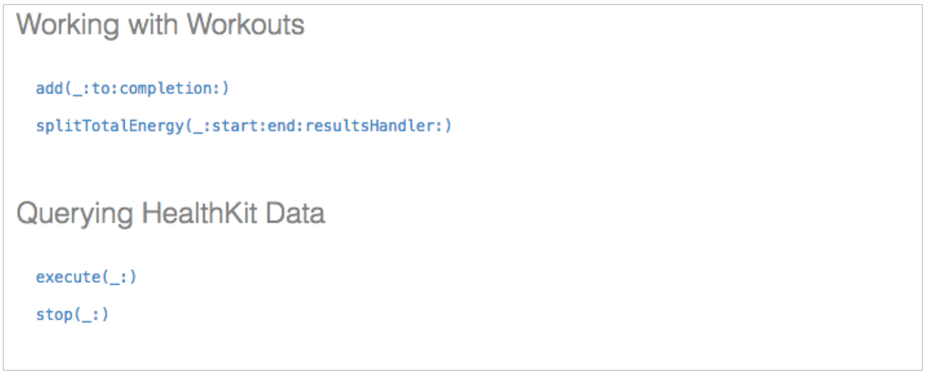
```
authorizationStatus(for:)
isHealthDataAvailable()
requestAuthorization(toShare:read:completion:)
handleAuthorizationForExtension(completion:)
```

Reading Characteristic Data

```
biologicalSex()
bloodType()
dateOfBirth()
fitzpatrickSkinType()
```

Working with HealthKit Objects

```
deleteObject(_: HKObject, withCompletion: (Bool, NSError?) -> Void)
deleteObjects(_: [HKObject], withCompletion: (Bool, NSError?) -> Void)
deleteObjects(of:predicate:withCompletion:)
earliestPermittedSampleDate()
saveObject(_: HKObject, withCompletion: (Bool, NSError?) -> Void)
saveObjects(_: [HKObject], withCompletion: (Bool, NSError?) -> Void)
```



Working with Workouts

```
add(_:to:completion:)
splitTotalEnergy(_:start:end:resultsHandler:)
```

Querying HealthKit Data

```
execute(_:)
stop(_:)
```

Figure 4.8: HealthKit API functions - From HealthKit Reference Framework[6]

4.2.2 Data structure

This section will encompass a description of HealthKit's data structure, covering class hierarchy, data types, as well as the form and structure of the actual storing mechanism(the centralized database). Before exploring the data structure in depth, there are certain aspects that must be addressed. Within HealthKit, data is stored in the form of objects in a local single instance encrypted object-oriented database (OODB). What this entails is that the structure for data storage contains objects with corresponding functions, predefined schemas and inheritance hierarchies; in contrast to traditional databases where data is stored in predefined tables. The class used to represent such objects is called *HKObject*.

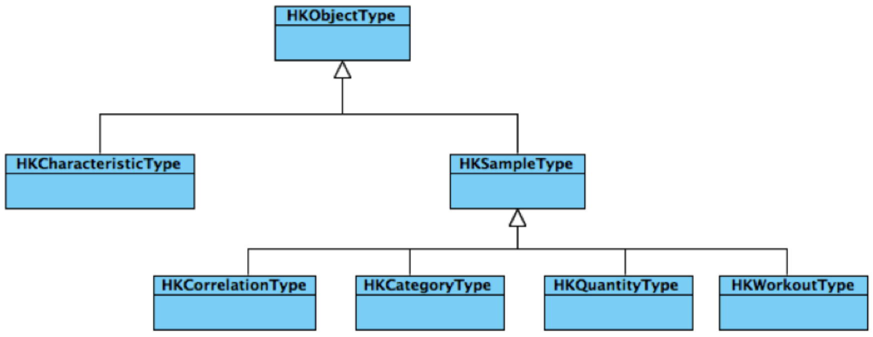


Figure 4.9: Data Type Hierarchy in HealthKit

A stored object conforms to one of 70+ predefined data types available in HealthKit's *HKObjectTypes*, which are instances belonging to one of five data types. In order to explain this in further detail, we need to examine the Figure 4.9 above, which illustrates the hierarchy of data types. At the top we see the *HKObjectType* class, with two child classes, namely *HKCharacteristicsType* and *HKSampleType*. Within *HKCharacteristicsType* we find the data pertaining to characteristics of the user that are constant, e.g. date of birth and blood type.

HKSampleType is again divided into subcategories that relate to the actual data types. These categories are determined by the nature of the measured data. The nature of data for an *HKQuantityType* for example, is quantity, where a data sample is represented by a numerical value and a unit used to measure blood glucose levels. A data sample of *HKCategoryType* interprets values based on predefined enums or categories.

HealthKit does not currently allow users to create custom data types, however due to the mechanism used to categorize data into five larger types, Apple has a possibility to open creation of custom subtypes to developers, such as the 70+ preexisting ones.

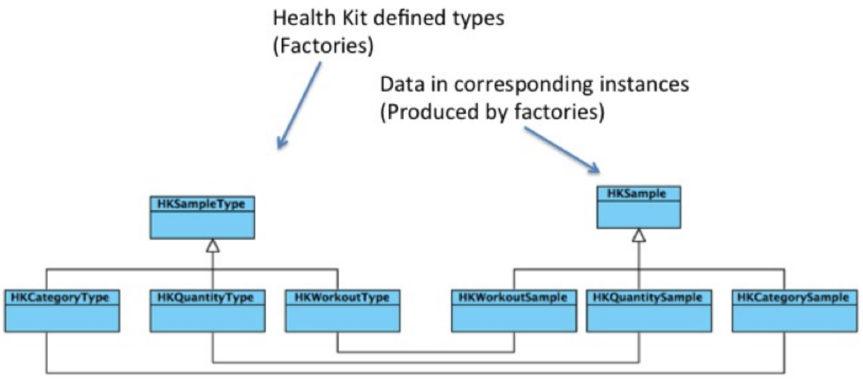


Figure 4.10: Relation between data types and data samples within HealthKit class hierarchy

The structural choices of data and the data types are reflected in the class structure, which is illustrated in Figure 4.10. The class of each data type has a corresponding class used to define a subclass of *HKSample* objects that contain the data values. Now that the overall structure has been covered, we will look at what a data sample consists of: Each sample requires a start and an end date in order to map when it was measured. Additionally, it needs a sample type to determine its sample kind. Based on the sample type, a sample contains either a value and a type identifier, or a quantity and type identifier. A value is raw numerical data evaluated in context of the type identifier, while a quantity, represented by the class *HKQuantity*, is formed by a measured numerical data and a unit to represent the data, e.g. meters or deciliters. A unit is also represented by its own class called *HKUnit*. Common for all samples is the device parameter, which contains an alphanumerical string called a *UUID*, which is a unique identifier for data source(phone, external sensor or another application).

Looking back at the start of this section, the HealthStore was defined as a single instance encrypted database, meaning that the applications accessing the HealthStore require authorization for each class of data readings. *"The result is that you consolidate all your health related data in one place, and can selectively give access to subsets of it to other applications on your iPhone handset (and to revoke permissions at any time)*" [54]. Due to this structure of having encrypted data located on the users device without external backup, Apple has instead of trying to define a protocol for data exchange, allowed developers to implement such functionality, guided by restrictions and regulations. The entire structure is possible to export in a XML format, alternatively using services such as Open mHealth, that converts the exported data into a JSON format[50]. Due to data being accessed through a centralized encrypted database, we had a concern regarding the data access and data storage between multiple applications in realtime. An interesting discovery made when going through various community sites and forums was that certain users complained about data loss and slow data access when accessing large amounts of data and trying to access it simultaneously from another application[12, 37].

4.2.3 Actor Network

This section will provide an overview of the different actors that are a part of an ecosystem where applications and technology are used with the platform. The overview is represented as an actor-network diagram.

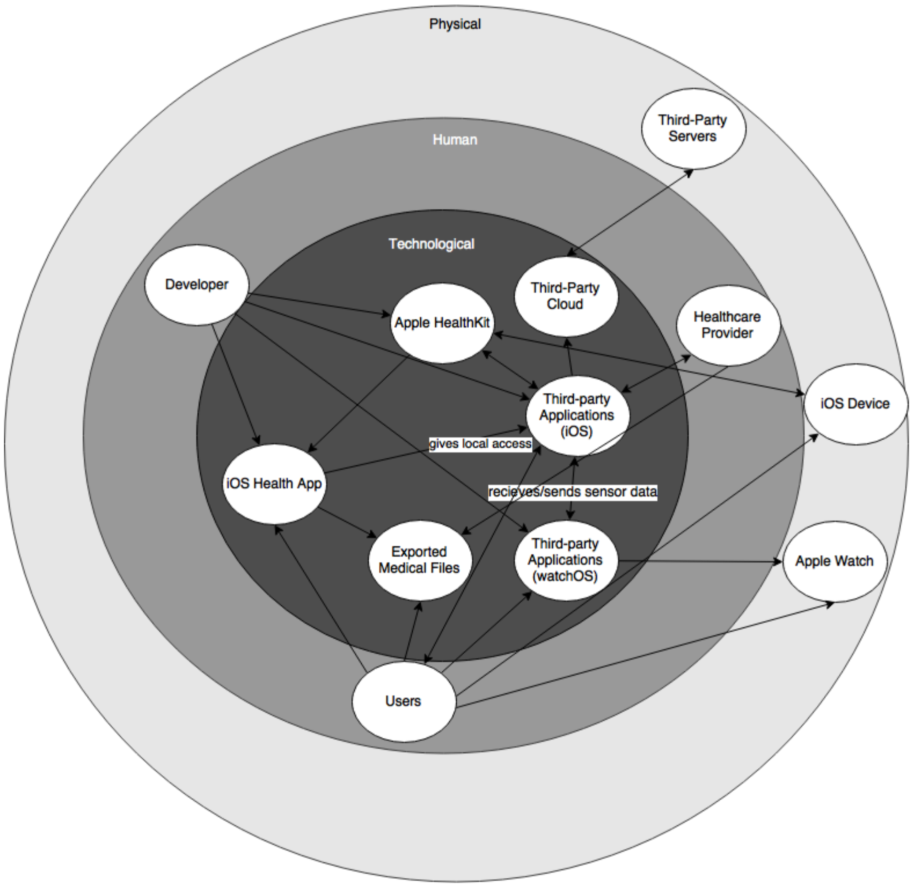


Figure 4.11: Actor-network diagram for the Apple HealthKit platform

Figure 4.11 shows the main actors within the ecosystem of the Apple HealthKit platform. The diagram shows that the Health App, as well as the HealthKit, plays a significant role in the ecosystem. The application is built with HealthKit, and the Health App handles the database and the permissions manager of the platform. An example scenario would be a user accessing the Health App from an iPhone to review his/her measured blood glucose levels from the past two days. The user would act on the application, which acts on the HealthKit by retrieving the data. Moreover, to store this data, the heart rate would have had to come from either an Apple Watch or another third-party application. As seen in Figure 4.11, the cloud option has to be implemented by the developers or some other third-party, as the data is only stored locally on the device.

Table 4.2: Description of Actor-network in Figure 4.11.

Actor	Description	Acts on
Technical		
Apple HealthKit	Framework part of the iOS SDK.	Third-party application, Health App, iOS Device
iOS Health App	Provides the user with a way to set permissions regarding data types. Views and exports health data	Third-party application, Exported medical files, Users, Apple HealthKit
Third-party applications (iOS)	Range of applications developed to integrate and take advantage of the functionality of HealthKit and wearable device to provide end users and healthcare staff with useful information.	Third-party watch application, Third-party Cloud, HealthCare Provider, Apple HealthKit, Users
Third-party applications (watchOS)	Applications developed for use on wearable devices to provide functionality enhancing the device's sensor technology.	Apple Watch, Users, Third-party iOS Application

Continued Description of Actor-network		
Actor	Description	Acts on
Human		
Developers	People responsible for developing an application using HealthKit framework.	Third-party applications (iOS and watchOS),Apple HealthKit, iOS Health App
Users	End users using one or more of applications provided in the network by HealthKit, including patients.	Apple Watch, iOS Device, iOS Health App, Third-party applications (iOS and watchOS), Exported Medical Files
Healthcare Provider	Healthcare staff who can access third-party application or exported medical files	Third-party application (iOS), Exported Medical Files
Physical		
iOS devices	iPhones running iOS 8 or higher.	Third-party Application (iOS), Users
Apple Watch	Wearable technology running watchOS, with a special application connected to accompanying third-party application on the phone.	Third-party applications (watchOS), Users
Third-party Servers	Servers for storing data from third-party applications, managed by third-party. developers/businesses	Third-party cloud
End of Table		

4.2.4 Application Development Process

To provide insight into the development process of an application using Apple HealthKit, this section will describe the various steps a developer goes through in a specific context, e.g. the development of an diabetes journal app. The first part of the process is to acquire the correct tools, in other words, Apple's IDE Xcode with the newest iOS SDK (currently iOS 9.3.2).

Once set up, the next step to creating an application with HealthKit is to check for availability and instantiate a store object instance. Seeing as how the HealthStore is a centralized database on the device, the HealthStore is a Singleton pattern instance, as instantiation of only one store is allowed. In order to read or write a certain type of data, HealthKit requires that application developers request authorization from the user to read or write data types, sometimes requiring both due to privacy concerns. Requests are implemented by calling the *requestAuthorizationToShareTypes* function on the HealthStore object. In the context of a diabetes journal, we would request to both read and write an *HKQuantityType* which has a type identifier of *HKQuantityTypeIdentifierBloodGlucose*. What this indicates is that we wish to have access to a *HKQuantityType* of type Blood Glucose.

Subsequently one can now implement the required functions needed for the diabetes journal. A useful feature of our application would be to store our blood glucose levels after measuring them. Storage of data in HealthKit is done in an object oriented fashion, meaning that data is stored in the database in the form of objects. HealthStore provides a function called *saveObject*, that takes an *HKSample* (or a subclass of it) and a completion function. In the case of the diabetes journal we measure blood glucose, therefore the *HKSample* subclass *HKQuantitySample* would be used. In order to create a HKQuantitySample one needs to use the *init* constructor which takes a start and end date of the sample, a specific *quantityType* (as the one created when requesting authorization in previous paragraph), a quantity represented by *HKQuantity* which takes a value and a unit (represented by *HKUnit*).

To retrieve data, HealthKit uses the notion of queries. Using a subclass of the abstract class *HKQuery*, we construct a query to retrieve exactly the data we require. From our use case, the diabetes journal app, we might want to retrieve the stored blood glucose data for one week at a time. To implement this feature, we can use a *HKSampleQuery*, which in the init constructor takes in an *HKSampleType* (in this case equivalent to the quantity type from the request process, as HKSampleType is a parent class of *HKQuantityType* class), a predicate, numerical limit of number of objects to be retrieved and a sort descriptor. When the query is created, the *executeQuery* function is called on the store object, passing the created query as the parameter.

Both saveObject and executeQuery have completion parameters, where a developer handles the further process of app and data.

4.3 Samsung Digital Health

In this section, we will provide an overview of the Samsung Digital Health(SDH) platform including its technical architecture, data structure, ecosystem and developers support.

Samsung's website states the following about what SDH's contributions are: "*SDH helps application developers and healthcare providers thrive in an open environment that connects sensors, devices and partner services. Users can experience various fitness and health services through Samsung Digital Health*"[13]. In others words, the aim is to provide developers and healthcare providers with better tools to support users in gaining improved knowledge of their health, and also designing goals and guide them towards reaching them. In the following sections, the different aspects that concern Samsung's stated contributions of SDH will be addressed.

4.3.1 Technical Architecture

In this section, we will look at the overall architecture of a system that implements SDH, further specified in the deployment diagram in Figure 4.12 below. Artifacts and components of such a system will be described from a technical perspective, focusing on their properties and functionality. This section also provides an overview of the different actors that are a part of an ecosystem where applications and technology are used with the platform. The overview is represented as an actor-network diagram.

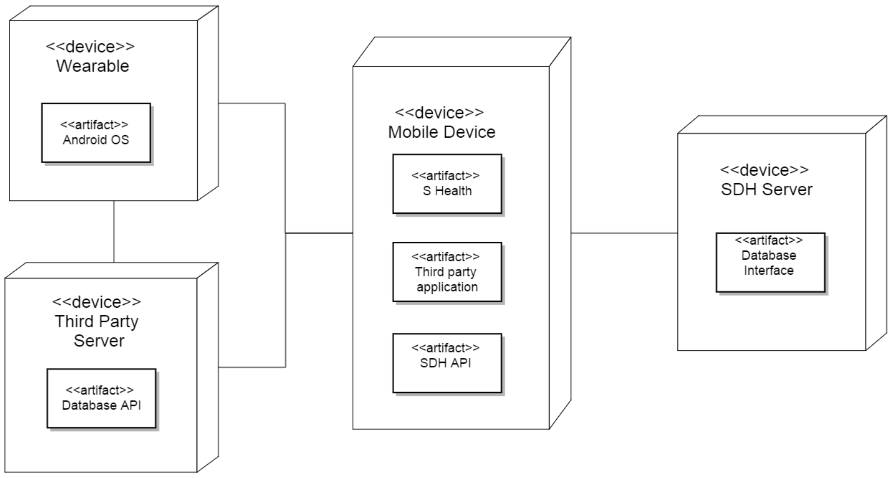


Figure 4.12: Diagram of physical deployment of artifacts on nodes for the SDH platform

Figure 4.12 depicts a diagram of the physical deployment of software components(artifacts) on hardware components(nodes) for a system where a third-party application is integrated with SDH. The boxes represent nodes and the rectangles within boxes the artifacts that are deployed on the given node. Wearable devices run Android OS and can interconnect with both a mobile computing device and a third-party server. The third-party server is deployed by a database with an API enabling data access. A mobile device is deployed by the platform's general purpose application S Health and an API, plus a third-party application. Artifacts deployed on the mobile device is the centerpiece that acts as a mediator between Samsung's and the third-party's application, wearables and servers.

SDH SDK

The SDH software evelopment kit(SDK) is a tool for enabling data access and exchange between Samsung's own mobile application *S Health* and third-party applications. The SDK is composed of two parts, namely the *Health Data Package* and the *S Health Service Package*. The two packages are not dependent on each other, and can be used concurrently.

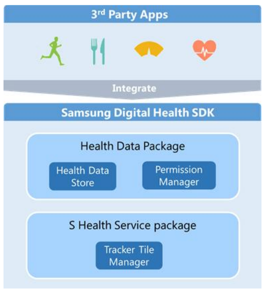


Figure 4.13: SDH's packages

S Health

S Health is an application available for devices accessible via Google Play Store. With S Health, an individual can track everyday activities, get coaching to reach goals, and download training programs to improve fitness. It enables measuring and understanding of personal fitness and health by tracking movement and collecting samples from built-in sensors in devices and wearables. As of the release of v.4.5 it is no longer only compatible with Samsung's devices, but also all non-Samsung Android devices[11] (Requires v4.4. Kitkat or above).

Health Data Package

The Health Data Package is a package with functionality for storing, accessing and sharing data in a secure manner. Data is stored in a store that keeps user data collected by S Health secure, and can be accessed through the provided APIs. The package also includes an interface for handling control of user permissions. It provides a simple way for users to control what data they want to share, and without such defined permissions, a third-party application will not be granted access to the data store.

S Health Service Package

While the previous package addresses access and storage of data, the S Health Service Package addresses data presentation. The basic building blocks of S Health are the Tracker and the Tracker Tile that provide categorization and visualization of data. A Tracker monitors or measures a given data type and a Tracker Tile is a box visualizing the data type. The package features a Tracker Tile manager that lets third-party developers implement support for posting or removing a Tracker Tile in S Health. A developer can, for example, define own Tracker Tiles to support a defined custom data type.

4.3.2 Data Structure

SDK does not provide an API to access data that is externally stored, e.g. in cloud storage, it is only possible for the application to read or write data that is stored by the S Health application on the same device. SDH does provide persistent storage of the collected data, which is implemented with options to synchronize data to a remote storage unit where the data is linked to a registered account, but as mentioned does Samsung not provide an API to access such data. This means that the SDK does not facilitate data exchange directly with applications and vendors, but data has to be accessed through the S Health application that acts as a hub that restricts data flow and access. From a developer's point of view this is a noticeable limitation of the transfer and sharing of health data collected by the applications, and as a consequence all vendors that want to utilize SDH's functionality have to succumb to its standard of storage, access and visualization of data with all the restriction and limitations that follows.

To store data in the store, the data has to be defined as a type of health data, and can either be stored as a predefined type(e.g. blood glucose or steps) or as a custom data type defined and customized by the third-party developer. A custom type provides properties not covered by the predefined types and can be defined with all new properties or with importing existing type. All data types are normalized according to units in the "*International System of Unit*", and data inserted to store must hence be unified accordingly.

4.3.3 Actor Network

This section will provide an overview of the different actors that are a part of an ecosystem where applications and technology are used with the SDK platform. The overview is represented as an actor-network diagram.

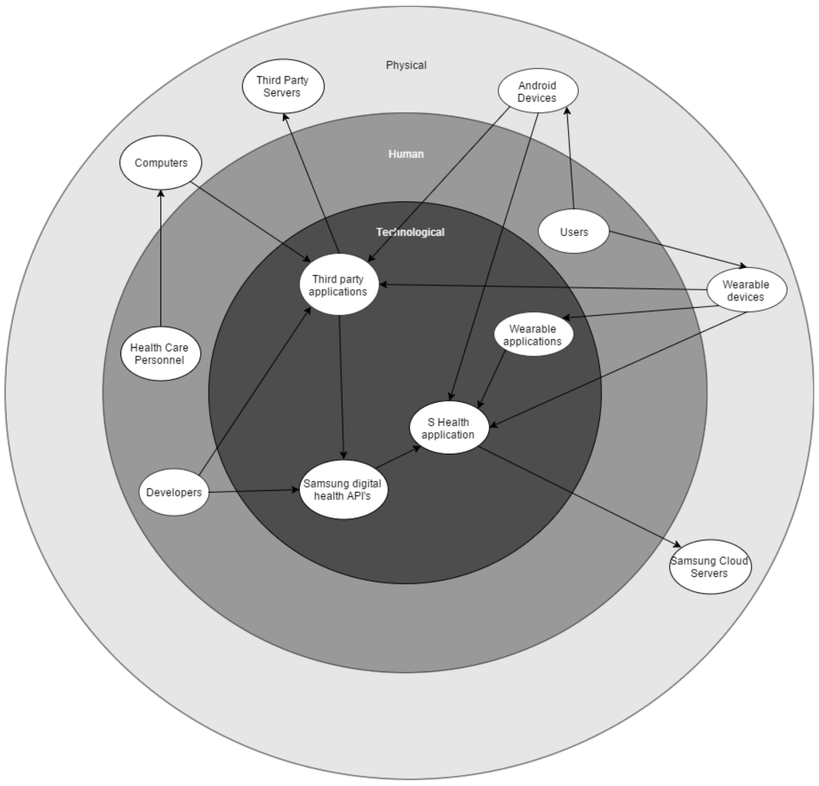


Figure 4.14: Actor-network diagram for the SDH platform

Figure 4.14 shows an overview of the main actors involved in the ecosystem of the SDH platform and describes how the different actors affect each other. The number of arrows in the diagram indicate that the actors with the most relations in the ecosystem are Technical ones: the S Health application and third-party applications that other actors depend on to interconnect. The S Health application acts as a mediator between third-party applications with their respective dependants and the store of collected user data. It handles permissions, retrieval, and storage of such data.

Table 4.3: Description of Actor-network in Figure 4.14.

Actor	Description	Acts on
Technical		
S Health application	Application for use on Android devices which automatically tracks data from sensors or manually by users, and visualizes data in a beneficial way for the user.	Samsung cloud servers
SDH APIs	Access points for accessing and read/write data from/to S Health.	S Health Application
Third-party applications	Range of applications developed to integrate and take advantage of the functionality of the SDH platform and wearable device to provide end users and healthcare staff with useful information.	SDH APIs
Wearable applications	Applications developed for use on wearable devices to provide functionality enhancing the device's sensor technology.	S Health

Continued Description of Actor-network

Actor	Description	Acts on
Human		
Developers	Persons developing applications for the SDH platform.	Third-party applications, SDH APIs
Users	End users using one or more applications provided in the network by the SDH platform, including patients.	Wearable devices, Android devices
Healthcare Personnel	Healthcare staff who can access applications that accesses, enhances or visualizes data from S Health or the SDH platform.	Computers
Physical		
Android devices	Devices that are running the Android OS.	S Health application, Third-party applications
Wearable devices	Wearable technology running Android OS. Needs to be a registered partner accessory with Samsung to act on S Health directly.	Wearable applications, Third-party applications, S Health application
Computers	Computers and other devices that can be used in order to access third-party applications.	Third-party applications
Samsung cloud servers	Samsung's server park.	

Continued Description of Actor-network		
Actor	Description	Acts on
Third-party Servers	Servers for storing data from applications, managed by . developers/businesses.	
End of Table		

4.3.4 Application Development Process

On a general level, the official development process of applications with SDH consists of the following steps:

1. Download the SDH SDK
2. Integrate the SDH SDK into your application and test with S Health or SDK's tools.
3. Publish the application integrated with the SDK in Google Play.
4. Apply for Partner Apps by filling in the request form.

As described in section 4.3.1, there are two packages offered by the SDK. Out of the two, only the Health Data Package was focused on in this research, because the two packages can be integrated independently and because integration of the S Health Service Package was not deemed relevant to the purpose of this research. The SDK Health Data package consists of an API with a range of classes to be employed, but here's a closer look at its most important classes along with a short description of their functionality:

- HealthConstants: Defines constants for predefined and custom data types.
- HealthData: Used to manage data in the form of objects.
- HealthStore: Provides a connection to the health data store to query data.
- HealthPermissionManager: Handles requests for permission from end user to read and/or write specific data types.
- HealthResultHolder: Interface representing the result of invoking function.
- HealthResolver: Used to insert, read, update, delete data, and provide some aggregate functions.
- HealthObserver: Implementation of the observer pattern, where observer receives a notification when subject data is changed.

The steps to access data along with the main APIs needed for each step is depicted in Figure 4.15 below. An application needs to initialize the health data service, connect to the health data store, and acquire data permission(s), before being able to query the health data store's data in S Health.

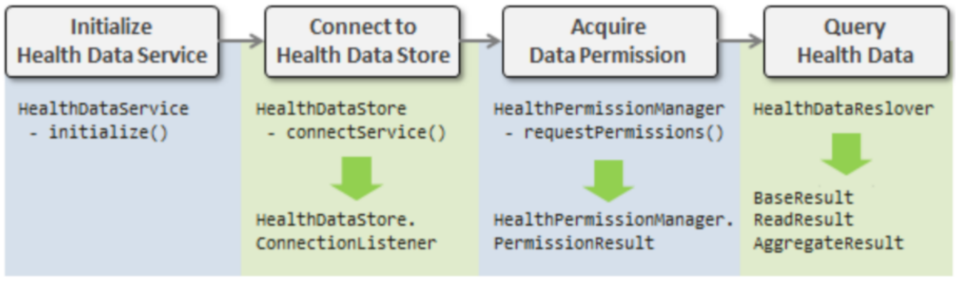

Initialize Health Data Service
Connect to Health Data Store
Acquire Data Permission
Query Health Data
HealthDataService - initialize()
HealthDataStore - connectService()
HealthDataStore. ConnectionListener
HealthPermissionManager - requestPermissions()
HealthPermissionManager. PermissionResult
HealthDataReslover
BaseResult
ReadResult
AggregateResult

Figure 4.15: Flow for data access

4.4 Findings

This section will present findings from Chapter 4. Most of the data presented in Chapter 4 are results of our literature study and did not directly lead to any findings, but the findings presented in this section were attained by analyzing the data and examining the relevant aspects of the mHealth Toolkits.

A finding from Chapter 4 is that Google Fit is different from both Apple HealthKit and Samsung Digital Health with regards to technical architecture, the way data is stored and the types of data that are allowed to be stored. Google Fit employs Google's cloud storage, which means that new data from any application that is connected to the Google Fit API will synchronize with Google Cloud Servers. HealthKit handles storage by connecting to the HealthKit API, and SDH through an interface. The data for both HealthKit and SDH is stored on the phone. In addition, Samsung gives the user the possibility to synchronize the data that is stored in the SDH application to the cloud. This is in contrast to Google Fit since the synchronization with cloud storage is not a required, but rather a choice of the user.

Another finding from Chapter 4 is that the setup process of Google Fit is more extensive than we believed initially. The code provided by Google through the Github repository consists of 150 codelines in total. When we tried to use the provided example code, we were not able to connect without modifying the code due to the use of the *await* function on asynchronous functions, which did not work when the application was deployed on a device running Android. Using the *setResultCallback* function solved this issue. We used six full working hours for the process of setting up the system and to be able to perform calls to the API.

Chapter 5

Development of Example System with mHealth Toolkits

In this chapter, the example system we developed with mHealth Toolkits will be described. First, we will discuss our architectural approach. Secondly, the system will be described. Thirdly the metrics we tracked in the development process will be presented, and lastly, the findings from the system will be presented. The system will include functionality for reading and writing data both internally on device and externally to another device or server. The findings from this chapter will contribute to answer RQ4.

5.1 Architectural Rationale

In this section, we will provide a rationale for the architectural approach of the developed example system. In the research process, we examined the way mHealth Toolkits can be used in a healthcare service to provide value for a doctor when treating a patient. After an initial search for ways in which data from mHealth Toolkits are and can be used in a healthcare service, we found the Mayo Clinic app[18]. The Mayo Clinic app is a patient app that can collect data the user has gathered with Apple HealthKit and send it to the Mayo Clinic servers. The Mayo Clinic app can also push patient data to Apple HealthKit, making the information flow bidirectional. In practice it functions as a portal between the patient and the clinic. The system is described in Figure 5.1 below.

In our research, we also found examples of sensors that can connect to the mHealth Toolkits to provide automated and more accurate data. An example is the iHealth Gluco-Smart application[17] that connects to the iHealth Glucometer[8]. This system lets the user read his/her glucose values and export them to Apple Health where the data can be distributed to the Mayo Clinic app.

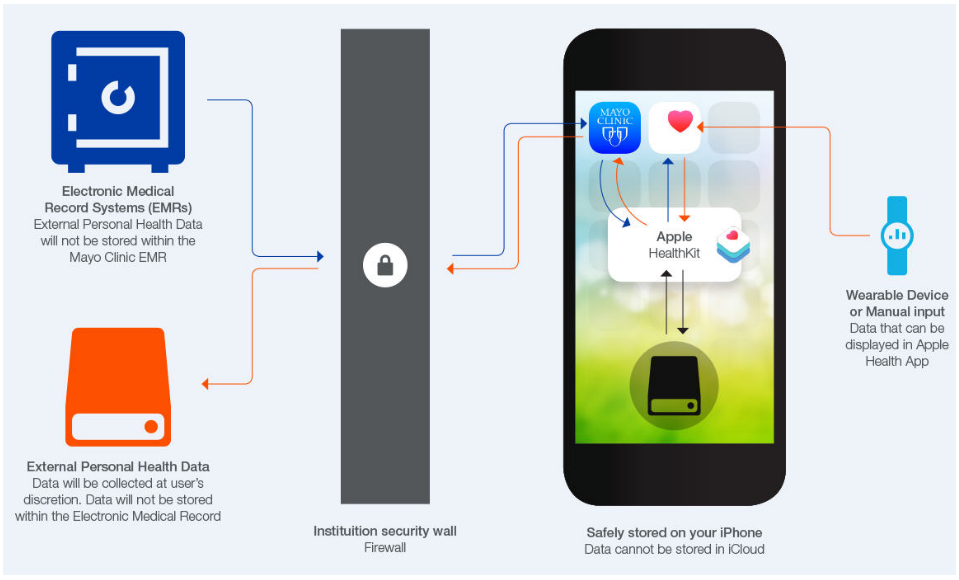

Electronic Medical Record Systems (EMRs)
External Personal Health Data will not be stored within the Mayo Clinic EMR
External Personal Health Data
Data will be collected at user's discretion. Data will not be stored within the Electronic Medical Record
Institution security wall
Firewall
MAYO CLINIC
Apple HealthKit
Safely stored on your iPhone
Data cannot be stored in iCloud
Wearable Device or Manual input
Data that can be displayed in Apple Health App

Figure 5.1: Architecture of the Mayo Clinic app

5.2 System Description

In this section the developed system will be described. The deployment diagram below in Figure 5.2 displays its architecture. The core components of the system are the applications for Android and iOS that includes a Patient-Hospital application(PH-app) and a Blood Glucose application(BG-app) for both Apple HealthKit and SDH, and the hospital API deployed on a Cloud server. Google Fit was excluded from this development due to not being intended to be used to store medical data, as earlier mentioned. Google Fit does not support storing of glucose related data because storing medical data in the

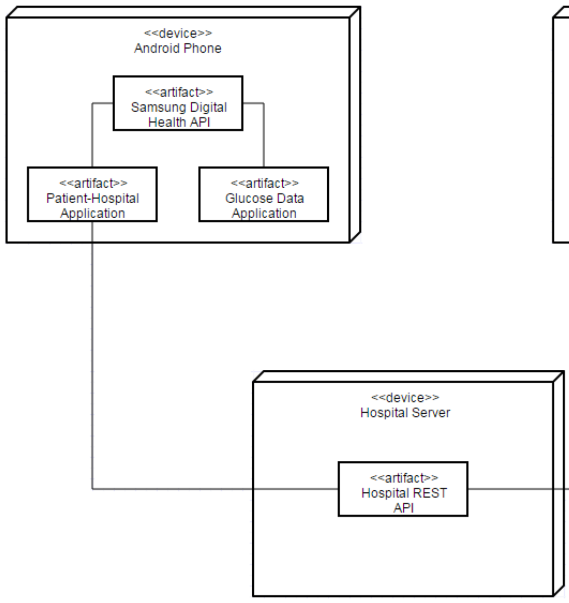


Figure 5.2: Deployment diagram of example system with the BG-app and PH-app for both HealthKit and SDH, and the hospital server

Cloud is not seen as an alternative as long as its cloud storage solution cannot guarantee the safety of the data.

5.2.1 Blood Glucose Application

The main task of the BG-app is to let the user read and write glucose data from and to the data store of the toolkit. The best way to accurately capture the data would be through a blood sugar device containing a Bluetooth sensor that the user connects to the phone. We investigated the possibility of acquiring such a device, but were not able to acquire one due to cost and time limitations. Therefore, we abstracted away the sensor support functionality and gave the responsibility of data input to the user. The purpose of the application was to make a calculator for blood glucose, where the user can input a blood glucose value and how many carbohydrates he/she is going to eat. The calculator will then return a suggested amount of insulin for the user to inject. The fact that the application provides value for the user, namely calculating the suggested amount of insulin, giving the user an incentive to use the application.

Apple HealthKit Solution

An implemented solution employing Apple HealthKit comprised of an iOS application. The *HealthKit* framework was imported and used as a tool to provide data storage and an interface to share the gathered data with other applications, provided that the proper user authorization was granted.

The main purpose of the application is to collect blood glucose measurements and calculate insulin needed. The application follows the same overall process as described in section 4.2.4 with regards to data access, requiring few additions for calculation of insulin. First, the application's GUI presents the user with a field to input a blood glucose measurement and intake of carbohydrates, but before enabling the user to use this functionality the application has to complete the following steps:

- Instantiate a store object using *HKHealthStore*.
- Request the authorization to access and store the needed data type.

The instantiation protocol can be found in Apple HealthKit Reference Framework[6], however, the information for the authorization request was found through an external source [43]. The user can then input the values requires to write the blood glucose value to the store and to calculate the amount of insulin needed. Once the user presses calculate, the inserted value will be passed to an appropriate function within *HealthKitDelegate* (a class made to handle all functions of requesting access, storing and accessing data from HealthStore). The function encapsulates the raw data within a HKQuantitySample and stores it in the Health Store as explained in section 4.2.2.

The application's purpose is to provide functionality for registering samples of specific data, in this case blood glucose data, and abstract it away from the information presented to the user (what amount of insulin the user needs to take). The application serves as an example of how applications that register health data from users, either by the use of sensors or manual input, can be stored within HealthKit and shared with other applications. HealthKit provides the user with a centralized data store containing all of the user's health data, limiting access to applications chosen by the user.

Samsung Digital Health Solution

The solution with SDH was an Android application where the SDK's Health Data Package was employed as a library to provide functionality for data exchange with S Health through its interfaces. As mentioned previously, the application's main purpose is to collect blood glucose sample data and calculate needed insulin. The application follows the same overall process as the one depicted in Figure 4.15 in section 4.3.4 to provide data access, and little other functionality was needed to complete the application. The application's GUI prompts the user to input a blood glucose measurement and intake of carbohydrates, but before enabling the user to use this functionality the application has gone through the three following steps that are required to write data to the store:

- Initialization of *HealthDataService*(service)
- Connecting service to *HealthDataStore*(store) with a *ConnectionListener* interface for handling the callback when application is connected or disconnected with the store.
- The *HealthPermissionManager* handles requesting the user for permissions to read the *BloodGlucose* data type, which is the only permission needed. The permission is requested when the user opens the app, and access to read glucose data from the store is not permitted until the user has granted the request.

The user can then input the values required by the application to write the blood glucose to the store and to calculate the amount of insulin needed. *HealthDataResolver's InsertRequest* interface is used to make a request to insert the blood glucose value, and the value has to be encapsulated in a *HealthData* object to be compatible with the InsertRequest. It is mandatory to set the data type and source device of the *HealthData* object for the store to be able to process and give an identifier to the data.

The purpose of this application is to abstract the functionality for storing samples of a specific data type, in this case blood glucose data, away from other functionality such as data processing. This application is an example of how applications that only collect specific types of data, either by user input or by sensor, can be shared with SDH's S Health application. The purpose of S Health is to collect sample data and to store it in its data store, providing applications with centralized access to all types of health data. Devices that want to use this application are therefore required to have S Health installed.

5.2.2 Patient-Hospital Application

The purpose of the PH-app was to make the user able to transfer data stored in the data store of the given mHealth Toolkit to a healthcare institution. The application enables the user to log in to its patient account and register with a doctor, in the role of a patient. When the user has finished the registration process, the blood glucose data can be transferred to the Hospital server from the data store of the mHealth Toolkit in question. We chose to only let the application export blood glucose data due to time limitations, but the application could potentially transfer all types of health data stored in the data store.

Apple HealthKit Solution

Similar to the application described previously, this section will be looking at a scenario where a healthcare institution wants to create an application that handles the communication and data exchange between the institution and the patient. The application enables exchange of user data between patients and their doctors to aid monitoring and treatment of diseases, e.g. Diabetes. The technical specification of the process involving the external server managed by the healthcare institution and an application utilizing HealthKit will be described in detail in the following paragraphs. The textual description is supplemented with sequence diagrams that depict the technical interaction behind the two main processes and can be viewed in Figure 5.3 and Figure 5.4 below.

In the initialization phase, which is illustrated by the sequence diagram in Figure 5.3, the application will have to perform the same availability check and authorization request as in the previous application(BG-app). Assuming that the device used is an iPhone, the *isHealthKitAvailable* function will first be called and following the user will be prompted to authorize the sharing of data types. Once the user has authorized use

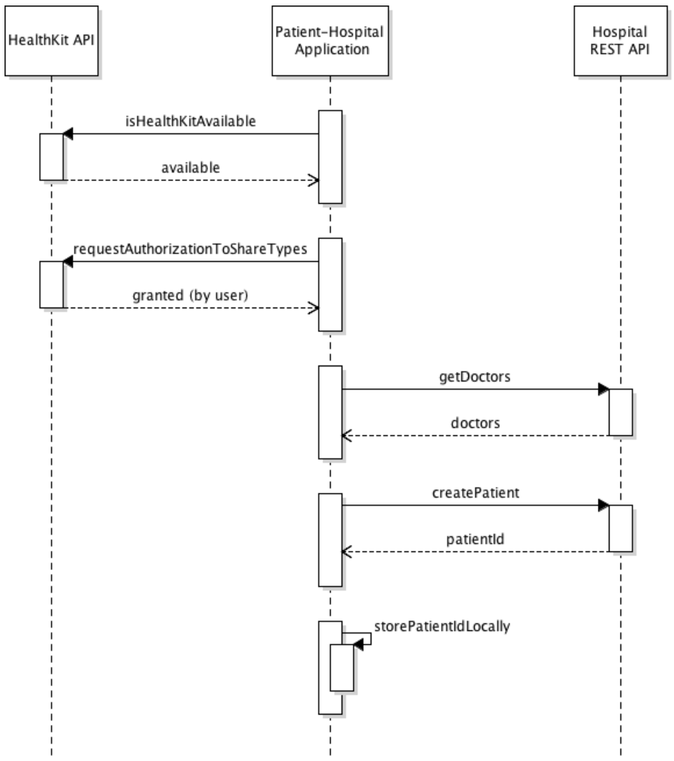


Figure 5.3: Sequence diagram of the process initializing HealthKit and call to API to register patient

of requested data types, the user will then be prompted to select a doctor from a list. In the background the list is retrieved using the hospital REST API access point by sending a *GET request*, and the results are presented to the user in a dropdown list. To create a GET request, we can use a class called *NSURLSession* built in iOS which takes a URL and a completion handler. The completion handler parses the JSON and defines the handling of the retrieved data. Passing the data to the GUI provides the user with options in the dropdown list. Selecting the doctor and pressing next, leads the user to a new screen prompting for a name to be submitted. On submit, a *POST request* is sent to the API to create a patient. The process is finalized when the application receives a response with a valid identifier(patientId) that is stored in the device's local storage.

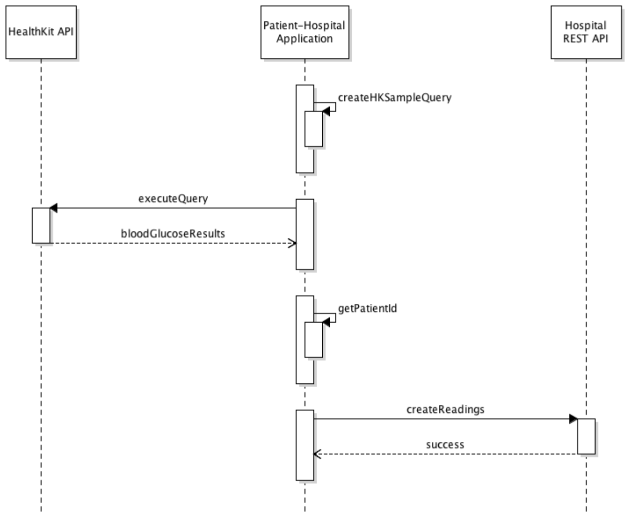


Figure 5.4: Sequence diagram of the process of reading data from store and sending it to server

The second part of the process is depicted as a sequence diagram in Figure 5.4, and it consists of reading the blood glucose value from the *HealthStore* and sending them to the hospital using the server API. As the API does not accept calls from a user that is not registered, this process cannot be performed before the initialization is completed. Reading the blood glucose values from store is done by creating an *HKSampleQuery* that requires a sample type (to which we supply an HKQuantityType with a type identifier for Blood Glucose). Below is the code needed to extract the last weeks values, assuming the user only measures blood glucose once a day.

```
func readGlucoseData(resultsHandler:((query:HKSampleQuery, results:[HKSample]?,
 error:NSError?)->Void)!) {
    let bloodGlucose = HKQuantityType.quantityTypeForIdentifier(
 HKQuantityTypeIdentifierBloodGlucose)
    let query2 = HKSampleQuery.init(sampleType: bloodGlucose!, predicate: nil,
 limit: 7, sortDescriptors: nil, resultsHandler: resultsHandler)
    store?.executeQuery(query2)
}
```

Listing 5.1: Code for extracting the latest 7 blood glucose values

Once the values are extracted, a session is created by using the *NSURLSession.sharedSession* function. Next a JSON object is created with the required parameters: the patientId retrieved from local storage and the results received from the query. After creating this object, a *NSMutableURLRequest* is created, setting the request type to POST and serializing the post parameters using *NSJSONSerialization.* The final step is to perform the actual sending of values to the server, which is done by calling the *dataTaskWithRequest* function on the session object.

As mentioned, this application only enables reading and sending of blood glucose values measured in the last few week, but it is possible to provide additional functionality due to the application's modularity. If for example a different criteria for the values is needed, just the *HKSampleQuery* needs to be modified. To add support for reading or writing of additional data types can be done by modifying the *requestAuthorizationToShareTypes* parameters and the query.

Samsung Digital Health Solution

If a healthcare institution wants to create an application with SDH that includes the functionality described at the beginning of the section, an application is needed that handles communication between the external server and the S Health app regarding the specific patient-hospital relation. A more detailed and technical description of how the main parts of this application were implemented with SDH and the Android platform will now follow. The textual description is supplemented with sequence diagrams that depict the technical interaction behind the two main processes and can be viewed in Figure 5.5 and Figure 5.6 below.

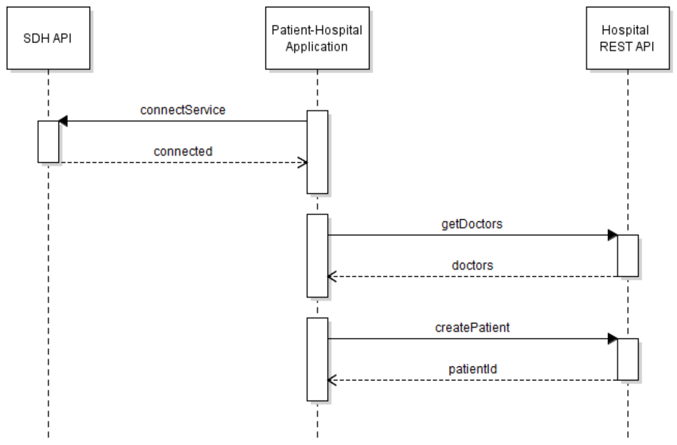


Figure 5.5: Sequence diagram of the process initializing HealthStore and call to API to register patient

This application uses the same interfaces as the BG-app for setting up connection to the store and handling permissions(except for requiring a read permission, not write). The first main process is initializing store and connecting, as well as registering the patient, which is depicted as a sequence diagram in Figure 5.5. Before using the application for data exchange, the user must grant the permission and go through a registering process. Here, the user is prompted with a list of available doctors to choose from, retrieved from the server API via a GET request. By selecting a doctor and entering his/her name the user can press submit, which leads to a POST request being sent to the API to create a patient. The process is finalized when the application receives a response with a valid identifier(patientId) that is stored in the device's local storage.

The second process consists of reading data from the store and sending it to the server, which is depicted as a sequence diagram in Figure 5.6. Here, the user would like to read and send blood glucose values, which can only be performed after the user is registered. This user action is a two-step process: reading values locally from store, and sending values remotely to the server API. The first step consists of using the *ReadRequest* along with a *Filter* to read blood glucose data measured on the current day from the store, set with a *ResultListener* interface to receive the corresponding result asynchronously. The second step is to read the patientId from local storage and enclosing it with the received result value(s) in a POST request to the server API, enabling the hospital to access and review the patient's blood glucose values.

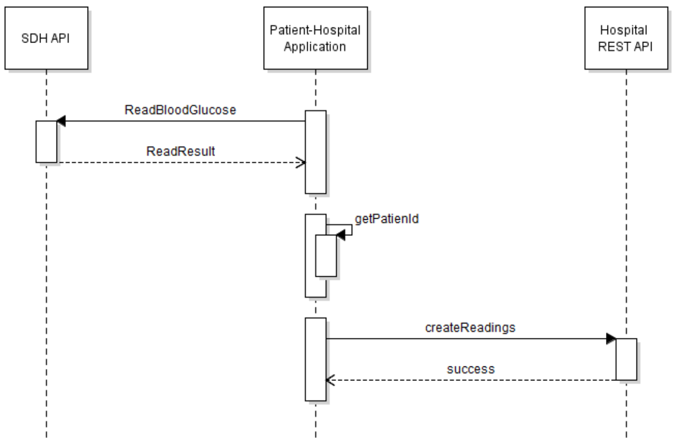


Figure 5.6: Sequence diagram of the process of reading data from store and sending it to server

As mentioned, the application uses the HTTP protocol for its request methods to communicate with the remote server API, more specifically by employing the Java class *HttpURLConnection*. Due to Android's single thread model, each HTTP request is processed separately by an *AsyncTask*(Android class) to avoid blocking the UI-thread. The data format used to exchange data with HTTP is JSON, and the *Jackson library* is used for converting Java objects to/from JSON. JSON combined with Jackson simplifies the data exchange as it provides a structure for serializing objects sent to/from the server API that is written in *Javascript*, and because Jackson maps the two different object representations used by the two artifacts.

This application only enables reading and sending of blood glucose values measured on a specific day, but additional functionality can easily be added due to the application's modularity. If for example a different criteria for the value(s) is needed, just the Filter needs to be modified. To add support for reading or writing of additional data types it can be done by implementing similar functions and HTTP requests, and by adding the correct permissions.

5.2.3 Backend

The Hospital Backend consists of a server with a database and a REST API that the PH-apps can use as an access point. The backend was written in Node.js and deployed on the Heroku platform. We have chosen not to implement any security protocol for the API access because the purpose of connecting to a REST API is to investigate how developers collect data from the mHealth Toolkit data stores and convert them to a format that can be sent to the REST API. Using an advanced authentication mechanism would only have resulted in a more complicated process than necessary. In addition, we were not allowed to use authentication protocols that are used within governmental healthcare as they are classified. For example, the id-porten authentication[7] that is used in Norway does not permit anyone without permission and authorization access to the development environment of the protocol.

A framework for *Node.js*, *Express*, was used to set up the REST API, which gave the ability to create routes and structure for the REST API. The database of the project was the NoSQL alternative *MongoDB*, which has a document object store. The Javascript library *MongooseJs*, customized for Node.js, was used for the connection to the MongoDB database. The format for all data exchanged with the API was in JSON. The reason for using the JSON format is that it is a lightweight, text-based, language-independent data interchange format[25].

5.3 Development Metrics

In this section, we will present the metrics we tracked during the application development. The metrics were chosen based on what quantitative data that is possible to track when developing such a system.

The primary quantitative data tracked during development of the example system were development time and lines of code. Total development time was tracked for each of the two applications described in section 5.2. While the total amount of code required was also tracked for each of the two applications; this metric was separated into three different categories for each of the two applications. Therefore, there were in total six different metrics that tracked lines of code, where each of them tracked the total amount of code needed to implement a particular functionality of an application. For the BG-app, the functionalities were: "*Initialize service and connect to Store*", "*acquire data permissions*" and "*calculating insulin and writing measurement to store*". And the functionalities for the PH-app were: "*Register/Create Patient(getDoctors, createPatient, store patientId locally)*", "*Read values from Store*" and "*Send value to server API(retrieve patientId, createReadings)*". All the described metrics were tracked for both the system employing SDH and HealthKit.

5.4 Findings

This section will cover the qualitative and quantitative findings from the development of the example system described in Chapter 5. The qualitative findings from the development with Apple HealthKit and SDH will first be presented separately, then the quantitative findings of the two Toolkits jointly.

5.4.1 Qualitative Data Findings - Apple HealthKit

The following section will cover the findings from qualitative data regarding the development of the example system with Apple HealthKit. During the development, notes of personal experiences were taken by the developer responsible for implementing the BG-app and the PH-app with HealthKit. The notes along with comments in the written code served as a basis for the qualitative analysis. The initial phase of the development was the creation of the BG-app. As it turned out, the BG-app required a majority of the critical HealthKit capabilities: initializing a connection to HealthStore, requesting authorization to write data and writing data. The description of each of these steps can be found in section 5.2.1.

The initial examination of HealthKit and its documentation revealed a poor structure. Certain figures and a large amount of text made the Toolkit seem overly complex. Taking the time to go through the text, but also discovering and watching the "*Introducing*

HealthKit"[15] video to perceive the structure helped make things more understandable. Once one has the overview of the structure, only a rudimentary knowledge of programming and a basic understanding of Swift is required to get started.

The initial stage of the implementation phase involved initialization of the HealthStore, which was done with a single line of code. Finding information relevant to the next stage regarding authorization for reading and writing data turned out to be challenging; searching through the reference documentation[6] provided by Apple was not always intuitive and certain parts of the navigation seemed broken. Due to this, third-party documentation in the form of tutorials[43] was used as a supplement to Apple's documentation. Eventually, a sample application provided in the documentation was discovered, but otherwise there were few code examples.

Implementing the write functionality required creating an HKSample with the user-provided information, and storing to the HealthStore utilizing the *saveObject* function (see section 4.2.4). This entails that Apple follows object-oriented concepts for storing data samples. Based on the structure, creation of data samples from the user-provided data was logical, due to being restricted to a limited amount of classes provided by HealthKit.

The development process of the PH-app was almost equivalent to the BG-app, differentiating only in the final step where the PH-app required implementation of functionality to read data, rather than just write. This process involved more time being spent browsing the documentation because the process of reading data followed traditional database principles using queries rather than the same object-oriented concepts for writing data in the BG-app. The implementation of the connection to the REST API was purposefully omitted due to lack of relevance with regards to HealthKit functionality.

5.4.2 Qualitative Data Findings - Samsung Digital Health

The development of the example system with SDH resulted in findings based on notes taken during the development process. These notes consisted of personal thoughts and experiences formulated by the developer responsible for implementing the BG-app and the PH-app with the SDH SDK. This section will present findings from the notes, regarding the development of these two applications and their common aspects.

At the start of the development of the BG-app an aspect distinctive to SDH was discovered, which was due to SDH's Android Studio requirement: the need to add metadata information in the *androidmanifest.xml* file. The documentation did not make it clear that permission values had to be specified in the xml-file in order to make the permission API work. The development process also established that the SDK draws a clear distinction between classes and interfaces. In addition, the use of a familiar naming conventions for classes and functions(such as "*insert*", "*observer*" and "*constant*") makes the class hierarchy more intuitive for the user. An exception to the intuitive class naming is the use of the word resolver in the *HealthDataResolver*(see section 4.3.4) class which was not immediately recognized as a link to reading and writing data. These findings were validated by the development of the PH-app, which consisted of the same experiences regarding naming conventions("*read*" and "*result*") when reading data with the HealthDataResolver.

The development of the second application, the PH-app, exposed negative aspects concerning class implementations related to reading and retrieving data. First of all, the need to call the *HealthDataResolver.ReadResult.getResultCursor* function was confusing as it was not clear that the *Cursor* class is used as the structure for iterating database records. Second of all, there was confusion around how the result is requested and retrieved with regards to the setting of result properties. More specifically, the *HealthDataResolver.ReadRequest.setProperties* function is used to define which properties of a given data type(in this example *HealthDataConstants.BloodGlucose*, see section 4.3.4) to be read, and the results are retrieved by passing a property to the *Cursor.getColumnIndex* function. As properties are used to both define what data to request and retrieve, it should be intuitive that these properties are directly related to a specific data type and results requested is structured with a column property, which is not the case.

After completing the development of the two aforementioned applications, common aspects could be used to extract findings regarding the overall development with the SDK. SDH simplified health data access and handling by providing an interface used for creating both custom and predefined types, enabling the use of numerous data types and measurements. It also provided a modular implementation of user permissions that was easy to use and that makes the user more consciously aware of data access. From the perspective of a developer possessing fundamental programming experience with mobile development with Java, and no previous experience with SDH, it was certainly achievable to create such a system as described in section 5.2 with the help of the available documentation with its examples and without any assistance from external resources.

5.4.3 Quantitative Data Findings - Apple HealthKit & SDH

The findings from the quantitative data generated from the example system development with both Apple HealthKit and SDH will be presented together in this section. These findings consisted of the results for the tracked metrics described in section 5.3, including time to complete the applications and amount of code lines required to implement each of the functionalities of the applications. Data from both of the developed applications were divided into three functionalities each and can be viewed in Table 5.1 below. All six functionalities are listed with a description and code, where the codes for the functionalities of the BG-app are shortened with code *BG-FX* and the functionalities of the PH-app are shortened with code *PH-app FX*, where X represents the numerical identifier of the given functionality. The implementation of needed GUI elements was excluded from these functionalities and their corresponding results.

Code	**Functionality(exclusive GUI code)**
BG-app F1	Initialize service and connect to Store.
BG-app F2	Acquire data permissions.
BG-app F3	Calculating insulin and writing measurement to store.
PH-app F1	Register/create Patient(getDoctors, createPatient, store patientId locally)
PH-app F2	Read values from Store
PH-app F3	Send value to server API(retrieve patientId, createReadings)

Table 5.1: List of functionalities comprising the BG-app and the PH-app

In Table 5.2 below, an overview of the results from the tracked metrics for each of the two Toolkits(HealthKit and SDH) is listed. The metrics include the amount of hours spent to complete the BG-app and the PH-app, and the number of code lines written to implement each functionality listed in Table 5.1 above. The following types of code lines were excluded from the results: empty lines, imports, class wrappers, and other code lines that were not specifically relevant to the given functionality.

Metric Toolkit:	**SDH**	**Apple HealthKit**
BG-app: time to complete(hours):	18	9.5
PH-app: time to complete(hours):	24	13
Number of code lines(BG-app F1)	20	5
Number of code lines(BG-app F2)	24	7
Number of code lines(BG-app F3)	20	9
Number of code lines(PH-app F1)	198	77
Number of code lines(PH-app F2)	38	6
Number of code lines(PH-app F3)	81	49

Table 5.2: List of measures from the development of the BG-app and the PH-app

Chapter 6

mHealth Toolkits Observation Study

This chapter will describe the observation study conducted with computer science students, where participants attempted to solve programming tasks in a predefined environment. In addition to the observations, this section describes the interview that was conducted with and the questionnaire that were filled out by each participant. The interview was held and the questionnaire supplied after the finalization of the observation. The questions were related to the participant's experience related to the tasks completed in the observation. After the observation study is presented and elaborated, the findings will be presented. The findings from this chapter will contribute to answer RQ1,RQ2 and RQ3.

6.1 Observations

The observations involved participants implementing solutions for predefined tasks in a predefined development environment, conducted in a laboratory setting. The pool of participants consisted of one group developing in an environment with HealthKit and the other with SDH(see Table 6.1 for a more detailed description of the development environments). The development consisted of having each participant attempt to solve four predefined tasks by programming in the given environment. For each task, the participant was given a task description and a maximum time limit of 30 minutes. Each task could result in two situations: either a participant completing the task within the time-frame or not. In both cases, the participant was asked to move on to the next task chronologically until all four were completed. In an attempt to minimize the occurrence and effect of non-Toolkit related issues, each task was structured with a pre-coded skeleton that provided the participant with a minimal setup to run the application, except that it lacked the Toolkit-related code needed to implement the required functionality.

The task description varied between the two environments, but the core functionality to be implemented was essentially the same. The selection of functionality for each task was based on the essential features each Toolkit provides. The following list describes the functionality to be implemented:

1. Connect to data store
2. Create permissions
3. Read value
4. Write value

The observation of each participant was conducted individually and before conducting the observation, the participant was given a brief introduction about what he/she was going to do in the study and why, before giving consent. Present during each observation were two people from the research team: one in the role of the main observer who provided the participant with information and another in the role of an observer secretary. Both observers were present overtly and took field notes during the observations. The field notes primarily serve as a supplement because screen recordings of the computer were used to provide a detailed account of all of the participant's actions during development.

As the researchers present were participant-observers with a passive role, i.e. the activities were observed in the setting but without participation[35], the participant could not contact the observers for assistance that might aid the participant in solving the task at hand.

Detail Provider	Apple	Samsung
SDK	HealthKit	Samsung Digital Health
IDE	Xcode	Android Studio
Programming language	Swift 2	Java
Computer	Mac	PC

Table 6.1: Details of the development environments for the two providers Apple HealthKit and Samsung Digital Health, used in the observation study

6.2 Questionnaires

Following the completion of all development tasks, the participant was asked to fill out a questionnaire that included five questions related to the participant's feelings regarding the Toolkit and its documentation. The D-T Scale was used as the metric scale for these question, which can be viewed in Table 6.2 below.

Question	Answer format	Qualitative/Quantitative data
Q1: How do you feel about the overall development experience?	D-T Scale	Quantitative
Q2: How do you feel about your ability to become familiar with with the Toolkit?	D-T Scale	Quantitative
Q3: How do you feel about the programming experience?	D-T Scale	Quantitative
Q4: How do you feel about the structure of the documentation?	D-T Scale	Quantitative
Q5: How do you feel about the descriptiveness of the documentation?	D-T Scale	Quantitative

Table 6.2: List of questions comprising the questionnaire answered by participants

6.3 Interviews

As with the questionnaires, the interviews were conducted post-development. Each interview was conducted individually with the participant, with the person conducting the interview being the main observer from the observation to avoid confusion and maintain consistency. The purpose of the interview was to gain insight into the participants' thoughts, feelings and experiences regarding the environment and the Toolkit that the participant had been exposed to. Audio tape recording was used to record the conversation that would later be transcribed and analysed. The interviews followed a semi-structured approach, chosen due to it being well suited for discovery. The structure involved having a list of 3 main questions to ask the interviewee, consisting of: "*In what way did the mHealth Toolkit enforce your programming creativity?*", "*What are your positive experiences regarding the development?*" and "*What are your negative experiences regarding the development?*". It was possible to change the order of the questions according to the conversation flow, and it was also possible to ask additional questions depending on the answers given and the subjects mentioned by the interviewee. Avoidance of ambiguity is crucial for discovery of themes, and in this respect, the ability to ask additional questions proved to be very useful in our interviews. The reason is that there was a strong tendency among the interviewees to give ambiguous answers and statements that the follow-up questions were successful at clarifying.

A major advantage of such a structure is that the interviewee was able to provide more details about the raised subjects and was also able to introduce own thoughts deemed relevant. On the other hand, it is important to note that it limits the discovery to personal accounts and feelings, and may hence not be used to draw generalizations about a wider population.

6.4 Metrics

In this section, the different metrics that were tracked during the observations and questionnaires that were based on data that can be measured will be described. As interviews by nature are qualitative, no interview metrics are presented.

6.4.1 Observations

Of the metrics tracked during the observation study, the most straightforward to measure was the number of tasks completed by a participant. This metric is valuable because it is understandable by outsiders and comparable across groups of participants. The second metric was task completion time, namely the time a participant uses to complete a given task, which is a good metric, also due to the previous reasons. Lastly, the time spent in the IDE and each of the available documentation sources individually and combined as a total was measured. The motivation behind this choice was to attempt to describe the documentation's affect on the participants' development process.

6.4.2 Questionnaires

In the case of the questionnaires, user satisfaction was the main metric measured with the D-T Scale(presented in section 6.2). In this case the satisfaction is related to the predefined environments described in Figure 6.1, where the user is an application developer with a fundamental level of experience with the given environment. The user satisfaction was measured quantitatively with a given set of questions answered by the participants.

6.5 Findings

This section will present the findings from all data generated from the observation study, including notes from video recordings and field notes, interviews and questionnaires answered by the students participating in the development study. The findings include both findings gained from qualitative and quantitative data. The qualitative findings from interviews, field notes and video recording notes are first presented for Apple HealthKit and SDH separately, with the list and description of the themes identified from the qualitative data analysis. Then the quantitative findings from the measures tracked in the video recordings and the answers from the questionnaires will be presented for HealthKit and SDH jointly.

6.5.1 Qualitative Data Findings

Apple HealthKit - Interviews, field notes & video recording notes

The following section will present the findings from the qualitative analysis performed on the data gathered from the process described in Chapter 6, specifically regarding Apple HealthKit. From the sources listed in the title, we identified relevant themes by analysing the transcribed data. For example, in an interview participant *AAPL01* said: "*Pleasantly surprised by the size and complexity of the Toolkit*", while *AAPL07* stated the following about the benefit of using HealthKit: "*... Removes the complexity of storing/retrieving values as well as in creating a backend/structure in order to do the same thing*".

These quotes along with others from the participants were grouped together to form a theme called "*Toolkit structure*" describing their commonalities. This is an example of how transcribed quotes are related to a certain theme, and similar relations exist for all of the remaining themes. The themes were discovered as a result of the process described in section 3.4.1, and the themes along with their description are shown in Table 6.3 below.

Following the discovery of the themes, the next stage was to categorize them further into parent themes. A parent theme describes the concept that mutually binds the discovered themes, e.g. documentation descriptiveness and documentation usefulness would be grouped within the parent theme documentation.

Theme	**Description**
Toolkit usefulness	Statements regarding usefulness of the mHealth Toolkit.
Naming conventions	Statements regarding naming conventions of methods, classes and terms within the Toolkit.
Class confusion	Statements regarding confusion/lack of information on what class to use.
Toolkit structure	Statements regarding the structure of the mHealth Toolkit.
Toolkit structure issues	Statements regarding issues around the structure of the Toolkit(classes, methods, objects).
Implementation issues	Statements regarding issues during/about implementation.
Concreteness	Statements regarding issues during/about implementation.
Lack of examples	Statements regarding the lack of examples within documentation.
Documentation Structure	Statements regarding the document structure.
Navigation issues	Statements regarding navigation issues within the documentation.
Example confusion	Statements regarding confusion around provided example in the documentation.

Table 6.3: List of themes and their description, identified from interviews with participants developing with HealthKit

During interviews, the participants were asked to elaborate on their experience. Candidate AAPL05 was quoted saying: "*..there were certain methods that were supposed to be used and these were defined to do no more and no less that exactly what was needed*" with AAPL06 concluding that HealthKit was:"*easy to learn, and fairly easy to use.*". These in addition to similar quotes from candidates AAPL03 and AAPL07, formed the parent theme "Toolkit general", which represents statements regarding HealthKit overall.

The parent theme "Class hierarchy and naming" was formed on the basis of quotes belonging to several categories. An example is AAPL04 stating that "*I'm not used to iOS naming conventions, as other languages follow the standard set/get method names.*", which belongs to the "Naming conventions theme", was grouped with the quote from AAPL02 stating "*I feel it is correct of Apple to provide building blocks (user ref. To HKUnit, HKSample, etc) and create something out of that. Happy with the way that was done*".

Both candidates provided valuable information regarding the class hierarchy or naming and are therefor categorized under the same parent theme.

When evaluating HealthKit, another aspect that appeared crucial to our participants was implementation of tasks. Participants AAPL03 and AAPL06 commented on the *isHealthDataAvailable* function, saying: "*Why can't it find isHealthDataAvailable() method in the store object?*" and "*Unsure what object to call the method isHealthDataAvailable with*" respectively. This lead to the creation of "Class implementation" theme.

Lastly, the majority of quotes consisted of thoughts or comments regarding the documentation. AAPL01 disclosed that: "*Documentation was good*" however noted that "*There are few examples compared to other frameworks*". Candidates AAPL03 and AAPL06 backed up the notion of lack of examples by saying "*good swift examples in certain parts*" and "*I feel they should have placed some code examples for use of this class*" respectively.

In Table 6.4 below, the discovered themes along with their parent themes are presented, along with the occurrences of both, for the interview data analysis. For each theme, the number of its distinct occurrences was counted across the interviewees(participants) that completed the Apple HealthKit version of the tasks. I.e. if there were more than one occurrence in the transcript of an interview with a statement related to a specific theme, it would only be counted once towards the total amount of distinct occurrences. In the same manner, the number of distinct participants related to each of the parent themes was counted across the aforementioned theme occurrences, not allowing a participant to be counted more than once.

Parent Themes(distinct #participants)	**Themes(distinct occurrences)**
Toolkit general(**4**)	Toolkit usefulness(4)
Class hierarchy and naming(**7**)	Naming conventions(1), Class confusion(4), Toolkit structure(5), Toolkit structure issues(3)
Class implementation(**2**)	Implementation issues(2)
Documentation(**7**)	Concreteness(2), Lack of examples(3), Structure(6), Navigation issues(4), Example confusion(3)

Table 6.4: List of themes and their parent themes, along with occurrences of both, identified from interviews, for HealthKit

Additionally, data was generated in the form of field notes during observations and from reviewing the video recordings during the analysis stage, to validate the identified themes and occurrences. Therefore, the format of the results from this data analysis is the same as described above with interview transcriptions, and can be viewed in Table 6.5 below.

Parent Themes(distinct #participants)	**Themes(distinct occurrences)**
Toolkit general(**0**)	Toolkit usefulness(0)
Class hierarchy and naming(**3**)	Naming conventions(2), Class confusion(2), Toolkit structure(0), Toolkit structure issues(1)
Class implementation(**4**)	Implementation issues(4)
Documentation(**2**)	Concreteness(0), Lack of examples(2), Structure(1), Navigation issues(0), Example confusion(2)

Table 6.5: List of themes and their parent themes, along with occurrences of both, identified from field notes and recording notes, for HealthKit

SDH - Interviews, field notes & video recording notes

The following section will present the findings from the analysis performed on the data generated from interviews, field notes and video recording notes generated in the process described in Chapter 6, concerning SDH. The transcripts from the interviews were used to identify relevant themes based on the analysis process described in section 3.4.1. The complete list of themes and their description can be viewed in Table 6.6 below.

Theme	Description
Toolkit usefulness	Statements regarding tasks and connecting to store being simplified by Toolkit.
Permission issues	Statements regarding struggle with understanding and using the Permission related classes.
Class relation confusion	Statements regarding difficulties with understanding how the various classes are related, especially classes related to reading values.
Interface for database R/W-access	Statements regarding difficulty with retrieving read results with the Cursor interface.
Class pattern issues	Statements regarding issues with implementations of Filter and Builder Pattern.
HealthData parameter issues	Statements regarding issues related to setting correct parameters of the HealthData object used for writing values, including confusion around mandatory parameters and usage of Offset parameter.
Intuitive Request-classes	Statements regarding the intuitivity of the ReadRequest and InsertRequest classes and how they simplified tasks solving.
Documentation eased task solving	Statements regarding the significance of the documentation for improved task productivity and task solving.
Descriptive documentation	Statements regarding the descriptiveness of the documentation including the ease to find relevant information and good examples.
Not descriptive documentation	Statements regarding not descriptive documentation including large of amount of time spent searching in and becoming familiar with the documentation.

Table 6.6: List of themes and their description identified from interviews with participants developing with SDH

Each of the identified themes described above were, in compliance with our qualitative data analysis approach, grouped into more generalizable parent themes, where each parent theme described the overarching aspect binding the themes together. Therefore, were each parent theme connected to the interviewees' statement via this relation to these themes.

An example of how the interview data was related to the identified themes is the first parent theme "Toolkit general", only comprised of positive statements of the theme "Toolkit usefulness", where participant PS04 stated that "*The code and the descriptive documentation gives you a better starting point than starting from scratch*" and participant PS06 stated that "*It offers a library that simplifies the tasks at hand*".

The theme "Permission issues" was based on the quotes by participant PS07 stating that he/she "*Used a lot of time to understand how the permissions were structured*" and participant PS06 stating that "*Permissions were a little cumbersome. Struggled with putting together three classes PermissionType, PermissionKey and PermissionResult to set up Permissions*". In a similar fashion were statements related to the theme "Class relation confusion", which together with "Permission issued" formed the parent theme "Class hierarchy and naming".

The parent theme "Class implementation" comprised of the themes "Interface for database R/W-access", "Class pattern issues", "HealthData parameter issues" and "Intuitive Request-classes". Regarding the theme "Class pattern issues" the statements included participant PS07 stating that: "*I found information in API-ref about ReadRequest, but I did not understand how to create a Request with parameters with Builder*" and regarding the theme participant PS02 stated that: "*I did not know the meaning of the word Offset. I had a hard time figuring out what to put in the Offset field, so I tried to run the program without it, but that did not work*". In a similar fashion were statements related to the remaining two of the abovementioned themes.

The parent theme "Documentation" comprised of negative and positive statements related to the themes "Documentation eased task solving", "Descriptive documentation" and "Not descriptive documentation". For example, related to the theme "Documentation eased task solving" participant PS01 stated that: "*In the case of both the tasks I completed, the documentation was crucial for my success*", and related to the theme "Not descriptive documentation" participant PS04 stated the following about a specific task: "*I didn't like the documentation here. I didn't figure out which functions to use, and that I had to use Filter for setting time range. It was poorly explained*".

For each theme, the number of its distinct occurrences was counted across the nine interviewees(participants) developing tasks with SDH. I.e. if there were more than one occurrence in the transcript of an interview with a statement related to a specific theme, it would only be counted once towards the final amount of distinct occurrences. In the same manner, the number of distinct participants related to each of the parent themes was counted across the aforementioned theme occurrences, not allowing a participant to be counted more than once.

These themes and their corresponding parent themes, along with the occurrences of both, for the interview data analysis are listed in Table 6.7 below.

Parent Themes(distinct #participants)	**Themes(distinct occurrences)**
Toolkit general(**9**)	Toolkit usefulness(9)
Class hierarchy and naming(**6**)	Permission issues(2), Class relation confusion(4)
Class implementation(**9**)	Interface for database access(3), Class pattern issues(3), HealthData parameter setting issues(7), Intuitive request classes(3)
Documentation(**9**)	Documentation eased task solving(8), Descriptive documentation(5), Not descriptive documentation(3)

Table 6.7: List of themes and their parent themes, along with occurrences of both, identified from interviews, for SDH

As mentioned, were data from field notes and video recording notes also analysed. The findings gained from these analyses were gained to supplement and check the validity of the identified themes and occurrences from the interview transcripts, and hence this list follows the same format as the previous one. These results can be viewed in Table 6.8.

Parent Themes(distinct #participants)	**Themes(distinct occurrences)**
Toolkit general(**0**)	Toolkit usefulness(0)
Class hierarchy and naming(**9**)	Permission issues(3), Class relation confusion(9)
Class implementation(**6**)	Interface for database access(1), Class pattern issues(2), HealthData parameter setting issues(4), Intuitive request classes(0)
Documentation(**8**)	Documentation eased task solving(4), Descriptive documentation(7), Not descriptive documentation(3)

Table 6.8: List of themes and their parent themes, along with occurrences of both, identified from field notes and recording notes, for SDH

6.5.2 Quantitative Data Findings - Apple HealthKit & SDH

The following section presents the findings from analysis of quantitative data from the conducted observation study described in Chapter 6. These findings were a result of an analysis of data generated from screen recordings of the participants' development during the observations, and the questionnaires completed by these participants post-development.

Screen Recordings

A vital part of the observation study consisted of screen recordings of the participants' actions, used as a basis for quantitative data analysis. Based on the observations metrics described in section 6.4.1, there was derived a list of measures relevant to developer efficiency and productivity. The list is as follows:

- M1: Average percentage of time spent in documentation
- M2: Corr(X, Y), Correlation between variable X and variable Y
  - Where X = Tasks completed for participant P(i).
  - Where Y = Mean(TS(D)/TS(IDE)), mean average of ratio between time spent in documentation and time spent in IDE in each of the 4 tasks for participant P(i).
- M3: Number of participants solving one or more task within time limit.

- M4: Average number of tasks completed within time limit.

Measure M1 gives information on the portion of time participants spent in documentation, regardless of whether they completed a task or not, to reveal the significance of each Toolkit's documentation. M2 shows whether or not there is a correlation between number of completed tasks and the ratio between time spent in documentation and IDE. The correlation coefficient illustrates the statistical relationship between variables, and can provide an indication to whether or not there is a dependence between the two. The results of M2 will be discussed in section 9.1.3 in light of this statement: *"Any coefficient between 0.3 and 0.7(plus or minus) is regarded as demonstrating a reasonable correlation"*[44, p. 258]. The reason for choosing percentage of time spent in documentation and using a ratio between time spent in documentation and IDE, was to be able to disregard the difference in completion time and whether or not a task was completed. Both M3 and M4 provide information on what Toolkit the participants had more success with and the degree of their success, with M4 providing information with more granularity than M3. The results of measures M1-M4 for Apple HealthKit and SDH are shown in Table 6.9 below.

Toolkit \Measures	**M1**	**M2**	**M3**	**M4**
Apple HealthKit	43,1%	0.7294	7/7(100%)	2,57
SDH	41,2%	0.1564	6/9(66%)	1,00

Table 6.9: Resulting measure for M1-M4, for each of the two evaluated mHealth Toolkits

Questionnaires

The following section will present the findings gathered from questionnaires filled out by observation study participants post-development. For each of the Toolkits, mean average values were calculated for all of the five questions. The D-T Scale determined the values (1-7) where 1 represents Terrible, and 7 represents Delighted according to the process described in section 3.4.2. The resulting score for Apple HealthKit and SDH can be viewed in Table 6.10. The referred questions can be found in Table 6.2 in section 6.2.

Toolkit \Question	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Total**
Apple HealthKit	5,0(0,76)	5,0(0,53)	4,9(0,64)	3,9(0,64)	4,2(0,64)	4,6
SDH	5,0(1,05)	4,8(1,13)	4,9(1,10)	4,6(1,26)	5,0(0,94)	4,8

Table 6.10: Mean average scores of questions from the questionnaires answered post-observation. Standard deviation for each question is listed in parenthesis

Chapter 7

Questionnaire with Experienced Developers

In this chapter, we will describe the process we went through of creating a questionnaire and distributing it to experienced mHealth Toolkit developers. The reason why we wanted to create and distribute the questionnaire is to gain insight into how developers are using the mHealth Toolkits in real life settings and how using them affects their creativity. First, the questionnaire will be described. Secondly, the process of distributing the questionnaire and the participation from developers will be elaborated. Lastly, the findings will be presented. The findings from this chapter will contribute to answer RQ3.

7.1 Description

In this section, the questionnaire will be described in detail. We created the questionnaire based on the questionnaire from the observation study described in Chapter 6. The D-T scale was used as the scale for quantitative answers. For qualitative data, we asked questions where the participants could write textual answers. Developers were first asked if they had experience with the mHealth Toolkits in question(Apple HealthKit, Google Fit and SDH). If they had experience with one or more of the mHealth

Toolkits, they were given questions about that(those) Toolkit(s). The questions for all Toolkits were the same, and they are listed in Figure 7.1 below.

Question	Answer format	Qualitative/Quantitative data
Q1: How do you feel about the overall development experience?	D-T Scale	Quantitative
Q2: How do you feel about your ability to become familiar with with the Toolkit?	D-T Scale	Quantitative
Q3: How do you feel about the programming experience?	D-T Scale	Quantitative
Q4: How do you feel about the structure of the documentation?	D-T Scale	Quantitative
Q5: How do you feel about the descriptiveness of the documentation?	D-T Scale	Quantitative
Q6: If found efficient, in what way did it increase your efficiency?	Text	Qualitative
Q7: How would you describe the Toolkit's effect on development creativity?	Text	Qualitative

Table 7.1: List of questions comprising the questionnaire answered by experienced developers

7.2 Process and Participation

The following section will describe the process of reaching out to developers with mHealth Toolkit programming experience. The target group of this questionnaire consisted of people that are developers and have experience with one of the Toolkits in question(Apple HealthKit, Google Fit and SDH). To reach out to the developers, we tried to identify and use all possible communication channels. We identified these as developer forums for the Toolkits in question, emails to developers of partner applications to the Toolkits(applications that have integration to the Toolkit in question) and computer science student forums.

In our process of distributing the questionnaire, we tried posting in mHealth Toolkit developer forums. Some of the Toolkit providers would not let us put the questionnaire up on the forum as they did not want to be involved in the research. For example, we got this answer from Apple: "*The developers can be contacted by the people following the information found on the App Store in case needed but it cannot be made through Apple, and we cannot provide information on the contacts we have for privacy matter*". We were unavailable to post in the Samsung developer forum because we never received a confirmation email after registering. We were able and allowed to post in the forum of Google Fit.

The next step was to contact all registered partner applications via email. We sent emails to their support emails, available in Apple AppStore and Google Play Store. These partner applications were found by searching in the various application stores and performing Google searches. The email contained the link to the questionnaire and asked the person who received it to forward it to developers with mHealth Toolkit development experience. In total, 63 different applications providers were contacted via email and we got 19 answers from the support departments of the respective applications that were not automatically generated by their email system. Some replied that they had answered the questionnaire, while other informed us that the questionnaire had been forwarded to developers, and others could not answer the questionnaire due to the company's security policy.

The questionnaire was also posted in a student group on Facebook with Computer Science students. The aforementioned group is for all students that are attending the computer science program at the Norwegian University of Science and Technology. The answers from this channel were mostly unusable because most respondents had no experience with any of the Toolkits in question.

In total, we received 22 answers. From these, only 12 answers contained usable data as the other 10 had no prior experience with the toolkit(s). Of the 12 answers, 4 were related to Google Fit, 4 to Apple HealthKit and 4 to SDH.

7.3 Findings

In this section, the findings from the questionnaires filled out by experienced developers will be presented. As mentioned in Chapter 7, we reached out to developers through forums, emails and posts in social media. The response consisted of only 12 answers usable in the analysis, which was a lower degree of participation than expected. First, the quantitative data will be presented, then the qualitative data.

Table 7.2 presents the results of the quantitative data collected from the questionnaires. For all of the questions for each Toolkit, we calculated mean average values using the D-T Scale values(1-7) where 1 represents Terrible, and 7 represents Delighted. The referred questions can be found in Table 7.1 in section 7.1.

Toolkit \Question	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Total**
Apple HealthKit	5,00(1,22)	5,50(0,87)	4,75(1,09)	5,50(0,50)	5,25(0,83)	5,20
Google Fit	4,75(0,83)	4,75(0,43)	4,75(0,83)	4,00(1,22)	4,75(0,83)	4,60
SDH	4,75(1,09)	5,00(0,71)	4,25(1,48)	4,00(1,22)	3,50(1,12)	4,30

Table 7.2: Mean Average Score results from questionnaire with experenced developers. Standard deviations is displayed in parentheses

From the answers to the questions that gave qualitative textual answers in the questionnaire, we deducted certain themes. The themes that most answers fell underneath were, as described in Table 7.3, complaints about the stability of the mHealth Toolkits, the added value of data sharing and API design of the mHealth Toolkits.

Theme	**Description**
Bugs and stability	Statements regarding bugs and stability of the mHealth Toolkit in question.
Added value of data sharing	Statements regarding what kind of value the mHealth Toolkit in question adds to the developer in terms of data sharing.
API design	Statements regarding the design of the API of the mHealth Toolkit in question.

Table 7.3: List of themes and their description, identified from textual questionnaires answers of experienced developers

Two of the developers that participated in the questionnaire complained about bugs and stability of the Toolkits. For example one of the participants wrote the following about Apple HealthKit: "*It has had so many bugs, and has many to this date, that it's frustrating to use. Hence, we try to avoid relying on it, which limits the gained room for creativity*". Another example is a participant that wrote the following about SDH: "*Bad SDK code bring bad code to project. Library fails periodically in production*".

Seven of the questionnaire participants mentioned the added value of data storage and sharing from the mHealth Toolkits in their answers. All of the seven developers that mentioned the added value of data storage and sharing from the Toolkits felt that they became more productive in developing an application since the Toolkits handled the data storage for them. For example, one of the participants wrote the following about Google Fit: "*It makes me more creative since I do not need to create my own backend and therefore can focus on actually implementing the application and adding features instead of making a backend*". Another wrote the following about Apple HealthKit "*It allows data sharing in ways that were not possible previously, so it generally allows for more creativity and use cases*".

Participants of the questionnaire had both positive and negative things to say about API design. Google Fit was criticized for its API design. For example, one of the participants wrote the following about the Google Fit API: "*API is not properly designed, seems to me too academic and too generic*". Another participant complained about the setup process of Google Fit by writing that "*The setup process is however quite extensive and could be simplified. I used one working day before I was able to make calls to the APIs. The most challenging part was to understand the example code for connection provided by Google on their GitHub*". The API design of SDH, on the other hand, received positive feedback from a participant answering that: "*API is simple but covers most cases and doing what you expect*".

We tried to use the same parent themes as we did in previous sections for the themes found from the questionnaire with experienced developers. The themes and their parent themes are listed in Table 7.4.

Parent Themes/Theme Source	**Themes(distinct occurrences)**
Toolkit general	Bugs and stability(2), Added value of data sharing(7)
Class hierarchy and naming	API design(3)
Class implementation	API design(3)
Documentation	-

Table 7.4: List of themes and their occurrence, along with corresponing parent themes, identified from questionnaires with experienced developers

Chapter 8

Summary of Findings

This chapter will summarize the findings from our research as presented in sections 4.4, 5.4, 6.5 and 7.3 respectively.

In section 4.4, we discovered that Apple HealthKit and SDH had more in common in terms of data storage as both stored data locally of the device, with Google Fit only using cloud storage. In addition it was discovered that the setup process of Google Fit, was quite extensive.

Section 5.4 presented findings from Chapter 5. This section described data obtained through implementation of an example system. It revealed that the system could be developed using both Apple HealthKit and SDH, with HealthKit using substantially less time and lines of code to develop. Additionally, the implementation revealed that within both toolkits there was confusion regarding class naming.

An observation study was the source of the findings in section 6.5. The qualitative data gathered from interviews revealed that both HealthKit and SDH were useful to the participants, and that use of Toolkits simplified the solving of tasks. Furthermore, concerns regarding the class structure and implementation were revealed, with participants using either platform complaining about specific aspects of the platform. Comments and quotes regarding documentation revealed that while SDH participants were mostly satisfied, HealthKit's documentation was criticized for lack of examples. The quantitative data supported these facts, with HealthKit scoring lower than SDH in the questionnaire filled out by the participants post-development.

The previous chapter collected questionnaire data from experienced developers, and the findings were presented in section 7.3. This questionnaire revealed that Apple HealthKit scored highest among the three, with developers commenting that either toolkit is stable/reliable enough to be used in production.

Chapter 9

Discussion

In this chapter, we will discuss the research conducted in this thesis. First we will interpret and discuss the findings of the research, and then we will revisit and answer the research questions stated in section 1.2. Lastly, we will evaluate the research in terms of its significance and in light of existing knowledge on the subject. All papers that are referenced in this chapter has already been introduced and we only use extractions from the previous works in the following sections(see section 2.2. for the complete description of previous works).

9.1 Discussion of Findings

In this section, the findings from the research will be discussed. The findings will be discussed in the same order as they were presented in chapter 8. First, the findings from the mHealth Toolkits chapter will be discussed. Then, we will discuss the findings from the example system development. Following that, findings of our observation study will be discussed, and lastly we will discuss the findings from the questionnaire with experienced developers.

9.1.1 Chapter 4 Discussion

In this section, the findings from chapter 4 regarding technical architecture; the way data is stored and the type of data is allowed to be stored will be discussed. The other finding regarding the setup process of Google Fit will be discussed in section 9.1.2 due to its relevance to the discussion of the examples system we developed.

As presented in section 4.1 and 4.4, Google Fit differs from Apple HealthKit and SDH with regards to what data types that can be stored and how the data is stored. Both Apple HealthKit and SDH use the device as the primary storage for data, while Google only stores data in its cloud storage. For the developer, this might seem irrelevant, but it is important to keep in mind the need for secure storage of health data. Developers that are developing mHealth applications must examine the rules and restrictions for data storage in a given country. In most countries, cloud storage is a violation of the requirements and laws regarding storage of health data. The previous failure of Google Health described in section 4.1.2 indicates that it is difficult to implement cloud storage of health data. Developers should therefore stick to solutions that store health data in a secure manner on the device such as Apple HealthKit.

Since Google Fit is not meant to store health data, it is reason to believe that Apple HealthKit and SDH are suited for developing mHealth applications, while Google Fit is not. This is the reason for our decision to exclude Google Fit from our developer observation study and our example system development. Additionally, Google Fit lacks a data type for blood glucose that we used in the observation study and the development of an example system.

9.1.2 Chapter 5 Discussion

The following section will discuss the selected mHealth Toolkits, with regards to the findings from the example system development presented in section 5.4. In addition to interpreting the findings through evaluation based on experience and previous work, this section will provide information to answer RQ4 along with providing partial information for answers to RQ1, RQ2 and RQ3. Do note that the discussion is written from a developer's point of view.

There are fundamental differences between the Toolkits with regards to the context of developing mHealth applications. Apple limits the development of iOS applications to OS X users and as a result, the development of HealthKit applications is limited to iPhone devices, as discussed in sections 4.2 and 5.2. Compared to HealthKit, SDH is based on the Android platform, which is open-source and provides a lower “entry cost” (the cost to create and publish an application a store). and therefore makes SDH available to the entire Android device portfolio[1]. What this entails is that Samsung has a larger opportunity to affect markets where developers have limited resources compared to Apple who seems to be targeting the modern healthcare market, evidenced by their cooperation with EPIC and Mayo Clinic[18] as referenced in the introduction of Chapter 5.

In order to understand what separates the selected Toolkits and thereby provide a partial answer to RQ1, their differences were examined through development of an example system. The example system should be thought of as a semi-realistic example of how a healthcare provider might facilitate the use of applications built by employing the selected Toolkits. Additionally, as mentioned by Greenberg[2007] in Chapter 2, Toolkits are supposed to abstract common and general problem areas within a domain. Considering this, we will discuss the aspects that affect productivity during development.

Delving into the setup process, findings from section 4.4 revealed that Google Fit, in comparison to HealthKit and SDH, requires a more extensive setup time and substantial amount of code. Setup time is defined as the time it takes to set up everything required to use the Toolkit. As Google Fit was excluded due to reasons described in section 5.2, only Apple HealthKit and SDH will be discussed. From the qualitative findings, we notice that the setup time of HealthKit was quite short according to the responsible developer, requiring only one codeline to instantiate the HealthStore. SDH required the developer to add information in the androidmanifest.xml file. In addition, with SDH the developer needed to download and install relevant jar-libraries from the Samsung’s online developer resources. When comparing the two, we conclude that HealthKit has

[1]Samsung Digital Health is restricted to Samsung devices running Android 2.3 or higher and/or non-Samsung devices running Android 4.4 or higher

a more straightforward setup, due to HealthKit being more tightly integrated into the iOS platform than with SDH is with Android.

HealthKit's higher level of abstraction is evident when inspecting the quantitative data. Development with SDH shows that it is inferior to HealthKit with regards to time and number of lines of code required to complete the example system. To implement the same functionality, HealthKit required approximately 50% lines of code less than SDH. For example, SDH required 20 lines of code to implement initialization and connection to the store, where the same was accomplished in 5 lines with HealthKit. The experience of the developers that implemented the system for HealthKit and SDH was not a cause for the resulting disparity between the two, as the they had approximately the same amount of experience. Other possible causes must therefore be examined.

Apple imposes structural limitations on the developer in the form of categories of classes: each sample type(data type) had a corresponding sample class(actual data object) and each sample type was defined based on predefined constants provided by Apple in the documentation(type identifier(s)). In practice, this meant that the developer could only create applications that used data belonging to one of the predefined types. SDH provided the developer with the option to create custom data types, based on a standard object. On one hand these restrictions are negative as they restrict the scope of applications that could be created. On the other hand, the creation of applications using predefined data types can be completed in less time.

Becoming familiar with the structure of a Toolkit or an API is usually done through its documentation. The structure and usability of the documentation of HealthKit and SDH will be evaluated in light of the findings relating to the developers' thoughts from section 5.4 in the previous chapter. Based on the findings presented in section 5.4.1, we conclude that a developer with no previous experience with SDH can implement the described applications with the help of the available documentation without assistance from any external resources. The same was the case with HealthKit, but there was a difference in the documentation used; external documentation in the form of tutorials was also used due to insufficient examples provided by Apple. This lead us to the belief that SDH has a more descriptive documentation with a superior structure, however, these findings alone are insufficient and to justify this statement additional findings supporting it will be discussed in section 9.1.3.

Regarding class architecture, the findings of section 5.4 revealed fundamental differences regarding the implementation process. SDH applies consistent concepts and naming related to the process of reading and writing data using the HealthDataResolver class, while Apple shows inconsistencies by applying object-oriented concepts to writing data with the saveObject function and traditional database concept with the HKQuery class for reading data. One could argue that the approach used by HealthKit confuses the developers. There were also some concepts within SDH that caused confusion, such as the implementation of the Builder and Filter design patterns, and especially the use of the Cursor class for iterating database records. Looking back at development time and lines of code in Table 5.2, the numbers suggest that developing with HealthKit is far more efficient, but as with the findings regarding the documentation, are these findings alone not sufficient to support this claim.

9.1.3 Chapter 6 Discussion

In this section, the findings from qualitative data presented in section 6.5.1 and the findings from quantitative data presented in section 6.5.2 will be summarized and discussed. The qualitative findings will be discussed first, followed by the quantitative findings.

Discussion of Qualitative Findings

The findings from the analysis of qualitative data generated from the observation study described in section 6.5.2 will be discussed in this section, for each of the parent themes individually. The findings from the interview data are primary, while the findings from field notes and video recording notes are supplementary.

Toolkit General: For HealthKit, there were 4 participants with statements regarding the general Toolkit usage, while there were 9(all participants) with SDH. They all stated that the given Toolkit was useful by making the tasks easier to solve. Both had no occurrences in the supplementary findings and hence did not validate the primary findings, but it is important to note that these statements were directly related to a question asked in the interviews and it was therefore difficult to observe such occurrences from the participants' actions. As these results comprised of more than half of the participants, they indicate that both Toolkits simplified task solving, although to a greater extent with SDH. Greenberg[2007] states that a Toolkit should "*Remove low-level implementation burdens*"[31], and it is likely that our findings are due to the removal of such burdens in the given Toolkits. These findings support the conclusion of Von Hippel and Katz[2002] that "*Toolkits for user innovation will eventually be adopted by many manufacturers facing heterogeneous customer demand*"[53], as time not spent implementing basic functionality can be instead be invested in user innovation.

Class Hierarchy and Naming: In the interviews, both HealthKit and SDH had many occurrences of statements regarding naming of classes and functions, and their class hierarchy; 7 for HealthKit and 6 for SDH. For HealthKit they were predominantly related to naming inconsistencies(notably the use of the HKHealthStore.saveObject function to write and the HKHealthStore.executeQuery function to read), but also related to the relations between classes of the hierarchy, while for SDH they were related to the relations between classes used to read values and to manage permissions. These findings were validated by the 3 occurrences in the supplementary findings for HealthKit and 9(all participants) for SDH. The reason why the number of occurrences of the supplementary findings can exceed those of the primary findings such as in this example, is because the participants were not conscious of or had forgotten certain events at the time of the interviews.

In a study of different variants of an API, Scheller and KühnScheller and Kühn concluded that "*If a class or method name didn't meet the programmer's expectations, he/she would need much more time finding it*"[51]. In light of this conclusion, we can infer that the identified issues with understanding the names of classes and functions of HealthKit affected the developer productivity negatively, compared to if it had been put more weight to the developer's expectations when designing the API and its naming. It is also reasonable to assume that the confusion around class relations within the class hierarchy decreased productivity for both Toolkits.

Class Implementation: Clarke[2004] states that: "*usability problems can exist in an API that are related to users not perceiving the affordances the API supports*"[24]. I.e. how the API classes are implemented affects the way developers perceive their affordances, and this aspect will be discussed here.

With HealthKit there were 2 occurrences of issues related to class implementations and 4 from the supplementary findings, where the prominent issue was related to the class function $HKHealthDataStore.isHealthDataAvailable$. The issue was due to the function being static, and thereby requiring to be called on the HKHealthStore class, not the instance fo this class. This distinction lead to confusion because the participants did not know that an initialization of the object was not needed, i.e. they did not perceive the affordance of this class function. The use of static functions is non-existent in SDH, hence SDH has a more consistent class implementation in this respect. There were 6 occurrences with SDH in the primary findings related to class implementation, validated by 9(all participants) occurrences in the supplementary findings. All themes consisted of issues, except for a theme where participants commented on the intuitivity of the Request-classes($HealthDataResolver.ReadRequest/InsertRequest$). The occurrences included three main issues that were due to participants not perceiving the affordances of the API implementations, possibly revealing an inadequacy in the API. The first issue was related to setting a mandatory parameter of the Health-Data class, the $HealthConstants.BloodGlucose.TIME_OFFSET$ that considers time zones, which was a cause of confusion due to the name itself, why it was needed and that this property was related directly to each type of *HealthConstants* class. In contrast, this was not an issue with HealthKit, as the handling of time offset is abstracted away by using the *NSDate* class. The second issue was related to SDH's implementation of the Builder and Filter pattern; there were problems with understanding how to use the Builder class to build a ReadRequest and the Filter class to filter read values by date time. The last issue was concerned with the use of the Android Cursor class for iterating the retrieved database records. Here, an alternative implementation for retrieving the result could instead involve the use of the *Iterator* interface, which is a part of the standard Java API. The last two issues seemed to occur because the implementations varied from what the participants expected based on their previous Java programming experience. In contrast, HealthKit does not require knowledge such design pattern implementations within the SDK.

Documentation: Robillard[2009] states that: *"A major result of the survey is that resources topped the list of obstacles to learning APIs. This is a good reminder that efforts to improve the usability of an API's structure need to be complemented by efforts to improve the resources available to learn them"*[49]. Among the available resources mentioned, the main resource for learning is the documentation of the Toolkits and the corresponding APIs provided by the Toolkit providers. The findings consisted of occurrences of statements regarding the benefit of using the documentation of HealthKit and SDH, both positive and negative. For HealthKit there were 7(all participants) occurrences in the primary findings related to the documentation specifically, and 2 in the supplementary findings. The occurrences included 3 participants describing the documentation as helpful and descriptive, and 6 participants criticizing its lack of descriptiveness. These findings are validated by 1 and 3 occurrence(s) in the supplementary findings, respectively. While for SDH, there were 9(all participants) occurrences in the primary findings, and 8 in the supplementary findings. The occurrences included 9(all participants) participants describing the documentation as helpful and descriptive, and 3 participants criticizing its lack of descriptiveness. The supplementary findings validate these findings by 8 and 3 occurrences respectively. According to these findings is SDH's documentation more descriptive and useful when learning its APIs, when compared to HealthKit.

Discussion of Quantitative Findings

In this section the quantitative findings presented in section 6.5.2 will be discussed for both Apple HealthKit and SDH, and compared to determine differences and similarities between the two. These findings include the measurements M1-M4 based on data from the observation screen recordings and the data from the questionnaires filled out by the participants.

Observation screen recordings

M1: The average percentage of time spent in documentation was 43,1% for HealthKit and 41,2% for SDH, resulting in a difference of 1,9%. These numbers show that for both Toolkits, the participants used a significantly less amount of time in the documentation than in the IDE, but the difference between the two is insignificant.

M2: M2 describes the correlation coefficient between the average number of tasks completed and the average of the ratio between time spent in documentation and time spent in IDE, averaged over the set of participants. For HealthKit the correlation was 0,7294. According to the citation introduced in section 6.5.2 stating that "*Any coefficient between 0.3 and 0.7(plus or minus) is regarded as demonstrating a reasonable correlation*"[44, p. 258], is this measure sufficient to conclude that there is a strong correlation between the two variables. I.e. in the case of HealthKit, increased time spent by a participant in documentation leads to an increased number of tasks completed.

With SDH, the measure was 0,1564, indicating a weak positive correlation between the variables with the given data set, but not sufficient to make any conclusion.

M3: M3 describes the number of participants solving at least one task within the given time limit. For HealthKit this measure was 7 out of 7 participants(100%), and with SDH 6 out of 9(66,7%), resulting in a difference of 33,3% in HealthKit's favour. As a significantly larger portion of participants developing with HealthKit managed to solved at least one task, it is reasonable to assume that the developers were more productive developing with HealthKit than with SDH.

M4: M4 describes the average number of tasks completed by the participants. For HealthKit the result was 2,57 tasks, and it was 1,00 for SDH, resulting in a difference of 1,57 tasks(39% of the total tasks). We consider that a difference in 39 percentage points in HealthKit's favour is sufficient to conclude that the participants were more efficient and productive in their overall development with HealthKit than with SDH.

Questionnaires The quantitative findings from the questionnaires with the participants in the observation study presented in section 6.5.2, consisted of the resulting mean average score(score) for each of the five questions answered and all five in total, for HealthKit and SDH individually. The D-T Scale[55] was used for scoring, with a score range of 1-7.

The mean average score of all five questions in total was 4,6 for HealthKit and 4,8 for SDH, resulting in a difference of 0,2 in SDH's favour. The difference is not significant, but the numbers show that both of the Toolkits got an above average score on questions concerning documentation and overall development. Concerning the scores of individual questions, only the score of question 4(Q4) and question 5(Q5) had significance as they were the only questions where a large variation in the scores was identified. For Q4, the difference between the two Toolkits were 0,7 in SDH's favour, i.e. the developers rated the structure of the documentation significantly better. For Q5, the difference between the two Toolkits were 0,8 in SDH's favour, i.e. the developers rated the descriptiveness of the documentation significantly better.

Both Q4 and Q5 were related to the Toolkits' documentation, and according to the D-T Scale the participants had mixed feelings about HealthKit's documentation, but were mostly satisfied with SDH's documentation. While on the remaining questions, concerning the overall development and the ability to become familiar with the Toolkits, the score difference between the two was negligible and participants were mostly satisfied with both.

We end this section by summarizing the discussion and comparison of the qualitative and quantitative findings of HealthKit and SDH. Both Toolkits simplified the overall task development process for the sample of participants involved in the study. Confusion due to the relations of classes in the class hierarchy also occurred with both of them. SDH proved to have a more intuitive naming of classes and functions, while the higher level of implementation caused less confusion in HealthKit compared to the class implementations in SDH. Regarding their documentation, a strong indication was found that SDH's documentation was more descriptive as there were significantly less occurrences of issues related to its structure. The quantifiable measures implied that HealthKit is more productive in terms of number of task completed and the number of participants completing at least one task. In addition, a strong positive correlation was identified between the time spent in the documentation and the number of tasks completed in development with HealthKit.

9.1.4 Chapter 7 Discussion

Our findings from the questionnaire with experienced developers presented in this section are essential to understand the way mHealth Toolkits affect experienced developers. The findings from the questionnaire will also contribute to answer research question 3(RQ3). As mentioned in chapter 7 and section 7.3; the participation was lower than we expected. Even though we used a variety of communication channels to reach out to developers, the number of participants was lower than expected. Reasons might be that there are few people with experience with the Toolkits in question and that our email did not reach the right recipients. A consequence of the low sample size is that the resulting findings of the questionnaires are uncertain, especially those found from quantitative data. We found three themes when analyzing the qualitative data. In the remainder of this section, the quantitative data will be discussed, before these three themes will be elaborated.

For the questions with quantitative data from the D-T Scale(1-7)[55], the results show that Apple HealthKit has the highest total average score with 5,20, which is higher than both SDH and Google Fit, with 4,30 and 4,60 respectively. The main reasons why Apple HealthKit has the highest average score are questions Q4 and Q5, which concern the documentation of the mHealth Toolkits. The fact that Apple HealthKit has a significantly higher score on the questions regarding the documentation gives reason to believe that Apple HealthKit has a better documentation than the other two Toolkits in question. Documentation is as described in section 2.2 a very important aspect of an Toolkit/API. As stated in Bloch[2006], "*Documentation matters. No matter how good an API, it won't get used without good documentation*"[21]. On another side, as earlier mentioned, we cannot deduce such statements with 100% certainty because of the low sample size in the questionnaire.

The first theme we found from the qualitative data analysis was complaints about bugs and stability of the mHealth Toolkits. The reliability of data storage is key when developing an application. When a developer states the following about Apple HealthKit "*However, it has had so many bugs, and has many to this date, that it's frustrating to use. Hence, we try to avoid relying on it*", it is reason to question the reliability of Apple HealthKit. The same goes for SDH, due to a developer that stated the following: "*Bad SDK code bring bad code to project. Library fails periodically in production*". The problem with these answers is that they only state that the mHealth Toolkits has bugs, but does not specify what kind of bugs. These bugs might have been fixed by the mHealth Toolkit providers, or they might be continuous problems that the providers have been unable to fix. If a user experiences a bug, it might not necessarily be an actual bug, just deficient Toolkit/API design. Clarke[2004] states, as discussed in section 2.2, that the affordance of a Toolkit/API is crucial in Toolkit/API design. The developer needs to be able to perceive all affordances of the Toolkit/API. The failure to do so might lead to the developer experiencing the Toolkit/API as buggy.

The second theme we found in section 7.3 is the added value of data storage and sharing of the Toolkits. The comments and overall impression from the participating developers described in section 7.3 shows that the data storage and sharing features of mHealth Toolkits are highly appreciated by the developers. The positive effect of the data storage and sharing features is that developers can make an mHealth application without the need of a backend service and also collect data from other applications. These findings are aligned with the findings in Greenberg[2007] where one of the findings is that the removal of low-level implementation simplifies the development process and allows the developer to give more focus to the high-level functionality of the application resulting in an improved application.

The last theme we found in section 7.3 is the API design of the mHealth Toolkits. Developers perceived the design of the APIs and the documentation differently. Some had positive experiences and some had negative experiences. We received three answers that concerned this theme. Two of them were negative comments about the API design of Google Fit. The negative comments gives reason to believe that the API design of Google Fit is not intuitive for developers. If the API design of Google Fit is not intuitive for the developer, the developers might prefer other options. As described in previous work in section 2.2, is a good API design essential for it to be easy to learn and use for developers.

9.2 Research Questions

In this section, we will revisit and answer the research questions we initially stated in section 1.2. In the following four subsections, each of the research questions will be answered based on our findings and discussion. They will be addressed in the same order as they were presented in section 1.2.

9.2.1 Research Question 1

The following paragraphs will review and answer research question 1(RQ1) described in section 1.2. The research question is as follows: *What separates the technical architecture, data models and application development process of the leading mHealth Toolkits and what aspects affect productivity for a developer?* The research question is answered by reviewing the findings from chapters 4, 5, 6 and 7.

The technical architecture and data models of the evaluated mHealth Toolkits differ in a few aspects. Firstly, Apple HealthKit and SDH store health data encrypted on the device, while Google Fit only stores health data in its cloud storage. SDH also has an optional possibility to synchronize data from the device to cloud. The types of data stored also differ between the three; compared to the other two, Google Fit explicitly state that it should only be used to store fitness data. When delving into the architecture of each Toolkit, we discovered that HealthKit provides data type classes for each category of data with accompanying data sample classes. On the other hand does SDH provide a clear distinction between classes and interfaces in it SDK. Google Fit and SDH both provide developers with a way to create custom data types, while HealthKit developers are restricted to predefined data types.

Regarding their application development process, there are certain differences between the Toolkits. The setup process is more extensive with Google Fit than with the other two. SDH is slightly more difficult to set up than Apple HealthKit where the functionality works straight out of the box. As earlier mentioned, does Google Fit store data in the cloud and connects to a REST API to access and store data, thus differing from HealthKit and SDH. These discoveries combined with the fact that Google Fit does not allow storage of health data, lead to our decision to exclude Google Fit from further investigation in our research. When it comes to the two remaining Toolkits, HealthKit requires fewer code lines than SDH. Based on our findings, we conclude that HealthKit provides the developer with more abstraction than SDH.

Based on the abovementioned aspects, we can conclude that mHealth Toolkits make developers more productive. Based on our findings, there is a low entry cost for developers both for Apple HealthKit and SDH. While SDH is considered simple, HealthKit on the other hand provides a higher level of abstraction, which in terms of development time and code lines means more productive programmers. However, there are issues regarding the stability of both SDH and Apple HealthKit as discovered through the questionnaire with experienced developers.

9.2.2 Research Question 2

In this section we will review and answer research question 2(RQ2) described in section 1.2. The research question is as follows: *How do new adopters perceive the mHealth Toolkits in terms of documentation and development process when creating a fundamental mHealth application?* This research question is answered by reviewing the findings and discussion from the observation study conducted with students.

The findings revealed that the participants found both Apple HealthKit and SDH useful when solving problems, with both Toolkits providing abstractions of low-level implementations of functionality, HealthKit more so than SDH. From the findings from the observations, we discovered that the participants using Apple HealthKit on average completed more tasks(1,57 tasks more) than with SDH, and all of them completed at least one task, compared to SDH where only 66% managed to do the same. Both sets of participants managed to eventually grasp the main Toolkit concepts, although there were several occurrences of confusion regarding what classes to use. The class confusion was more prominent with HealthKit; the participants were confused by its naming conventions and certain classes with similar names, thereby making it difficult to differentiate the classes. With SDH the main issue was the implementation of permissions and retrieving read results from store.

Regarding the documentation, the participants had mixed feelings. Some liked the provided documentation and some found it difficult to understand. When extracting themes regarding challenging aspects of the documentation we identified the following themes: concreteness, lack of examples, structure, navigation issues, documentation eased task solving, descriptive documentation, not descriptive documentation and example confusion. Apple received a lower score on the questionnaire we gave the students with regards to the documentation. On average, the participants also spent more time in the HealthKit documentation than in the Samsung documentation, with 43,1% for HealthKit and 41,2% for SDH.

9.2.3 Research Question 3

In this section we will review and answer research question 3(RQ3) described in section 1.2. The research question is as follows: *What is the view on mHealth Toolkits seen from the experienced developers and new adopters based on their experience?* This research question will be answered by reviewing the findings and discussion for the questionnaire answered by experienced developers and the observation study conducted with students.

Our answer to this question is based on the findings from the both the questionnaire answered by experienced developers and the questionnaire answered by new adopters(participants of the observation study). From the experienced developers, we found that all three Toolkits (Apple HealthKit, Google Fit, and SDH) provided a simple way to read and write data. While the Toolkits did simplify this process, some found HealthKit and SDH to be unstable abd unreliable due to bugs in the source code. When asked to rate their overall thoughts according to the D-T Scale[55], HealthKit scored the highest among the experienced developers, where they on average were mostly satisfied/pleased. Regarding Google Fit and SDH the general opinion was mixed/mostly satisfied among the experienced developers. Overall, the new adopters were mostly satisfied with both HealthKit and SDH, with SDH scoring 0,2 higher mainly due to HealthKit's lower scores on the questions regarding its documentation.

9.2.4 Research Question 4

In this section, we will review and answer research question 4(RQ4) described in section 1.2. The research question is as follows: *What mHealth Toolkit is best equipped to be utilized in development of a mHealth service in terms of developer productivity?* This question will be answered by reviewing the findings and discussion from the development of an example system with Apple Healthkit and SDH.

Our findings show that using Apple HealthKit when developing a mHealth service leads to both shorter development time and fewer code lines, compared to development with SDH. The collected data shows that using Apple HealthKit leads to fewer code lines than SDH for the whole subset of functionalities implemented. The findings give reason to believe that Apple HealthKit is better equipped to be utilized in development of a mHealth service in terms of developer productivity.

9.3 Research limitations

In this section, we will evaluate the research in terms of significance and the existing knowledge on the subject. We will discuss the significance of our contribution and what changes that could have been made to increase the degree of its significance.

We chose to include Google Fit in the study even though it does not directly address health data. The reason is that Google is one of the main actors in the technology world today and it is reasonable to assume that they want to take part in the future of the mHealth industry. Even though it is only addressing fitness data currently, it might change its scope to include health data in the future. Google is a networking company, while Samsung and Apple have their primary focus on hardware, which may be the reason why they have chosen their respective data storage mechanism. Google uses a cloud architecture in order to make the users dependent on their data storing network. Apple and Samsung, on the other hand, makes the users dependent on their hardware devices by storing the data on the devices. As we have seen in this study, the most realizable of the two abovementioned is the on-device storage-approach, due to the success of Apple HealthKit with the Mayo Clinic and the previous failure of Google Health.

Regarding the development of the example system, the chosen design was based on the architecture of the pre-existing Mayo Clinic App[18] because we felt it reflected a prime use case of the technology. Despite this, our solution could have been improved by reaching out to hospitals in order to establish what requirements were needed to develop a more applicable and realistic system.

The sample set of participants selected for the observation study described in Chapter 6, were selected according to convenience sampling resulting in recruiting computer science students from our university. This strategy was advantageous because it let us recruit a larger sample size and spend significantly less time recruiting, but the drawback is that these students were acquaintances of us, resulting in our handling of and communication with them being especially prone to subjectivity. This subjectivity might have affected the formality of the conducted observation study and the data generation process, which could have been avoided by recruiting a more varied sample set.

In the questionnaire answered by experienced developers, we tried to contact the developers in question for follow-up questions. We wanted the participants to elaborate their textual answers, but it was difficult to reach out to these developers since the answers were given anonymously. We could have anticipated that we would receive such answers and incorporated follow-up questions into the questionnaire beforehand. A better solution would have been to give the participants the possibility to leave their email in the questionnaire for follow-up questions. Then we could have sent individual follow-up questions to the participants in order to make them elaborate their answers.

In spite of the abovementioned issues that could have been addressed differently, the findings from the research are still valid. Some of the findings could have been augmented further in order to make them more comprehensive. The research is significant because, as mentioned in chapter 1.3, there is a lack of previous work on mHealth Toolkits. We found related papers regarding Toolkit/API usability and improved efficiency of Toolkits, but none of them were directly related to mHealth or any of the evaluated Toolkits. It is important to emphasize mHealth because it is a growing field.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

In this study, we have studied the phenomenon of developers' efficiency and productivity in the context of their development of mobile health applications. The units of the analysis performed to describe the given phenomena and its context were the following mHealth Toolkits: Apple HealthKit, Google Fit, and Samsung Digital Health.

In order to investigate the selected mHealth Toolkits we composed four research questions, concerning the development process of experienced developers and new adopters, and the fundamental differences between the evaluated Toolkits. The research questions were answered by the findings generated by a literature study that reviewed previous work on the field, development of an example system, an observation study with students, and an online questionnaire with experienced developers.

We have found that when designing a mHealth Toolkit to be utilized by developers when developing mobile health applications and services, there are several concerns that need to be addressed in order to increase the efficiency and productivity of the developers, including aspects related to the usability of the Toolkits' API and their corresponding available resources. In light of these concerns, we conclude that the present state of Apple HealthKit, Google Fit and Samsung Digital Health is not satisfactory from the view of application developers requiring a Toolkit that enhances their performance. Despite this, the evaluated Toolkits show potential and if they address the identified concerns they might be viable solutions in foreseeable future.

10.2 Future Work

This section will describe the authors' thoughts with regards to future work, based on the experiences from the research. Certain steps on what future work will consist of will be elaborated in the paragraphs below.

As mentioned in previous chapters, did some of the experienced developers that answered the questionnaire complain about bugs in the mHealth Toolkits. As we mentioned in section 9.3, we should have reached out to the developers in question for follow-up questions. We did not do so, hence further work both for the field and the Toolkit providers would be to identify these bugs of the Toolkits and address them. The abovementioned can be achieved with more in-depth questionnaires and interviews with developers, or by performing system testing of the Toolkits. It is important to investigate these bugs further because making developers able to rely on the mHealth Toolkits as primary storage for mHealth applications is key to the success of the Toolkits.

Developing a healthcare service by utilizing mHealth Toolkits should, in addition to be tested with large data quantities, be evaluated by inspecting whether or not the encryption process affects application performance and thereby increasing the amount of issues that the developers have to address. In addition, it would be beneficial to develop applications similar to the described BG-app and PH-app, only without the help of the Toolkits, and then compare the results to further evaluate the Toolkits' impact on developer productivity and creativity.

Bibliography

[1] Community: Using iphone | apple support communities. https://discussions.apple.com/community/iphone/using_iphone.

[2] Github - googlesamples/android-fit. https://github.com/googlesamples/android-fit.

[3] Google fit - fitness tracking - android apps on google play. https://play.google.com/store/apps/details?id=com.google.android.apps.fitness.

[4] Google fit | google developers. https://developers.google.com/fit/.

[5] Google fit developers - community - google+. https://plus.google.com/communities/103314459667402704958.

[6] Healthkit framework reference. https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Framework/index.html#//apple_ref/doc/uid/TP40014707.

[7] Id-porten | eid.difi.no. http://eid.difi.no/nb/id-porten.

[8] ihealth portable wireless blood glucometer - apple. http://www.apple.com/shop/product/HJ152ZM/A/ihealth-wireless-blood-glucometer-with-50-test-strips?fnode=4a.

[9] Official google blog: An update on google health and google powermeter. https://googleblog.blogspot.no/2011/06/update-on-google-health-and-google.html.

[10] S health - android apps on google play. https://play.google.com/store/apps/details?id=com.sec.android.app.shealth.

[11] Samsung introduces the next generation of s health, expands compatibility. http://www.samsungmobilepress.com/2015/09/22/samsung-introduces-the-next-generation-of-s-health,-expands-compatibility-to-other-android-devices.

[12] Why is the health app so slow? : iphone. https://www.reddit.com/r/iphone/comments/2sj1w9/why_is_the_health_app_so_slow/.

[13] (2012). Samsung introduces s health application for gsiii. http://www.samsung.com/uk/news/local/samsung-introduces-s-health-application-for-galaxy-s-iii.

[14] (2014). 25 experts weigh in on the apple health app.

[15] (2014). Introducing healthkit - wwdc 2014 - videos - apple developer. https://developer.apple.com/videos/play/wwdc2014/203/.

[16] (2015). Google platform overview. https://developers.google.com/fit/overview.

[17] (2016). ihealth gluco-smart on the app store. https://itunes.apple.com/us/app/ihealth-gluco-smart/id571576516?mt=8.

[18] Anderson, B. (2014). Mayo clinic news network. http://newsnetwork.mayoclinic.org/discussion/mayo-clinic-app-integrated-with-apple-health-now-available-for-free-download/.

[19] Apple. Healthkit - apple developer. https://developer.apple.com/healthkit/.

[20] Apple. Hkhealthstore class reference. https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HKHealthStore_Class/index.html.

[21] Bloch, J. (2006). How to design a good api and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507. ACM.

[22] Brent, R. No, phones aren't more accurate than fitness wearables | wired. `http://www.wired.com/2015/03/fitness-tracking-test/`.

[23] Buttarelli, G. (2015). Reconciling technological innovation with data protection.

[24] Clarke, S. (2004). Measuring api usability. *Doctor Dobbs Journal*, 29(5):S1–S5.

[25] Crockford, D. (2006). The application/json media type for javascript object notation (json), 2006a. *URL http://tools. ietf. org/html/rfc4627.*

[26] Cross, D. Google fit is a bit sh*t - it doesn't sync! * dan is cross. `http://www.daniscross.co.uk/2015/06/google-fit-is-a-bit-it-does-not-sync/`.

[27] Dictionary.com. Toolkit search definition. `http://dictionary.reference.com/browse/toolkit?s=t`.

[28] Gill, P., Stewart, K., Treasure, E., and Chadwick, B. (2008). Methods of data collection in qualitative research: interviews and focus groups. *British dental journal*, 204(6):291–295.

[29] Google. Fitness data types | google fit | google developers. `https://developers.google.com/fit/android/data-types`.

[30] Google. Terms and conditions | google fit | google developers. `https://developers.google.com/fit/terms`.

[31] Greenberg, S. (2007). Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159.

[32] Guest, G., Bunce, A., and Johnson, L. (2006). How many interviews are enough? an experiment with data saturation and variability. *Field methods*, 18(1):59–82.

[33] Henning, M. (2007). Api design matters. *Queue*, 5(4):24–36.

[34] Holzworth, E. and Stokes, A. How will apple healthkit and google fit affect health apps? an illustrated guide. http://littlegreensoftware.com/blog/mhealth/how-will-apple-healthkit-and-google-fit-affect-health-apps-an-illustrated-guide

[35] Kawulich, B. B. (2005). Participant observation as a data collection method. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*, volume 6.

[36] Kay, M., Santos, J., and Takane, M. (2011). mhealth: New horizons for health through mobile technologies. *World Health Organization*, pages 66–71.

[37] kevinjalbert. Anyones health kit app stop working? | apple support communities. https://discussions.apple.com/thread/6568048?start=165&tstart=.

[38] MA, C., HA, B., KG, V., and MS, P. (2015). Accuracy of smartphone applications and wearable devices for tracking physical activity data. *JAMA*, 313(6):625–626.

[39] marketsandmarkets.com (2015). Mhealth solutions market by connected devices & services – 2020 | marketsandmarkets. http://www.marketsandmarkets.com/Market-Reports/mhealth-apps-and-solutions-market-1232.html.

[40] Marshall, M. N. (1996). Sampling for qualitative research. *Family practice*, 13(6):522–526.

[41] Miles, M. B. and Huberman, A. M. (1994). *Qualitative data analysis: an expanded sourcebook*. Sage Publications.

[42] Morse, J. M. (2000). Determining sample size. *Qualitative health research*, 10(1):3–5.

[43] NatashaTheRobot.com (2014). Healthkit: Getting fitness data. https://www.natashatherobot.com/healthkit-getting-fitness-data/.

[44] Oates, B. J. (2005). *Researching information systems and computing*. Sage.

[45] Orzechowski, J. Why is mhealth important? | university of illinois at chicago. http://healthinformatics.uic.edu/resources/articles/why-is-mhealth-important/.

[46] PCMag. Toolkit definition from pc magazine encyclopedia. http://www.pcmag.com/encyclopedia/term/52987/toolkit.

[47] Pope, C., Ziebland, S., and Mays, N. (2000). Analysing qualitative data. *Bmj*, 320(7227):114–116.

[48] PwC (2012). Emerging mhealth: Paths for growth.

[49] Robillard, M. P. (2009). What makes apis hard to learn? answers from developers. *Software, IEEE*, 26(6):27–34.

[50] Schaefbauer, C. 3 painful lessons learned building with healthkit. http://www.openmhealth.org/3-painful-lessons-learned-building-with-healthkit/.

[51] Scheller, T. and Kühn, E. (2012). Influencing factors on the usability of api classes and methods. In *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, pages 232–241. IEEE.

[52] scuderiadank Reddit, U. Having problems with google fit? don't expect a fix any time soon : Android. https://www.reddit.com/r/Android/comments/3aa2f3/having_problems_with_google_fit_dont_expect_a_fix/.

[53] Von Hippel, E. and Katz, R. (2002). Shifting innovation to users via toolkits. *Management science*, 48(7):821–833.

[54] Waring, I. An initial dive into apples new health app (and healthkit api). http://www.ianwaring.com/2014/06/05/an-initial-dive-into-apples-new-health-app-and-healthkit-api/.

[55] Westbrook, R. A. (1980). A rating scale for measuring product/service satisfaction. *The Journal of Marketing*, pages 68–72.

[56] Zahid, I., Ali, M. A., and Nassr, R. (2011). Android smartphone: Battery saving service. In *Research and Innovation in Information Systems (ICRIIS), 2011 International Conference on*, pages 1–4. IEEE.

Appendix A

Acronyms

API	Application programming interface
App(s)	Application(s)
BG-app(s)	Blood Glucose application(s)
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of things
Jar	Java Archive
Js	Javascript
JSON	JavaScript Object Notation
mHealth	Mobile Health
NoSQL	Not only Structured Query Language
NTNU	Norwegian University of Science and Technology

OODB	Object-oriented Database
OS	Operative System
PH-app(s)	Patient-Hospital application(s)
PwC	PricewaterhouseCoopers AS
REST	Representational State Transfer
RQ	Research Question
SDH	Samsung Digital Health
SDK	Software Development Kit
TDT	Subject prefix for Computer science classes at NTNU
UI	User Interface
USD	United States Dollars
UUID	Universal Unique Identifier
WHO	World Health Organization
XML	Extensible Markup Language

Appendix B

Access to generated Data

In order to be as open as possible regarding our research, data generated from the observation study through observation, interviews and questionnaires can be accessed using the URL displayed below.

URL: http://bit.ly/thesisDataTR

Appendix C

End User Issues

The use of health platforms is currently in the start phase and as a result, there are some issues and problems experienced by the users. In the following section, the issues with the platforms from end user's point of view will be addressed. The user reviews of each of the platform's application[3, 1, 10] will be evaluated to shed light on what kind of issues that end users are experiencing when using the application. These problems are crucial for the developers to keep in mind when they develop applications with the toolkits. There were three main categories of issues that the users were experiencing, which included: data accuracy and synchronization, battery drain and the number of third-party applications that are supported by the different platforms.

C.1 Accuracy and Synchronization

According to reviews and forum posts[3, 1, 10], the first group of issues users experience are related to data accuracy and synchronization problems. Some of the equipment used for tracking seems not to be accurate enough. For example, some users write in their reviews that smartwatches and smartphones are not accurate enough when tracking step count and distance. Sensors tracking health-related data should be accurate because inaccurate sensors lead to unsatisfied users. Blog posts and articles question the accuracy of both wearable trackers and smartphone trackers for fitness data[38, 22]. In the research, they are mainly concerned with step count, which is a metric for fitness tracking. The user has no option to validate the step count sensor data, but as long as the data has small deviations from the actual value, it will still be useful for the user. The demand for accuracy is higher in the health sector than with fitness, for example if hospitals were provided inaccurate metrics and sensor data it might lead to a misdiagnosis and potentially fatal consequences. Therefore are accurate sensors that interconnect with the mHealth platforms crucial to the success of these platforms. Partnerships with hardware providers are essential in developing accurate sensors, and mHealth platform providers need to realize the importance of this area to get accurate sensors and hardware that delivers measurements with a low degree of uncertainty.

According to posts in the forums and reviews[3, 1, 10], some data that is tracked with different devices tend to be lost when synchronizing the different devices. Lost data is an important issue because the applications need to be accurate to attract users. One example is the synchronization between the smartwatch and smartphone with Google Fit, where steps recorded on the watch tend to be lost. Readings from the phone seem to be preferred over readings from the smartwatch[26, 52, 5]. The synchronization problem has been there since Google Fit was released in October 2014. If one goes for a run carrying only the smartwatch, the steps might be lost when the watch is resynchronized with the phone after the run. Google are currently working on the problem, but there exists an opinion that there's not put enough resources into fixing the bug[5]. This may again lead to fewer and more unsatisfied users of Google Fit. Another example is when several users lost all or some of their historical data after an update of the S Health application. This problem shows that users are very interested in keeping their historical data in a safe place and always have access to the data.

C.2 Battery Drain

Another problem several users are experiencing is that the automatic tracking function of the three applications discussed drains much battery power of their smartphones. The application has a built-in automatic tracker of position and steps. The position tracking is highly dependent on the utilization of GPS-signals from the phone. GPS signals are very power consuming for the phone[56]. The different platform providers are constantly working on improving the applications to maximize the utilization of the GPS signals. They are working towards reduced and more efficient usage of GPS data.

C.3 Third-party Connectivity

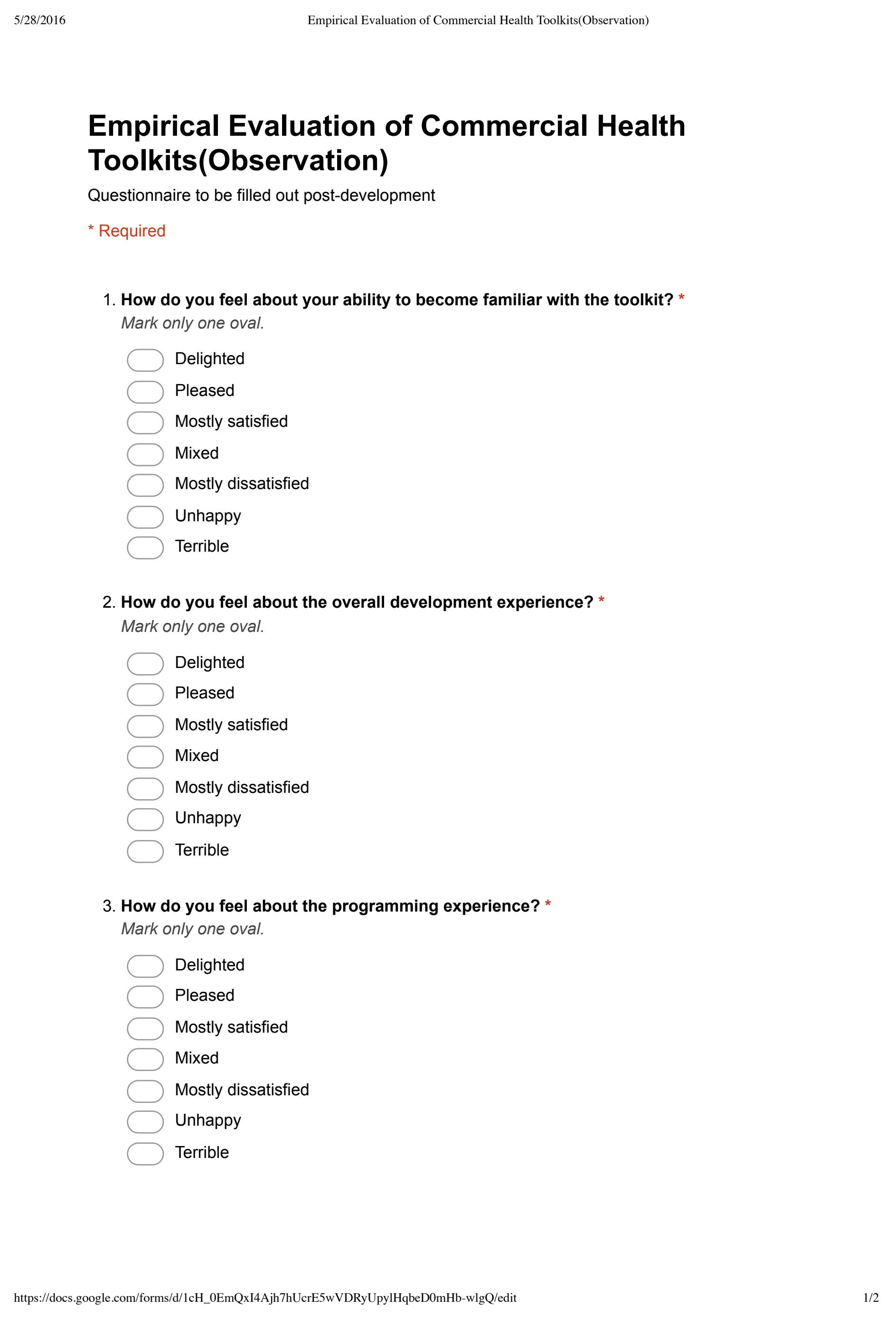
As earlier discussed, many applications are opening up for the possibility of integration with the different platforms. There are still some third-party applications lack support, and some of the users complained about it in their reviews, e.g. the lack of support for integration with FitBit which is the case for all three applications. For example, if a user has a FitBit for tracking fitness data, the applications would be useless because they are not compatible with each other. The obvious solution would be to provide integration with the various platforms that lack support. The problem is that the other devices have a competing solution, and the competitors will not give up their solutions. The solution to it all could be a universal exchange protocol for fitness data, which makes all devices compatible with all platforms.

Appendix D

Questionnaires

Below are the PDF versions of the questionnaires used in data gathering process of the thesis

D.1 Observation Study Questionnaire



Empirical Evaluation of Commercial Health Toolkits(Observation)

Questionnaire to be filled out post-development

* Required

1. **How do you feel about your ability to become familiar with the toolkit?** *
   Mark only one oval.

   ◯ Delighted
   ◯ Pleased
   ◯ Mostly satisfied
   ◯ Mixed
   ◯ Mostly dissatisfied
   ◯ Unhappy
   ◯ Terrible

2. **How do you feel about the overall development experience?** *
   Mark only one oval.

   ◯ Delighted
   ◯ Pleased
   ◯ Mostly satisfied
   ◯ Mixed
   ◯ Mostly dissatisfied
   ◯ Unhappy
   ◯ Terrible

3. **How do you feel about the programming experience?** *
   Mark only one oval.

   ◯ Delighted
   ◯ Pleased
   ◯ Mostly satisfied
   ◯ Mixed
   ◯ Mostly dissatisfied
   ◯ Unhappy
   ◯ Terrible

4. **How do you feel about the structure of the documentation?** *

Mark only one oval.

- ⬭ Delighted
- ⬭ Pleased
- ⬭ Mostly satisfied
- ⬭ Mixed
- ⬭ Mostly dissatisfied
- ⬭ Unhappy
- ⬭ Terrible

5. **How do you feel about the descriptiveness of the documentation?** *

Mark only one oval.

- ⬭ Delighted
- ⬭ Pleased
- ⬭ Mostly satisfied
- ⬭ Mixed
- ⬭ Mostly dissatisfied
- ⬭ Unhappy
- ⬭ Terrible

6. **Would you consider using the toolkit in further work?** *

Mark only one oval.

- ⬭ Yes
- ⬭ Maybe
- ⬭ No

D.2 Experienced Developers Questionnaire

5/28/2016 Empirical Evaluation of Commercial Health Toolkits

Empirical Evaluation of Commercial Health Toolkits

Questionnaire for developers with experience

* Required

1. **Do you have any previous development experience with Apple HealthKit?** *
   Mark only one oval.
   - Yes
   - No *Skip to question 10.*

Apple HealthKit

2. **How do you feel about the overall development experience?** *
   Mark only one oval.
   - Delighted
   - Pleased
   - Mostly Satisfied
   - Mixed
   - Mostly dissatisfied
   - Unhappy
   - Terrible

3. **How do you feel about your ability to become familiar with with the toolkit?** *
   Mark only one oval.
   - Delighted
   - Pleased
   - Mostly Satisfied
   - Mixed
   - Mostly dissatisfied
   - Unhappy
   - Terrible

https://docs.google.com/forms/d/1PZjexdvok53rr31faLehIiuPkY1D4zYhL_BniTQyJY8/edit 1/7

4. **How do you feel about the programming experience?** *
Mark only one oval.
- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

5. **How do you feel about the structure of the documentation?** *
Mark only one oval.
- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

6. **How do you feel about the descriptiveness of the documentation?** *
Mark only one oval.
- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

7. **Did the Toolkit increase your efficiency when creating a product?** *
Mark only one oval.
- Yes
- No

8. **IF found efficient, in what way did it increase your efficiency?**

9. **How would you describe the Toolkit's effect on development creativity?** *

....

....

....

....

....

Google Fit

10. **Do you have any previous development experience with Google Fit?** *
Mark only one oval.

- ⬭ Yes
- ⬭ No *Skip to question 19.*

Google Fit

11. **How do you feel about the overall development experience?** *
Mark only one oval.

- ⬭ Delighted
- ⬭ Pleased
- ⬭ Mostly Satisfied
- ⬭ Mixed
- ⬭ Mostly dissatisfied
- ⬭ Unhappy
- ⬭ Terrible

12. **How do you feel about your ability to become familiar with with the toolkit?** *
Mark only one oval.

- ⬭ Delighted
- ⬭ Pleased
- ⬭ Mostly Satisfied
- ⬭ Mixed
- ⬭ Mostly dissatisfied
- ⬭ Unhappy
- ⬭ Terrible

13. **How do you feel about the programming experience?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

14. **How do you feel about the structure of the documentation?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

15. **How do you feel about the descriptiveness of the documentation?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

16. **Did the Toolkit increase your efficiency when creating a product?** *
Mark only one oval.

- Yes
- No

17. **IF found efficient, in what way did it increase your efficiency?**

18. **How would you describe the Toolkit's effect on development creativity?** *

Samsung Digital Health

19. **Do you have any previous development experience with Samsung Digital Health?** *
Mark only one oval.

- Yes
- No *Stop filling out this form.*

Samsung Digital Health

20. **How do you feel about the overall development experience?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

21. **How do you feel about your ability to become familiar with with the toolkit?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

22. **How do you feel about the programming experience?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

23. **How do you feel about the structure of the documentation?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

24. **How do you feel about the descriptiveness of the documentation?** *
Mark only one oval.

- Delighted
- Pleased
- Mostly Satisfied
- Mixed
- Mostly dissatisfied
- Unhappy
- Terrible

25. **Did the Toolkit increase your efficiency when creating a product?** *
Mark only one oval.

- Yes
- No

26. **IF found efficient, in what way did it increase your efficiency?**

27. **How would you describe the Toolkit's effect on development creativity?** *

....

....

....

....

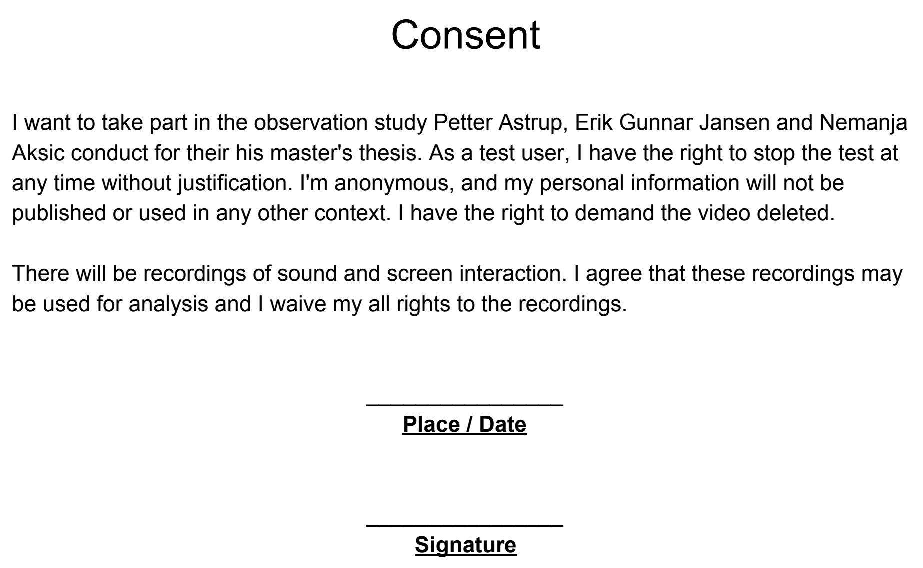
....

Appendix E

Consent Form

In order to be able to use sound and video recordings for the observation study, the participants had to fill in consent forms. Below is the PDF of the consent form used for the thesis.

E.1 Consent Form



Consent

I want to take part in the observation study Petter Astrup, Erik Gunnar Jansen and Nemanja Aksic conduct for their his master's thesis. As a test user, I have the right to stop the test at any time without justification. I'm anonymous, and my personal information will not be published or used in any other context. I have the right to demand the video deleted.

There will be recordings of sound and screen interaction. I agree that these recordings may be used for analysis and I waive my all rights to the recordings.

____

Place / Date

____

Signature