placeholder

Master thesis Computer Science

Dennis Cardinaels

Promotor: Professor Mieke Haesen

Assistant: Jens Brulmans

2018 – 2019

Preface

TODO

Abstract

TODO

Contents

1 Introduction

2 The Electronic Health Record

Health information systems have become an integral part of health care. They support patient care as well as administrative and financial tools. At the heart of these systems lies the electronic health record. An electronic health record (EHR) is a repository of electronically maintained information about an individual's health status and health care, stored such that it can serve multiple legitimate uses and users of the record[1]. An electronic health record system (EHRS) provides tools to manage and interact with these records. These tools include reminder generation, data analysis, and decision support. It helps the clinician to organize, interpret and react to medical data. A record where the health information is managed by the patient, is called a personal health record[2].

2.1 Moving away from paper

For modern medicine, traditional paper-based medical records are not suited for today's world filled with technology. The drawbacks of information on paper are obvious when compared to digitally stored information.

Storage Paper records need to be stored in a safe location and require a lot of physical space. More so, the organization of these records is a daunting task in case they are fragmented across multiple locations. This is a process that wastes time and therefore increases health care costs. Also, losing or misplacing records is a possibility. If this for example happens to lab results, tests have to be redone which again wastes time and increases costs. Finally, the amount of paper added to the records on a weekly basis is significant and very eco-unfriendly[3]. The ink and paper costs should not be underestimated. Storing data in a digital format solves these issues, but raises several other questions such as: shall we store the data on premise or choose a cloud provider? What is our backup policy?

Data quality The medical information available on paper is static and more effort is required from clinicians in processing it. For example, comparing data values by reading two forms side by side. What if the data is illegible due to poor handwriting or worn down over time? What if data is incorrectly read from the computer display or simply not written down? These medical errors directly impact the quality of care that is delivered[4][5]. Computation is possible for digital records which allows data aggregation, data processing and statistics generation. Also, a computerized system detects false data input and ensures that all data fields are filled in correctly. A summarizing paper noted that the use of EHRS leads to more complete, accurate, comprehensive, and reliable data compared to paper-based records[6].

Medical data such as printed x-ray images are prone to data loss, which in turn increases the chance of medical errors. A high resolution monitor allows zooming, whereas the paper is severely limited in this regard. In case of audio

files, bundling for example a cassette tape with a paper dossier is problematic in terms of storage and transportation. The last two examples can both be stored digitally and paired to a digital record.

Accessibility Paper records are bound by their location and can often only be accessed by one person at a time. Transferring the paper record requires a lot of manual work: either the clinician sends the actual record or a copy of it, or converts the record to a digital format. One study described that updating several copies of medical records was a very cumbersome process for clinicians[7]. If such a copy is updated, how can these changes be reflected in the other existing copies? Where are these copies even located? Converting the paper records to a digital format raises several other questions: where can the data values found in the paper record be stored in the digital system? What if there are extra notes scribbled on the document? What if values are missing? Data transfer and dealing with changes seems simpler for digital records, however the interoperability of different systems is complex. This is discussed in section 2.3.

Security While paper records can be safely stored under lock and key in a basement, several security issues still exist. For example, the storage location can be broken into. This allows the perpetrator to retrieve, alter, or destroy critical medical information. Other possibilities of data loss include flooding and fires. These are scenarios that can't be predicted and can happen at any time.

A common strategy to detect data alteration and prevent data destruction is to create regular backups. However, for paper-based medical records this is simply not feasible. As mentioned before, the resources required to create copies of such a large data set is enormous. For digital records this is a much easier process, but in this case other security issues arise. These are discussed in section FILl. Also, data breaches have important privacy implications. This topic is discussed in section 2.6.

Electronic health record systems deal with most of the issues mentioned in the previous paragraphs. However, there are other factors which are responsible for the continuing presence of paper in clinical environments. Three categories are predicted to be the cause of paper generation: policy requirements, suboptimal system design, and EHRS user interface flaws[3]. The study focused on the last source with the following subcategories:

- Cumbersome interface design leads to handwritten notes on paper.
- The EHRS is not well integrated into the clinical workflow. This leads to paper-based workarounds which *do* align with the workflow.
- The visual organization of the data in the EHRS is incompatible with the mental model of the clinician, which leads to manual transformation of patient data.

Notice that these subcategories relate to usability and human-computer interaction issues. Twenty persons practicing varying roles in a clinical environment were interviewed concerning the use of paper in their workflow. Afterwards, recurrent paper-based workaround strategies highlighted during analysis were categorized. Eleven distinct categories were found, ordered by frequency of use:

- Efficiency: paper use enhanced perceived or actual efficiency.
- Workarounds related to the clinician's knowledge of the health system, skill level with technology, and ease of use of the health system. Example: using paper because the software is difficult to use.
- Memory: cases where paper is used as a reminder tool.
- Sensorimotor preferences: using paper as a means of having something concrete to deliver or to quickly jot down some notes.
- Awareness: using paper to make clinicians aware of new information.
- Task specificity: cases where the health system lacks specificity, is not customizable, or sends too many alerts (alert overload).
- Task complexity: paper processes are used because the health system does not support it. An oncology order is an example of a complex task, tailored specifically for each patient.
- Data organization: using paper when data is poorly organized on the computer screen.
- Longitudinal data processes: using paper instead of the health system when data needed to be tracked over time.
- Trust: using paper as a form of providing proof the health system cannot provide.
- Security: using paper when the health system is insecure.

While these workarounds can improve efficiency, they also circumvent the health system which minimizes medical errors frequently made when dealing with paper records. However, they also imply that the EHRS is not in line with the workflow of the clinicians. Several of these workarounds are caused by poor usability, which is the topic of section 2.4.

2.2 Components

As mentioned before, EHRS do not simply store patient records. They consist of many components which ultimately define how well they perform in health care. We define the following five functional components as key[1]:

Integrated view of patient data An EHR must allow storage of a wide range of data types. This can be text, numbers, images, video, and others. Some data can still be on paper due to lacking support of the EHRS, as mentioned in section 2.1. Data standards were developed to store simple and complex data types, ranging from numeric values to x-ray images. Medical data standards are the central topic of section 2.3.1.

Clinician order entry The procedure in which the clinician enters treatment instructions is called order entry. An order entry system assists the clinician during the decision-making process to ensure that the instructions are correct. It reduces medical errors as a result of more complete and correct data as mentioned before in section 2.1. Also, costs associated with the use of paper are saved.

Clinical decision support A decision support system aids the clinician by suggesting actions when certain situations occur. If for example, a patient is due for vaccination, the system notifies the clinician by presenting a pre-filled order which only needs to be confirmed or denied. The system can do this for a bulk of patients, so manual checkups are not required, thus saving time. Decision-support implementations benefit from artificial intelligence. As such, these systems are often complex, due to the many parameters involved in the decision-making process.

Access to knowledge resources Clinical questions often arise during the clinician's workflow. Instead of asking colleagues or searching through multiple manuals, the clinician can consult the EHRS for literature. Thanks to the internet, large sources of information are readily available. Also, if the EHRS is aware of the current medical context, searching for the required information takes even less time.

Integrated communication and reporting support Communication lies at the heart of health care delivery. It is common for patients to receive care from several clinicians spread across multiple institutions or departments. As mentioned in section 2.1, using paper as a form of communication should be avoided. Therefore, the availability of communication tools directly affects the quality of care. The use of standards and high interoperability of health systems facilitate efficient communication (section 2.3).

Each of these components must be present in health software. This example of a prescription management tool, features all five components:

- Present list of medication currently taken in a clear manner.
- Create, edit, or remove prescriptions.
- Provide alerts for drug-drug and drug-allergy interactions. Suggest medication dosage with respect to parameters such as weight.

- Provide extra information regarding medicines, such as side effects.
- Send prescription orders to the pharmacy.

An EHRS features many tools such as prescription management and many examples are given in section 3.4. Ideally, all tools are bundled in a single software package. In case this is not possible, multiple applications are used to provide all the required functionality. In section 2.3 the advantages and drawbacks of both scenarios are discussed.

2.3 Interoperability

Health care institutions can opt for a monolithic EHRS or combine multiple EHRS to achieve all required functionality. There advantages and drawbacks to both[8]. Elements that influence this choice include IT infrastructure, safety risks, the volume of care, and frequency with which patients move facilities. It is possible that a single EHRS does not satisfy the requirements of an institution. A reason for this is that certain branches require specially tailored software for their clinical practice, which the current EHRS in use lacks.

Functionality wise, a monolithic EHRS tends to appeal to more general types of care, whereas it lacks in very specific ones. Also, vendors that offer these all-in-one solutions tend to have less experience with these special types of care which reduces the chance it will be added to the system. Vendors of specific EHRS do have this expertise and can tailor the system to the needs of the customer. In this case, combining a system that supports general care with special care systems seems like the best choice. However, other factors have to be considered.

The advantage of a monolithic system is that the data it uses is centralized. This ensures that all data is accessible anywhere throughout the system and is easier to maintain. When multiple systems are in place, data has to be exchanged between them. As a result, searching for data is more difficult. As mentioned in section2.1, due to different data structures, data exchange is difficult. To solve this issue, an intermediate EHRS can be developed. This system serves solely for the purpose of data collection and transformation. All other systems search for data in this intermediate system. However, this leads to another system, requiring additional costs, development effort, and maintenance.

2.3.1 Standards

When excessive diversity creates inefficiencies and affects effectiveness standards are required[1]. A hospital contains many independent units spread across primary, secondary, and tertiary care. These units use software best fit for their practice and all record different types of data. For example, the hospital admissions system records patient diagnosis, the pharmacy records prescriptions that were handed out, and the laboratory system records test results. Inevitably, transfer of data between these units is required.

To coordinate multiple systems, data transfer is essential. Nowadays too many different systems exist to create point-to-point interfaces for. Standards try to resolve this problem by defining guidelines software system should follow in order to communicate data. An efficient standard requires that data can be easily stored and presented towards the users of an EHRS. Also, security measures, such as authentication and access control, need to be interwoven with these standards.

The use of standards in an EHRS leads to easier integration which in turn leads to lower development costs. Integration of many proprietary data structures leads to difficult to maintain software and a higher chance of components breaking. New medical devices and software that comply to these standards prevent this.

One of the most used and implemented standard set is Health Level 7. These messaging-based standards are created to exchange clinical and administrative data. Another standard such as DICOM is used for the communication and management of medical imaging information[9]. Medical device interface standards are created by IEEE. Many more standards exist, but these are the more prominent ones.

2.4 Usability

2.5 Recent advancements: telehealth

2.6 Privacy

Privacy refers to the desire of a person to control the disclosure of personal health and other information[1]. On the other hand, confidentiality is the ability to control the release of personal health information to a care provider under the agreement that the information will not be spread or used further. Security is the protection of privacy and security, which is achieved through policies, procedures, and safeguards.

We can ask several questions related to health information data access and storage: who owns the data? Is it the health provider or the patient? What type and how much of data needs to be stored? Who can read the data? Who can write to the data? Can someone access specific health information without consent? In order to deal with the some of the challenges concerning data access and storage, these questions need to be answered[10].

An important issue surrounding privacy is medical data access. For example, researching the medical data of a large population can uncover looming threats or an epidemic. Also, disease incidence and intervention analysis benefits from data pooling. However, medical data should not be too accessible. As more and more people gain access to health data of the population, the chance that data is misused or confidentiality is broken increases. Therefore, striking a balance between free information access and protection of privacy and confidentiality is difficult. An argument that promotes free access is to anonymize the data by removing identifiers such as the person's name. However, the individual pieces

of information can still reveal the their identity.

Compared to paper records, electronic records are easily accessible, such as through the internet. This calls for extra security measures to prevent data breaches. If someone wants to access a medical record, this person must first authenticate himself. In turn, the system checks if the person is allowed to access this information. Regardless of the outcome, the audit system logs the access attempt. Such systems need clear rules and policies defined by the institution on who can access what. Another obvious security measure is data encryption.

To further prevent malicious use of health data, medical staff should follow education and training programs to make them aware of the privacy rules and policies that are in place. In case of privacy violation, the person in question should be punished appropriately.

3 Design

fdsafdsa

3.1 Proposal

The proposal features four core concepts which will be combined to form one cohesive solution: a dashboard, telemonitoring of chronic diseases, customization, and integration. As mentioned in the introduction, multiple applications and integrating them pose several difficulties: data is spread and stored multiple times. Interaction between these applications is initially not present and integration with existing systems requires a significant amount of work from the IT staff of the institution. As more applications are added, this becomes increasingly difficult. To remedy this problem, each application becomes a module which can be plugged into the dashboard system.

A module-based approach towards the dashboard allows easy customization and integration. Each module will serve its own purpose by showing data for one particular health parameter. As mentioned in section FILL, parameters such as heart rate and blood pressure are often measured to monitor multiple chronic diseases. Integration is handled by the fact that each module is independent and does not require other modules to work properly. This also helps with customization as a clinician can choose which modules are relevant for the patient in question.

The modules that will be developed are for use in chronic disease monitoring. However, modules can be developed for purposes more fit for in-patient care. Examples are real-time monitoring, documenting diagnoses, and viewing of lab results. Which modules will be implemented and what each one of them tries to achieve is described in section FILl. The implementation section will provide more insight on how this independence is achieved and how future modules can be easily integrated.

This combination allows the clinician to customize the dashboard according to the needs of each patient, facilitating effective care. Data is found rapidly, albeit summarized or detailed, and not hidden away behind multiple screens. Developers should be able to create modules that will fit into the dashboard, negating the use of multiple applications. The following section describes how the system will work and how the users interact with it.

3.2 Guiding principles

The translation of the aforementioned proposal to an intuitive prototype is guided by several principles. It was paramount during the design and implementation phases that the final prototype provided a positive user experience. This final prototype is the result of a series of steps defined by the MuiCSer framework, which is explained first. Hereafter, several design principles are mentioned that serve as helpful guidelines during the design process of a dashboard application.

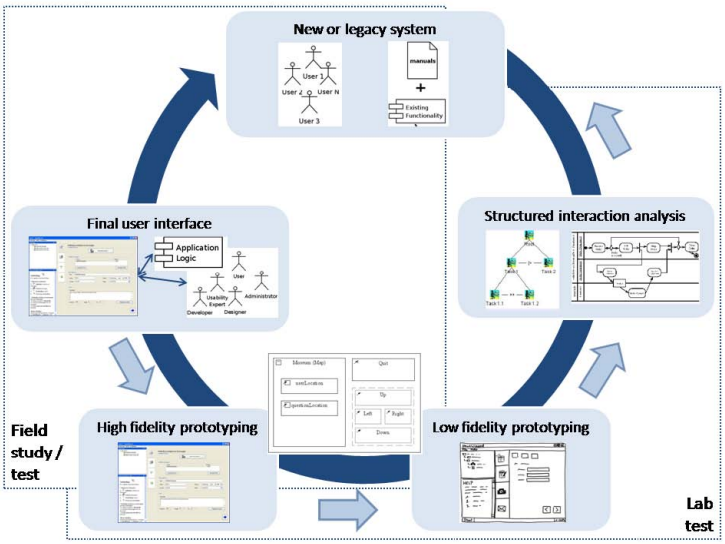

Figure 1: The MuiCSer process.

3.2.1 MuiCSer

This section describes the MuiCSer framework, designed for user-centered software engineering processes in a multidisciplinary context[11]. This framework focuses on optimizing the user experience during the entire software engineering cycle to ensure that the needs of the end user are fulfilled. By combining user-centered design methodologies and software engineering principles, the user experience of the final product can be improved substantially. Given the multidisciplinary context of this thesis, MuiCSer guided the entire development process leading to the final prototype described in section 4.

The MuiCSer process is summarized in figure 1. After each phase, the result is evaluated, verified and validated to ensure that the required functionality is present. In turn, the received feedback can be used to reiterate over the previous phase. On the figure, this is denoted with the light blue arrows, while the single dark arrow represents the overall direction of the process. What follows is a brief description of each phase:

1. New or legacy system The MuiCSer process starts with either an existing system in need of improvement, or with a new system which needs to be built from the ground up. This requires an analysis of the tasks and needs of the user, as well as the objects and resources required to perform these tasks. Personas and scenarios are the resulting artifacts of this phase. First, personas describe the personalities of the potential end users including hobbies, skills and the environment they surround themselves in[12]. Its goal is to uncover behavior patterns which can be of use when designing a user interface. Second,

scenarios tell stories describing the use of a fictitious system from the point of view of a persona[12]. To summarize, personas and scenarios try to sketch the usage of the system for which a design must be made. These artifacts are found in section 3.3.

2. Structured interaction analysis During this phase, the results of the analysis are used to create task models. These models specify concrete tasks and goals which can be dissected into specific actions or steps the user has to take. These artifacts lay the foundation for designing a user interface which supports these tasks and goals. Task models were not created. Instead, via a brainstorm session (section 3.4), common tasks of a health system were grouped to form modules. The user interface will be designed to support these modules.

3. Low fidelity prototyping After the task definition, low fidelity prototypes are created. Paper sketches and mockups are examples of low fidelity prototypes. Without spending too much time and resources, presenting these prototypes to the end-user or customer can yield valuable feedback. However, there is often no functionality present. Typically multiple versions of these prototypes are created until the customer is satisfied, after which the development of a high fidelity prototype starts. Low fidelity prototypes were created for all modules defined during the brainstorm session and are found in section 3.5.

4. High fidelity prototyping The development of high fidelity prototypes requires a lot more effort compared to low fidelity prototypes, as they offer functionality closely resembling the final product. However, the end user can now test both the design *and* the functionality. This yields much more meaningful feedback. Section 4 describes the development of the high fidelity prototype.

5. Final user interface When the latest iteration of the high fidelity prototype satisfies all user requirements, the final user interface can be created. The high fidelity prototype may serve as the starting point of the final product in order to save time and resources. As a final step, the task models are checked against the resulting interface to check if all required functionality is present. No final product was created during the period of this thesis. However, the results of the usability test described in section 6 should determine if it is feasible to commit further research towards the prototype.

3.2.2 Dashboard design

The proposal mentions a dashboard. However, dashboards are used in many contexts. Common examples include a car speedometer and stock analysis. In computer networking, dashboards are used to quickly interpret network traffic. In our case, we want to empower the user to create his or her own dashboard. To facilitate this goal, several design principles should be kept in mind.

While some dashboards provide fancy graphics, many miss the key point of what they are supposed to accomplish. The primary goal of a dashboard is clear communication, which is achieved through effective design. A formal definition is as follows: "A dashboard is a visual display of the most important information need to achieve one or more objectives; consolidated and arranged on a single screen so information can be monitored at a glance."[13]. This definition contains four key characteristics:

Dashboards are visual displays. It is important that data is represented visually. Graphics such as charts allow more efficient communication compared to textual information. For example, trends and outliers are easier spotted in a line chart when compared to a table containing the same data. Instead of using different visualizations for the sake of variety, one should *always* choose the visualization which best suits effective communication of the data in question[14].

Dashboards display information needed to achieve specific objectives. The data that is shown, must be relevant to the job at hand. Sometimes data needs to aggregated from multiple sources after which it can be tailored according to the context wherein it must be presented. This manner of processing data yields effective data visualizations.

A dashboard fits on a single computer screen. In order to see as much information at a glance, scrolling should be prevented. If multiple screens are present, then it is no longer a single dashboard. This leads to another question: what type of display is the information shown on? In today's world, computer displays come in many shapes and sizes. Therefore, a responsive layout can be very beneficial, but scrolling becomes inevitable as the screen decreases in size. As an example, it is impossible to present the same dashboard built for a Full HD monitor on a smartphone display. While the latter display may have the same resolution, the content will be too small to read comfortably.

Dashboards are used to monitor information at a glance. Important data should be immediately noticeable, whereas specific details should be hidden. Therefore, by summarizing or aggregating the data a more effective visualization may be achieved. However, if the user wishes to view the detailed data, the dashboard should provide means to do so. Also, careful thought must be given to what information is of importance so it is never hidden.

To create an effective dashboard, the user-base must be well known and understood. What type of users are we dealing with? What are their characteristics? These are important questions to ask, as one user may not comprehend one visualization while the other can. Therefore, the focus should be put on the user during the design of the dashboard[15]. We can ask the following questions to better understand the needs of the user:

- What metrics does the user need to see?

- What context does each metric require to make it meaningful? Do we need to visualize the variance, target to reach, trend. . . ?
- What visualization best communicates the metric?

On that end, sketches and mockups are helpful during the design process. Multiple iterations each incorporating feedback received from the users, ensure that the end result is satisfactory.

3.3 Personas & scenarios

The first step of the MuiCSer involves creating personas and scenarios. These artifacts should give us a better understanding of the system's users and how they will use it. Each persona and scenario pair tries to highlight problems the users face and how the new system tries to solve them.

3.3.1 Jake

Persona Jake is a 25-year-old male who currently works as a nurse at the hospital in his city. He has been working there for four years and lives alone in his apartment. Still being a young adult, Jake grew up with technology. As a result, he experiences no difficulties when using new software on his computer or smartphone. His hobbies include music, playing the guitar, and video gaming.

Currently, the workflow at the hospital is dated. A new health information system was introduced to summarize and gather all medical data in a single program. Because this is an all-in-one system, a lot of features that Jake does not need still clutter the screen. Navigating the system is difficult and customization is not present. Jake wishes to only view the features he uses most while hiding the features he does not need.

Scenario Jake starts his first day using the new system by reading the manual that is accompanied with it. He starts the application which shows a list of patients for which he can create a personalized dashboard. When Jake opens the dashboard associated with a patient, he notices that many modules can be selected.

After adding a few modules to the dashboard, Jake has all the functionality he needs to do his work. During this process, he added a wrong module which was easily removed. Hereafter, Jake resizes and moves the modules until he is satisfied with the layout. Over time, Jake updated the dashboard of several patients by adding a module due to changes in his workflow. All he had to do was to open the module list and select the module he needed.

By examining the new module Jake quickly notices that it looks similar and is customizable in the same manner as the other modules. Jake feels he is in control of the system and convinced it will boost his productivity.

This scenario highlights the benefits of customization. Hiding unwanted and adding useful components allow for a clear dashboard to be displayed. The user is in control and is able to quickly view data of importance.

3.3.2 Dan

Persona Dan is a general practitioner since he graduated from university. He is on the job for 21 years and he is the preferred doctor in his town. Throughout the years Dan has used a multitude of systems and he always tries new ones to improve his workflow. Because he has been a general practitioner for such a long time, he has 500 patients that visit him at least twice a year. Some patients, especially the elderly, visit as much as once a month.

Most of Dan's patients visit for illnesses such as fever and a cold. To diagnose these illnesses, a description of the patient's symptoms suffices most of the time. If Dan performs such a diagnosis, he promptly adds it to the electronic health record of the patient.

However, if a patient visits that has a lot of problems regarding his blood pressure, then Dan has to perform a more complex diagnosis involving historical data. In the current system that Dan uses, it is very difficult to search for and work with this data. But what bugs Dan the most is that he has to do it every time this patient visits.

Scenario Dan tries a new health information system which allows customization for each specific patient. Because the diagnoses of most patients can be very different from time to time, Dan creates a default module group which is displayed for each patient, unless that patient has a specific module configuration. The default module group includes past diagnoses, known allergies, patient information (blood type, last weight, height...), and a current medication list.

The first patient of the day describes what sounds like a fever. Dan confirms this and hands the patient a prescription. The diagnosis is added to the respective module and the prescribed medication to the module containing prescription data. Dan did not need other health data to perform the diagnosis. Therefore, Dan does not change the configuration of this patient.

The next patient, an elderly woman, visited Dan for the second time this month. Dan knows from the past that it will probably be a heart related issue. Dan searches for a 'heart' module and adds it to the configuration of the elderly woman. Now both the default module group and a heart rate module are present, which is unnecessary according to Dan. He removes some modules of the default module group. Now, the next time the elderly woman visits, that configuration will be automatically loaded.

One specific patient had broken his arm three times in less than a year. When the patient came for a routine visit, Dan immediately added a module to easily view x-ray photos and view them in a timeline.

Providing customization options as described in this scenario has immediate benefits. Initially, it will take some time to configure each dashboard for all

the patients, but the system will try to make this process quick to complete. Once the configuration is finished, time spent during a consultation can decrease dramatically.

3.3.3 Emily

Persona Just like Dan, Emily has been a successful general practitioner for quite some time. However, she has different needs of an EHRS. Between each patient visit, there is a period of time in which Emily does not know what happens to patients regarding certain parameters. For example, if a person regularly measures his heart rate and blood pressure because of cardiovascular disease, then it is important that the doctor is made aware of these values. If Emily notices a negative trend, she can contact the patient to schedule an early consultation.

In this case, Emily would like to visually see the progression of the heart rate and blood pressure values. Also, easy side-by-side comparison of both parameters would be useful. Currently, she needs to navigate from one screen to the other to do the comparison.

Scenario Emily recently received notice of a new platform that allows 3rd party applications to easily integrate with it. Now, a mobile application can send new values towards the platform. This allows Emily to monitor the patient remotely, which saves precious time for both parties. The platform can be customized to display multiple charts of data values.

A patient who recently had a cardiac arrest is continuing rehabilitation at home. However, the patient has chest pains and pays Emily a visit. She tells the patient to regularly measure his blood pressure and heart rate, and to take note of these values in a mobile application. After they have scheduled the next visit, the patient is sent home. As the next few weeks pass by, Emily notices a climbing trend on this patient's blood pressure chart. She tells herself to regularly check up on this patient as there is currently no reason to panic.

After another two weeks, Emily takes another look at the blood pressure data. The chart clearly shows that the blood pressure values keep rising. Emily decides to call the patient to schedule an early visit. The system helped Emily to intervene as soon as the situation seemed to worsen.

This scenario gives an example of interoperability and highlights the effect of data visualization. By allowing the caregiver to monitor the patient at home the onset of a disease can be detected.

3.3.4 Anna

Persona Anna is a software engineer working at the local hospital. From the early 2000's, she was responsible for the integration of health software systems. As such, she has experienced the continuous change associated with health software firsthand. Each unit of the hospital uses different software, while they all

share a central system to view patient records. The interaction between these applications is one of Anna's responsibilities.

As time goes on, new software and medical instruments that improve workflow are purchased. Sometimes these replace old systems, otherwise they are added to work alongside them. Both are becoming increasingly difficult to do, as the codebase of the EHRS grows. When an older system gets replaced, Anna has to weed through the code and delete little pieces, hoping other parts wont break. It is a frustrating task as often band-aid solutions are applied.

Scenario A new EHRS was installed that should improve integration of other applications. Anna skims through the documentation and starts the transition process. She quickly notices that for each device or application she needs to create a back end data structure and a module that represents it in the EHRS. The new EHRS can integrate with minimal downtime.

After the initial transition period, everything is put into place. Not soon after, new bedside monitors with accompanying software were purchased. These replace the old bedside monitors and consequently, the software. As a first step, Anna reworked the back end data structure to work with the readings from the new device. The data of the old device is converted to the new data structure, so no data is lost. Anna designed the new module to be similar in appearance and functionality. After implementing the module, the staff barely notices any change and workflow resumes with minor disruption.

This persona and scenario highlight the importance of interoperability. Significant time and effort can be saved if this process is simplified. Also, maintenance is easier due to looser coupling of the modules.

3.4 Module definition process

3.5 Application definition

3.5.1 General structure

3.5.2 Modules

4 Implementation

This section describes the implementation process. The design highlighted the usage of the dashboard through its personas and scenarios, while the low fidelity mockups gave an early glance at its appearance. However, at the heart of this application lies the ability to customize and easily integrate new components. To provide these features, a lot of thought went into choosing the best libraries fit for the job.

First, an overview of the global scope of the application is given, after which we go into more detail. The dashboard consists of two major parts common to web development. The back end is described first. This section covers topics such as the chosen frameworks, the REST API, and data storage. For the front end part, the framework which supports our module-based design and its underlying principles are explained. The contribution of both parts towards a customizable dashboard which allows simple integration, is explained further in their respective sections. It is without question that during the implementation small changes were made to the design. These changes are also documented for both parts. During the entire development process, no effort was put towards privacy safeguards and standard support as they are out of the scope of this thesis.

4.1 Overview

The first major choice that was made, was the type of application to develop: a native application or a web application. It was a relative simple choice, but nonetheless an important one. Native applications are developed, in most cases, for one type of operating system. If a developer wants to support both Android and iOS devices, separate applications need to be developed for each of them. Maintaining multiple applications requires a lot more development effort. Java, for example, has the benefit that it is cross-platform thanks to its virtual machine. However, regardless of platform, platform specific changes need to be made to tackle bugs. One of the largest drawbacks of native applications is updating them. This process often requires the reinstallation of the application. As a result, rolling out software updates in hospitals is a difficult task.

Web applications saw a recent surge in popularity. Web applications today are sometimes very similar in user interface and functionality. An example is Office 365 Web Apps, which shares a lot of functionality from its native counterpart. While these web applications also require updates, these happen on the server side. When the user refreshes the web page, the latest version of the application is automatically loaded. This simplifies the roll-out process significantly. Also, all operating systems that support web browsers, are capable of running this type of applications. As a result, developers only have to develop one version. Mobile devices, therefore, are capable of viewing these applications. However, to fit their small screen, developers need to support a flexible layout.

With this in mind, the choice to create a web application was evident. Users can view the dashboard regardless of the device they use, without sacrificing

functionality. It should be noted that complex image processing applications will not fit easily into the dashboard, due to their computational complexity. Web applications are limited in this regard. Transferring the computational burden to the server side also has the drawback of latency, which the user can experience as slow.

A typical web application consists of two parts: the back end and the front end. In software engineering principles, these refer to the separation of the data access layer and the presentation layer of the web application. For example, the back end fetches data requested by the front end. In turn, the front end presents the retrieved data towards the user. Due to the modular design of the dashboard, each module needs to have its own back end (data retrieval and database models) and front end functionality as summarized in figure. As seen in the figure, there is no interaction between the modules. Furthermore, the front end has a general presentation in which the modules reside. The next sections describe the back end and front end in detail. Also, before the implementation began, I only had basic knowledge of HTML, CSS, and JavaScript. As a result, the learning curve was personally quite steep.

4.2 Back end

The back end consists of a REST API which handles the requests sent by the front end. Each request that the back end supports performs an operation on the database. Examples of such requests are fetching, posting, and deleting data. In this section we describe the REST API, its structure, and the operations it allows. Hereafter, we mention how the database program was chosen, its models and their relations.

4.2.1 REST API

Node.js was used to create the back end server. This server environment was chosen due its large user base and available libraries. The REST API itself was created with the Express framework and tested with Postman. The documentation of the REST API present in the prototype can be found in appendix. We now describe the general structure of the API and the operations it supports.

General structure The API structure starts at the user. Each user has a unique identifier which is passed on to the subroutes. To get the information of the user, we send the following request:

```
GET /user/123456
```

If there is a user present with the id '123456', the API sends a response containing the user information. Modules that want to use data from a certain user, need to prepend their request with `/user/id_value`, as this is the main route. For example, both a heart rate and a blood pressure module want to retrieve their respective values of a certain user whose id is '1a2b3c'. The modules each send the following request:

```
GET /user/1a2b3c/heart
```

```
GET /user/1a2b3c/bp
```

Each module is responsible for its own subroutes and operations it supports. As an example, the threshold values can be retrieved differently for each module, depending on the developer's choice:

```
GET /user/1a2b3c/heart/thresholds
GET /user/1a2b3c/bp/misc_values/thresholds
```

The subroutes are simple to integrate thanks to the Express framework. All the requests a subroute handles are stored in its own file. The main route imports this file so it can forward the subroute towards that module. Suppose we have a `route/heart.js` file that we want to integrate in the main route `user.js`. This is done as follows:

```
const heartRoutes = require('.routes/heart'); //import routes
router.use('/:userId/heart', heartRoutes); //forward routes
```

Only two lines are needed to import all the necessary requests from a module. The main route, and all subroutes of all modules present in the prototype are documented in appendix.

Operations The modules can create new, retrieve, delete, and update data entries. Respectively, these are the following request types: POST, GET, DELETE, and PATCH. In the prototype however, data deletion is not possible because it concerns valuable medical data. For threshold values, deletion is also not possible as every patient is required to have these. Creation of the thresholds for a patient is only possible there is currently no entry present for that patient. Again, appendix documents all implemented operations for all modules present in the prototype.

4.2.2 Database

At the very beginning, SQL was used in the back end, but it was quickly switched out for MongoDB. MongoDB uses JSON-like documents to store data. Groups of these documents are stored in collections. A feature of MongoDB is that the documents inside a collection do not need the same data fields. Suppose we update our patient information page to also show their phone number. In SQL, tables would need to be redefined and the existing rows updated to have a default value for this attribute. In MongoDB this is not necessary as patients with this new attribute can be added without a problem to the existing collection. These flexible documents facilitate integration as no schema rules need to be defined. This allows developers to create their own document schemas and pair them to existing users by solely referring to their id.

The prototype runs a local MongoDB database server where all dashboard data is stored. Each module is responsible for its own documents. As such, every module has a value collection, which contains all values of all users, and a threshold collection, where one threshold entry exists for each user in the dashboard system. No module accesses data from another module. The database is managed via the Mongoose modeling library for Node.js. By defining models,

Mongoose helps us by managing relationships between data. Also, it translates source code objects to their representation in the database.

The models present in this prototype are straightforward, as shown in figure Fig. It is possible that one blood pressure entry can have an additional field, such as whether it was seen or not by the clinician. However, the current implementation of each module, ignores extra data fields. If in the future data fields are added, the front end part of the module must adapted in order to present these to the clinician.

4.3 Front end

Ultimately, the front end is what the user sees. From the onset of the implementation process, the search for a library which supports a modular approach began. It was clear that Web Components would be used to create the individual modules, which will be explained in the next section. There are several libraries that support the creation of Web Components. Google's Polymer library was eventually chosen, due to its extensive documentation. The following sections describe the Web Component principles and the structure of the front end. For the latter, the customization aspect and the library used to achieve this are explained.

4.3.1 Web Components

HTML provides us with standard elements such as `h1` and `img`. Web Components allows the developer to create custom elements which can be used in the same manner as the ones HTML provides. Suppose a calendar Web Component was created, a developer could add the element to any web page as follows: `<calendar></calendar>`. The actual definition of the calendar module resides in another file which is imported on the page where it will be used. This approach supports reusability, and above all, easy integration which is a primary goal of our dashboard application. At the heart of Web Components is the shadow DOM.

Shadow DOM We assume the reader is familiar with the DOM concept. The shadow DOM can be seen as a scoped subtree inside a custom element. The root of this subtree is called the shadow root. The structure, styling, and behavior of the children within the shadow DOM do not affect any elements outside of it. Therefore, the appearance and functionality of the custom element is encapsulated and hidden at the document level. Only events fired by elements in the shadow DOM can be pickup outside the shadow DOM boundary. Figure Fig visualizes this concept. This idea of encapsulation is important for our approach towards the dashboard, as each module is responsible for its own appearance and functionality. Developers of custom modules should not need to worry about what happens outside of the shadow DOM.

4.3.2 Structure

The dashboard features two panels. On the left, only small modules can be pinned, while both the small and large modules can be placed on the right panel. The left panel is thin and is automatically hidden in case of small screen estate. Should too many modules be present on the right panel, a scroll bar appears. The left panel will stay in place if scrolling happens in the right panel to show the information of the small modules that reside there. Figure Fig shows the empty dashboard.

The right panel allows the ordering of the modules via drag-and-dropping them into place. Several libraries were tried out to achieve this functionality, but many of them did not support dragging Web Components. As a result, the front end was rebuilt several times. Eventually, Packery.js was used. This library allows free placement of the modules and tries to arrange them to maximize space efficiency. In case of resizing, the modules are rearranged automatically. Modules are added via a dialog which can be accessed by pressing the plus-sign button in the top-right corner. The same applies for the smaller panel on the left side. However, the modules in this panel can only be realigned vertically.

4.4 Result

Throughout this section multiple screenshots show the final prototype in action. Extra screenshots can be found in appendix Ref, as some modules are very similar in design and functionality. The threshold settings menu is found at the start of that appendix. Changes visible in a screenshot compared its low fidelity counterpart, are explained in the paragraph where it is referred to. First, the small and large modules are shown. Hereafter, multiple complete dashboards are presented to show the different possibilities in terms of layout.

4.4.1 Small modules

Figure Fig shows four small modules, with each showing data of different time periods. For easy arrangement, all small modules have the same width. Note that there is no heart rate data for the time period selected in that module. The only changes present in the small modules are also applied to the large modules. One of these changes was the addition of a small handle bar at the top of each module. This allows safe dragging and dropping. Previously, the entire module was draggable, which resulted in unexpected button behavior, but could also lead to accidental dragging.

The second and last change, was the replacement of the menu button in the top-right corner. This menu was replaced by buttons for each action it would contain: resize, update thresholds, and delete. As a result, a click is saved. The unit of measurement, which was originally in the title, had to be moved to the tables below to make room for the buttons.

4.4.2 Large modules

The larger modules saw additional changes on top of the two mentioned in the previous section. When the large modules were completed, the right-hand side had a lot of whitespace. To remedy this, the statistics were placed beneath the chart. This had the added bonus that the modules became narrower. All large modules have a fixed width of 716 pixels, which is the minimum screen resolution width without needing to scroll horizontally.

The charts were generated using the C3.js chart library. Not mentioned in the design, most charts feature a line showing the average of the values for the selected time period. Indicating the lowest and highest values is difficult to do in C3 and was left out of the prototype. Hovering over the plotted data points shows the exact values, as shown in figure Fig for medication. Labelling the axes and indicating the measurement units was overlooked and added later for all charts. All charts are simple to understand and no "chart junk" is present.

4.4.3 Dashboard

As mentioned in section 4.3.2, different screen sizes are supported, and the left panel is hidden if the available screen space is too narrow. This section shows several examples of dashboards on smaller and larger screens. The empty dashboard was already displayed in figure Fig. The plus-signs on top open the menu seen in figure Fig. Only small modules can be added to the left panel. Figure Fig shows a very wide dashboard, which was automatically resized on a smaller screen to the one shown in figure Fig. If the screen is even smaller, the left panel is hidden as seen in figure Fig. Pressing the button in the top left corner shows the left panel on top the other modules (figure Fig). Extra dashboard layouts are found in appendix Fig.

5 Usability test

6 Discussion

7 Conclusion

A Appendix

A.1 Usability test documents

References

[1] Edward H. Shortliffe and James J. Cimino. *Biomedical informatics: Computer applications in health care and biomedicine: Fourth edition.* 2014.

[2] Paul C. Tang, Joan S. Ash, David W. Bates, J. Marc Overhage, and Daniel Z. Sands. Personal health records: Definitions, benefits, and strategies for overcoming barriers to adoption, 2006.

[3] Jason J. Saleem, Alissa L. Russ, Connie F. Justice, Heather Hagg, Patricia R. Ebright, Peter A. Woodbridge, and Bradley N. Doebbeling. Exploring the persistence of paper with the electronic health record. *International Journal of Medical Informatics*, 78(9):618–628, 2009.

[4] Shereef M. Elnahal, Karen E. Joynt, Steffanie J. Bristol, and Ashish K. Jha. Electronic health record functions differ between best and worst hospitals. *American Journal of Managed Care*, 2011.

[5] Richard Hillestad, James Bigelow, Anthony Bower, Federico Girosi, Robin Meili, Richard Scoville, and Roger Taylor. Can electronic medical record systems transform health care? Potential health benefits, savings, and costs. *Health Affairs*, 24(5):1103–1117, 2005.

[6] Kristiina Häyrinen, Kaija Saranto, and Pirkko Nykänen. Definition, structure, content, use and impacts of electronic health records: A review of the research literature. *International Journal of Medical Informatics*, 77(5):291–304, 2008.

[7] H. J. Tange. Consultation of medical narratives in the electronic medical record. *Methods of Information in Medicine*, 38(4-5):289–293, 1999.

[8] T. Payne, J. Fellner, C. Dugowson, D. Liebovitz, and G. Fletcher. Use of more than one electronic medical record system within a single health care organization. *Applied Clinical Informatics*, 2012.

[9] Peter Mildenberger, Marco Eichelberg, and Eric Martin. Introduction to the DICOM standard, 2002.

[10] Marci Meingast, Tanya Roosta, and Shankar Sastry. Security and privacy issues with health care information technology. In *Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings*, 2006.

[11] Mieke Haesen, Karin Coninx, Jan Van Den Bergh, and Kris Luyten. MuiCSer: A process framework for multi-disciplinary user-centred software engineering processes. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008.

[12] Sabine Madsen and Lene Nielsen. Exploring persona-scenarios - Using storytelling to create design ideas. In *IFIP Advances in Information and Communication Technology*, 2010.

[13] Stephen Few. *Information dashboard design: The effective visual communication of data*. 2006.

[14] Stephen Few. Intelligent Dashboard Design. *DM Review*, 2005.

[15] Richard Brath and Michael Peters. Dasboard Design: Why Design is Important, 2004.