Communication solution for ICU patients: a case study

Thesis for MSc. Computer Science, University of Twente, 20-5-2015

Researcher: J. Thunnissen (s0176745)

Graduation Committee
Dr. Pim van den Broek
Prof.dr.ir. M.J.A.M van Putten
Prof.dr.ir. M. Aksit

ABSTRACT

This thesis describes the design and development of an augmentative and alternative communication software system for use by disabled patients in intensive care units. For this challenge we adopt the principles of agile software development methodology. We repeatedly analyze the requirements from patients, medical specialists, nurses, physiotherapists and family by testing intermediate working versions of our software. For each development cycle the design challenges and evolution of earlier design are discussed. We continuously evaluate the development process and conclude that most of the agile principles effectively contribute towards solving the design problem. The final product is positively received by all stakeholders.

INTRODUCTION

The goal of this research is to tackle a software system design problem. This is done using knowledge of the software design process, as acquired by the researcher during his master Computer Science. A problem case is introduced and this case is extensively analyzed as part of the software design process. The chosen case is communication for disabled patients on an intensive care unit (ICU). The designed software system is appropriately named "CommIC". The case is chosen, because of its challenging nature and real-world demand for a solution. Going through the design process, we cooperate with doctors, patients and other stakeholders who all have different backgrounds and who often have very limited experience with computers.

Although the ICU context is the focus of the research project due its challenging environment, we aim to produce a solution that is applicable to communication problems outside of the ICU as well. In practice this means we prioritize implementation of new features not just based on demands within the ICU, but on applicability outside of the ICU as well.

Until now, the Computer Science program has only offered limited exposure to complex real-world scenarios such as this ICU problem case. Being successful requires technical engineering skills, as well as social and management skills. The development process is started by a team with two other developers, similar to most real-world software projects. Coordinating this development team to efficiently and effectively tackle the problem case is intended as an important aspect of the project. Unfortunately, quickly after the start of the project the other developers quit contributing. Only student in technical medicine J. Benistant is involved throughout the project as a minor code contributor. His involvement is quite important though, as his domain knowledge allows him to contribute many valuable insights in the problem domain.

This research proposal is structured as follows. Section 1 shortly introduces to problem case. The problem domain of this case is extensively analyzed in section 2. Here we look at the aspects of the problem domain that might be relevant for our software system. This includes creating an overview of possible patient afflictions and other limitations, an example case of patient interaction on the ICU and a description of the current use of hardware and software within hospital context. Based on our findings in section 1 and 2, section 3 identifies the main problems in the domain, followed by a preliminary requirements analysis of any solution to these problems.

In section 4 the solution domain is explored. We study existing communication technologies for people with disabilities, called Augmentative and Alternative Communication (AAC) technology. The operation and usage by existing AAC software systems is examined. We finish by determining the extent to which existing AAC solutions solve the problems identified in section 3.

The research questions are laid out and explained in section 5.

In section 6 we discuss in detail the methods and practices that are used to build CommIC. We argue for adopting an agile design approach and we discuss concretely the practices that are followed. Special attention is given to the task of requirements analysis, as we work in a complex environment with diverse stakeholders. Moreover, the ICU patient is an important stakeholder which we have to approach with great care.

Section 7 describes the development cycles. In each cycle we address the implementation of a number of user stories, short descriptions that capture a requirement of the system without volatile detail. The selection of user stories is made based on their priority and implementation cost. Throughout implementation testing of the system is given the needed attention. The design is kept as simple and clean as possible. We rely heavily on refactoring for this. The team and stakeholders are constantly in close contact to minimize the feedback loop. At the end of each iteration, the software is evaluated together with these stakeholders. From the feedback new user stories are created and existing ones are updated.

At the end of this report we evaluate the final product and discuss the answer to the last research questions (section 8). A glossary and referenced sources list can be found in section 8 and 10 respectively.

This project is supervised and graded by a graduation committee consisting of primary supervisor Dr. P.M. van den Broek, external supervisor Prof.dr.ir. M.J.A.M van Putten and chairman Prof.dr.ir. M. Aksit.

ACKNOWLEDGEMENTS

We thank Dr. Pim van den Broek and Prof.dr.ir. M. Aksit for their guidance and feedback during the project. Their knowledge of the software development process and research methods were valuable to improve the quality of the research. Credit is also due for Prof.dr.ir. M.J.A.M van Putten for contributing his perspective from the medical world.

Special thanks goes to J. Benistant, whose experience with both the technical and medical world has been invaluable in our communications with the stakeholders. He has contributed his domain knowledge and has helped presenting and testing the system on the ICU.

R. Damink has taken on the role of our primary contact within the ICU. He has guided the adoption of CommIC on the ICU, has overseen the systems usage in our absence and has helped to collect valuable feedback from medical personal. His involvement was key to the successful introduction of the system to the ICU.

Further thanks goes to V. Silderhuis, B. Beishuizen, M. Braakhuis, B. Tempert and other personal and patients of the ICU of the Medisch Spectrum Twente hospital. We highly appreciate their feedback and insights and their participation in the many trials.

CONTENTS

1 CASE DESCRIPTION

This case description has been formulated based on a discussion with our contacts at the hospital Medisch Spectrum Twente. First there is our project supervisor within the MST, Prof. Dr. ir. M.J.A.M van Putten. He is clinical neurophysiologist at MST and professor of clinical neurophysiology at the University of Twente. Our main contact within the ICU is Dr. B. Beishuizen, who qualified in ICU healthcare and has a lot of experience working with patients on the ICU. He has provided us with a lot of invaluable insights into the workings of the ICU. The source for the majority of the statements made in this case description and the following domain analyses are discussions with Dr. Beishuizen and other medical personal and patients on the ICU of the MST.

On the ICU of hospitals many patients are unable to communicate in a normal way. They often cannot speak because they are mechanically ventilated and intubated. For many different reasons, such as trauma, or ICU acquired weakness, patients can have limited or no motor control. Some can only move a finger or control their eye movement. Furthermore, patients sometimes suffer from sensory deprivation. All these conditions limit their communication abilities.

This handicap in communication can have several negative effects on the patient. His wellbeing is reduced by the inability of the patient to communicate with loved ones. Communicating with loved ones can be a great comfort for IC patients in their unfortunate and often stressful state. This is even truer in end-of-life situations. ICU patients often become socially isolated and feel helpless as a result of the lack of communication. This can lead to more stress and panic attacks. Social isolation also increases the risk of the patient suffering from delirium. (T.M. Brown, 2002)

Patients are often able to communicate by alternative means, such as eye blinking, pointing to a letter board or by forming words with their mouth. These ways of communicating are very slow, often tiresome for the patient and prone to misunderstanding. This basic alternative form communication does often not allow patients to completely and clearly communicate their message. This can cause frustration.

Apart from communication with other people in the room patients very often have limited ability to communicate with the rest of their environment. For example, they might not be able to control the TV, use a phone to call someone, switch the lights on or off or sound the alarm in case of emergency. This is cause for further social isolation, discomfort, boredom and might possibly endanger the patient. Of course, there are also a lot of patients who are too sick to feel the desire to communicate.

Medical staff is adversely affected by the communication problem. Communicating basic information with the patient, if possible, takes up a lot of their valuable time. The limited communication possibilities might make it difficult to help the patient to the best of their abilities. For example, they need to know whether the patient is feeling any pain, where this is located and how severe it is. To help make a diagnosis of the patient's health, doctors want to have questions answered by the patient.

2 PROBLEM DOMAIN ANALYSIS

2.1 PATIENT COMMUNICATION DISABILITIES

In this subsection, we make an inventory of ICU patients' afflictions that are a common cause of communication disabilities. There are both physical and mental conditions that negatively impact communication capacity. During a patient's stay on the ICU most of these conditions are subject to change, meaning, new disabilities can be formed or existing ones can become worse or be alleviated. We have roughly identified four causes. They are disease, sensory deprivation, medical treatment and mental incapacity.

2.1.1 Disease

Diseases are a very common cause for communication disabilities. They are normally the reason the patient lies on the ICU to begin with. In Appendix A a non-exhaustive, but varied list of example diseases is described, including the disabilities that they can cause. Based on the diseases we explored, we have made an overview of possible physical patient limitations that can cause communication difficulties.

Patients can have limited motor control. Either they cannot move at all, or their muscles are very weak and movements only happen very slow, tiresome and with minimal force. It is also a possibility that the patient can move parts of his body, but has limited control over the movement. He could have spasms or reduced coordination. Muscle control can be limited in any part of the body, but facial muscles and especially the eyes are often spared or only hindered in late stages of a disease.

Often a direct result of limited muscle control is difficulty with speaking. Reduced muscle control around the throat, mouth or larynx or difficulties in controlled breathing can cause a patient to be difficult to understand. Either he can only speak very softly or has difficulty with certain sounds. Possibly, speech is completely impossible.

Diseases can also cause a reduction in cognitive abilities. Patients might have difficulty remembering, might be disoriented and can have very short attention spans. Basic mental exercises such as simple problem solving or even speaking a language can be made impossible.

2.1.2 Sensory deprivation

Sensory deprivation can also be a cause for communication difficulties. With lots of elderly patients on an ICU, it is not uncommon for patients to have difficulty hearing or seeing. Possibly, patient can only hear loud noises, low tones or nothing at all. Eyesight can be reduced to a blur, difficulty to differentiate between colors or complete darkness. Naturally, reduced eye muscle control can also contribute to reduced eye sight.

2.1.3 Medical treatment

The medical treatment that a patient receives can have certain (side) effects that limit communication capability. Here we give multiple examples.

Body parts can be restrained for medical reasons. Commonly, patients should not move certain parts of their body that need rest to heal. In many cases, they are forcibly prevented from doing so. Broken bones, wrapped in plaster, are a common example.

Sedating patients against pain is very common. It can cause the patient to behave lethargic and much less receptive and responsive to external stimuli. Other medication, especially when administered in large amounts, can have similar effects on the mental state.

Mechanical ventilation is very commonly applied to prevent patients from suffocating. It can be either applied through a mask that is placed over the patient's mouth and nose, through an endotracheal tube or a tracheostomy tube. These tubes are inserted into the patient's blowpipe respectively through the mouth or through an incision in the front of the neck. Using the vocal cords to produce specific sounds is not possible during mechanical ventilation.

2.1.4 Mental incapacity

An abnormal mental state with reduced cognitive abilities or just lacking knowledge can form a great obstacle for effective communication.

As a result of the dismal situation they are in, patients experience pain, fear, anxiety, lack of sleep, tenseness, nightmares, and loneliness. It is found that patients who are mechanically ventilated through endotracheal tube frequently experienced "spells of terror, feeling nervous when left alone and poor sleeping patterns" (A.J. Rotondi, 2002). This mental stress causes patients to have very limited attention spans and to be easily exhausted.

ICU Patients often suffer from reduced functioning of the memory. After their stay, some have no recollection of their time on the ICU at all. "Of 554 [prolonged mechanically ventilated] patients … two thirds remembered the endotracheal tube and/or being in an intensive care unit" (A.J. Rotondi, 2002).

This reduced memory has different causes. Prescribed drugs, administered to patients in ICU, can have effects on memory. "Opiates, benzodiazepines, sedative drugs such as propofol, adrenaline, and corticosteroids can all influence memory" (C. Jones, 2000). Not only the drugs themselves, but also the withdrawal effect from previously administrated drugs can contribute to memory loss. Furthermore, the stress induced sleep deprivation reduces the ability to memorize. In addition to these causes, it is hypothesized that "there is a process that affects memory negatively for external events but enhances memory for internal events. [This could be induced by] the physical constraints and social isolation experienced by the ICU patients and the life-threatening nature of the illness" (A.J. Rotondi, 2002). These causes suggest that a communication system could reduce patients' memory deficiency.

Delirium is a condition of acute confusion that commonly develops for patients in the ICU. In fact, it prevails for 80% of terminally ill patients (T.M. Brown, 2002). A major symptom is impaired cognitive functioning, including reduced functioning of memory, orientation, attention span and planning skills. Furthermore, consciousness is reduced. The patient can be variably alert and aware of his surroundings. Also his perception is disturbed, as it is not uncommon for patients to suffer from illusions or even hallucinations. Finally the delirium can affect a patient's mood, which can result in the patient acting with total indifference or with great anxiety.

Delirium is most commonly caused by prescribed medication or infections, and can also be caused by withdrawal from addictive substances. Risk factors for developing it include amongst others social isolation, sensory deprivation and unfamiliarity with the new environment. Due to this, for treatment and prevention of delirium it is advised to employ clear and repetitive communication by staff and with family and friends. Also some aids help the patient orientate and alleviate his sensory deprivation, such as a

clock and adequate lighting. Furthermore, it is important to demand as few as possible from the patient's cognitive abilities.

Finally, it is possible that the patient simply does not have the required knowledge to communicate. The IC houses patients from all backgrounds. It is possible that a patient is unable to speak Dutch or English. Another scenario is that the patient is not able to read or write. Knowhow about interacting with a computer can also not always be expected.

2.2 PATIENT INTERACTION ON THE ICU

To form a good picture of the problems faced by patient, family and medical staff, we have visited the MST ICU during one day and has shadowed several members of the medical staff. Patients and medical staff have been observed in their activities and interviewed about their experience in order to gather requirements for the software system. The approach used to observe and interview is described in subsection 6.5. Although more thorough observations and interviews are necessary, these can already be used as an important requirement source for the first release plan.

Here we include one report of observation and interview of patient Martijn and his doctor. Martijn has been observed as he communicates with a doctor and with his wife at his bed side.

Martijn is unable to speak due to being mechanically ventilated through a tracheostomy tube. A few hours a day he has enough strength to breathe on his own and he is decoupled. During this time he is able to speak, though just a little. During the rest of the day he communicates by nodding and shaking his head, alternated with using his hands to make simple signs or just pinching someone else's hand. Martijn tries to form words with his lips, but most of the time his wife does not understand what he wants to say.

All these means of communication are very tiresome for Martijn and he can only communicate very slowly, giving no more than one response every 10 seconds. After one or two minutes he needs to close his eyes for half a minute before being able to continue. He signals that pinching is still less tiresome than his other means of communication. Martijn confirms that he feels very helpless, isolated and anxious in his current condition.

His doctor says she lets the amount of communication depend on how tired Martijn is. When he is tired she limits social interaction and she only asks medically relevant questions. Ideally, she can also ask open questions, which he sometimes can answer if he has the energy to breath without the ventilator. For example, she would ask how stuffy he feels or what symptoms he has experienced.

Martijn's story illustrates some of the requirements that have been identified during the day at the ICU. All of these are included in the requirement analysis in section 3.

2.3 SOFTWARE AND HARDWARE IN ICU CONTEXT

New technology introduced into the ICU environment must comply with very strict rules. All physical devices must be completely save and must be kept clean. Traditional computer and peripherals are very difficult to keep in such a clean state, because filth can accumulate inside where it is not easily removed.

ICU personnel, like many patients, has only basic experience in working with computers. Any form of logical insight in what to do and what not to do with a computer that provides AAC cannot be assumed.

For example, cleaning personal might guilelessly disconnect and reconnect an input device that the patient is using during cleaning. They might even forget to plug it back in altogether.

Software that interacts with a patient is also subject to restrictive limitations. It may not put unnecessary stress on patients, since in their fragile state this could have serious health consequences, such as contributing to forming a delirium. There also exist many ethical restrictions on what can and cannot be asked from patients. Ethical guidelines for patient interaction in the context of research are specified in the Wet Medisch-wetenschappelijk Onderzoek (Ministerie van Volksgezondheid, Welzijn en Sport, 1998).

3 PROBLEM ANALYSIS

From the problem domain analysis and case description, we can identify the following main problems.

- Difficult communication between medical staff and patient related to caregiving.
- Difficult communication between patient and his/her visitors.
- The patient is unable to call for help from medical staff.
- The patient is powerless, isolated and disoriented.

Furthermore, there are a number of challenges any potential solution should overcome in order to resolve these problems.

Patients often have limited cognitive abilities. The software system should thus be as simple as possible to use by the patient. The need for remembering how to perform a task, solving problems of any kind or attention should be minimal.
Patients have varying limited physical abilities and are often exhausted. This means the software system should support different means of input in such a way that each patient can control the system efficiently with minimal effort.
ICU Medical staff is very busy and most staff members have very little knowhow about computers. The software system should be very easy to learn and use for the medical personal and require little of their time.

The requirements of any solution to these problems are extensively analyzed over the course of the project. A major effort is made at the start of the project to identify the initial requirements for the first release. These are listed in section 7.1.1.

4 SOLUTION DOMAIN ANALYSIS

4.1 AUGMENTATIVE & ALTERNATIVE COMMUNICATION

In this subsection we describe methods to bypass some of the physical disabilities identified in subsection 2.1. These methods are encompassed by the term **Augmentative and Alterative Communication** (AAC), which we define as all forms of direct person to person communication using speech and facial expressions. We all use AAC when we make gestures, write, type or use a computer mouse. In this project we are only interested in AAC that can be used to replace traditional oral communication for those that have lost the ability for this.

There are many forms of AAC available. Below is a comprehensive list of AAC media that can be used as an alternative to oral speech:

- Eye blinks
- Forming words with the mouth
- Lip movements (munching)
- Respiratory modulation
- Tong movements (clucking)
- Facial expressions
- Finger, toe, arm, leg or facial movements
- Gaze direction
- Brain activity (EEG)
- Muscle activity (EMG)
- Hand movement (writing, pointing with a finger, forming signs)

In order to employ these communication media to practical use, a multitude of practices and technological aids have been developed. In the following subsections we discuss those that are in use today to help people with impairments.

AAC practices can be divided over two categories. There are analog practices and those supported by computers. The computer is in its very nature an AAC device, as it relies on communication other than speech, or at the very least enhances this form of communication. Although traditionally computers have been designed to aid people that do not suffer from communication impairments, over the last two decades computers have proven to be very useful as an AAC tool for the impaired as well. Today, they are widely adopted to this end, supplementing or even replacing traditionally used analog forms of AAC.

4.1.1 Analog AAC practices

4.1.1.1 Letter boards

A very common way to communicate with impaired patients is using letter boards. A board is held up in front of the patient, showing the letters of the alphabet and perhaps some other meaningful symbols. The patient can form words by pointing at the letters one by one. It is a popular way to communicate, because of its simplicity and expressiveness. The letter board can be used in combination with a pointing stick for more precision.

4.1.1.2 Sign language

Sign language is also very common, yet more difficult to use. The communicating parties must agree on the meaning of signs. Signs can be anything from certain hand movements to eye blinks. The most basic of sign languages, commonly used in communication with people with locked-in syndrome, is to blink once for 'No' and twice for 'Yes'. Naturally, this form of communication is extremely inefficient and one sided. The second party has to guess what the patient might want to communicate in order to ask the appropriate question. For patients with more motor control, sign languages such as those adopted by the deaf-community are much more powerful. The downside of these languages is that they require a lot of effort to learn by both communicating parties.

4.1.1.3 Electrolarynx

The Electrolarynx is a small device that, when pressed to the throat, functions as mechanical vocal cords, thus allowing the user to speak. This also works for mechanically ventilated or orally intubated patients. It is an extremely powerful and efficient way to communicate, that makes almost normal conversation possible. It is not without downsides though. It takes weeks of practice to learn how to speak with an Electrolarynx. Also the patient should be able to press the device against his throat whenever he wishes to speak, which makes it unusable for patients with insufficient motor control. There is also the limitation that not all sounds are pronounceable through the Electrolarynx, so the user has to avoid words with these sounds.

4.1.2 Computer based AAC practices

There exist a great number of digital devices that leverage the alternative communication media listed at the start of this section. These devices translate the patient behavior to digital signals to control a computer. In this subsection an exhaustive list is given of the AAC devices that are used in practice.

4.1.2.1 Switches

Switches come in all forms and sizes and provide the patient with a mono-stable binary input. This means that a switch has two states. The switch is in a rest state when the patient is not pressing it and in an active state while he is pressing the switch. Switches can be controlled in many different ways. Every part of the body that the patient can freely move can be attached to a switch. This can be, amongst others, any finger, a toe, the jaw or an eyebrow. Multiple switches can be used simultaneously by independently movable body parts.

4.1.2.2 EMG

Electromyography (EMG) is a technique for evaluating and recording the electrical activity produced by skeletal muscles. It functions very similar to switches. Instead of pressing a switch, electrodes measure nervous signals to a muscle. It can be used when the patient can nervously control a muscle, but not actually move the corresponding body part. "EMG is particularly suitable for people with severe motor disabilities, for example, people with high levels of spinal cord injury or with locked-in syndrome" (E. Naves, 2012).

Nerve signals are difficult to reliably measure though. Electrodes attached to the skin can only measure superficial muscles and even then it is hard to narrow down the signal to a single muscle. The measurement's reliability can also be reduced by fat, so EMG works best for skinny people. To decrease the skin's electrical resistance during measurements, cleaning and commonly abrasion are required.

4.1.2.3 Eye trackers

Eye trackers follow the direction of a patient's gaze based on pupil position. This is called **gaze tracking**. They are used in combination with a screen in order to determine where on the screen the patient is looking. In Appendix A we have listed multiple afflictions by which the patient can lose almost complete motor control, but being unable to control the eye muscles is very rare. Eye trackers can for example still be used by patient with a high lesion or with locked-in syndrome. They also form an intuitive and efficient interface, as they basically substitute the mouse. Mouse clicks are simulated by looking for a longer time at a specific spot or by using the eye tracker in combination with a switch.

4.1.2.4 Facial expression recognition

As loss of control of the facial muscles occurs less often than other muscles, patient remain capable producing complex facial expressions. These expressions come in many forms, allowing facial expression recognition software to interpret them as many different inputs. The recognition works by analyzing camera pictures to determine if the patient looks happy, sad, angry, surprised, etc.

4.1.2.5 EEG

Electroencephalography (EEG) is the recording of electrical activity along the scalp. EEG measures voltage fluctuations resulting from ionic current flows within the neurons of the brain. EEG is a form of non-invasive Brain-Computer Interface (BCI) as opposed to invasive BCI. This means it is less reliable, but does not require surgery to hook up the patient. EEG is the most studied potential non-invasive interface, mainly due to its fine temporal resolution, ease of use, portability and low set-up cost.

Obtaining reliable EEG signals in an active field of study, but different methods have yielded useful results. One team of scientists has been able to attain precise and continuous three dimensional movement of a virtual helicopter (A.J. Doud, 2011). They used different motor imaginations for different movements, for example, imagining moving the right arm upward would elevate the helicopter. Another method has yielded reliable results using the so-called P300 brainwaves, which occur involuntarily when people see or imagine something they recognize and may allow BCIs to decode categories of thoughts. Both of these methods use natural occurring brain behavior and can thus be used with little practice. It is also possible to learn how to control certain brainwaves for BCI, but this takes a lot of practice.

4.1.2.6 Pointing devices

The ubiquitous computer mouse can, off course, also function for AAC. The same counts for its many variants such as the joystick and trackball. For patients with sufficient hand motor skill, this is an excellent option. A less common pointing device is the head tracker, which translates the direction in which the head turns to (commonly) a screen position. Normally, this works by attaching a reflective dot to the head that is followed by a camera. It is also possible to attach this dot to other movable body parts to track their movement.

4.2 EXISTING AAC SOFTWARE SYSTEMS

Needlessly to say, we should look at existing AAC software systems that try to solve the identified problems and try to overcome the identified challenges. The severity of these problems can only be judged in light of (a lack of) existing solutions.

There exist dedicated communication computers that make use of the input devices listed in the domain analysis and also programs that do so on a regular computer. In this subsection we have a critical look at

these existing AAC solutions. In order to critically evaluate these solutions, we determine whether these solutions are likely to satisfy the identified requirements. In o an overview is given of the requirements satisfied by each solution.

For a great number of requirements it is difficult to determine to what degree these are satisfied. This can only reliably be done by testing the software system in ICU context. In this report we only provide an estimate based on the features and interface. It would be of great value to our software system's design to know how well the other AAC solutions performed on certain requirements. This way we know which aspects to copy and which to avoid. Preceding the first development cycle we try to arrange these trials with different solutions. The developers of these software systems have to approve this though, as they are not freely available. The necessary hardware needs to be borrowed from the developer in order to use their software. Whether this is feasible remains to be seen.

4.2.1 GoTalk Pocket

The GoTalk Pocket is an all-in-one communication solution. This little portable device has five sets of six programmable messages. The user simply presses a button and the corresponding message is spoken aloud. One button switches between the sets of messages.

It is straightforward to use and the portability is a great advantage for people that are not bound to a sickbed. It offers efficient communication for common messages between patient and staff. However, more complex communication or messages that were not programmed are not expressible. It is also not usable by patients unable to press a button. The main limitations are:

- Not usable by patients who cannot use a button.
- For many patients other forms of input than a buttons might be more suitable.
- Difficult to use for users with limited vision.
- The patient needs help to use the device, because in most cases he cannot hold it himself.
- There are limited output possibilities; only basic preprogrammed messages can be communicated.
- There is no explicit menu structure, so the user needs to rely on memory or communication is slow and frustrating.

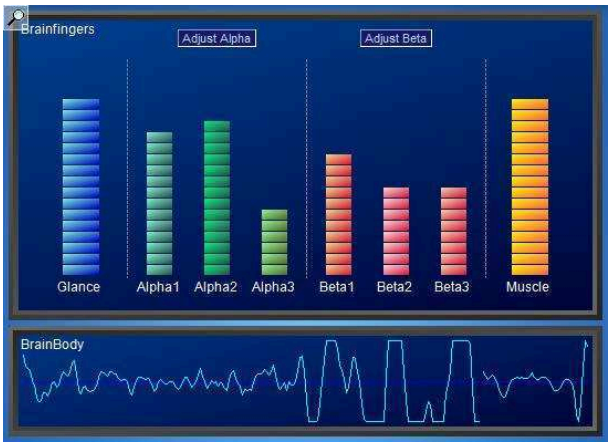
4.2.2 Voice

This solution is currently used on the ICU of UMC St Radboud. VoICe is an iPad app that is designed to be used through a normal tablet touch interface or with a single external button. It offers a simple intuitive interface with several output possibilities. It supports common communication scenarios built-in, such as specifying where the patient feels pain or what emotion he is feeling. This allows for relatively fast communication. There is also a virtual keyboard and drawing board for communication. Finally the app allows to send and receive (video) messages. The main limitations are:

- Not usable by patients who cannot use a button.
- Patients with insufficient hand motor control can only use a single button as alternative input, whilst other forms of input might be more suitable for these patients.
- Difficult to use for users with limited vision.
- The patient needs help to use the app, because in most cases he cannot hold the tablet himself.
- There are limited output possibilities.
- The system does not support other languages.

4.2.3 Brainfingers

Brainfingers is a program that allows the user to control a computer, using a combination of EMG and EEG signals. The user must wear a sensor band on his forehead, which senses and responds to surface electrical signals generated from eye muscle movement, and brainwave activity detected at the forehead. Each of these signals is a binary input that can be coupled to a configurable output. It is possible to control keyboard buttons and/or control a mouse. A training program is included that teaches the patient to use the program and assesses his abilities. The main limitations are:



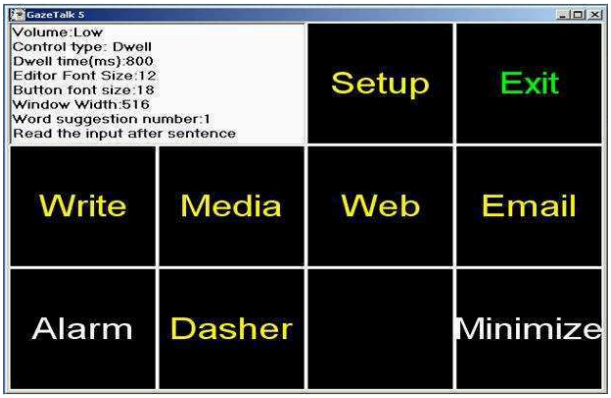
- Does only support EEG and EMG as an input, whilst for many patients other forms of input are much more suitable.
- It does not provide an efficient way to communicate, but should be used in combination with other AAC solutions for this.

4.2.4 SIDE

Side is targeted to patients with locked In Syndrome and allows them to type text, browse the web, send e-mails and interact external devices (television, radio, lamps…) using a single button. It is also usable if the patient has limited eyesight or eye movement that can only see a small portion of the output screen. Options are shown one by one, centrally on the screen. Creating custom option menus for image based communication is possible. The main limitations are:

- Not usable by patients who cannot reliably use a button.
- For many patients other forms of input than a single button might be more suitable.
- Though the system seems extendible, there are currently limited output possibilities.

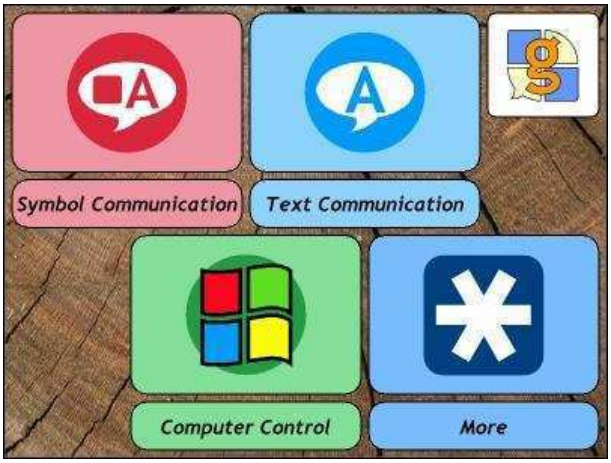
4.2.5 GazeTalk



GazeTalk is a predictive text-entry system that enables the use of an eye tracker with a low spatial resolution (e.g., a web-camera based eye tracker). GazeTalk tries to be both sufficiently feature-complete to be deployed as the primary AAC tool for users whilst remaining flexible. As input it supports eye tracking, head tracking, mouse, joystick, or any other pointing device. As output it supports text-to speech, web browsing, reading and sending email, playing videos and reading documents. The main limitations are:

- Not usable by patients who cannot use eye tracking or another pointing device.
- For many patients other forms of input than a pointing device might be more suitable.
- Though the system seems extendible, many desirable output possibilities are currently not supported.
- Not usable for users with limited vision.
- The system is not usable by illiterates.

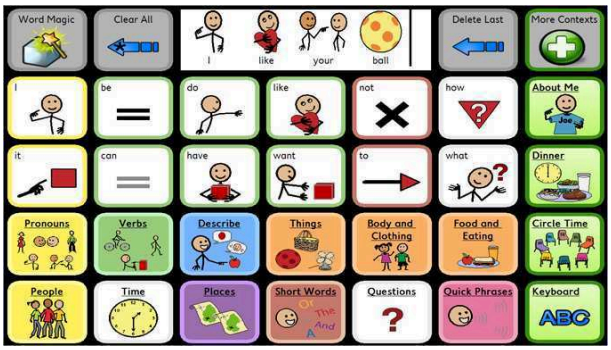
4.2.6 The Grid 2



The Grid 2 allows people to use eye gaze, switches, head pointer, touchscreen, mouse and keyboard to control a computer. It also has a number of built-in features, including voice output communication using symbols or text to build sentences, sending and receiving email and sms messages, browsing the web and listening to music. The main limitations are:

- Not usable by patients who cannot a reliably use pointing device with reasonable accuracy or a switch.
- The input needs to be configured before use, especially for non-basic input.
- Not usable for users with limited vision.
- There are limited easy to use output possibilities built-in to the application. This is somewhat made up for by the possibility to control the full computer for users that have the ability to accurately control the cursor. They can with extra effort use other programs for these features.

4.2.7 Tobii Communicator



This system allows communication through mouse, head and eye tracking, possibly in combination with a switch or keyboard to select predefined options for symbol or text based communication. Additionally it offers the possibility to control a computer by controlling the cursor. There is also support for controlling your environment by infrared switches. The user is offered a choice of tiles with conversation options for quick communication. There is also a full keyboard available and a screen to specify the details of pain or other discomfort. The main limitations are:

- Difficult to use for patients who cannot reliably control a pointing device.
- For many patients other methods of input than a pointing device might be more suitable.
- Difficult to use for users with limited vision.
- There are limited easy to use output possibilities built-in to the application. This is somewhat made up for by the possibility to control the full computer for users that have the ability to accurately control the cursor. They can with extra effort use other programs for these features.

4.2.8 Conclusion

Based on the functionality matrix we can conclude that all of the existing AAC solutions have shortcomings.

Although, some of these solutions suffice well for a particular group of patients, there is no catch all solution. Unfortunately, deploying different solutions on a per patient basis is undesirable. It is costly to

license multiple packages and it is more demanding for the medical personal to learn how to work with different solutions, especially determining when to use which software system.

None of these solutions is specifically targeted at the context of the ICU. They are designed to be used by people with long term communication disabilities. They can be carried along as they go through their daily life or assist the user at home. This context is crucially different from the ICU context, most importantly since the user's mental condition is very different. All the stress, exhaustion and confusion and anxiety form a big part of the communication problem we try to solve. This means the system should ask very little of the patient's mental and physical abilities, should provide support for patients orienting themselves in their environment and should be very reliable. None of the existing solutions is especially suitable for these challenges.

Our new software system, CommIC, tries to combine the strong points of existing AAC systems and add features missing from them altogether in order to make it more suitable for the ICU environment.

5 RESEARCH QUESTIONS

The goal of this project is to design and build an AAC solution for an ICU environment that solves the problems that were identified in section 3. The limited time for this project prevents us from developing a catch-all solution that offers optimal support for all patient limitations and communication wishes. Choices have to be made. We look for a solution that can realistically be built within this projects timeframe, yet with maximum problem solving capacity.

We try to answer the following research questions:

1. Which problems do occur on the ICU as a result of patients' communication disabilities?
2. How severe are these problems? (Number of patients, level of impediment for medical personal, etc.)
3. Which AAC solution can be designed and built within the timeframe of this research that reduce the severity to these problems as much as possible?
4. To what level does this new solution reduce the severity of these problems?

Answering research question one and two gives us valuable insight into the requirements of the solution sought after by question 3. As part of this research question the AAC software systems CommIC is designed and built with the help of the development team. The answer to the final research question tells us how successful this software system is solving the identified problems.

6 METHODOLOGY

6.1 AGILE DEVELOPMENT

During this project, the agile development approach is used. This section describes this approach and the software development methods and techniques that are brought into practice. We have chosen for an agile approach, in favor of the more traditional Waterfall model. In the next subsection we compare the two methodologies and justify our decision. The information on agile development covered in the section is largely based the book 'Agile software development' (Martin, 2003).

The agile approach requires rapid iteration through short **development cycles**, each containing requirement analysis, design, implementation and testing. At the end of a cycle, a working product is produced; either a prototype or a partial implementation of the final system. It is used to gain quality feedback regularly and quickly during development. The intermediate tangible result makes it easy for the client to give concrete feedback. This feedback is used as a source for new requirements for the following development cycle(s). This possibility for frequent early feedback is very powerful. The MIT Management Review analyzed software development projects and found two strong correlations. "The less functional the initial delivery, the higher the quality of the final delivery" and "the more frequent the deliveries, the higher the final quality".

The philosophy at the basis of this approach is the realization that, in the real world, requirements are never static and misunderstandings about them between client and developer cannot completely be prevented. Stakeholders never know what they want exactly in advance. In fact, change is seen as a good thing, as every time during development the requirements are changed, the understanding of the customer's needs is improved. Furthermore, being responsive to change gives a competitive advantage, as the software can react to the latest developments in the market. Agile embraces change as an essential part of the software life cycle.

Agile has a few other commandments. First of all extensive communication between developers and the stakeholders is very important to minimize size of the feedback loop. Face-to-face contact is the most effective and efficient medium of communication, over any form of documentation. For effective communication with the customer, ideally someone who represents the customer and his needs is present in the same room as the developers. Conversation is preferable over formal documentation or contract specifications, to facilitate the changing nature of requirements.

To foster communication within the development team, agile teams should be self-organizing. Responsibilities should be communicated to the team as a whole rather than to individuals and the team decide how best to fulfill them. Every developer is aware of what the others are doing and they all share responsibility. Frequent meetings are held to reflect on team progress and modifications are made to work more effectively.

This brings us to another point, namely that the only way to measure progress is by working software. If 40% of the targeted functionality is working than 40% of the project is done. No amount of documentation or project infrastructure influences that. That is not to say documentation and infrastructure are bad, but they should never be seen as ends in themselves. If investing in them is not guaranteed to pay back in the increased ability to create working software, it should not be done.

Another believe is that the pace of development should be sustainable. Working overtime and dealing with time pressure has a negative impact on software quality. Developers that are tired make mistakes.

Software design is always important for maintainability and reliability. Although agile developers allow software design to change, they never allow it to '**rot**'. If by changing the functionality the current design is violated or another design is more suitable, they refactor the software. They never allow themselves to create a mess. The practice of 'cleaning up later' is not done, in favor of cleaning up right away. The principle of simplicity is essential here. Agile programmers try the solve problems in the simplest, cleanest way possible in order to maximize the work that is not done and minimizing time spent on redesign. Simplicity also prevents the team from being vested in the initial design, postponing necessary changes. Simplicity is maintained according to a simple principle. If it is not reasonably certain that a certain abstraction or infrastructure is needed later, it is not implemented. In section 6.3 an overview is given of commonly used agile design principles to maintain simplicity and cleanliness of code, whilst maintaining expressiveness.

6.2 AGILE VS WATERFALL

We have chosen for agile because we believe in its philosophy of embracing change during development. The conviction that change is inevitable and even desirable matches with our personal experience. Moreover, it is especially useful in this project. The complex nature of the product and clients that have difficulties communicating make it likely that the requirements are subject to change more than usual. Furthermore, the frequent and early feedback made possible by agile has proven to result in better software quality.

The traditional Waterfall approach, on the other hand, is not very responsive to change, because the end product is developed at once. Therefore, the inevitable changes in requirements are detected much later and are much harder to correct. "Traditional approaches assumed that if we just tried hard enough, we could anticipate the complete set of requirements early and reduce cost by eliminating change. Today, eliminating change early means being unresponsive to business conditions" (J. Highsmith, 2001).

However, there are a few advantages to the Waterfall model that should not be so discarded without thought. First, the extensive project planning that the Waterfall approach starts with, makes for a clear and predictable project. This is pleasing to customers who know in advance how much time and money the project is going to cost. This advantage over the relatively vague agile approach, is mitigated though. A successfully executed project plan often does not yield a satisfactory end product right away. Frequently extensive changes are necessary, invalidating the original estimates in time and cost. Additionally, agile uses tools to continuously measure current and estimate future progress. We therefore think that this advantage of the Waterfall model is negligible.

Another advantage due to the extensive documentation is a decrease in vulnerability to changes in the team. If a developer is removed from the project someone else can easily continue his well-documented work. We believe that this is also a negligible advantage. Agile makes up for lack of explicit design documents by optimizing knowledge spread throughout the team (through shared responsibility, extensive communication and frequent reflection) and simplicity in design. Additionally, testing throughout the project makes it unlikely for someone new to the project to break existing functionality. The software tests, furthermore, form an excellent documentation of each software class's functionality and how to interact with this functionality.

A final advantage of the Waterfall approach is that the extensive customer contact that agile demands is time consuming for him. It is often preferable for clients to only be involved during the planning phase of the Waterfall model, then throughout the project. If this frequent customer contact cannot be arranged, agile is not a good idea. In this project frequent contact can be arranged by having multiple contacts within the medical staff, such that always someone can be found who has time to speak. It is also very desirable to arrange a workplace at the MST hospital. From there it is also easily possible to enter the ICU for contact with another vital stakeholder, the patient. These measures are realistic and thus should form no obstruction for a successful execution of agile development.

6.3 AGILE DESIGN PRINCIPLES

6.3.1 Single Responsibility Principle

The Single Responsibility Principle (SRP) states that classes or modules contain only elements that are functionally related. If this is the case, they are called **cohesive**. This is a desirable property, because each responsibility is an axis of change. If responsibilities are coupled in a module it has to be modified more frequently. Each time one responsibility is changed it is risked breaking the other and the whole module needs to be recompiled. It also becomes more difficult to independently test a responsibility. Separating responsibilities is, off course, only desirable in agile development if the responsibilities are actually subject to change.

6.3.2 Open-Close Principle

The Open-Close Principle (OCP) prescribes that software should be open for extension, but closed for modification. This means that a module is designed in such a way that changes of a specific kind can be implemented by adding new code only. There should be no need to modify and recompile the existing module. In agile this means that whenever a change is made to the code, the code is refactored in such a way that similar changes in the future can be added without modification. For example, if output code that prints to the command line needs to be modified such that it can also output to a file, both output methods are made to be extensions of an abstract output class. Adding new outputs in the future will then only require the extension of this abstract output class.

Of course, introducing new abstractions later in the development cycle might be problematic. Agile tries to minimize the chance of this happening by means of extensive acceptance testing, short development cycles and intimate customer contact. Still the risk of this being necessary remains and we think this is one of the weak points of agile.

For this reason we chose to violate agile's simple design principle in this respect. We apply the open-close principle in combination with **commonality/variability analysis** (CV-analysis), which is conducted for each new software component. For each aspect of the problem domain that needs to be modelled, we look at which concepts are static and which are dynamic. Static entities are constant in their behavior or responsibility. Dynamic entities could have different ways of functioning, depending on the context. Differentiating between these two allows us to predict the appropriate abstractions to the software. These abstraction are immediately implemented with the new component.

6.3.3 Liskov Substitution Principle

The Liskov Substitution Principle (LSP) states that subtypes must always be substitutable for their base types. In other words, a procedure that expects a certain type as an input should work with any subtype

of that type. Unexperienced programmers might be tempted to write a procedure that explicitly tests for the subtype to determine what to do. This, however, violates the OCP principle as new subtypes can now break or require extension of this function. The principle is especially important for external users of a type. If a subtype violates reasonable assumptions users can make about one of its base types, the base type might behave in ways unexpected to its users. Take as an example a Square subtype that extends a Rectangle type and overrides its SetWidth and SetHeight methods to ensure the Square's invariant of equal width and height. Now there is an external function that takes a Rectangle as input and assumes that setting its width to 2 and its height to 3 results in a rectangle with area 6. This is a very reasonable assumption but it is not valid if the passed Rectangle is in fact a Square!

A good way to avoid LSP violation is to interpret the is-a relation between a subtype and its base types as pertaining to behavior. The square example does not follow this interpretation. Although a square definitely is a rectangle, its behavior contradicts that of a rectangle.

6.3.4 Dependency Inversion Principle

The Dependency Inversion Principle (DIP) states that high-level modules should never be dependent on low-level modules. A violation would mean that a change in low-level details could influence high-level functionality. A good way to achieve this is making sure that all relationships between components never end on a concrete object, but always on an abstract interface. This so-called service interface of a low-level module is owned by the high-level clients that use it. They can make changes to this service interface if different low-level functionality is required, never the other way around.

6.3.5 Interface Segregation Principle

'Fat' classes with lots of functionality cause their clients to be coupled in undesirable ways. One client forcing a change in a fat class can affect many other clients. When it is not desirable to break the 'fat' class into smaller more cohesive classes, functionality can be separated through multiple interfaces. For each group of clients that use a specific subset op functionality, an interface is provided. This causes those clients only to depend on the methods they actually use and makes them independent of other (groups of) clients.

6.4 AGILE IN PRACTICE

As mentioned earlier, within every development cycle the requirements analysis, design, implementation and testing are conducted. This results in an intermediate product, which is then evaluated as part of the requirements analysis of the next development cycle. There exist different concrete agile practices for performing these tasks. Two of the most well-known and commonly used methods are Extreme Programming (XP) and Scrum. The two methods are very similar, the main difference being that Extreme Programming is more prescriptive in the practices to be used. How we implement the agile methodology in this project is based on XP and Scrum guidelines. In this section we elaborate on those practices used. If there is a difference between XP and Scrum, an argument is made for choosing between the two.

Note that for this project there is no literal customer and the term is used interchangeably with client and stakeholders to refer to the group of people that in one way or the other have to interact with CommIC.

6.4.1 Product owner

The product owner is defined as the person who defines and prioritizes the user stories. The product owner is part of the development team and should work closely together with the developers. Developers and product owner are aware of each other's problems and help each other out. The product owner is someone who represents the customer within the team. He can be the client for whom the software is built, but this is not necessary. As argued in section 6.2, intensive contact with the client can be arranged, through multiple contacts within the MST. However, making (one of) them a part of the team is not realistic due to their lack of time and software engineering experience. The researcher therefore fulfills the role of product owner, backed up by close contact with the medical staff and ICU patients. The role of product owner is Scrum terminology. In XP this role is called the 'customer'. This name is ambiguous, thus possibly causing confusion. It is not used in this report.

Since the product owner is the link between the development team and the stakeholders, he is also responsible for keeping the stakeholders up-to-date about the status of the project.

6.4.2 Scrum master

This team role is specific to Scrum. The Scrum master makes sure the project progresses as intended. He ensures that all the agile practices are indeed used. He normally chairs team meetings where he lets the team reflect on their progress and challenges them to improve. Since developers in this project do not all have experience with the agile practices used, the role of Scrum master is especially valuable. This role is fulfilled by the researcher, who has the most experience with agile development.

6.4.3 User stories

Detailed requirements are likely to change throughout the project. Instead we use user stories to represent functional requirements. A user story describes a functional requirement in a few words. "A user story is a mnemonic token of an ongoing conversation about a requirement" (Martin, 2003). Based on the sense of detail perceived by the product owner while talking with the customer, the implementation cost of each user story is estimated.

A good quality user story should be implementable, testable and independent of other user stories and its implementation cost must be estimable. It this is not the case, it needs to be made less abstract, split up into smaller or merged into larger user stories. Not all functional requirements have to be immediately formulated in this way. Concretizing low priority user stories not until later in the project often makes sense, as they can better be understood in the context of a working product.

Throughout the project all not yet implemented user stories are maintained in a prioritized list. We call this list, in accordance with Scrum convention, the **backlog**.

It is important to realize that user stories are no replacement for non-functional requirements. These requirements cannot be perceived as an independently implementable piece of functionality. The project thus maintains a separate list of such requirements that have to be taken into account throughout development.

6.4.4 Short Cycles

Working software is produced every two weeks. At the start of each two week iteration cycle an **iteration plan** is made. This plan consists of a collection of user stores that is to be implemented during this cycle. The number of user stories that is selected depends on a budget the developers set themselves based on

their experience with previous cycles. Each user story is divided up into tasks which are implemented in any order that makes logical sense. At the end of each cycle the working product is reviewed with the stakeholders. We call each cycle a **sprint** in accordance with SCRUM terminology.

A **release plan** is made for every six iteration cycles or so, spanning three months of work. The end result is a major delivery, called a **release**, usually one that can be put into production. Like the iteration plan, the release plan consists of selected user stories fitting within a developer set budget. An iteration plan cannot be modified during an iteration. A release plan, on the other hand, can freely be updated at any time by the product owner.

6.4.5 Acceptance tests

The details of a requirement are captured in acceptance tests specified by the product owner. This happens directly preceding or during the implementation of a story. They are tests that can be run automatically and repeatedly to verify that that the software behaves according to the customers wishes. They make sure that once a requirement is implemented it is never again broken. Writing acceptance tests requires the system to be testable at a high architectural level, which leads to more decoupling at this level. For example, the business rules should be decoupled from the GUI in order to make them accessible for testing. Often acceptance tests are written in a special scripting language. Developing this language is probably not worth the investment for this project, since the researcher, who fulfills the role of product owner, has sufficient expertise to write the acceptance tests in plain without a software domain specific language. Also the size of the project is probably insufficient to return the investment by increased test writing efficiency.

6.4.6 Pair programming

This is an XP specific practice. All production code is written by pairs of programmers rather than individuals. This provides rapid feedback, which significantly increases the quality of the code. While one programmers writes code, the other looks for errors and improvements. These roles change frequently. The pairing of programmers is changed at least once a day, so that every programmer works on most parts of the software. This facilitates the spread of knowledge throughout the team and reduces the vulnerability for losing knowledge if losing team members. Lack of a standard shared workspace and the uneven number of developers means that pair programming is not always feasible during this project, but it should be employed given the opportunity. Studies have shown that pair programming, contrary to intuition, does not reduce efficiency. It does however significantly reduce the defect rate.

6.4.7 Test-driven development

Test-driven development means that all implementation code is written in order to make a unit test pass. The developer is constantly switching between writing new unit tests and then writing code to make them pass. This iteration takes a minute or so. This heavy focus on tests makes it possible to make heavy use of refactoring without worrying this might break any existing code. It also forces the code to be testable. This forms a strong drive towards decoupling. One more advantage is that the developer is forced to look at his code from another perspective. He has to be concerned with its interface as well as its function, resulting in conveniently callable code. Finally the unit tests form excellent documentation of the code that is always up to date and correct. It shows good examples of how to work with the code.

6.4.8 Collective ownership

Every programmer shares responsibility for the whole project and no individual is responsible for a feature or module. This stimulates active involvement and communication between developers. No one is confined to a specialty, allowing everyone to broaden his skills.

6.4.9 Continuous integration

A non-blocking versioning system is used. Every hour or so developers check in their code, meaning they merge their code with the main branch. Then they run every test they just wrote to see if their new code is still working and also every other test written to check if their code does not break any existing functionality. If they broke something they fix it before finishing the check-in and publishing their changes.

6.4.10 Sustainable pace

As mentioned in our description of the agile approach, it is important for developers to conserve their energy and alertness. XP and Scrum adapt this principle. Working overtime is only allowed in the last week of a release.

6.4.11 Open workspace

This is an XP specific practice. The programmers work together in an open workspace, each within hearing distance of the other. This minimizes the gap to communicate and allows every developer to know the state of the other. This seemingly distracting environment, has surprisingly been shown to increase productivity by a factor of two (University Of Michigan, 2000). Like pair programming, working in the same workspace is not always practiced during this project. Often it is convenient to work from home.

6.4.12 The planning game

The planning game is the process of constructing the iteration and release plans. The product owner determines how important each user story is. The developers estimate how much time implementation costs. The developers also provide a budget for the customers to fit the amount of work they expect to finish. The cost of a user story is an arbitrary amount of points that has only relative meaning. A feature with a cost of 6 points is expected to take twice the implementation time than a feature worth 3 points. If it is difficult to estimate the number of points to assign to a user story, it should be split into smaller or merged into a bigger user story.

The budget of points is determined using the project velocity. The **project velocity** is a number of points implemented per unit of time and can be calculated based on previous iterations. Based on the cost and priority of the user stories and the available budget the product owner selects the user stories for the next cycle. In Scrum this works slightly different. There the developers select the user stories. In practice this makes little difference though, since always the user stories with the highest priority are selected first, as we do for this project. If at the end of an iteration cycle not all user stories are implemented, the cycle ends anyway. The velocity is adjusted accordingly and possibly the release plan as well. Due to this planning game the customer has a good idea about the duration and cost of a project.

After the selection of the user stories is complete, these are divided into programming **tasks**. These tasks should be implementable in 4 to 16 hours. Now one by one the developers sign up for tasks. Developers assign a number of task points to each task they choose, which is spent from a personal budget. Each

developer knows from experience how much task points he can complete and should spent accordingly. If not all stories are divided this way, the product owner should retract user stories, or vice versa add more. Halfway through the iteration, progress is evaluated and tasks are redistributed. Possibly, user stories not yet worked on are pulled out of the iteration plan.

6.4.13 Daily Scrum meeting

This Scrum practice is a daily max 15 minute developer meeting with the goal to update each other on their work and to identify problems. Each developer answers the following questions: What have you done since yesterday? What are you planning to do today? Any impediments / stumbling blocks? Solutions to identified problems are not discussed in the meeting, but between the Scrum master and the developer in question. The daily Scrum meeting is substitute for the XP practices of pair programming and the open workspace, which both stimulate the propagation of knowledge throughout the team and provide a quick internal feedback loop. Since pair programming and the open workspace principles cannot be fully implemented, we use Scrum meetings whenever the former cannot be applied.

6.4.14 Sprint retrospective

This Scrum meeting is similar to the daily Scrum meeting, but is held at the end of each development iteration. Each developer reflects: What went well during the sprint? What could be improved in the next sprint? Like the daily meeting, this practice fits well with the project and is used.

6.4.15 Simple design

Following this agile principle, XP programmers tend to create designs that are as simple and expressive as possible. During design they only consider the user stories in the current iteration. They do not worry about user stories in following iterations. If these ask for a different design, the design is migrated. This means an XP team does not start implementing the infrastructure of a new product straight away, but only when user stories ask for it. This design philosophy is captured by three mantras: 1. Consider the simplest thing that could possibly work. 2. You are not going to need it. 3. Once and only once. The second mantra should be understood as never to implement abstraction or infrastructure that you are not reasonably certain of you need. The third mantra means that redundancy is never accepted. If the same code is needed in two places, than an abstraction can and should be introduced that captures the common functionality. This focus on eliminating redundancy results in reduced coupling.

6.4.16 Refactoring

To prevent code rot from hacks to the design and to keep the code as clean, simple and expressive as possible, the design should be continuously updated by means of refactoring. The existence of acceptance and unit tests for every feature and functionality makes it save to refactor. If refactoring breaks any functionality, it is immediately detected and fixed.

6.4.17 Metaphors

Metaphors are typical for XP. They form the big pictures that tie the modules and components of an application together in an intuitive way. They are nothing more than names for parts of the software that intuitively convey their purpose and function. Metaphors prescribe that higher level parts of the application are assigned such names and these are used by the developers in talking about the system.

6.5 IDENTIFYING REQUIREMENTS

There are multiple stakeholders involved in this project and we converse with all of them. The patient stakeholder group is a special case. They have to be approached with great care. In this section we explore how we go about identifying requirements from the different stakeholder groups.

Requirements have to be identified for the initial release plan after every iteration. The identification procedure in both situations is different. The requirements identified after an iteration are based on the evaluation of the latest prototype or working product. For the initial release plan we use the problem domain analyses in section 3 to extract them.

We can identify multiple stakeholders that interact with CommIC. Each of them might be faced with different problems, thus all of them should be consulted when identifying requirements. The stakeholders are the patients, their family, doctors, physiotherapists, dieticians, nurses, the hospital pastor, social workers, the hospital and its ICT department.

Agile advocates face-to-face contact as the best medium to communicate with stakeholders. However, extracting all information by talking to stakeholders is undesirable for a number of reasons. Firstly, medical personal has often very little time. Secondly, through hasty conversation, important details might be skipped and many problem scenarios might not be mentioned. Thirdly, many of the patients for whom the system is intended have great difficulty conversing about their problems. By observing the patient as he communicates, we can mitigate these problems. Therefore we use observation besides conversation as a source of requirements for both the initial release plan and intermediate product evaluations.

In some cases it is undesirable to gather requirements out of personal meetings between patient and family, due to privacy concerns. In these scenarios it is best to talk to the patient and his family after the meeting, while it is stressed that the patient or his visitor do not have to answer a question if they do not want to. As an alternative, observing the patient and the pastor is an option, as their interaction, like a family visit, is social in nature.

6.5.1 Requirements for initial release plan

Many initial requirements are captured from possible communication problem scenarios on the ICU. We identify the patient interaction scenarios in which communication is desirable, what should be communicated in these scenarios and what limitations patients are faced with in these scenarios.

While formulating requirements from very specific scenarios, it is desirable to zoom out of the encountered scenarios into more general problems. This is done by identifying the variable elements that are part of a problem. We stimulate the implementation of our user stories to incorporate abstraction that is needed for expected variability. Additionally, we can make more general claims about the effectiveness of our own or existing software systems. A related advantage is that it helps us to measure the effectiveness of our solution in dealing with a certain problem on multiple separate occasions. In practice we are unlikely to find an identical problem scenario twice. Thus, it is difficult to test an exact problem scenario. Instead, we test with any concrete scenario that falls in a more general category, for which it is a lot easier to find a patient interaction scenario. In generalizing specific scenarios we should make sure not to push ourselves to violating the agile minimalist design principle. If it is not reasonably certain (yet) that we need an abstraction, generalization should probably be avoided.

To identify communication problem scenarios, firstly we observe different stakeholders as they go about their work on the ICU and interact with patients. During patient interaction, the following is the focus of our attention:

- How is communicated? (With what tools?)
- What is communicated?
- Is the communication effective? Why not?

Alongside the observations, we interview the stakeholders, about their experience with patient communication. The interviews are used mainly to identify additional (non-observed) problem scenarios and to determine the frequency and the level of discomfort experienced by the stakeholders in these scenarios. This information translates to a priority level for the related requirement.

- What aspects of your communication with patients do you experience as frustrating? To what level?
- What are your goals during patient interaction?
- What do you need from the patient to achieve these goals?

The goal-question is very important. Its purpose is to gain insight in the meaning of interaction scenarios. This allows us to systematically identify other scenarios that support each goal. For each goal that is identified, we identify the supporting interaction scenarios and for each of these scenarios we identify the experienced problems and severity:

- How often do you need this?
- Is it possible to communicate about this with the patient?
- What misunderstandings do occur during this communication?
- How frequently do these occur?
- How problematic are the consequences of the misunderstandings?

6.5.2 Product evaluation

After each iteration the latest working product is shown to and evaluated with the relevant stakeholders. Whenever possible, this is done though trials with ICU patients. Note that it is not very feasible to test CommIC with simulated patients. The mental and physical condition of ICU patients is a crucial factor in their communication and this cannot be reliably copied to a simulation.

Stakeholders are demonstrated newly implemented functionality and, when appropriate, they are observed and interviewed as they test the functionality of CommIC in the context of problem scenarios that are relevant for testing the user stories implemented in the last iteration. Such a problem scenario always includes a stakeholder goal and obstacles (normally patient disabilities) that the software should help to overcome. We determine to what level the newly implemented features are satisfactory and possibly identify new problems and thus new requirements to avert these.

In order to take the learning aspect CommIC into account, patients receive sufficient time to acquaint themselves to CommIC until they feel confident to work with it. They receive instructions, but not full lessons, since staff would not have time to give these in real practice.

While patients familiarizes themselves with the software and during the rest of the trial, their behavior is observed in order to determine the aspects of the software that are difficult to learn or use.

Following each test scenario, the stakeholders, other than the patient, is asked the following questions.

- Did you achieve you goal? Why not?
- What aspects of the communication did you experience as frustrating? To what level?
- Are there matters that you would like to communicate, but that are too impractical or impossible in the current situation?
- Can you think of additional functionality to CommIC that might be helpful?
- Can you think of possible problems that could occur using CommIC?

When the condition of the patient allows it, the above questions are asked to the patient as well. In many cases answering is too difficult for the patients. Unfortunate, as that has the potential to yield valuable information. We have created a small alternative list of patient-friendly questions that helps to further understand the communication problems from the patient's perspective.

Asking question to ICU patients is challenging as we should take into account their physical capability to answer and their mental condition. Because of their often confused, exhausted and anxious condition, questions are formulated in a very simple unambiguous way. Also the amount of questions is small in order to not exceed the patient's limited attention span. A literature study reveals that multiple quality of life questionnaires exist, that are designed by domain experts especially for questioning ICU patients. Amongst these are the PAEEC quality of life questionnaire (R. Rivera Fernandez, 1996) and the commonly used SF-36 Health Survey. Unfortunately, none of these questionnaires inform about the information useful to us, such as the questions listed below. However, we have copied the format and tone of these questions. Here are the questions and their multiple choice answers.

- Was the communication tiresome?
  - No.
  - Yes, a little. I feel more tired now than before.
  - Yes, it was exhausting
- Do you think that the doctor (the nurse, your wife …) understood you?
- Do you feel helpless?
  - No, I still feel in control.
  - Yes, a little
  - Yes, I feel I have no control over anything or anyone.
- Do you feel isolated?
  - No, I still participate in the world around me.
  - Yes, a little.
  - Yes, I feel separated from the rest of the world.

The Wet Medisch Onderzoek (Ministerie van Volksgezondheid, Welzijn en Sport, 1998) makes it obligatory that approval is obtained from the hospital's medical ethics review board for ICU trials with CommIC. This board ensures ethical conduct by researchers interacting with patients. The trails performed in this research have been approved by said board.

Another obstacle for these trials lies in the fact that the software and hardware should be approved by the IT department of the hospital to be safe for usage inside the ICU. The most important concern with hardware is that it must always be clean and save to use. The software needs to run from a computer in the patient's room. One option is to install the software on the computer already available in the room. This computer, however, is connected to the secured internal network of the hospital, which means software on this terminal could potentially access or compromise sensitive data on this network.

Obtaining approval for multiple AAC solutions for these experiments is very challenging. The best option is thus to use a mobile computer. This has been an important motivation in the decision to develop CommIC for a tablet. See section 7.1.2.

As soon the minimum viable product is released, we will also collect feedback from the stakeholders using the CommIC without us being present. We provide feedback forms alongside the system. The form is found in Appendix G.

For the evaluation at the end of the project, we test CommIC extensively using the same approach as the earlier trials. This way, we determine the effectiveness of the system in satisfying all the identified requirements.

6.6 Language & tools

Implementation of CommIC's software is in C#. There exist multiple reasons for this choice, but the primary one is that we are familiar with this language. Secondly the language is much better suited than Java to communicate with hardware, as it supports unsigned primitive data types native to hardware. Finally, C# offers many additional features over older languages such as Java, such as native language constructs for defining events, operators and higher order functions (delegates). The fact that C# is not cross-platform is not a problem, since most of the hardware we need to communicate with is only available for Windows anyway. C# applications also run on Microsoft's Surface tablets, allowing a portable installation.

A great advantage of using C# is the excellent tooling available. For example, Visual Studio supports 2-way synchronization between class diagrams of the software and the code. This means that by constructing low-level class diagrams code is automatically generated and vice-versa. This makes the redundant work of creating a low level class-diagram and implementing the outline of those classes obsolete. It also ensures that the class diagram of the software is always up-to-date. Visual Studio comes with many equally useful tools to streamline development.

Another very useful feature that is used is **code validation**. This is a heuristic software quality analyses tool that uses many best-practice rules. Fixing problems reported by code analysis helps to increase maintainability, reliability, security and portability of our code. These heuristics are also used as software quality metric in our evaluation.

We use other tools provided by Visual Studio as software metrics as well. These tools include dependency analyses, test coverage, cyclomatic complexity and class coupling.

Visual Studio and C# offer the integrated **WPF GUI framework** and tooling to quickly build complex interfaces. This is a huge advantage, as it allows us to focus on the software functionality and usability of the interface, rather than graphics design.

During development the team makes use of several tools, such as a unit testing framework, a versioning system, a user story tracker and progress measurement tools. These tools all serve to support the agile development process.

The agile development methodology is demanding in terms of internal communication. The developers that work on this project are often separated due to constraints in working hours and physical location. In order to facilitate the agile process and administration in this context, tool support is necessary.

We make use of the agile process management tooling offered by Microsoft's Team Foundation Server (TFS). The software offers an online collaboration environment where user stories and development cycles can be managed. It gives an overview of who is currently working on what, what they already finished and what they still have to do. This information is enriched with statistics that give useful insights in the projects progress, such as the velocity. User stories can contain all information that is useful to define in the agile process, such as their business value, expected implementation costs, subtasks and what higher level goal they are a part of. Furthermore, TFS is tightly integrated with visual studio, minimizing the overhead of using this tool. It also allows to easily couple your code to the user story or task it implements, maximizing tractability from each line of code to its external goal.

Continuous integration is achieved by adopting TFS version control. A big advantage of this tool is the feature of check-in policies. These are forced process rules that each developer should follow before he is allowed to integrate his code with the central repository. The check-in policies ensure that code is always coupled to user stories, that code analysis is performed and that each new code is accompanied by a descriptive tests. Furthermore, new code is rejected by TFS if it cannot be built or there are unit tests that fail. These policies ensure code quality and the adherence to the development process.

TFS version control also has a disadvantage as supposed to version control tools such as git and mercurial. These tools are designed with the central philosophy that branching and merging during software development are normal practices. This matches my experience as a programmer. With multiple programmers working on the same software simultaneously branches naturally form. If not properly supported by the versioning protocol this causes a large overhead in constant merging and conflict resolution. Like Subversion and CVS, with the TFS version control tool branching and merging are relatively complicated processes that cannot constantly be used to reflect actual development workflow. In practice this means that uploads to the central repository happens less often, decreasing the continuity of the integration. This is a minor problem that does not outweigh the advantages.

Skype also fulfills an important role during development. It substitutes for the shared workspace principle. If multiple developers are working at the same time they are in constant contact through a group call. Skype is also the medium for the daily scrum meeting. If developers cannot be simultaneously available for this meeting. They report their progress and problems through the chat.

Tests are written using the Visual Studio unit testing framework and Moq mocking framework. This testing framework has features similar to JUnit and is integrated in Visual Studio and TFS. Writing unit tests is further supported using mock objects. These are simulated objects that mimic the behavior of real objects. There are multiple object characteristics that make it useful for the object to be mocked. The following are listed on the website Agile-code (Maksimovic, 2012):

- The object supplies non-deterministic results (e.g., the current time or the current temperature)
- The object has states that are not easy to create or reproduce (e.g., a network error)
- The object is slow (e.g., a complete database, which would have to be initialized before the test)
- The object does not yet exist or may change behavior

- The object would have to include information and methods exclusively for testing purposes (and not for its actual task).

And additional advantage of mocking is the ability to write independent unit testing for coupled components. If the behavior of one component depends on another's, it cannot directly be independently tested. This means that breaking a dependency causes both the unit tests of the broken component and of the dependent component to fail. This defeats the purpose of the unit test which should only test a single 'unit'. Substituting component dependencies by mocks solves this problem.

7 DEVELOPMENT

7.1 PROJECT START

7.1.1 Initial requirements analysis

Since we are using an agile development approach, our goal is not to exhaustively identify and formalize the requirements in advance. During development many of the initial requirements are updated and new ones are identified.

Requirements are grouped per stakeholder goal. In the list below we differentiate between two types of requirements. There are those that can be implemented as separate functionality of CommIC. These are the user stories that populate the backlog. There are also requirements that state a property that should be adhered to by the system as a whole and that have to be taken into account with the design and implementation of each user story. We call them **global requirements**. Below, they are printed in italics.

Each requirement is accompanied by a priority that has been assigned by the stakeholders. The stakeholders were told to give a priority form one to five, where one means that the feature would be nice to have some day and five indicating that the product is unusable without this feature. Additionally, they were told to more or less equally distribute the requirements over the available priorities. Otherwise, we find that stakeholders have the tendency to assign most requirements either a five or a one, providing us with a lot less information. Priorities have been coded into the list using size for immediate recognition.

Requirement with priority 1
Requirement with priority 2
Requirement with priority 3
Requirement with priority 4
Requirement with priority 5

Medical personal and family want to use the system with multiple different patients.

1. **The system can be setup for new patients or switched to an existing patient.**
2. *The system is portable*.

Medical personal wants patients to be (physically) able to communicate with them.

3. The system can be controlled with the brain.
4. **The system can be controlled with a finger/toe/hand/foot/arm/leg/facial muscles.**
5. **The system can be controlled with the eyes.**
6. **The screen and eye tracker can be correctly positioned.**
7. The system filters accidental input from spasms.
8. *The system does not require good hearing to use.*
9. ***The system does not require good eyesight to use. (blurry, colorblind, limited eye movement)***
10. *The system is usable without eyesight.*
11. The system enables patients to form normal sentences.

Medical personal wants patients to be able to quickly respond to them.

12. **The system allows patients to easily tell they do (not) understand.**
13. **The system allows patients to easily answer a yes/no question.**

Patients want to orient themselves.

14. **The system allows patients to know the time.**
15. **The system can tell patients their location.**
16. The system can show patients a live video of his surroundings.
17. The system can show patients personal photos.
18. The system can act as a mirror for the patient.
19. The system can tell patients the reason for their hospital admission.
20. The system can show patients their treatment history.

Patients want to communicate about their state.

21. **The system allows patients to easily convey an emotion (happy / afraid / restless / sad / angry / gloomy).**
22. **The system allows patients to easily ask about their current health condition and prognosis.**
23. **The system allows patients to easily tell they have an itch or feel uncomfortable, where this is and how severe this is.**
24. **The system allows patients to easily tell they feel nauseous / stuffy / thirsty / hungry / sleepy / restless / dizzy.**

Patients want to communicate about their desires.

25. **The system allows patients to easily ask for rest / food / drink.**
26. **The system allows patients to easily ask for radio / TV and a specific channel.**
27. **The system allows patients to easily request the lights be turned on/off.**
28. The system allows patients to easily request to cycle with their hands or feet or use the squeeze balls.
29. The system allows patients to easily ask for a doctor.
30. The system allows patients to easily ask for a friend or family member.
31. The system allows patients to easily request mechanical ventilation be turned on/off.
32. **The system allows patients to easily call for help.**
33. The system allows patients to easily say they does not feel like it.

Patients want to be distracted from their unpleasant situation.

34. The system offers entertainment in the form of video / TV / music / (audio)books / games.
35. The system offers remote communication through e-mail / messaging / phone calls.
36. The system offers a possibility to browse the web.
37. The system offers mental exercises.
38. The system allows patients to easily make small talk.

39. The system allows patients to watch/listen to video/audio/books provided by their family.

Medical personal wants to know information relevant to treatment.

See requirements 21, 24

40. **The system allows patients to easily tell what kind (stabbing/nagging/cramping), where, when and how much pain they are feeling.**
41. **The system allows patients to easily describe a change in how they feel.**
42. The system allows patients to easily describe the circumstances leading up to their hospital admission.
43. The system allows patients to easily list their pets, especially birds (possible sources of infection).
44. The system allows patients to easily tell where, when and how long they have traveled.
45. The system allows patients to easily list their allergies (antibiotics, iodine, brown patches, X-ray contrast fluid, pain medication).
46. The system allows delirious patients to easily describe what they see/hear.

Patients want the software system to require little physical and mental effort to use.

47. The system offers easier access to functionality that patients personally use more often.
48. **The system preserves patients' configuration across multiple usage sessions.**
49. ***The patient UI can be localized to the patient's culture.***
50. The system can translate written messages from communication partners to the patient's language.
51. ***Using input devices requires as little effort as possible.***
52. ***The system's speed and complexity matches patients' varying capabilities.***
53. ***Using the system does not require memorization.***
54. ***The system is usable within a short attention span.***
55. ***The system is usable for lethargic/slow patients.***
56. ***Using the system requires minimal problem solving capacity.***
57. ***Using the system requires minimal information processing capacity.***
58. ***Using the system does not require experience using computers.***
59. ***Using the system does not require literacy.***

Patients and medical personal want to be able to rely on the software system's functionality.

60. The system notifies medical personal of sudden loss of connection of input or output.
61. ***The system is sturdy enough to withstand patients' delirious episodes.***

Medical personal does not want the patient's wellbeing or healthcare process to be at risk or obstructed.

62. ***Using the system is always save.***
63. ***The system's hardware is shaped such that it can be easily cleaned.***

64. ***The system's hardware is never an obstacle for treatment.***
65. ***The system forms a minimally obstruction to the patient's field of view, such that he can see medical personal that talks to him.***
66. **The system's sound volume is not bothersome or cause patient privacy concerns during visiting hours.**
67. **The system does not produce too much or accidental sound at night or while the patient is sleeping.**

Medical personal wants the usage of the system to be easy and quick.

68. Turning on system power start the software.
69. **The system allows patients to leave a message for later.**
70. ***Communication through the system is quick.***
71. *Setup of the system for new patients is quick.*
72. ***The system continues to function after it and the patient are repositioned many times per day.***

Physiotherapists want the patient to move optimally during their ICU stay

73. ***The system is usable lying flat, sitting straight up or semi-straight up.***
74. ***The system is usable outside of the bed during therapy.***
75. ***The system does not obstruct movement exercises of any body part and continues to function afterwards.***
76. **The system allows patients to easily tell what physical activities they normally partake in and for how much time.**
77. **The system allows patients to easily tell how they, before the ICU, were limited in their mobility and movements.**
78. ***The system is usable from a chair next to the bed.***
79. *The system is usable for patients lying on their side.*

7.1.2 Initial design decisions

Although no systematic prioritization of the initial user stories has been made, in depth discussions with our contacts at ICU have given us a clear picture of the initial direction of the project. Based on these initial insights we have made two important design decisions regarding the hardware. The software is developed to be run on a tablet and after the first release cycle CommIC should be controllable with gaze direction and the touchscreen. We

It was necessary to make these design decisions early before the start of the first development cycle, to allow us to acquire the necessary hardware, arrange financial support for the necessary investments and to setup the development environment needed for our hardware.

The reason for developing the software to run on a tablet is the advantage of portability. Having a portable system means it is easy to deploy it for the patients who need it. Having a separate permanent system in each patients' room regardless of whether it benefits them is undesirable. The investment costs is higher. The portability is also very useful during development and testing when only a single

system is available for both development and testing with many different patients. Another advantage of the tablet is that it is relatively easy to obtain approval for its usage on the ICU. It does not require a lot of space, it is easy to keep it clean, it has no sharp edges for patients to hurt themselves and it does not require a power outlet. This last advantage is more important than it seems, because having a device that is connected to net power in direct contact with a patient is difficult to receive approval for. Safety has to be proven beyond doubt. The final advantage of using a tablet is the touchscreen. Although many patients are not able to use it, it provides a quick and intuitive interface for those who can and for medical personal. Support for touchscreen input is added as a concrete user story extracted from the broader abstract requirement that CommIC can be controlled with a finger/toe/hand/foot/arm/leg/facial muscles.

The choice for gaze direction as the main initial input method is straightforward. It suffices in the input needs of a large percentage of the targeted user base. The majority of patients that are mentally capable of some communication can control their eyes. Any input requiring other movement or force is often very tiring for patients and cannot be relied upon in most cases. Additionally, gaze direction has the potential to be a very efficient means of input and requires little learning. These advantages are confirmed by the most promising existing AAC software systems. The Grid, Tobi Communicator and GazeTalk all use gaze direction as their primary means of input.

The tablet and eye tracker of choice are the Microsoft Surface Pro 3 and the EyeTribe. The EyeTribe is the first affordable eye tracker on the market targeting broad consumer base. It comes with extensive support for third party developers to develop applications to make use of the eye tracker. This support for third party development and the low investment costs make it a great choice for this project. Gaze tracking requires a lot of data to be streamed, so USB 3.0 is required. This is the primary reason for developing for a Microsoft Surface Pro, which are currently the only tablets on the market with USB 3 support. Another important advantage is the ability to run third party windows software on the tablet, as it runs Windows 8.1. Integrating existing accessibility software saves a lot of development time and allow us to take advantage of existing quality solutions.

7.1.3 Startup process evaluation

Prioritization the user stories is the first step of development. We have spoken with two stakeholder representatives for the nursing staff, a medical specialist and a physiotherapist, all working on the ICU. During these meetings a number of difficulties have been encountered. At the start of the project most of our user stories are still very abstract. It is difficult for stakeholders to grasp the exact meaning of a user story in practice without any prototype or design yet available.

We have found that the only way to let stakeholders understand the value of user stories, was to give a picture of how we currently thought this feature would impact the use of CommIC in practice. This is a problem, as it is difficult not to influence the direction of thinking of the stakeholder this way. The goal of letting the stakeholder decide what is important or not is vital for delivering a system that satisfies his needs. This goal is undermined if the importance that the stakeholder claims for a feature is pushed in the direction of our own thoughts on this matter. We have tried to give a minimalistic picture of each user story as a compromise between giving a tangible idea of each feature and influencing stakeholder judgment.

The different stakeholders generally agreed with each other's prioritization. Where there were differences we assigned the highest priority.

7.2 SPRINT 1

7.2.1 User stories

The first goal is to create a **minimal viable product** (MVP). This means a minimal implementation of CommIC that is usable in practice. Since we defined requirement priority five as necessary is such a system, the first sprint should address only priority five requirements. It is wise to not incorporate all of these requirements at once though, but to divide the first feedback loop into multiple iterations. It is possible that intermediate iterations might not be able to deliver useful stakeholder feedback. However, the iterations allow us to limit the number of requirements in focus to keep them manageable and to incorporate the feedback from the implementation of previous sprints into the design of the following sprints.

The user stories listed below are implemented in the first sprint. The second user story refers to all requirements that start with "The system allows patients to easily ask/tell". Naturally, these user stories are not independent, as they are supposed to be. They share a common implementation. We have thus extracted this common implementation into a separate user story.

- The system can be controlled with the eyes.
- The system allows patients to easily communicate common messages.

A number of global requirements are particularly relevant for the design in this sprint.

- *The system continues to function after it and the patient are repositioned many times per day.*
- *Using input devices requires as little effort as possible.*
- *The system's speed and complexity matches patients' varying capabilities.*
- *Communication through the system is quick.*

7.2.2 Design

7.2.2.1 Gaze tracking

The first functionality implemented for the MVP is the gaze tracking. Most of this functionality is already provided by The EyeTribe, which provides a convenient high level API to communicate with the hardware, perform calibration and display eye position. Integrating this functionality into our software leaves little degree of freedom and does not require any major design decisions.

Figure 1 gives an overview of the package structure of the software. The Main package contains the main application and contains only our own code, though the EyeTribe Client and Tools package are modified to suit our needs. The eye tracker communicates with a closed source server via USB. The EyeTribe Client communicates with the server through http network messages. In turn, our own code sends commands to the client and listens for events by extending several interfaces and registering these extensions as observers at the client. The tools package does the same and contains logic and UI components for an eye tracker status window and the calibration process.

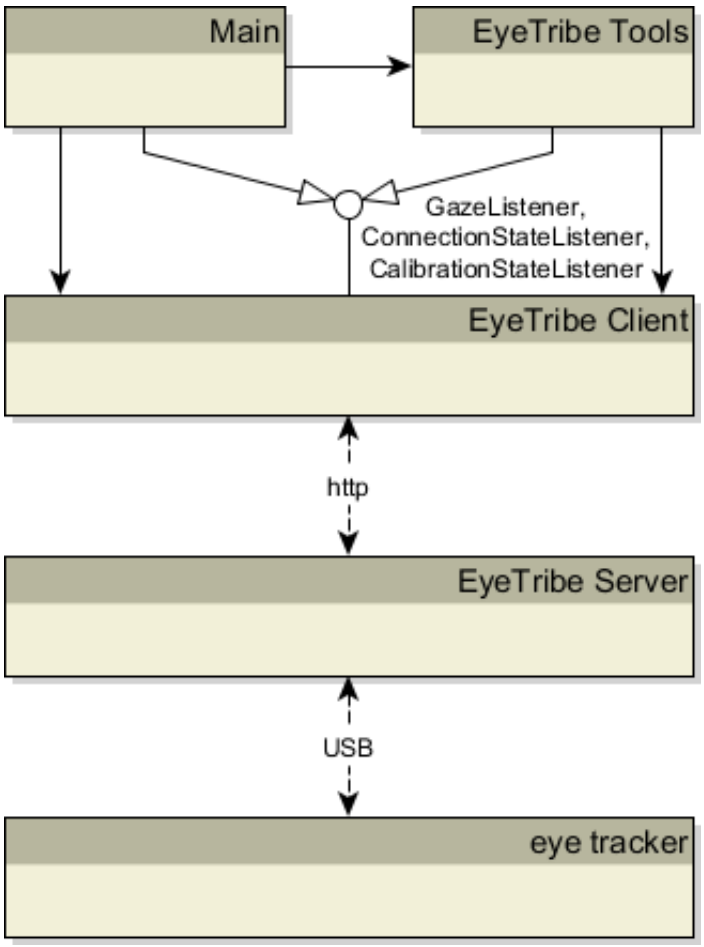


Figure 1: High-level components

7.2.2.2 Choice

Second on the agenda is the ability of the patient to choose an output option. There are many user stories that require that the patient can easily communicate a certain message. Only a few of these messages should be provided to the patient at any time in order to limit the amount to information he needs to process. The logical consequence is that the messages should be placed into a hierarchy that the patient can navigate. Each category can contain messages and deeper categories. Furthermore, the messages that are available and their hierarchy should be dynamic. Due to the diverse nature of the patient group the interface needs to be very flexible. This stems directly from requirement 52: The system's speed and complexity matches patients' varying capabilities. The choices offered to the patient have to be handled in a generic way and form an integral part of the design.

It becomes clear that the message output option should be implemented with the command pattern. This allows us to separate the dynamic representation and invocation of these output options in the UI from their implementation, which is of no concern for the UI. The command objects can also be used to create the dynamic hierarchy we need using a composite pattern.

A quick CV-analysis based on the user stories tells us that communicating a message is just one of the possibilities that the patient should be able to easily choose from. For example the patient can play a piece of music or send an e-mail. Since we already introduced the command pattern, our implementation can be easily extended with this functionality. We simply introduce the new concrete command types in addition to the message command. It is likely that we encounter many additional commands during development. We know now that our design is open to this kind of extension, following OCP. In most

cases a new type of Choice only differs in the action taken upon making the choice. Implementing a separate class for each such action causes needless complexity. Instead we use composition to create a generic `CommandChoice` class that delegates `MakeChoice` to a higher order function (`Action`).

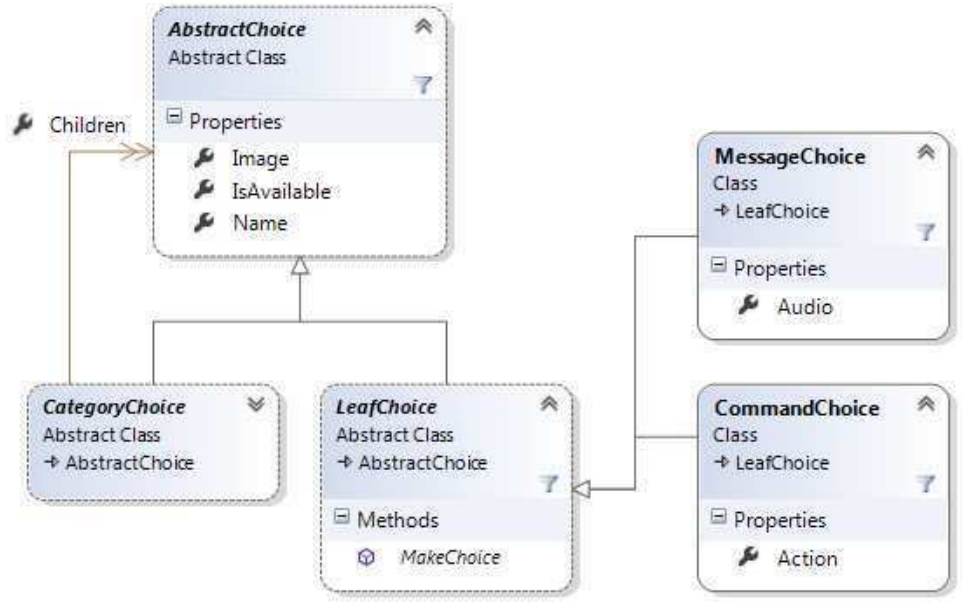


Figure 2. Initial design of Choice

Our CV-analysis provides us with some useful properties for the classes. Each `LeafChoice` can off course be made, which yields the abstract method `MakeChoice`. In order to make a choice the patient should be able to identify each choice. A name and an image seem adequate. A message always has some audio coupled to it to speak the message aloud. Some user choices should not be available for certain patients, such as the choice to call for help that patients might abuse. A disable option is added. Naturally, at least the properties of audio and name are variable due to the multi-lingual requirement. We ignore this in the design, because Visual Studio has built-in support for internationalization. All culture dependent properties can be defined in a resource file that can be swapped depending on the culture context.

An interesting question that springs to mind is whether the parental relationship between a choice and its category is unique. Existing user stories suggest a choice can be moved to belong to a different category. Consider the patient group with bad eye sight. Only a few choices at a time can be presented to them visually. The same can be said for patients that are very slow or confused. Presenting less choices at a time is preferable to them. To some extend this can be solved by spreading choices in each category over multiple pages, ordering them by likelihood. The existence of many pages greatly increases the time to lookup certain choices and the required attention span needed to find a choice. It is desirable to offer large categories to users who can handle many choices at once for maximally efficient communication and at the same time very small categories.

What does this mean for our design? We could introduce a multiple independent category hierarchies. This does not seem a good idea though, since a patient that deteriorates or improves his condition might change the used category size. The confrontation with a changed hierarchy can be very confusing. The better option is to introduce sub-categorization within each category that can optionally

be used as an extra navigation level. This approach does not require alteration of our design, since categories and subcategories have the exact same structure. The presentation logic can determine whether to expand or collapse a category. This only needs to be determined once when the software starts or the configuration is changed. So, we add an `IsCollapsed` property to `CategoryChoice`.

Presentation logic such as this does not belong in our model classes. This would violate SRP. We can solve this by applying the visitor pattern. A visitor can traverse the choice hierarchy to determine which categories are to be collapsed or expanded. This should not be done too lightly though, due to the added complexity. While the solution better adopts the SRP, it goes against agile's simple design principle. Since at this point we do not expect other operations to be performed on the choice tree, not opening the design for this type of change seems preferable. We add the logic to the `CollapseHierarchy` method in `Choosable` instead.

Let us consider another user story. Patients should be able to indicate the kind of pain they are feeling, where this pain is located, how much pain they are feeling and how frequently. This a significantly different message from the ones we reasoned about until now. The four aspects; kind, location, severity and frequency are all part of the same message, yet each is a different choice. Clearly we were wrong to abstract the concept of message to choice. The pain example is not a unique case, other user stories such as communicating about travel history have multiple sub-messages as well. Furthermore, this seems true as well for the `CommandChoice`. Take for example the choice of music, consisting of artist, genre, album, etc.

Thus, every `LeafChoice` can have several subchoices. Each subchoice is an additional choice category with several concrete choices or even more categories. The structure of these choices seems identical to our current `CategoryChoice`. The difference is in the behavior of making the choice. A choice consisting of subchoices is different from a `CategoryChoice` in that all of its children are chosen. It also differs from `LeafChoices`, because making a choice is this case is not the same as performing an action associated with that choice. Choosing a subchoice might result in an action – such as speaking it aloud – but this is not necessarily always the case. More interestingly, choosing a choice with subchoices should result in an action, but only after all the subchoices have been made. We could introduce a special `SubchoicedChoice` class. There is, however, a more simple option is to simply let behavior depend on the existence of subchoices.

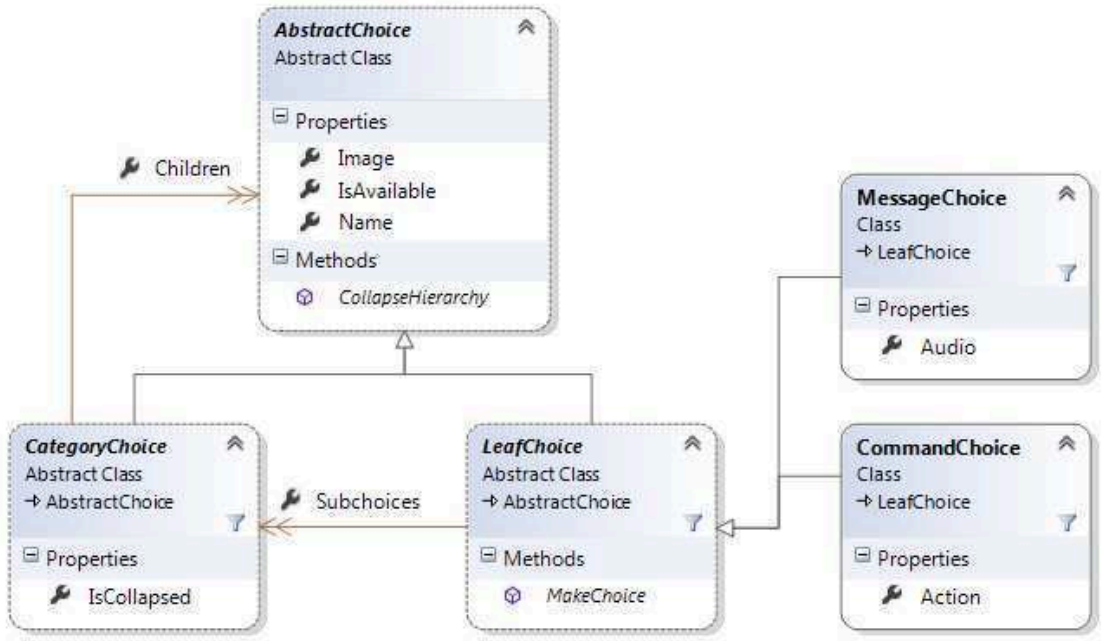


Figure 3. Improved design of Choice

7.2.2.3 MVVM

We have chosen to use the **Model-View-Viewmodel design pattern** (MVVM) as opposed to the more traditional MVC pattern. The MVVM pattern is a three layer pattern that is used for loose coupling between the user interface and business logic. The layers are **view, viewmodel** and **model,** each of them discussed below. (Microsoft, 2014)

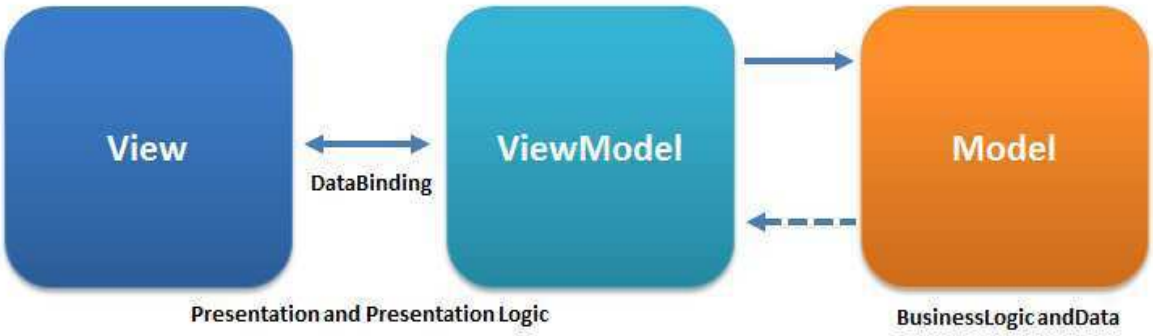


Figure 4: MVVM layers

The view layer defines the user interface elements. Only visual elements are contained, including controls, styles and visual behavior such as animations. The view depends on a **data context** that is provided by the viewmodel or by the model directly. The view includes references to properties and commands defined in the data context. This is called **data binding**. Within WPF a special syntax is used to declare these bindings. Via specific WPF interfaces that can be provided by the viewmodel layer, two-way binding is also provided. This means updates to bound properties can be made in the UI. For each binding it is also possible to specify converters to make sure the data has a correct format. The dynamic reference to the data context is the only dependency of the view. It is thus mostly independent. It is easily

possible to change this reference to another viewmodel or model (that offers the same properties and commands).

The model layer forms the body of the application and contains all the business logic and data. The model objects can be observed by viewmodels or views. This is either achieved by encapsulation or through the implementation of a change notification interfaces. Viewmodels and views listen to the events declared in these interfaces to forward these to the UI. Either way, the models do not need to know about their observers and thus are independent of them.

The viewmodel layer is the lubricant between the view and model layer. Although is common for the view to bypass the viewmodel layer, communicating through a viewmodel offers many benefits.

The main purpose of viewmodels is to act as a façade between the view and model layer. The **façade pattern** defines a façade as an interface between different subsystems that hides the complexity of server subsystem to the client subsystem and reduces the number of dependencies. This means a viewmodel provides a simple interface to one or more views through which complex models can be accessed. Also model data can be converted in such a way that it can be easily consumed by the UI. By use of viewmodels, model classes never have to worry about the representation of their data.

Furthermore, viewmodels extend the data from the model with extra functionality. It present model data and functionality in a way that views can data-bind to it. Data is presented through special properties and functionality through use of the command pattern. The later means that an action that the user perform through the UI is contained in a separate object, which ensures a cohesive viewmodel layer. Presentation logic is also housed in the viewmodel. This includes sorting, filtering and grouping of data.

Moreover, viewmodels can release models from the need to validate their data. Viewmodels can house logic to check the consistency and validity of user input. This might not always seem desirable, as what constitutes valid data is an inherent property of data itself. The application validation to user input on the other hand should not be a responsibility of the model, in accordance with the SRP.

Finally, viewmodels have a purpose to contain state information that is shared between multiple views. This shared state does not belong in a model object, because it would break when multiple views instances exist. Furthermore it would violate SRP. Implementing this shared state in the views would lead to code duplication. This same can be argued for shared UI behavior, which is also implemented in the viewmodel layer. A good example of such shared behavior is the gaze tracker that is part of many UI screens, but only when eye tracking is enabled.

There are several advantages of using this MVVM over MVC. MVVM offers a loose coupling with the model, following DIP. In MVC the view and controller are tightly coupled with the model. They have direct knowledge of all the servicing model classes and all there complexity. Changes in the model often affect the view and controller. Using the viewmodel layer as a façade together with data binding removes this coupling and the need for a controller. Furthermore, there is no tight coupling between the view and controller. The view is independent and can be reused, whereas in MVC the controller listens directly to view components and also modifies the view directly. Finally, the use of the command pattern in the viewmodel layer ensures for clear separation of responsibilities over command classes. The logic contained in these commands would in MVC be grouped together in methods of the controller, violating the single responsibility principle.

Finally, let us demonstrate the use of MVVM by applying it on our choice model. In the diagram below a schematic overview is given of our implementation. Notice that the dashed line represent (data)bindings.

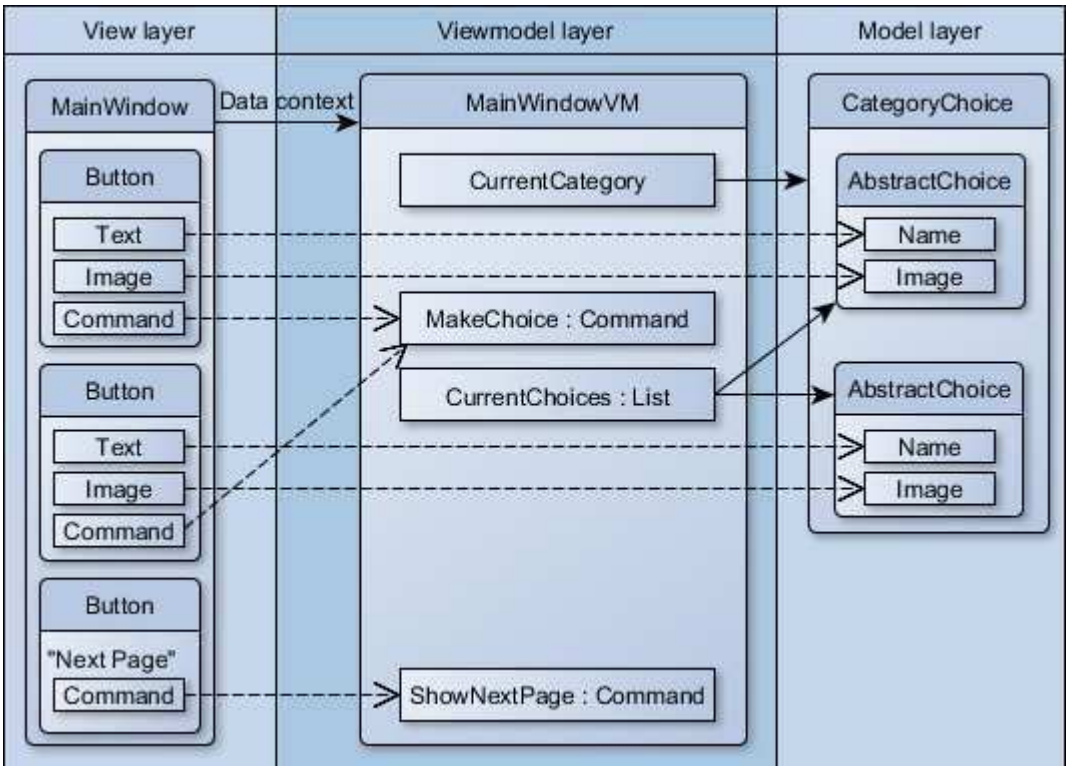


Figure 5. Application of MVVM

Our choices are represented in the UI by buttons that are part of a `TileView`. This view has a `ChoicePage` as its `DataContext` which provides it with the current choices it should display. `TileView` only has to display a button for each `AbstractChoice` part of `ChoicePage`'s `CurrentChoices` property. Buttons databind to `AbstractChoice`s directly. Each of these buttons also binds to the `MakeChoiceCommand`. This command either updates the `CurrentChoices`, if the made choice was a `CategoryChoice`, or executes a `LeafChoice`'s action. One special button in the view can issues a `ShowNextPageCommand`, which updates `CurrentChoices` with choices within the current category that overflowed the current page. All presentation logic that determines which choices are visible when is handled by the `ChoicePage`.

The viewmodel hides some model complexity by only presenting the view with abstract choices. For the view it does not matter what concrete type the choice has, thus it does not know about this.

7.2.3 Process evaluation

Soon after the start of the first sprint it became apparent that the planning of this sprint was very unrealistic. Planning in the early stage of a project is never easy and there are several factors that enlarge this problem. First of all, we still have to familiarize ourselves with the new tools and languages and we found that several application wide design decisions had to be researched before we could effectively make progress. Secondly, all of the developers have overestimated the amount of time they could spend on the project. Thirdly, the researcher, who is the main programmer, has almost no experience building graphical user interfaces, while this forms the major part of the implementation effort. Consequently, it is impossible to deliver any working prototype within the original timeframe of the first sprint. We are

forced to delay the delivery date of the minimal viable product to an unspecified date. After this initial release we resume the intended short sprint cycles. This means the agile practice for sprint planning is not used until the first release. However, user story selection for implementation happens as planned, just not on a budget.

We have made several agreements about the involvement of stakeholders. Our main stakeholder representative for the nursing staff is R. Damink. He is involved throughout the project and is the main contact of the product owner within the ICU of the MST hospital. For the first sprint we have made a number of concrete agreements with R. Damink, to help streamline our agile development process.

To minimize the feedback loop, the minimum viable product should be directly deployed on the ICU for continuous real world testing. We shall present and demonstrate the product to the medical staff. Then R. Damink supervises its usage on the ICU in our absence and collect feedback on our behalf. We also agree to supply him with the feedback forms (Appendix G) to accompany CommIC to further facilitate the process. At the start of each consecutive development cycle, we discuss this feedback with him to identify new user stories and priorities. Even though R. Damink has no time to be a full time product owner, he is always reachable for questions by phone during working hours. This continuous testing of the system and the immediate contact compensates for the lack of a domain expert in the role of product owner.

R. Damink has also agreed to help us set up usage tests on the ICU to test specific features. This entails helping us select suitable patients and arranging the support of other medical staff during these tests.

Additional contacts have been arranged within the medical specialist and physiotherapist stakeholder groups. They are respectively V. Silderhuis and M. Braakhuis. Both of them are available on call and are prepared to be involved with field trials.

7.2.4 Feedback

The first sprint does not yet deliver a prototype that is sufficiently evolved to receive useful feedback from non-technical stakeholders.

7.3 SPRINT 2

7.3.1 Requirements

The requirements listed above are concerned with the positioning of CommIC, initially the screen and the eye tracker. This is an almost completely separate part of the system and is thus addressed in a separate sprint.

- The system is portable.
- The screen and eye tracker can be correctly positioned.
- The system is sturdy enough to withstand patients' delirious episodes.
- Using the system is always save.
- The system's hardware is shaped such that it can be easily cleaned.
- The system's hardware is never an obstacle for treatment.
- The system is usable lying flat, sitting straight up or semi-straight up.
- The system is usable outside of the bed during therapy.

- The system does not obstruct movement exercises of any body part and continues to function afterward.
- The system is usable from a chair next to the bed.

7.3.2 Design & Process evaluation

The result of this sprint is the design and construction of a prototype mechanical arm, part of the MVP. Although the problem addressed by this sprint is not of a software engineering nature, the development philosophy of agile development does extend to this problem. In fact, the design of the arm has gone through multiple feedback iterations using 3D models as intermediate working products. Also, thanks to CAD models, even design simplicity and refactoring are concepts that fit this mechanical engineering challenge.

The development of the arm has not been completed. Construction of the first physical prototype has taken up significant time. MST's department of medical technology has inspected the arm and given useful feedback on necessary improvements before the arm is allowed on the ICU. Implementing these improvements would have delayed the research project significantly. A temporary solution is in use that attaches the tablet and eye tracker at the end of an existing arm borrowed from the hospital. This arm was originally used for positioning an EEG device. It does not satisfy all the requirements of our system. Particularly, the arm is more difficult to precisely position and keep in place. Also it does not support all of the desired degrees of freedom for positioning the tablet. However, this arm is certified as completely save and can thus on short term be used for trials and deployment of CommIC.

Although significant effort has gone into design and construction of the arm, this problem falls outside the scope of this research and is not further addressed here.

7.4 SPRINT 3

7.4.1 User stories

This sprint addresses the localization requirement(s) of the system. The localization mechanism is necessary infrastructure that needs to be built from the start, as it is pervasive in all the UI. We implement the following user story.

- The patient UI can be localized to the patient's culture.

One additional user story needs to be taken into account while choosing a localization strategy, though it is not yet implemented:

- The system can translate written messages from communication partners to the patient's language.

During design and implementation it appeared that this second user story implies two separately implementable features. We replaced the user story in the backlog with the following two.

- The system can dynamically switch between output message localizations.
- The system allows the communication partner to input messages in the output message language and translate them to the patient's language.

7.4.2 Design & Process evaluation

Originally, the localization requirements were not planned to be addressed in a separate sprint. However, the problem proved to be a lot more complex than anticipated. The second requirement suggests that it can happen that patients and communication partners do not have a language in common. This raises a number of problems. When the patient configures CommIC to his own language, the output messages he produces should not change language as well. Allowing a Polish patient to speak in Polish to medical personal is not very helpful. The input and output of CommIC should thus be separately localized.

Localizing the output is necessary, since foreign patients likely will receive foreign visitors that speak (only) his language. Moreover the output language should be dynamically changeable at runtime, since the patient needs to be able to alternately speak to his family and to medical personal. These new requirements are problematic, as the default internationalization strategy used in WPF applications, which we originally adopted, only supports loading a single culture on application startup.

We tried using third party internationalization approaches, but none of these had proper tool support for translation. Also many solutions do not offer resource types other than strings. These resource types can alternatively be internationalized using paths and type converters. However, since our application makes heavy use of resources such as icons and audio files this is not an ideal solution. These resources would have to be managed manually in this case, allowing runtime bugs to occur.

Microsoft does offer resource and internationalization tools that satisfy our requirements. They were not designed to integrate with WPF, but can be made accessible to the UI. The only downside is that the image format supported by the internationalization tools is incompatible with WPF. Conversion to a compatible format is inefficient. We have made the compromise to put image resources in the WPF's own resource framework, which makes them not localizable. At this time we do not foresee the need to support this.

7.5 SPRINT 4

7.5.1 User stories

In this sprint we implement the user stories that form the concrete choices that we made the general implementation for in sprint 1. These user stories are still not independent. The overall GUI structure has to be designed with all choices in mind. This allows us to distribute the choices over categories such that navigation is optimal. Additionally, we should reason about the full range of possible choices to make an educated UI design, as is explained in this section. For these two reasons all user stories starting with "The system allows patients to easily ask/tell…" are taken into account during design, but only the ones that are listed below are (fully) implemented.

- The system allows patients to easily convey an emotion (happy / afraid / restless / sad / angry / gloomy).
- The system allows patients to easily ask about their current health condition and prognosis.
- The system allows patients to easily tell they have an itch or feel uncomfortable, where this is and how severe this is.
- The system allows patients to easily tell they feel nauseous / stuffy / thirsty / hungry / sleepy / restless / dizzy.
- The system allows patients to easily ask for rest / food / drink.

- The system allows patients to easily ask for radio / TV and a specific channel.
- The system allows patients to easily request the lights be turned on/off.
- The system allows patients to easily request to cycle with their hands or feet or use the squeeze balls.
- The system allows patients to easily ask for a doctor.
- The system allows patients to easily ask for a friend or family member.
- The system allows patients to easily request mechanical ventilation be turned on/off.
- The system allows patients to easily say they do not feel like it.
- The system allows patients to easily describe a change in how they feel.
- The system allows patients to easily list their pets, especially birds (possible sources of infection).
- The system allows patients to easily list their allergies (antibiotics, iodine, brown patches, X-ray contrast fluid, pain medication).
- The system allows delirious patients to easily describe what they see/hear.

During the design of the GUI that offers the choices, special attention should be given to global requirements concerning patients' physical and cognitive abilities. After all, transferring the information of what choices are available and allowing the patient to pick one of them are the main interactions between patient and CommIC. The relevant requirements are listed below.

- The system's speed and complexity matches patients' varying capabilities.
- Using the system does not require memorization.
- The system is usable within a short attention span.
- The system is usable for lethargic/slow patients.
- Using the system requires minimal problem solving capacity.
- The system requires minimal information processing capacity.
- Using the system does not require experience using computers.
- Using the system does not require literacy.
- Using the system does not require good eyesight. (blurry, colorblind, limited eye movement)
- Using the system does not require good hearing.

In the next subsection we describe a UI design intended to satisfy these requirements. This yields the new user story "Controls should be made implementing the look and functionality of the basic semantic constructs", which we implement during this sprint.

7.5.2 Design

7.5.2.1 Cognitively effective visual design

There are a number of UI design principles that need to be taken into account to make a UI that is **cognitively effective**, optimizing the transfer of information to the user. For this design we use the principles proposed in "The Physics of Notation" (Moody, 2009). This heavily referenced paper synthesizes these principles by combining theories and empirical evidence from a wide range of fields. These include communication, semiotics, graphic design, visual perception, psychophysics, cognitive psychology, HCI, information visualization, information systems, education, cartography, and diagrammatic reasoning.

"Newell and Simon showed that human beings can be considered as information processing systems. Designing cognitively effective visual notations (of information) can, therefore, be seen as a problem of

optimizing them for processing by the human mind, in the same way that software systems are optimized for particular hardware. Principles of human graphical information processing provide the basis for making informed choices among the infinite possibilities in the graphic design space." (Moody, 2009)

Before we can discuss the principles we need to introduce some terminology. Our goal is to encode information into a **visual notation**. A visual notation consists of a set of **graphical symbols** (the **visual vocabulary**), a set of compositional rules and definitions of the meaning of each symbol (the **visual semantics**). The visual vocabulary and compositional rules together form the **visual syntax**.

Graphical symbols can be anything from lines to 3D graphic images or textual elements. They are used to symbolize (perceptually represent) elemental building blocks of information (**semantic constructs**). The meanings of graphical symbols are defined by mapping them to the constructs they represent. The information bearing part of a UI is composed of symbol instances (**tokens**), arranged according to the rules of compositional rules.

For example, one of the main semantic constructs is the choice and a cognitively effective visual notation clearly distinguishes between graphical symbols that can be chosen and those that cannot. Each concrete choice for the patient is represented by a token of the choice graphical symbol.

The solution space of visual design consists of eight **visual variables**, as identified in the book "Semiology of Graphics" (Bertin, 1983). The visual variables define a set of atomic building blocks that can be used to construct any visual representation in the same way the periodic table can be used to construct any chemical compound. Each variable has an infinite number of physical variations but only a finite number of **perceptible steps**, the minimum variation that can clearly be perceived.

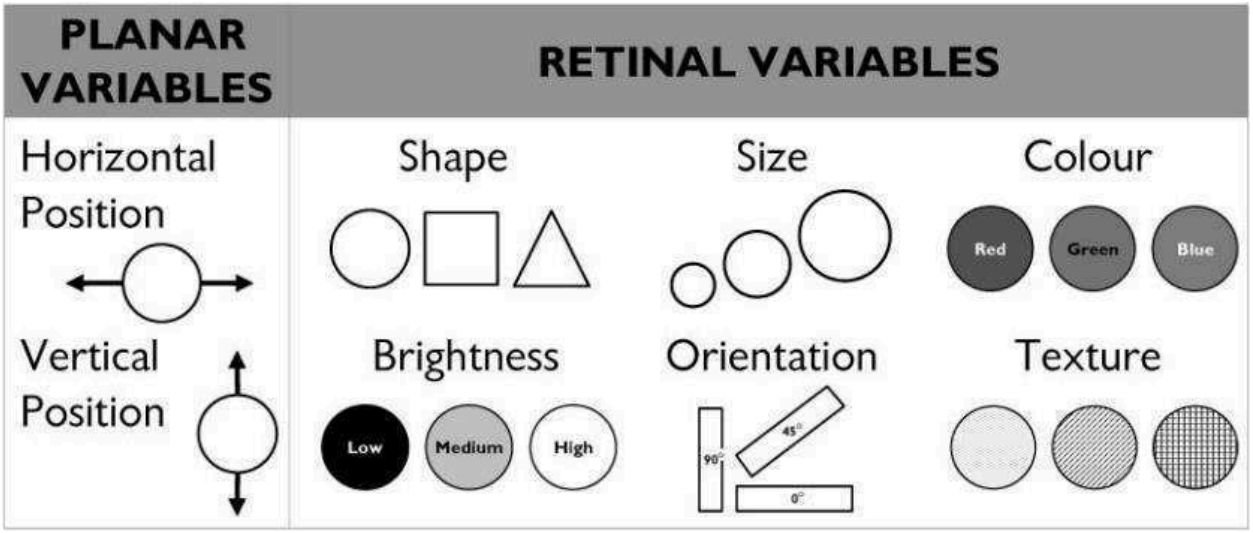


Figure 6. Visual Variables

In the following three sections the design of our GUI is described. The semantics constructs are identified, the visual design principles applied and the visual syntax constructed. Although represented in this order, in reality these three steps have been mixed up in an iterative process. Using the agile principles, we have been able to apply the feedback from each step to the previous ones.

7.5.2.2 Semantic constructs

Below, the semantic constructs are listed that can be synthesized from the requirements. Not all of them could be identified trivially. Many were synthesized using the visual design principles. This is explained in the next section.

Some semantic constructs have subtypes that can be inferred from their properties. For example Option has i.a. properties Favorite and Selected. This results in the subtypes Default Option (an option that is neither Favorite nor Selected, Favorite Option, Selected Option and Favorite Selected Option. Not all combinations of properties are always possible. The visual variables for the default subtype can be overridden by inferred subtypes.

- **Option**: An input option that can be chosen. Choosing an Option is the sole form of interaction. There are two subtypes of the Option construct.
  - **Navigation Option**: An Option to navigate the GUI. There are three different subtypes.
    - **Cancel**: Cancel the current input and return to the root Category or Module.
    - **Next**: Go to the Page to the right.
    - **Previous**: Go to the Page to the left.
  - **Choosable Option**: An Option to choose a Choosable

  In addition to these subtypes it is useful to differentiate between Options based on a semantic property of their instances. The property below was extracted from the requirements.
  - **Selected**: Whether an Option is (being) selected.
- **Gaze pointer**: Indication of the location on the screen where the user is looking, according to the eye tracker.
- **Choosable**: A convenient abstraction of construct types that can possibly be chosen through an option. There are multiple such constructs.
  - **Module**: A separate part of the program offering a certain type of functionality.
  - **Category**: A category containing similar Choosables. There is one property.
    - **Defining**: whether a category forms a context that helps define its children's semantics.
  - **Message**: A message that is relayed to the listener.
  - **Action**: An action that is performed by the software.

  Additional subtypes can be inferred from the following property.
  - **Virtual**: Whether a Choosable is abstract and needs further details.

  In addition to these subtypes it is useful to differentiate between Choosables based on semantic properties of their instances. Below are possibly useful properties that we extracted from the requirements.
  - **Favorite**: Whether a Choosable is frequently chosen by the user.
  - **Disabled**: Whether a Choosable is disabled
  - **Visual appearance**: what the Choosable looks like.
  - **Order**: how an ordinal Choosable is ordered relative to other ordinal Choosables.
  - **Spatial position**: where a Choosable is located relative to other Choosables.
  - **Shape**: the 2-dimensional shape to the Choosable.
- **Chooses relation**: specifies that selecting a certain Option chooses a certain Choosable.
- **Subchoice**: A choice that further defines a Choosable. A Subchoice is itself never chosen, but sequentially presented automatically. The subchoice has a subtype induced from a property.
  - **Focused**: Whether a Subchoice is in being made.

- **Choosable Parent – Child relation**: Indication that a Choosable is a child of a certain other construct. Different relations can be identified, depending on type of Choosable involved.
  - **Module – Choosable relation**: Indication that a (top-level) Choosable is part of a certain Module.
  - **Category – Choosable relation**: Indication that a Choosable is part of a certain Category.
  - **Message/Action – Subchoice relation**: Indication that a Subchoice defines a certain Message or Action.
  - **Subchoice – Choosable relation**: Indication that a Choosable is part of a certain Subchoice.
- **Subchoice sequence relation**: Indication that one subchoice follows/is followed by another.
- **Page**: Subset of Choosables in same Category that is shown simultaneously.

Note that the concept of choice has been (re)named to Choosable. During our in-depth analysis of the semantic constructs, we realized that "Choice" is an ambiguous term. A choice can refer to both a thing that can be chosen and to a range of possibilities from which can be chosen. Until now we have used these different meanings interchangeably, as our earlier defined category and leave choices fall under the first interpretation and subchoices fell under the second. Our design needs to be refactored to reflect this change. From here on use the term "Choosable" to indicate an item that can be chosen, while choice (as in the Subchoice construct) indicates a set of Choosables from which only one can be chosen.

Using the identified semantic constructs as building blocks a semantic structure of the patient GUI has been created, showing all Choosables and Subchoices with their parent-child relations. The structure incorporates all requirements of the form of "the system allows patients to easily ask/tell…". The organization was guided by the visual design principles as explained in the next section. Figure 7 displays this structure. For convenience, the "Pictures…" Category has been collapsed and is separately displayed in Figure 8.

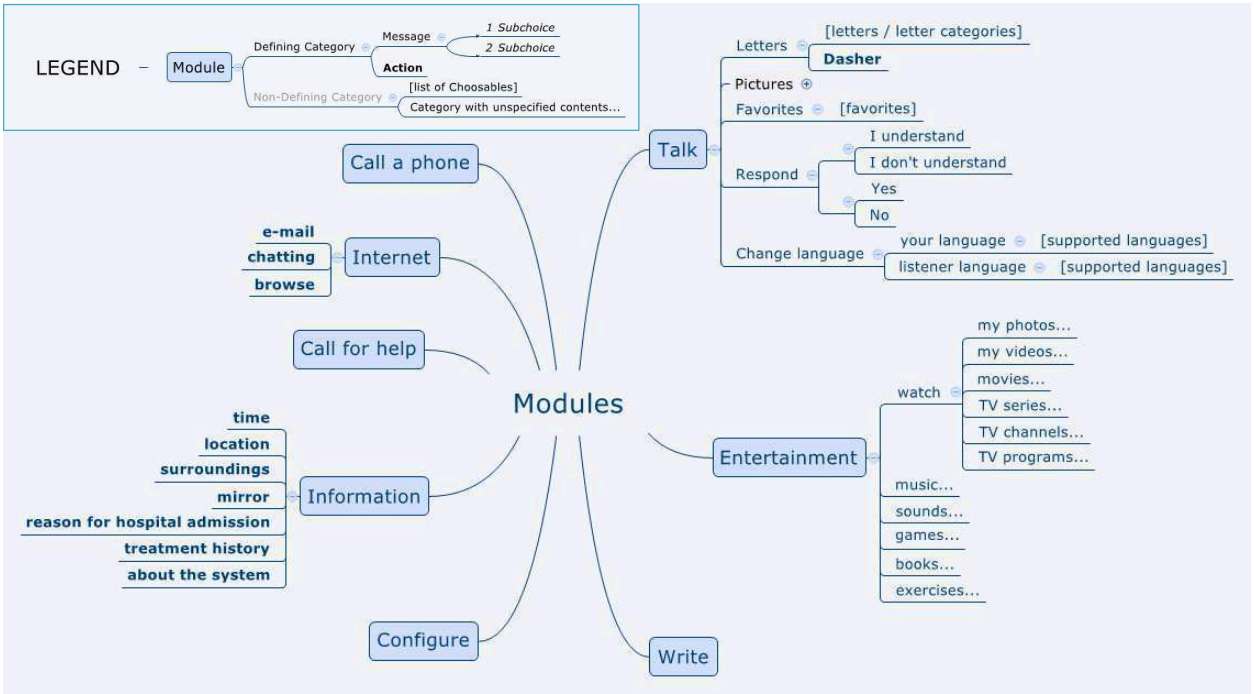


Figure 7. Semantic structure of patient UI

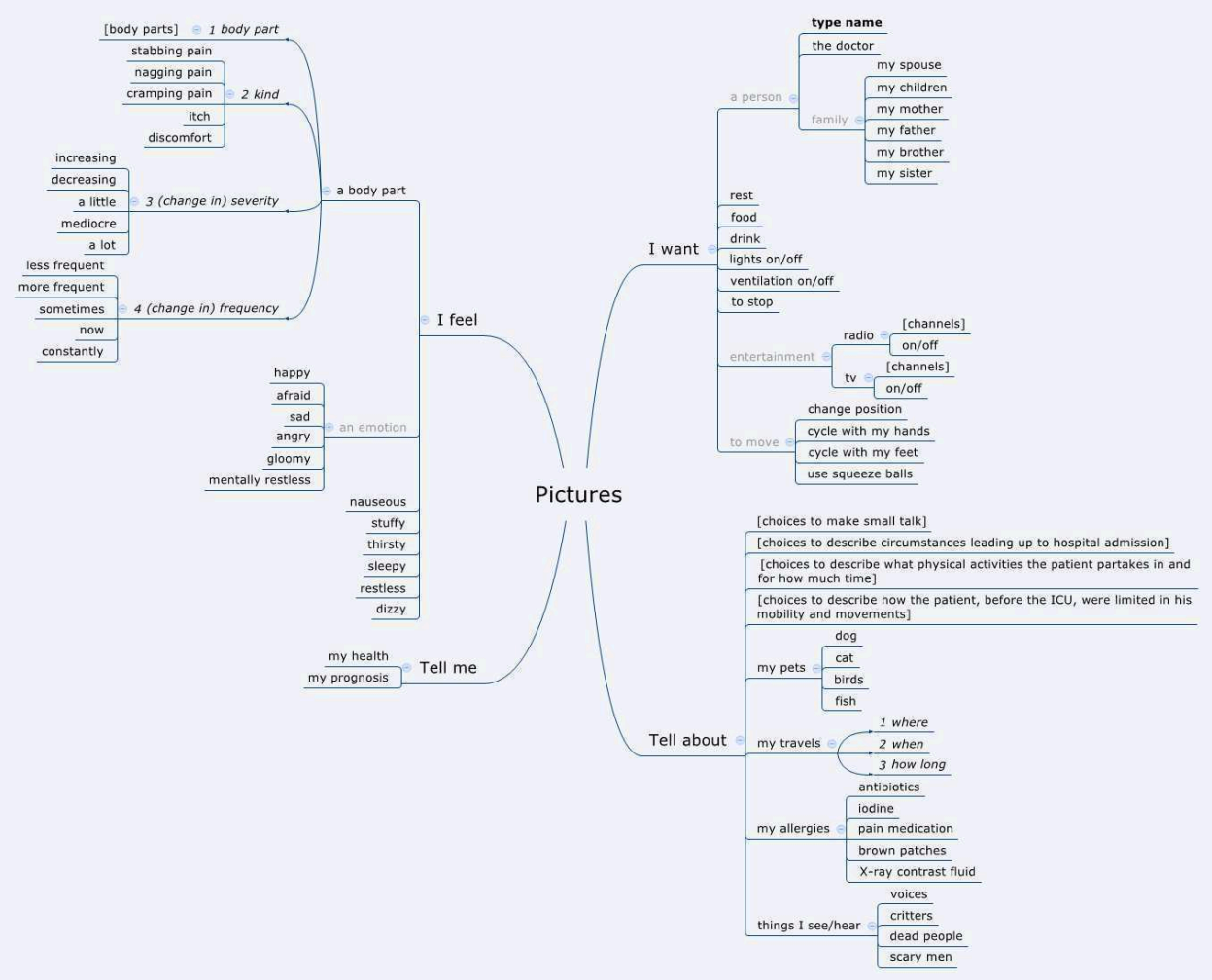


Figure 8. Semantic Structure of picture talk

Incidentally, the semantic construct we have identified serve a purpose as metaphors. No all of them identify high level components, but these definitions prove useful for talking and reasoning about the system throughout the project.

7.5.2.3 Visual design principles

In this subsection the visual design principles and their application to the design (emphasized in the text) are discussed.

7.5.2.3.1 Perceptual Discriminability

Perceptual discriminability is the ease and accuracy with which graphical symbols can be differentiated from each other. Symbols should have sufficient **visual distance** to be easily and quickly discriminable. This distance is measured by the number of visual variables on which they differ and the size of these differences measured by the number of perceptible steps. A larger visual distance makes it easier to discriminate symbols and the likeliness of confusing one for another is reduced. *The symbols of our semantic constructs should have pairwise sufficient visual distance*.

What is perceptible can differ per user group. A good design ensures different symbols are perceptually different for all of these groups. From the requirements it follows that *we are restricted in the use of color and texture.* Colorblind patients might not be able to discern symbols that differ solely by color. Patients that see blurry might not be able to discern textures. *Background textures shouldn't be used at all*, as a blurry background texture can be seen as reduced brightness instead.

This does not mean these visual variables color and texture should not be used to differentiate between symbols, but that they should be used in addition to other variables. This practice to use multiple variables to discriminate is called **redundant coding** and increases the visual distance. *We use redundant coding in our GUI design whenever possible*.

It is important that each graphical symbol has a unique value on at least one visual variable. Identifying a symbol based on a unique combination of visual variables is cognitively demanding. Our GUI *avoids this*.

Another important implication is that *discrimination should not be text dependent*. Different texts have zero visual distance as they are permutations of an identical symbol set. It should only be used as a last resort, as there are better ways to manage graphic complexity (see section 7.5.2.3.8). This design choice agrees with the requirement to design for illiterates.

Discriminability problems can also occur when dissimilar symbols are used to represent the same or similar constructs. This is the problem of **visual-semantic congruence**: the visual distance between symbols should be consistent with semantic distance between the constructs they represent. Generally, similar symbols should be used to represent similar constructs. This concept is applied in our UI design by keeping the visual distance between similar semantic constructs small. *The visual distance between the similar Choosable subtypes and between the similar Option subtypes is kept small. The visual distance between Choosable tokens in the same category should be even smaller, as they have similar semantics.*

7.5.2.3.2 Semiotic clarity

Ideally, there exists a one-to-one correspondence between semantic constructs and symbols. This means we should avoid the following anomalies.

- **Symbol redundancy** occurs when multiple graphical symbols are used to represent the same semantic construct.
- **Symbol overload** occurs when multiple different constructs are represented by the same graphical symbol.
- **Symbol excess** occurs when graphical symbols do not correspond to any semantic construct.
- **Symbol deficit** occurs when there are semantic constructs that are not represented by any graphical symbol.

We apply this principle by identifying the semantic constructs of the information we wish to transfer to the patient. This is done in the previous section. *For each of the semantic constructs we aim to define a unique graphical symbol*. As discussed later, we have to compromise and introduce some symbol overload and symbol deficit.

7.5.2.3.3 Semantic Transparency

The principle of semantic transparency dictates that symbols' visual appearance should suggest their meaning. Ideally, we use **semantically immediate** symbols, whose meaning can be deduced by novice readers based on appearance alone. This is very important since remembering a symbol's meaning can

be very difficult for patients whose memory is not functioning well. Using icons that match the semantic concept's visual appearance is the best way to achieve complete transparency. The major part of patient decisions are made based on the interpretation of Choosable and Subchoice tokens. *It fits to make these semantically transparent by representing each with an icon*.

Semantic immediacy is not always a realistic goal, especially for more abstract semantics. Transparency can still be increased by hinting at underlying semantics, which makes it easy to memorize them. M. Tichelaar describes multiple ways that concepts can be transparently represented in pictures (Tigchelaar, 2013). Concepts can be transparently represented by an image of something (commonly) associated with it, such as a search action by a magnifying glass. Another option is representation by a person, such as Albert Einstein representing science. Picturing an (overdone) characteristic or something that belongs together with the concept is also an effective possibility. An example of this is picturing a grandfather as a hunchbacked man with a walking stick. Finally it is also effective to picture something with a similar sound, such as rain representing pain. These substitutions should not be used. Although great mnemonics, they are very confusing for first time users who are likely to misinterpret the symbol literally. *If a Choosable cannot be literally pictured by an icon, we picture an association, representational person, property or something that belongs to it*.

A worthwhile consideration is the use of a door shape to represent Options. Although this is a semantically transparent symbol through association, a square shape is more convenient as it makes optimal use of the screen space. This is vital for patients who cannot use gaze tracking precisely.

Another way to visually hint at semantics is to visually represent its properties. From the semantic structure in Figure 7 and Figure 8 we extracted applicable properties of Choosables, which have been listed in the previous section. These properties are represented in the UI as follows.

- **Visual appearance** is represented by icons as argued.
- The **Order** of, for example, frequencies of pain occurrence is represented by ordinal visual variables.
- Relative **Spatial Position** of Choosables, such as body parts, is preserved in the UI when possible.
- **Shape**, such as of countries, is preserved when possible.

Another tool to help us create semantic transparency are semantically transparent relationships (see Figure 9). These have been found to have an intuitive meaning independent of context. *We use sequence notation as the representation for Subchoice sequence relations and subclass notation to symbolize the Choosable Parent – Child relations*. It is not desirable to encode the Message/Action – Subchoice relation using subclass notation as well. As the Subchoice – Choosable relation needs to be visualized at the same time. In many cases, this requires an inconvenient amount of vertical space.

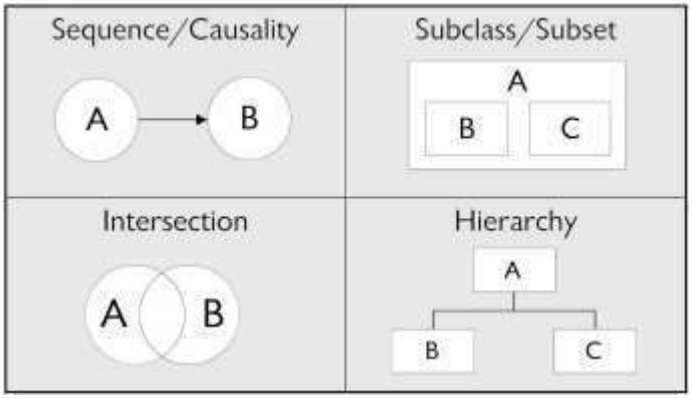


Figure 9. Semantically transparent relations

7.5.2.3.4 Complexity Management

The principle of complexity management prescribes to abide by the **perceptual limit** to prevent **cognitive overload**. The perceptual limit is the maximum amount of information the human mind can effectively process at a time. Visual notations must provide mechanisms for modularization and hierarchical structuring. *Hierarchical structuring is incorporated in the design of the UI by means of Categories. Modularization is applied by separating different functionality in visually separate modules. These different functionalities include speaking, entertainment, using the internet, messaging, etc. The concrete modular and categorized structure of our UI is given by Figure 7 and Figure 8 in the previous section.*

7.5.2.3.5 Cognitive Integration

Complexity management is closely associated with cognitive integration. When information is spread out across multiple modules or levels, the context of the information in focus should always be clear. There are multiple ways to integrate symbols into the bigger not displayed structure they are part of.

A top level summary should be provided. This gives the user a **cognitive map** of the system, which can serve to contextualize the currently focused symbols while navigating. *We use the Modules and top level Categories as the cognitive map. This is effective if all descendent Choosables are clearly represented by their top-level ancestor.*

To help the user cognitively integrate the current symbols in the provided cognitive map, the context should be clarified in multiple ways. People navigate, whether in a city or a UI, following four stages:

- **Orientation**: Where am I?
- **Route choice**: Where can I go?
- **Route monitoring**: Am I on the right path?
- **Destination recognition**: Am I there yet?

A good UI should support all of these contextual elements. The patient should be shown the context of his position in the hierarchy. The *Category – Choosable relation, Message/Action – Subchoice relation, and Subchoice – Choosable relation of the Choosables in focus are always displayed. The Module – Choosable relation is only displayed at the root level of a module due to screen space constraints*. Furthermore it should be clear what all available interaction possibilities are and how they affect the patient's position in the hierarchy. We achieve this in multiple ways. First of all, *Options are clearly distinguishable from constructs that cannot be chosen.* Secondly, *Navigation Options are clearly separated from other Options*. Thirdly, *we*

introduce the property Virtual to the Option symbol. The helps to clarify changes in level and to visualize endpoints.

Virtual Choosables are high level Choosables that still need details filled in. Categories and the Messages and Actions that have Subchoices are virtual. Choosing them is always followed by an additional choice between child Choosables to further specify what exactly is meant. This is equivalent to navigating down in the hierarchy. **Non-virtual** Choosables on the other hand provide fully defined input, with no more details to fill in. These are Messages and Actions without Subchoices and Modules. For example, if the patient specifies he is wants to see family, this is a virtual Choosable, as it is logically followed by a non-virtual Choosable of which family member he wishes to see. If the patient specifies he feels pain, he selects a virtual Choosable, since further details of the location, kind, etc. need to be specified to define the pain. The patient first specifies that the kind of pain is a stabbing pain. This Choosable is non virtual, as no further details need be provided to define the kind of pain. Mind that this Choosable is followed by another choice, as the parent non-virtual Choosable is not fully defined yet.

This gives rise to the alternative variation in behavior that could be encoded instead of the virtual/non-virtual encoding. We could explicitly encode whether a Choosable is the **final** Choosable in a sequence or an intermediate Choosable. This encoding does not map nicely to the semantics though, because whether a Choosable contained in a Subchoice is the final one is based on the arbitrary order in which the Subchoices are made. Also, the user can see whether a Choosable is final using the virtual/non-virtual property. The final and non-virtual properties correspond, except for some subchoices. Non-virtual Subchoices are only final when they are not followed by another Subchoice. This means that if we clearly symbolize the Subchoice sequence relation, the patient can see if a Choosable is a final one.

The question rises whether we could show a complete representation of the hierarchical position. Displaying only the parent of a Choosable gives an incomplete picture if there are multiple ancestors. However, displaying all ancestors requires a lot of screen space and increases graphic complexity. We compromise by only textually representing the ancestry of the focused Category. *Focused Categories have a label describing not just themselves, but their whole hierarchical ancestry.*

7.5.2.3.6 Visual Expressiveness

The visual expressiveness of an interface is the number of visual variables carrying information. The maximum visual expressiveness is therefore 8 (using all visual variables) and is called **visual saturation**. This is desirable. "Using a range of visual variables results in a perceptually enriched representation that exploits multiple visual communication channels and maximizes computational offloading" (Moody, 2009). Visual expressiveness differs from visual discriminability in that it looks at visual variation across the entire visual vocabulary, as opposed to pairwise visual variation between symbols.

We achieve visual saturation by assigning semantics to all unused variables remaining after our initial mapping of variables and constructs. This results in redundant coding, a desirable property following the perceptual discriminability principle. Particularly this applies to the choice of icons. *Icons are chosen such that they make use of a broad range of visual variables*.

Color is a very important visual variable as it recognized the fastest of all variables. It is thus desirable to *use color to encode important semantic constructs*. We determined earlier that it is not safe to rely on color, due to possible color blindness of the population. In light of this new information, we should have a closer look at the limitation of color. This visual variable can still be used for the group of colorblind people.

There are different forms of color blindness. First of all, **monochromia** or **achromatopsia**, conditions of not being able to see any color, are extremely rare. The most prevalent forms of color blindness are **protanopia** and **deuteranopia**. They account for about 8% of the population. People with these forms have difficulty discriminating colors in the red-orange-yellow-green region of the spectrum, which all appear yellow to them. A third form of color blindness, **tritanopia**, prevents people from seeing colors in the blue to purple end of the spectrum. It is very rare with less than 1% of the people being effected, thus we do not take it into account. *This allows us to safely use the color visual variable along the yellow-blue axis to encode information. Instead of yellow we could also use reddish or greenish colors. We can even use them together, as long as they have significantly different levels in saturation.*

Visual variables and semantic constructs cannot be arbitrarily mapped. It is important to take into account the **level of measurement** of the variable (whether it is an ordinal, nominal or interval type) and the **capacity** of the variable (the number of perceptible steps). If the variations of a semantic construct outnumber a variable's capacity or level of measurement of a semantic construct is greater than the variable's, it cannot be (fully) encoded by this variable. *Visual variables and semantics are mapped based on their capacity and level of measurement.*

7.5.2.3.7 Dual Encoding

It has been stated earlier that text labels have a visual distance of zero and do not help with cognitive offloading. This does not mean they are not useful. Just as redundant coding with visual variables helps to achieve greater discriminability, the use of text should be used to redundantly encode meaning. Neither text nor visually transparent symbols are likely to achieve 100% visual transparency for every user. Using them together improves the chance of correct interpretation.

We use dual encoding by adding text labels to every Choosable. Furthermore, there is a referentially transparent way to textually encode the virtual property of Choosables. We add an ellipsis (…) as suffix to the text labels of every virtual property.

Subchoices also benefit from dual encoding, as the meaning of their tokens are very important as well. However, in most cases there is insufficient space to display text for all Subchoices. We compromise by showing only a text label for the currently focused Subchoice.

7.5.2.3.8 Graphic Economy

It is important to limit the number of different symbols (**graphic complexity**). This is especially important when the users are novices or if the symbols are not fully transparent. Research has shown that humans are limited to reliably differentiate between 6 semantic constructs for single visual variable. Graphic economy differs from complexity management in that it is concerned with the number of different symbols rather than the number of tokens.

Graphic complexity can be reduced in three ways. Firstly, there is the possibility to reduce or partition the semantic complexity. As a result, the user has to differentiate between fewer entities at a time, thus reducing the number of symbols needed. Our software is partitioned into modules, but the semantic constructs we identified are the same in these modules. An alternative modularization that does partition the semantics does not seem possible. Reducing the number of semantic constructs is a more realistic option. T*he Choosable property favorite can be removed for cognitively less capable users.* These users do not make intensive use of the UI and therefore do not profit much from the Option favoritism feature.

Secondly, we can reduce the graphic complexity by increasing the visual expressiveness. We already plan on visually saturating the interface. It is thus save to have more than six different types of symbols. However, to be on the save side with cognitively challenged patients we should still aim to reduce our symbol set further.

Finally, there is the possibility to introduce symbol deficit. There are a number of semantic constructs whose meaning can be easily made clear through the visual grammar, without explicit representation. Also we can aggregate symbols that are semantically similar. The rest of this paragraph lists these cases.

We do not visually represent the semantic construct Page. The existence of multiple Pages is made clear from the available Navigation Command symbols. *If the patient can navigate to the next Page, a Next Navigation Option is displayed. If he can navigate to the previous Page, a Previous Navigation Option is displayed*. Having more than three Pages in undesirable due to the inefficiency of the navigation. This means the user can always infer which Page is focused from the available navigation options.

As argued in the paragraph on semantic transparency, the Message/Action – Subchoice relation can't be semantically transparently displayed using subset notation. *In the absence of an alternative compact transparent symbol, we choose to hide this relation completely*. The relation should instead be inferred from the visual grammar, as *we exclusively place Subchoices always next to their parent Choosable*.

The Chooses relationship is subject to symbol deficit as well. We believe this relationship is clear from the visual grammar: *The Choosable chosen by an Option is placed inside it*.

The types of Module, Category, Message and Action have semantic differences that are not very relevant for the user. Their common behavior, being chosen through an Option is, is the most apparent. Therefore it makes sense to introduce the Choosable parent type and only visually represent this construct. The only significant difference in behavior among the Choosables that should be represented is whether they are virtual or not, as explained in the section on cognitive integration. *We aggregate the symbols for Module, Category, Message and Action into 2 symbols, one for virtual Choosable and 1 for non-virtual Choosable.*

Similarly to how we aggregated the different Choosable subtypes, we can aggregate the relations that connect these different types. For this reason we introduced the Parent – Child relationship. There is one symbol for all its subtypes (except the Message/Action – Subchoice relation).

7.5.2.3.9 Cognitive Fit

Cognitive fit theory states that different representations of information are suitable for different tasks and different audiences. There are especially big differences between an expert and a novice audience. Novices have more difficulty discriminating between symbols, are more affected by complexity and have to consciously remember what symbols mean. Simplifying the interface for novices has an adverse effect on experts, due to the **expertise reversal effect**.

Using CommIC requires roughly only one task, making a selection from the available Choosables, but the expertise of the patients using it varies. Patients that have more experience using CommIC or have better cognitive capacity should not be hindered by an oversimplified UI. Experts especially are hindered by low complexity. We already designed *flexible complexity management in the form of dynamically shown/hidden Categories* to deal with this problem.

This feature can result in the contents of subcategories being displayed together with direct children of the focused category. In many cases no reference to an expanded subcategory is necessary. This is the case for categories whose child Choosables have clear meanings without the context of their parent category, such as "respond" and "watch". We call these **Non-defining categories**. On the other hand, there exist categories that present a defining context for their children, such as "Tel me" and "My videos". They are called **Defining categories**. We introduced the defining property of Category, as it is desirable to display them differently. Display of expanded non-defining categories adds no semantics, but wastes screen space and increases graphic complexity. They are hidden. Defining categories should always be shown.

To further improve the cognitive fit *the Options for the more advanced or less likely chosen Choosables are hidden altogether for novice users*.

It is not a good idea to vary perceptual discriminability or the graphic economy based on the expertise of the patient, as many expert users start using CommIC as novices. Suddenly changing the look of the interface when they are experienced enough is confusing.

7.5.2.4 Visual syntax

Table 1 displays the mapping of semantic constructs and choice instance semantics to symbols. The visual syntax has been constructed iteratively in conjunction with our analysis of the visual design principles and identification of the semantic constructs. Within each iteration we took several steps. First we made sure each semantic construct is maximally semantically transparent. Then we assigned at least one unique value to a visual variables for each construct. These values are in italics in the table above. As noted in the section on perceptual discriminability, a unique texture only is insufficient. In the next step we checked and improved the pairwise visual distance between constructs taking the rule of visual – semantic congruence into account. Finally, made the syntax visually saturated. Each visual variable has been applied to its full capacity when possible.

One exception is the position of child Option symbols. They can be placed anywhere under the focused category. There is the transparent possibility to use the order to encode Choosable favoritism. Options for frequently used Choosables appear left and those infrequently used appear on Pages to the right. However, since favorites change during usage, this makes the position of all Options dynamic. This is likely to cause confusion among users and increased lookup time while using Options whose earlier position is remembered. Instead we introduce the grammar rule that *child Options are statically positioned in order according to expected frequency of use*. This is not accurate for all patients, but it can be fine-tuned by user tests to achieve an on average efficient navigation.

Another restriction to the position of child Options is their selectability. We expect that future versions of the product require other forms of selection specifying a screen position. This is the case for selection by means of hardware buttons. A commonly used technique in AAC software is **scanning**. The software automatically selects subsets of Options in a predictable sequence. When the desirable subset is selected the user presses his button. The process is repeated for the subsets of the selected subset to zoom in until a single Option is selected. To support this predictable subset selection mechanism, *Options are positioned in a grid*, such that rows, columns and blocks can form such subsets.

Symbol			Position	Size	Shape	Color (blue-yellow axis only)	Brightness	Orientation	Texture (border only)
Option	Default		*	normal	*rectangle*	*uniquely colored background*	*bright*	horizontal	solid
	Selected			*bigger*	*	*		*	thick border
	Navig	Cancel	*top left*	*	*hexagon*	red			solid border
		Next / Previous	top left / bottom right child		*rectangle with arrow head*	*(light) grey*		to left / right	
	Favorite		*		*	*		*	*double border*
	Virtual				*rounded rectangle*				*dashed border*
	Similar / Opposite		grouped with similar/opposite Choosable children		*	*identical for similar, contrast for opposite Choosable children*			*
	Order		matching child Choosable order		*shared pointy border with neighbors matching Choosable order*	*gradient background matching child Choosable order*		in direction of order	
	Position		matching child Choosable position		*	*		matching child Choosable orientation	
	Shape		*		matching child Choosable 2D shape				
	Disabled			smaller	*crossed out rectangle*		*Dark*	*	
Gaze pointer			*moving with gaze*	smaller	*radially fading point*	*red (black)*	avg./ dark	horizontal	none
Choosable			*inside Option / in header of subclass notation*		diversely shaped icons	diversely colored icons	avg.		none
Sub-choice	Default		*next to parent Choosable*		*round*, icon inside	*grey*			*dotted border*
	Focused				*oval*, icon inside				thick border
Choosable parent – child relation			*filling screen except top row / expanded Defining Category: within focused category*		*subclass notation*	parent background color in background			solid border
Subchoice seq-uence relation			*	1D	*sequence notation*	none	black		none

*Table 1. Mapping of semantic constructs to symbols. The *-sign means a visual variable is undefined by the symbol and can thus vary or (position) can be determined by grammar. Italics mark unique values for a visual variable.*

The size variable has been defined relative to the 'normal' size. To achieve cognitive fit, everything is scalable matching patient accuracy and ability to process information. A normal size is defined to be the minimal area needed for reliable selection with using the eye tracker. The size visual variable has limited capacity as to not waste valuable screen space and to ensure visibility for people with bad eye sight.

We considered using a shape of a door to represent Options, as doors commonly have an association with options. However, using a complicated shape for the many choice tokens that displayed would congest the interface. Furthermore, as the user is constantly choosing, the symbol for choice is remembered quickly.

The constructs of Option and Choosable have proven to be very difficult to be visually separated. In earlier iterations we did not even identify them as different constructs. This is problematic though, since Choosables can appear outside the context of an Option. Most importantly, this is the case for the focused category that appears above its children. We found earlier that it is very important to separate Options from non-Options, thus encoding this difference is necessary. We have not been able to visually design a compact completely separate visual representation for both constructs. Therefore, we are forced to make a compromise. We represent Choosable properties not through the Choosable symbol, but as part of the Option symbol. This means these properties are not represented at all when Choosables appear outside of an Option context. This is not a big problem, as this only happens after the Choosable has been chosen and the properties that serve to aid in the interpretation of the Choosable have done their job.

As the subclass notation only makes sense when the parent is displayed, this notation is only indicates the relation between a Defining Category and its child Choosables. Category – Choosable for expanded non-defining Categories should nevertheless be displayed, as the similarity of the children is valuable for providing structure in large Choosable sets. We repeat the solution for the Choosable properties, by introducing the Similar property to the Option symbol. A set of Similar Options indicates they are children of the same Category. Using both the Similar Option symbol and the Choosable parent- child relation amounts to symbol redundancy. This is a necessary compromise.

The hexagon shape has been chosen for the Cancel Option, to make the button resemble a traffic stop sign. Some countries adopt a different shape for the stop sign, namely a rectangle pointing down with a circle touching its corners. This can serve as adequate alternative in our localized interface.

The symbols and grammar rules combine to form the GUI depicted in Appendix C.

7.5.2.5 Class design

The visual syntax has to be supported in code. Next we discuss how our current design needs to be modified to support this UI.

The different functionalities harbored in the identified semantic constructs should mostly be implemented in their own model class. These classes should expose all properties that are represented by symbols. For example, the `Subchoice` class implements the `IsFocussed` property. Additionally they are linked through databinding to their representing symbols, implemented as UI components in the view layer.

The Option and Choosable construct form exceptions. Option has only visual behavior and therefore does not need to be implemented on the model layer. On the other hand, the Option symbol visualizes nearly all the visual properties of the Choosable it represents. The actual Choosable symbol is only an icon. Clearly implementing a separate UI component for Choosable needlessly complicates the UI design. Instead we implement one UI component for both for both symbols. This component is data bound to a Choosable to represent its properties and its icon.

WPF makes it very easy to dynamically generate UI components for data items. This is ideal for our implementation of dynamic paginated categories, where Options could be generated for each Choosable. There exists a problem though. Navigation Options are used interchangeably with Choosable Options as Page content since Pages can contain the Next/Back Navigation Option symbols. These

cannot be generated from Choosables. We add to this the observation that there is hardly any difference between our implementation of the types of Navigation Options and of an Action. The main difference that an Action operates within the model layer and an Option in the view layer. Recalling the simplicity guideline of agile design we judge it best to unify the two. We use `ActionChoosables` to represent Navigation Options within our model layer Choosable data structure. This approach has the disadvantage that the ActionChoosables carries logic to manipulate the viewmodel layer. This results in tighter coupling between these two layers, but we believe the increased simplicity of design outweighs this.

The `Choosable` composite structure has been expanded with the newly identified Choosable types. `CategoryChoosable` has been substituted for an abstract `CompositeChoosable` class and its children `Category`, `Module` and `SubChoice`. Recall that subchoices were implemented as `Categories` earlier, as explained in subsection 7.2.2.2. Since the needed abstraction is introduced anyway, we now make `Subchoice` into a separate class. The differences between `SubChoice` and `Category` currently only exist in the UI, but it is likely that there will be future differences in behavior. The introduction of `CompositeChoosable` makes the design open to similar changes (adding more composite Choosable types) in the future.

Now that we uniformly call the model logic from the different Options, we no longer need separate Commands in the Viewmodel layer for each type. For Messages, Actions and Navigation Choosables alike, the Options simply call `MakeChoice` on their Choosable. This is not the case yet for the different kinds of `CompositeChoosable`. Each Choosable we added required us the change the `OnChoosableChosen` Command in the viewmodel, checking explicitly for the subtype to call the appropriate logic. Or assign a different viewmodel Command to different Option types. Both methods require knowledge of the specific Option type and violate OCP for adding new types. A solution is to move the viewmodel manipulation logic into the `Choosable` subtypes. This allows us to uniformly delegate the handling of choosing an Option via `OnChoosableChosen` to `Choosable.MakeChoice.` In fact, the mediation of the viewmodel layer can be skipped. We let `Choosable` implement the `ICommand` interface so `Options` can call `Choosables` directly.

We already moved navigation to the model layer, creating a dependency of the by Choosables on the viewmodel layer. This solution makes that coupling tighter. We believe this is not actually a problem, the boundary between the model and viewmodel layer is ours to define. Observing the tight coupling of Choosables with our viewmodel layer and the direct dependency of view components on Choosables, we realize that Choosables naturally belong in the viewmodel layer. We move them there.

The logic we moved from `OnChoosableChosen` is now spread out on different levels of the hierarchy. Upon calling `MessageChoosable.Execute` we play an audio message and call the overridden `LeafChoosable.Execute`, here we set the current Module as the currently selected Choosable, so a new Choosable can be chosen. We also call `AbstractChoosable.Execute` where we implement logic that should happen on any Choosable choice, such as logging the Choice for debug purposes. The structure is also known as the Call Super Anti-pattern. It is a bad design, since being dependent on subtypes calling overridable methods is prone error when this is forgotten. We improve this design using the template method pattern. `LeafChoosable` and `CompositeChoosable` children no longer override Execute, but move this logic to the `OnChoose` hook method. When a Choosable is chosen. `OnChoose` is called by `LeafChoosable`'s and `CompositeChoosable`'s Execute method. These Execute method still require to call

`AbstractChoosable.Execute`, but since this layer in the hierarchy is not likely to be extended it is better to not add the complexity of another template method.

The Option properties Similar, Order and Position cannot be implemented trivially like the other visual properties. They share the problem describing or being dependent upon the relation with other Options in the same `CompositeChoosable`.

As we have seen, Similar is very similar to the implicit parent-child Choosable relation. We could implement it not as a property, but a `CompositeChoosable` subtype named `SimilarityGroup`. However, the only behavior that seems to be dependent on similarity is the collapse behavior, namely a `SimilarityGroup` should never collapse. Rather than adding a new type we add the `Similarity` property to `Category` and add some logic to Category's implementation of `CollapseHierarchy`. For the Order property we can use the same solution. We implement the property as the `IsOrdered` property in `Category`.

Options with the Position property should know a way to position the Choosable accurately relative to other Choosables of the same parent independent of the size. Their common reference is their parent. It seems sensible to store position as normalized coordinates relative to the parent. So (0,0) is defined as the upper left corner of the area taken up by the parent and (1,1) as the bottom right.

Figure 10 shows the improved design of Choosables.

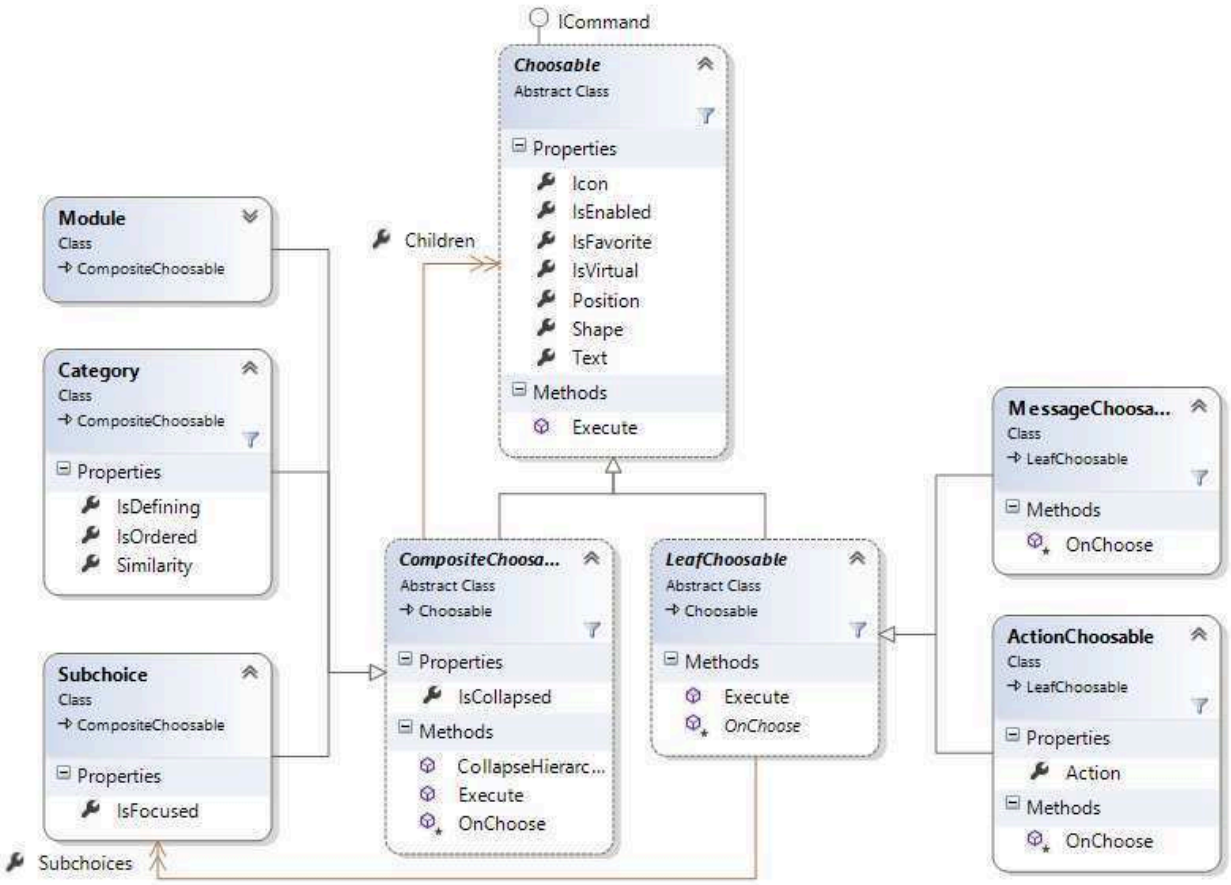


Figure 10: Improved design of Choosables

7.5.3 Process evaluation

The implementation of the UI design has proven to be an extremely time-consuming task. This is due to our lack of experience with GUI implementation in general and the steep learning curve of WPF. For this reason this sprint does not finalize the implementation of the GUI design in order to receive earlier feedback. We split the user story "Controls should be made implementing the look and functionality of the basic semantic constructs." into separate user stories for each semantic construct. The following ones have not yet been implemented: Subchoice, Subchoice sequence relation and Choosable Parent-Child relation for collapsed Defining Categories. Option has been partially implemented, still lacking the properties Selected, Disabled, Order and Similar/Opposite. These will be addressed in future sprints.

During this sprint it has become clear that the implementation of CommIC will be done almost exclusively by the researcher. The other members of the original development team have very little time to contribute. Consequently, the discussed teamwork aspects of the agile approach cannot be used for this project, or are used to a limited extend.

In particular, Scrum meetings are held much less frequently. This is not enough to keep other minor contributors involved to the extent where they can effectively contribute on their own. Instead, J. Benistant occasionally contributes to small independent parts of CommIC with low complexity, such as the logging functionality. Another approach we use to allow occasional contributions by minor developers is pair programming. The rapid feedback from pair programming allows minor developers to quickly become updated about the particular component they are working on and prevents misinterpretation of requirements.

Until this point, test driven development is not adopted. The majority of code describes interface look and behavior rather than independent logic with verifiable results. Another significant part of the coding effort deals with handling gaze server input. It is not possible to write tests for this logic without building complex mock objects imitating the eye tracker server. The implementation cost would be inappropriate for the benefit of the tests. To test the UI and input we thus must rely on white box testing. During the demonstration of the software we found bugs, previously undiscovered. This should not happen again in the future. Starting next sprint we will extend our development process with systematic black box testing.

An unfortunate result of lacking automatic tests is that the heavy use of refactoring causes a lot of overhead. Frequently, bugs are introduced by refactoring. Without automatic tests these are often not immediately discovered and difficult to trace. A great deal of development time is thus lost tracing bugs and also to frequent black box testing to discover them.

7.5.4 Feedback

At the end of this sprint CommIC is not yet in a state where it is testable with real patients. Its functionality has been demonstrated to nurses and medical specialists and they have been observed and questioned while using CommIC. We have gathered the following feedback from them.

- Choosable similarity might be more clearly depicted by identical background color than by identical Option color. We implement the similarity property thusly.
- The gaze pointer indicating where you look on the screen, makes the UI too busy. We can solve this by replacing the gaze pointer by a filling pie shape overlay over an Option that is being selected.

This way current selection progress is still made clear and as long as the user keeps looking at the same option the interface does not abruptly change with each small eye movement.

- The Cancel button should be replaced with a back button, as patients are likely to make wrong choices.
- While white box texting and demonstrating the current version, it becomes clear that CommIC malfunctions too often, mostly due to the eye tracker not working correctly. The problem is not one we can solve at the source, since it is a firmware problem of the eye tracker. In order to achieve the reliability that our stakeholders want, we will report that the problem occurred and then automatically restart the whole system.
- CommIC should anonymously log user input. We need to gather information on system usage in order to detect problems, prioritize development and improve interface design. This new user story expresses our own needs as developers. Until now we did not specifically inquire about the needs from the developer stakeholder. The agile methodology does not require this, since like other requirements it is not realistic to predict them and they will come up automatically during development. Another developer user story that has come up is be wish to remotely monitor the system for problems and possibly fix them. We add a new encompassing stakeholder goal to CommIC: "Developers want to remotely monitor and remedy problems and collect usage data."
- In our analyses of a cognitively low-demand GUI design, we restricted ourselves to visual elements only. We could further improve the interface by providing information using other senses as well, as this is another form of redundant coding. We should add auditory feedback to user input. This takes the form of speaking the name of an option while the user selects it. This should be clearly separated from the audio message that the patient wishes to convey. This is accomplished by having audio feedback spoken more softly and rapidly.
- The Favorite symbol is not semantically transparent and causes confusion. We should show the Favorite symbols as bigger instead of using double borders. This is the only transparent improvement we and the stakeholders can think of. Unfortunately, this visual state has already been chosen to represent focus. However, focused Choosable will become bigger only when focused. This behavior clearly separates them from statically bigger Favorite Options and which mitigates the symbol overload. We thus believe the proposed change will improve overall clarity.

7.6 SPRINT 5

7.6.1 User stories

This sprint focusses mostly on the feedback from the last sprint. At the end of the sprint all basic functionality should be usable by medical personal. If this is the case, CommIC will be deployed to the ICU for real-world testing.

- The Gaze pointer should be replaced by a filling pie shape overlay over an Option that is being selected.
- The Cancel button should be replaced with a back button.
- If an error occurs within the eye tracker firmware, the system should report the problem and automatically restart.
- The system should anonymously log user input.
- Audio feedback should be softly and rapidly played while the user selects an Option.
- The system publishes its debug log for remote access.

- The system can be remotely updated.

Additionally, we partially implement the requirement stating that the system should be controllable by finger/toe/hand/foot/arm/leg/facial muscles. This user story is not implementable and should be split up into smaller, less abstract user stories that are. In this sprint we will ensure CommIC is usable by means of touchscreen input only, since this is easily implementable. Fully implementing this user story will also require input via a hardware switch.

- The system can be controlled by the touchscreen.

7.6.2 Design

The user stories in this sprint can be implemented fairly trivially and there are hardly any noteworthy design decisions. The only design change is the separation of `MessageChoosable` and audio output functionality, which was extracted to its own single responsibility class `AudioOutput`. Now audio output functionality is used by multiple clients to implement audio feedback and involves logic (audio feedback should not override Messages) and customization (loudness / pronunciation speed). This functionality could have been recognized as a separate responsibility earlier, but it was a lot more limited and not worth the increase in design complexity.

7.6.3 Black box tests

We have made a black box test plan for systematic test coverage of important or complex functionality of the system. Application of these systematic tests has brought numerous bugs to light, such as the eye tracker failing to connect when CommIC starts too quickly after boot and the pie chart Option overlay filling too quickly when reaction time is configured as high. The complete test plan, including test cases added in future sprints, can be found in Appendix D.

7.6.4 Process evaluation

Systematic black box testing has had significant improvements on software quality. It has let us to find problems that could otherwise have been overlooked. More importantly it has allowed us to find problems before demonstrating the end product of the sprint to our stakeholders. This is of vital importance for a professional software company. Furthermore, using our systematic test plan we have to spend less time black box testing than in earlier sprints and have found bugs earlier, which often made them easier to trace. We conclude that systematic block box testing is a valuable addition to our development methodology.

7.6.5 Feedback

The end product of the sprint was demonstrated to the specialists and nurses working on the ICU and the reception was positive. With our main contact we decided to move towards a release of CommIC. Over a two week period we gave demonstrations of the system to reach most of the medical personal who get to work with CommIC. Additionally the first test with a real patient and medical personal have been done.

We want to prevent any unclarities among personal on how to use CommIC in our absence. To this end, a user manual is written for medical personal and family with step by step instructions and pictures to operate the system. It is designed to be as easy to understand as possible and make little assumptions about the knowledge of the user. The result is a manual that covers many pages, despite the simplicity of CommIC. During the trials we let medical personal use CommIC with patients with the help of only this manual. Due to its size medical personal was reluctant to use it and in some cases used the system

incorrectly. We conclude that a less detailed and less graphic single page manual is more useful. This improved manual found in appendix F.

During the demonstrations various points of feedback came to light, most of this feedback was easy to process into software improvement. Continuous improvements have been made between the different presentations. Basically, these two weeks serve as rapid development iterations. Rapid feedback on small improvements have enabled us to fine-tune CommIC for optimal usability. Below we list the received feedback and system improvements in response.

- Numerous changes to the available Choosables have been suggested and implemented.
    - The Message "Tell me about my death" was too prominently visible in the interface and might negatively confront users. It is disabled.
    - The Message "I want to move my legs up or down" is added.
    - The Message "I want to sit in the chair" is added
    - The Message "In want to pee/poop" is added.
- The selection pie draws too much attention. While selecting people follow the pie with their eyes rather than look at the centre of the Option. This accidently can cause the gaze to be measured as outside of the Option, terminating selection. We make the Option much smaller and more transparent. However, this causes confusion. Users are less aware of the selection progress and users repeatedly mistake audio feedback upon Option selection for having selected the Option. We have updated the selection pie to a size and transparency that is a compromise between drawing too much or not enough attention.
- The audio feedback upon selection makes the interface busy. It is disabled by default and should only be used for illiterate patients.
- Messages consist of the one by one pronunciation of the names of the defining Categories, such as "I want" "To sleep". In some cases this does not product grammatically correct sentences, such as "Tell about" "My pets" "A Dog". This might be confusing for patients. Two Choosable properties have been added. If `Category.IsSilent` is set to true, its pronunciation is skipped. If `Choosable.ContextualText` is non empty, this property is pronounced instead of the Choosable name. This way we can couple grammatically correct sentences to Messages, without defining extra properties on the majority of Messages. Setting the "Tell about" category to silent and defining the "My pets" `ContextualText` property to "I have", we now form the correct sentence "I have a dog".
- It is not always clear when CommIC should be calibrated. In almost all cases the patient or system has moved since the last time CommIC started or a different patient is using it. In these cases a calibration is desirable. We now reset the calibration each time the system starts, whereas previously the eye tracker sometimes would remember the last calibration. This makes the procedure to start using gaze tracking uniform every time CommIC starts; always calibrate. No knowledge of the calibration state is required, nor a decision on whether or not it is a good calibration.
  Furthermore, in the event that a calibration attempt is unsuccessful, CommIC notifies the medical personal and instructs them to check the eye position and attempt recalibration.

The demonstrations have also produced feedback for more complex improvements.

- The possibility to spell words is in high demand, we make this into a high priority.

- Selecting an Option by gaze is often unsuccessful. Numerous test subjects among personal have a lot more difficulty then ourselves in achieving accurate calibration and selecting Options. Their gaze position is often very unstable, either by measurement errors or inability to focus on their gaze. As a result, selecting an option becomes very difficult. The moment the gaze is measured outside of a selecting Option, the selection is aborted.

This first patient trial is intended for orientation and first impressions. There is no need for a specific test plan. We intend to verify the basic working of CommIC, namely whether nurse and patient can complete the positioning and calibration of the system and whether the patient can successfully choose Options and navigate the menu.

We will refer to the patient in this trial as patient A. Patient A is lying on the ICU for a couple of days. His exact condition is unknown, but he appears to have the symptoms of locked-in syndrome. The patient is able to understand and respond to yes/no questions by means of eye blinks. After a short explanation of the functionality of the software system, the patient indicates that he understands and is willing to participate in the trial.

We position the eye tracker in front of the patient, but the eye tracker does not succeed in getting a very stable recognition of his eyes. As we run the calibration, the patient seems to be following the dot on the screen with his eyes. He also claims that he has no problem doing so. Nevertheless the process fails to yield a usable calibration. After a number of attempts, we attempt to have the patient use a calibration with our own eyes. This does not yield usable control. At this point patient A is too tired to continue. We adjust the calibration procedure to run slower and use calibration points that are bigger and more clearly visible. An hour later we retry calibrating with the patient, but results have not improved.

Multiple explanations are possible for the calibration failure. Firstly, it is possible that patient A was unable to concentrate for the required duration on the calibration points. This was not observable by looking at his eyes. If this is a factor it can only account for a small error. Secondly, it is possible that the patient was not able to focus his eyes on the picture of the calibration points. At the next trial this should be verified by asking the patient. Finally, we know that the detection to the patient A's pupils is unclear. We tried different angels to no avail. The only cause we can think of is the fact that the patient's eyes are slightly less open due to muscle weakness and exhaustion. In the coming trials we will instruct the patient to open his eyes further. Patients are often not capable of doing this or maintaining this effort for a long time though.

The feedback from the demos and the patient trial makes us conclude that the release of CommIC should be delayed for one more sprint.

7.7 SPRINT 6

7.7.1 User stories

This sprint addressed the yet unaddressed feedback from the previous sprint. Most importantly, we implement several new user stories that provide a solution to discovered calibration and option selection difficulties. The end product is the minimum viable product that is to be used on the ICU.

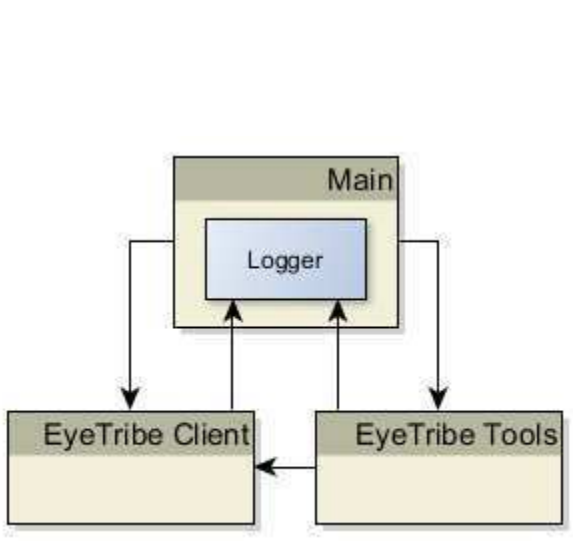
- Losing gaze detection for a short time should not cause an Option selection process to be halted.

- When the gaze leaves an Option, its selection progress should not be immediately reset, but gradually be reduced.
- Gaze should 'stick' to Options. The screen area that the user's gaze needs to exit before selection of an Option stops is bigger than the size of the Option.
- In order to gather more information on calibration in real world trials, the process should be logged in detail.
- The minimum accepted calibration quality should be lower.
- Sampling of individual calibration points should be automatically retried if their quality is poor.
- The speed of calibration should be configurable.
- Calibration works best when the eyes are close to the screen (40 cm rather than 70cm). The tracker box should only show a green background when the user is in close range.
- The system allows patients to form normal sentences.

7.7.2 Design

As we add logging functionality to the eye tracker client code and calibration code, the build of our software breaks, due to a cyclic dependency. The Logger class implements logging functionality and is part of the Main package. The Main package depends on the EyeTribe Client package and EyeTribe Tools package. Both of these packages now depend on Logger and thus on the Main module.

In (Martin, 2003) two methods for resolving cyclic package dependencies are described. Firstly, we can apply the Dependency Inversion Principle by moving the abstract interface of the logging functionality to the dependent package. This is not a solution, since Logger is a singleton and DIP works on interfaces, not object instances. The other solution proposed by Martin is to move the shared class dependencies to a separate package. This approach is effective and is illustrated below.



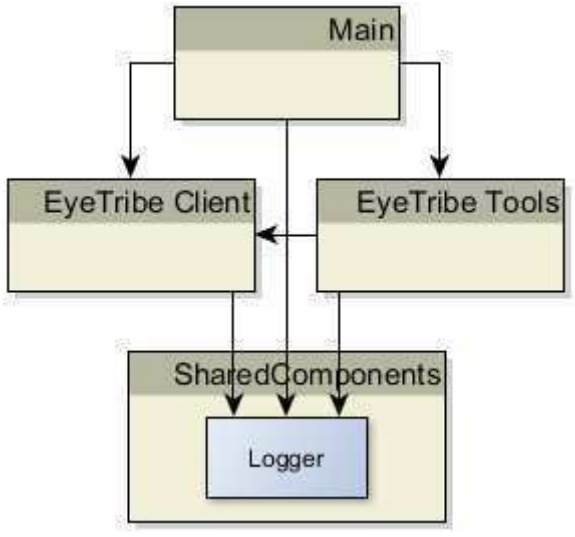


Figure 11 (left): Cyclic dependency, Figure 12 (right) the problem resolved

Until now, the gaze of the user is translated to a mouse pointer position in order to interact with the UI. Gazing at an option causes a `Mouseover` event. This easy solution is not ideal, since gaze and the mouse pointer behave differently. Particularly, the mouse pointer is always on screen, while the gaze is not. We have implemented workarounds for this incompatibility, such as moving the mouse pointer to the top

left corner when the gaze is off-screen. Additionally, these gaze events are difficult to separate from touch events, which also control the mouse pointer. This causes problems when gaze and touch control are used interchangeable and both require a different response.

To solve these problems we implement our own input device. WPF allows us to extend from a touch device class. This is convenient, since touch input, like gaze, can be off-screen. Our gaze input device listens to gaze events from the tracker and triggers touch events in the appropriate location. We can easily detect whether these events originate from our gaze input device, which allows us to separate between gaze events, touch events and mouse events.

The functionality to form normal sentences looks similar to common Message Options. For each letter a Choosable is created within the "Letters" Category. There are number of relevant differences though. Firstly, the choice for a letter is not a Message, but an Action. We can implement this Action with the `CommandChoosable`. The command performed is not independent of the Action that spawned it, since it should write the letter associated with the Action. This makes composition not an ideal approach to implement the Action. Rather, we extend `LeafChoosable` with a new child type `TypeChoosable`.

Secondly, the user will often type multiple characters in sequence, which means it would be convenient if after typing a character the "Letters" Category is selected rather than the root "Talk" Module. In fact, the "Letters" Category behaves like a Module rather than a Category and is implemented as such.

Thirdly, letters can be categorized flexibly. The alphabet can be partitioned in groups of arbitrary numbers of letters. We should make use of this in order to make navigation to any letter as fast as possible. This means using the minimum Category size while all letters categories can still be displayed on a single Page. We require a different approach to categorization, where categories are created dynamically, rather than statically. A Category factory could produce the category hierarchy for a Module. Abstract factories are unsuitable, since they product a limited set of predefined products. A factory method is also unsuitable. Such a method would need most of the information that is currently embodied by our static hierarchy to produce it. Since we have to still statically define this structural information, the input and output types of the factory would be mostly redundant. The same is true for an implementation of the builder pattern. It seems that rather than generating the Choosable hierarchy, a transformation of the statically defined hierarchy is in order. We rename the `Choosable.CollapseHierarchy` method to `TransformHierachy` and add the new `DynamicCategory` child type of `Category` overriding this method.

Finally, while the "Letters" Module is selected, an output area should be shown in the header bar. What is shown in the header bar seems dependent on the current module. Currently we are not aware of cases in which the current Module and header bar content are not mapped one to one. With agile's simple design principle in mind, we add a property to the Module describing the header bar content.

The complete Choosable class diagram is shown below.

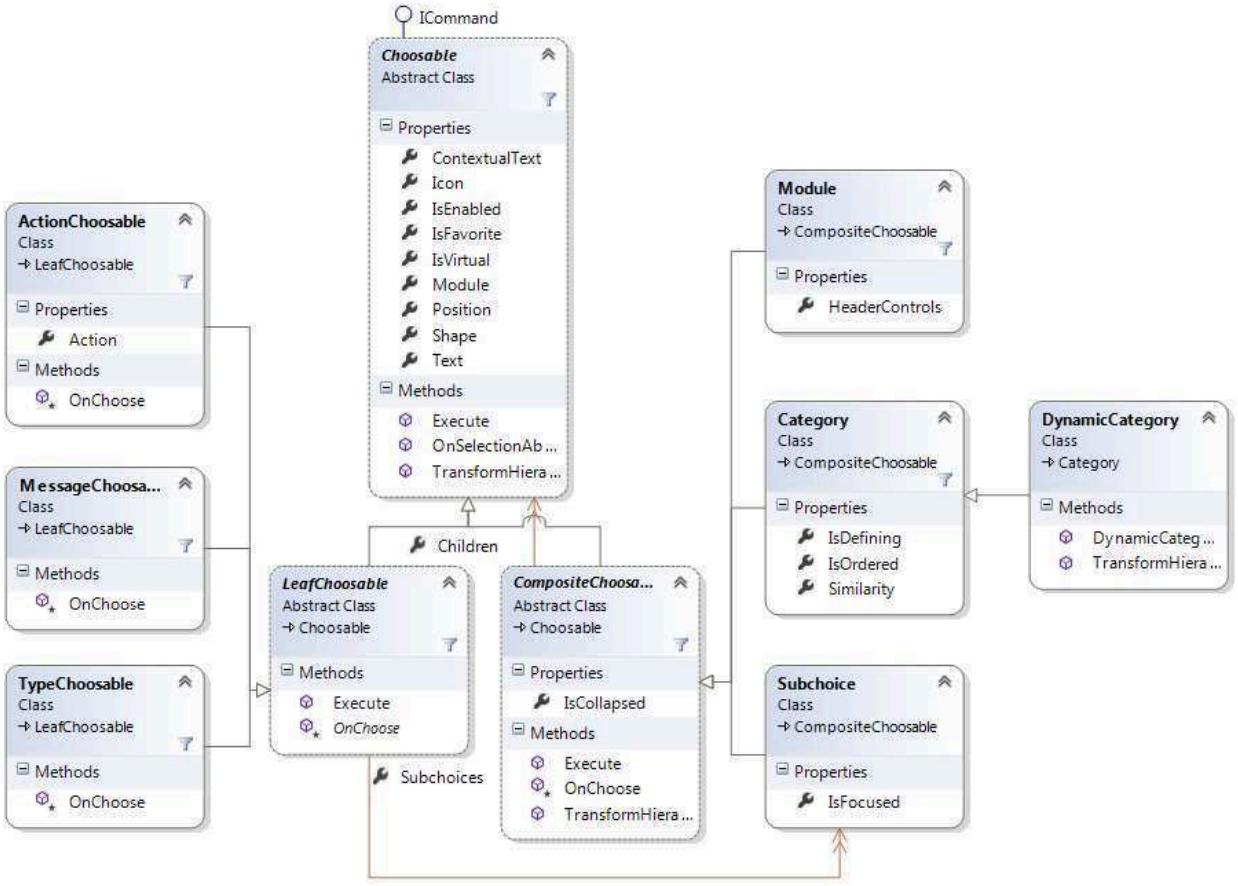


Figure 13: Choosable class diagram

7.7.3 Process evaluation

After the first trials with patient A, we realized we lacked relevant information about the patient to interpret our results. We do not collect information about patients, other than our own observations and the answers to the three usability questions we ask them. We do not collect this information for the sake of patient privacy and because it is difficult to collect this information. With future trials we should try to collect it nonetheless. Below is a list of patient characteristics we will inquire about during future trials.

- Whether the patient is literate.
- How much computer experience the patient has.
- Whether the patient is colorblind, nearsighted or farsighted and how severe.
- Whether the patient currently is wearing lenses.
- How well the patient can hear.

7.7.4 Feedback

At the end the sprint we have tested the software system with medical personal and again with patient A. The implemented improvements were accepted. Numerous nurses used CommIC and they were all able to select Options without trouble. The trials with the patient were also more successful. We confirmed that he could see the screen sharply after positioning the tablet in front of him, since blurry vision was a hypothesized cause of the calibration difficulties during the previous trial. The patient was able to complete the calibration process without trouble. We observed him familiarizing himself with the

software for about 5 minutes. He navigated the Choosable hierarchy without problem and appeared to decisively choose Options. We continued by asking him to communicate how he felt using the images in the Option menu. Patient A gave a relevant response. We instructed him to look at certain Options by pointing at them and he was able to do so. He was not able to reproduce these steps without our guidance. We also instructed him to type a word using the Letters Module. The patient typed many meaningless characters. We retried the trial a few days later with similar results.

We conclude that the patient did understand the basic working of the interface and how it responded to his gaze. He does however not appear to grasp the meaning of the responses. We are uncertain what the cause of that is. The most likely explanation is that the step from basic sensory interaction to real-world semantics is too complex for the patient in his current condition. This step may indeed be cognitively demanding, if the patient has limited experience using computers or is illiterate.

After concluding the patient trials, we and our contacts within the medical staff agree that CommIC can be released and used in practice. At the start of the research project the hospital's HR-department and medical ethics review board have given permission to conduct our research and deploy CommIC on the ICU. We have an approved document describing the purpose and procedure of our research that each patient needs to consent to before using the system. Consent can be obtained from family or from the patient with a yes/no signal after explaining the document's content. Unfortunately, as we organize the deployment of CommIC an involved medical specialist disagrees with the proposed consent procedure. He wants the patient to explicitly consent or decline with CommIC logging their data through the interface. We decide to delay release until this new requirement is implemented in the next sprint.

Even though the medical specialists are not officially concerned with the ethical components of our research, as important stakeholders their opinions are just as important as those of the HR department and the medical ethics review board. Because until now they were not seen as stakeholders in these matters, relevant feedback has been received later than it should have. In order to support agile's rapid feedback goal the product owner should in the future more critically asses his/her assumptions about each stakeholder's interests.

7.8 SPRINT 7

7.8.1 User stories

In this sprint we implement more priority 5 user stories, including some UI semantic construct functionality and look.

- The system enables patients to easily tell what kind (stabbing/nagging/cramping), where, when and how much pain they are feeling.
- The system enables patients to easily tell where they have an itch.
- Patients should be prompted whether to store their usage data.
- Option should have a Position property
- Option should have a Shape property
- The Favorite property of Option should be modified to show Options bigger rather than with a double border.
- The common messages menu structure should be improved.

The first two user stories are not independent as they share the functionality of selecting a location on the human body. We rewrite these user stories to the ones below. Only the first one has highest priority and is implemented in this sprint.

- The system enables patients to communicate the location of pain/discomfort.
- The system enables patients to communicate the nature of their pain (kind, frequency, severity).

7.8.2 Design

The introduction of a prompt asking whether CommIC can store user data, means that Options should be placed in dialogs. This should not be a problem, since Option was designed as an independent UI component. Unfortunately, the existing dialog buttons exhibit behavior not supported by Options. In particular, dialogs have a timeout after which the default button, which is selected, is chosen automatically. This cannot be trivially unified with Options where the state of being selected is connected to receiving input. Mainly, only an Option that is looked at is selected. In order to support dialog behavior we decouple these two concepts into separate properties. Options can be **Focused** and/or **Selecting**. When an Option receives input, it is Selecting and Focused. When no Option is looked at, one can still be Focused, as is the case for the default Option in a dialog. We can now modify the dialog to give focus to the default Option and run the countdown to auto selection only while no Option is Selecting.

The possibility of selecting a location on the human body has already been taken into account when designing Choosables and Options. The straightforward implementation simply draws Options in the shape and on the relative position specified by their Choosable. Unfortunately their exists a major drawback to this approach. Namely, logical subdivision of the body into body parts leaves many parts (head an appendices) to small or thin to be selected right away.

An alternative approach to get around this does away with the body's subdivision into parts and gradually zooms in to the location on the body the patient wishes to select. This method is very intuitive since it takes away the complexity of navigation through multiple selection levels. Additionally this allows for faster selection of the desired part. Following the principle of cognitive integration, we observe several difficulties with this solution though.

Firstly, the user has no means of recognizing his destination. It is not trivial when zooming in will be finished. We could define a maximum zoom level based on the amount of precision that is desirable. The amount of precision differs per body part though. It should be possible to zoom in towards an eye but allowing a user to zoom in this far on the thorax would result in a view lacking the distinguishable features required for orientation. A related problem that needs to be considered is how to communicate this selected location. An arbitrary location on the body cannot be named. The only way to communicate this location to the listeners is to freeze the image and allow them to look on the screen, which is not practical. We conclude that a subdivision of the body into named parts is desirable. As the user zooms in on a part it can become focused as soon as it is large enough. At this point the UI zooms in on the part, slowing down as the part almost fills the view to indicate that the destination has almost been reached.

Secondly, the user is not provided with a clear route choice, since individual destination Options might not yet be clearly visible from low zoom levels. In many cases they should not be, to abide by the patient's perceptual limit following the principle of complexity management. Fortunately, we designed Options to be uniquely distinguished from non-choosable parts of the interface by their high brightness. This means that even though individual Options might not be recognizable, the entire bright area of all

Options combined is clearly visible and provides a clear route choice. Additionally, the UI should only zoom in towards valid route choices.

Thirdly, the user cannot clearly monitor his route, since it is unclear where exactly the user is zooming in towards when no destination Option is focused yet. This is solved by reintroducing the Gaze Pointer. Since the Gaze Pointer marks the point that the user is zooming in towards, it should only be displayed when the user is zooming in, which is the case when the user looks at Options. Therefore we only display the part of the Gaze Point overlapping with Options.

Another issue is posed by the need for undoing navigation. Zooming in could be undone by a back button, but when the user is zoomed in far this is not suitable for minor corrections. A better solution is to employ side scrolling. If the user looks close enough towards a side of the view the view starts panning into that direction. This is intuitive because it happens automatically when the user follows the part they want to select with their eyes. When the view zooms towards the wrong point the part moves towards a side and by following the view starts panning in the direction of the part again.

Additionally, the view should start zooming out again so that the user can always return to a higher zoom level for orientation. It should be possible to correct you destination until selection is complete, but at the same time it is not desirable that the view starts zooming out while the user is looking at his intended destination. The selected destination should therefore not fill the whole screen while selecting. A destination Option is thus reached before if fills the screen. This possibly reintroduces the problem of unclear destination recognition, although arrival at the destination is still recognizable due to the slowing zoom. A possible solution to this is visualization of the view region that should be filled. This means the introduction of another symbol, increasing graphic complexity and also increasing the chance for cognitive overload. We do not implement this solution unless it later appears to be required.

The design of this new UI component does not introduce new semantic constructs. We should make sure that no new symbols or symbol properties are introduced either to prevent symbol redundancy. This is not completely possible. Several symbols or symbol properties cannot be preserved in this UI.

- A Focused/Selected Option's border should be made thick. However, there is no displayed border to begin with. Instead we make a border appear to indicate the Focused property. The Option should also be made bigger in relation to other Options, this is unwise as it would distort the combined figure of the Options.
- Choosables are not represented by icons. We can and should on the other hand display their text.

7.8.3 Process evaluation

At the end of the first sprint we decided to delay sprint time planning until after the first release. We considered adopting the planning game again for this sprint. However, our capacity to predict user story implementation time over the last six sprints has hardly increased. We have to conclude that sprint planning is not a useful methodology for a single developer who is not an expert programmer.

7.8.4 Feedback

Due to the hard deadline at the end of the research project, we have not finished the pain pointer in time and are thus not able to receive feedback on it.

We perform two more patient trials, starting with patient A and his nurse. This trial is more successful than the previous ones. The patient has sufficient mental clarity to answer several questions using the

common messages menu. We also ask him to write his name in the letters module, but he is still incapable of spelling. Following the previous trial, we concluded that we lacked information about the patient to properly analyze our observations. We therefore inquire about the needed details about the patient. He is literate and has experience using computers. Furthermore he never uses glasses or lenses, is not colorblind and has good hearing. The only explanation we have for the patient's inability to spell is that his language skills are damaged, a likely possibility according to his nurse.

The second trial we perform is with a patient who appears to be awake and can understand and respond to our questions with eye blinks. We try to calibrate CommIC, but are unsuccessful. The patient cannot keep his eyes fully open for more than a few seconds.

Our observations during this trial yielded feedback. We have also received feedback trough a filled in feedback form and by interviewing several members of the medical staff who used CommIC.

- After choosing a Message or Action the user should not immediately be able to select an Option again to prevent accidental input. This has been immediately remedied by showing a non-interactive message window with the Choosable's text for a few seconds. This solves another problem as well, namely that it might be unclear when a Message has been chosen, especially for deaf people who don't hear the Messages pronunciation.
- When the patient is tired or temporarily does not want to give input he sometimes accidentally inputs messages. It should be possible to stop and resume input.
- A nurse asked how to turn on the system. Indeed the on/off button of the tablet is not very apparent. We have added an on/off label next to this button.
- The user manual should include a checklist to judge whether a patient can use CommIC. We conclude from the second patient trials and from the experience of other medical personal as well that many usage attempts of CommIC are unsuccessful due to limited patient capability. This waste of time goes against the requirement that the system should require little time for medical personal. Based on previous trials we add simple patient criteria to the manual that increase the chance for successful deployment of CommIC. The patient should be able to operate a tablet with his hand or to use eye tracking, the patient should be able to:
    - follow a simple command
    - keep his/her eyes open well
    - follow your finger at 40 cm in front of his/her face with his/her eyes
- Despite earlier attempts to improve the calibration interface for medical personal and providing simple instructions in the manual, there exists confusion about when CommIC should be calibrated. Sometimes personal forgets to position the system correctly before starting calibration. The button labeled "Recalibrate" is sometimes interpreted as an instruction rather than an option to recalibrate. When the calibration quality is bad or has deteriorated is not clear that and how the calibration can be improved. Whether CommIC is successfully calibrated is not always understood.

We conclude that a redesign of the workflow to setup eye tracking is necessary. CommIC should simply not allow for incorrect input and give step by step instructions to the user. To allow this new workflow we need to automatically detect whether the eyes are positioned correctly. We also streamline the recalibration process by dividing the calibration in a test phase and calibration phase. If an existing calibration exists, the tracker is only recalibrated if the calibration can be improved.

The redesigned workflow is depicted below using a state diagram. States that display messages to the user are drawn as rectangles and transitions that are triggered by user input are labeled with brackets.

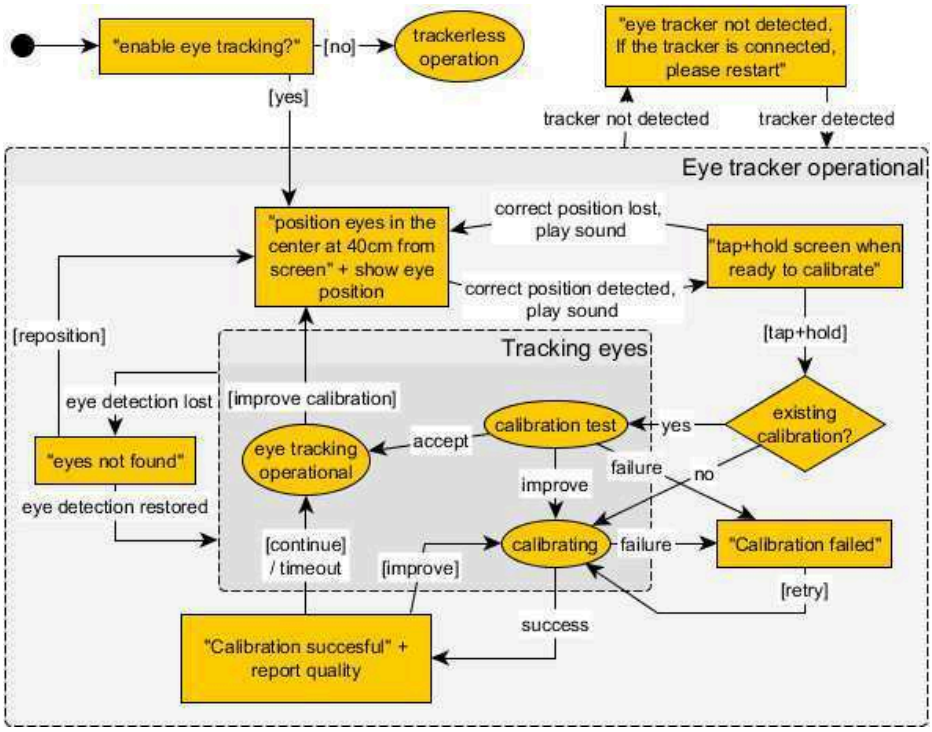


Figure 14: New gaze tracking setup workflow

Further feedback has been received from non-stakeholder users about the arrangement of common messages. Multiple changes have been proposed to improve the cognitive integration of the common messages. These changes have immediately been implemented and tested with patient A by asking him to communicate messages with Options in the changed menus. The improvements were all verified, except for the first one.

- "I am troubled by" might be more intuitive than "I feel", since practically all feelings in this Category are negative. Patient A has more trouble finding Messages in the "I am troubled by" Category than before, so this change is reverted.
- "I understand" is not useful, as the user can say "I don't understand" if he does not.
- "I want > breathe by myself / assisted" is more intuitive than "I want > to pause/resume ventilation".
- "See/speak with" as a Category within the "Images" Category is more intuitive and decreases the average navigation path length in comparison to "I want > A person"
- Within this same Category the "Type name" Action should be replaced with "A friend". This way all basic categories of people likely to be requested are available through common messages. In case specificity is desired, the patient can follow the Message by typing letters in the "Letters" module.
- "Tell about > my pets" should be moved to "Tell me about > my pets", since patients often worry about pets they left at home. Originally, this communication option was added so that medical personal could ask what pets a patient has when this is relevant for treatment. However, this is relevant at the start of hospitalization when the patient is rarely in a state of mind to respond.

8 EVALUATION AND RECOMMENDATIONS

Here we answer the final research question: To what level does this new solution reduce the severity of the identified problems? For the initial requirements analyses we listed a number of goals. We discuss how well each of them has been achieved. Discussions and usage testing with all stakeholders form the basis for these analyses. Finally, we look back at the used methodology and evaluate its contribution to our successes and failures.

8.1 COLLECTION OF FEEDBACK

For each implemented user story and each global requirement we asked the relevant stakeholders about their satisfaction. Appendix E lists all requirements and the received feedback. It is not always trivial for stakeholders to determine whether they are satisfied about a certain feature based on a demonstration of the system. Also, in the case of patients, it is difficult to find out whether they are satisfied. When appropriate, tests were conducted with the stakeholders to determine their satisfaction. Appendix E includes descriptions of such tests.

It is important that these acceptance tests are agreed upon by the stakeholders whenever possible, since formulating reliable tests requires that we know the exact requirements we try to satisfy. The agile approach assumes this is not the case. The only proper way to test whether a user story is correctly implemented is by stakeholder feedback.

Some acceptance tests refer to standard tasks. These tasks are composed to test basic understanding and ability to use the interface.

- Communicate how you feel.
- Ask about your pets.
- Tell me your name.
- Communicate that your ears hurt.

All patient tests have been conducted with only patient A, since he has been the only ICU patient capable of using CommIC during the final phase of the research. Trials with patients who were not able to use the system have also yielded some valuable feedback, but in all other cases we had to conclude that the patient did have the mental or physical capabilities of our targeted patient group. In order to collect more patient feedback, we interviewed an ex-ICU patient, patient C, about her personal experience and that of others during the ICU patient return day. This is an event where recovered ICU patients return to share their experience and give feedback to medical personal about their care. This feedback is also represented in Appendix E.

Patient C was a patient on the MST ICU one year ago. During the last days of her stay she was awake and fervently attempted to communicate with her family without success. She has experimented with CommIC and is confident that is would have been very helpful to her. The lack of communication was the hardest part of her ICU stay and this judgement is shared by most other ICU patients on the ICU patient return day. She also stresses the value of CommIC as a source of information and orientation. Many patients are in a constant state of confusion as they have the same blurry view all the time. Patient C could only see a clock on the wall, which she was unable to read. She became obsessed with the time and the strange symbols on the clock.

Patient C and her family attempted to communicate through a tablet during her stay, but this was unsuccessful. She constantly underestimated her ability to point to the screen. This cost enormous amounts of energy for her and even though she had the necessary motor control, she could not point to more than one item on the screen before running out of energy. This experience is shared by many other patients as well.

8.2 STAKEHOLDER GOAL ANALYSIS

The requirements matrix allows us to draw conclusions about how well the stakeholder goals have been reached. On the whole, of the 23 priority 5 requirements 74% has been satisfactorily implemented and another 17% is mostly satisfied. This leaves 9% of the must-have requirements (mostly) unsatisfied, but this does not mean the system is unusable. To gain a more meaningful picture of what has been achieved, we look at the separate stakeholder goals that CommIC supports.

CommIC's first goal is to enable communication between patients, medical staff and family. From the requirements evaluation we can conclude that CommIC has made important improvements toward this goal. All indispensable functionality has been (almost) satisfactorily implemented.

The ability for patients to form sentences is achieved for many patients, but can still be improved by means of word or letter prediction and alternative input systems like Dasher (The Inference Group, sd). Implementation of these improvements will be time consuming. However, we believe entering an entire sentence will never be possible within the limited attention span of many CommIC using patients.

Furthermore, the system should still improve for people with blurry vision. Patient C has indicated that blurry vision is very common among patients. Personally she saw everything blurry, especially further away, even though she normally has no vision impairment. Supporting patients with blurry vision deserves a higher priority than it was originally assigned. We have performed additional trials with a user with blurry vision. He has about 10% of normal visual capacity. He indicates that some to the text in CommIC should be bigger. Additionally some icons can be clearer. This is the case for icons containing thin lines and icons that use contrast of dark on a bright background, rather than bright on a dark background. For example, white text on a black background is more readable.

Overall, we conclude that CommIC has achieved its goal of enabling patient communication within the bounds of what can realistically be achieved within the scope of this research project.

Several goals have not yet received any attention, since they lack priority. These are the goals to enable patients to orient themselves, the goal to provide them with distraction from their unpleasant situation and the goal to allow the system to be easily used with multiple patients. From our talks with patient C, we conclude that the provision of orientation and distraction should be given a higher priority. The supporting requirements are important future work.

CommIC's design goal to require little physical and mental effort for its usage has been the biggest challenge of the project. From our feedback we conclude that this challenge has been overcome for the most part. Unfortunately, we have not yet been able to satisfy the requirement that the system should automatically adjust its speed and complexity. In our design we have put extensive effort in making system complexity and speed dynamic. The reason for lacking stakeholder satisfaction is that stakeholders cannot use this property of the system yet. The system can but does not adjust its

complexity and speed depending on their needs. Making this possible is a high priority for future work. Based on our analysis of this requirement and resulting design effort, we expect that once this future work is completed the implementation of this requirement will be satisfactory.

Our success towards achieving this goal has a level of uncertainty. During the project we only had a few patients available to test these requirements and only in our trials with Patient A we have been able to verify that CommIC is usable in real-world situations. Tests with different patients might still yield unexpected problems and bring further improvement opportunities to light.

Moving on, we may conclude that the goals of making CommIC's functionality reliable and to never put the patient's wellbeing or healthcare process at risk or obstruct it have for been achieved. All high priority requirements supporting them are satisfied.

Medical personal wants the system to be easy and quick to use. This goal still needs work. In particular, the workflow redesign proposed in sprint 7 needs to be implemented. The need for this improvement surfaced late, when the system started to be used in practice. It was not observed during or following the first demos and trials on the ICU. This is likely caused by the fact that the system was initially used only by medical personal that had recently attended a demonstration. Later the system was also used by personal that had not attended a demonstration or that had forgotten the demonstrated procedure. We were wrong to assume the system is only used by medical personal with some experience. Performing trials with completely unexperienced personal will be beneficiary in the future.

Another goal that is successfully supported is that of physiotherapists wanting patients to move optimally during their ICU stay. This goal has primarily been relevant for the design of the mechanical arm which is not discussed in this report.

Finally, our own goal as developer to remotely monitor and remedy problems and collect usage data, has almost completely been achieved.

Despite the fact that not all requirements that are deemed necessary have been fully satisfied, CommIC has already demonstrated the potential to add great value to ICU patient care. Feedback from all stakeholders has been very positive. We are confident that CommIC can be further developed to a great product that will reach countless hospitals, patients, nurses, medical specialist, physiotherapists and family members.

8.3 METHODOLOGY

The use of agile software development principles has effectively contributed towards solving the design problems. In particular, the short feedback loops have allowed CommIC to be repeatedly improved as it fostered a continuous growth in our understanding of the problem domain.

As evaluated after each sprint, not all planned agile methods have been successfully adopted. The Scrum practice of sprint planning and predetermined development cycle duration have been dropped as it appeared to be inapplicable to the small non-expert development tea. Our flexible sprint planning has in our experience not had a negative impact on stakeholder satisfaction. Additionally, we shifted our focus from test driven development to systematic black box testing, as this is more suited for testing of GUI components and complex gaze input handling. This has been effective.

Our own planned adjustments to the agile approach were effective. We believe that the adoption of commonality / variability analysis has been a useful addition on top of agile's design simplicity principle. The central architecture of our choice model has been expanded upon many times during development, but due to our initial analysis no major design changes were necessary. The use of goal based requirement analysis has also been valuable. Reasoning from and continuously enquiring about stakeholder goals has successfully guided the collection of requirements and has helped unearth new requirements that might otherwise have been overlooked. During the project we have tried to keep all requirements organized by their (high level) stakeholder goals. Although this was convenient for our analysis earlier in this section, we have not seen any advantages to this organization. Rather, it was a distraction that inhibited the flexibility of user stories. We conclude that the traditional agile use of an unstructured backlog would have been more suitable.

9 GLOSSARY

AAC

See Augmentative & Alternative Communication.

Acceptance test

An automated test to check requirement satisfaction. See section 6.4.5.

Action

A type of Choosable, an action that is performed by the software.

Anamnesis

Information that the patient can tell medical personnel concerning the prehistory and relevant conditions surrounding his affliction.

Augmentative & Alternative Communication

AAC systems attempt to compensate and facilitate, temporarily or permanently, for the impairment and disability patterns of individuals with severe expressive and/ or language comprehension disorders. Augmentative/alternative communication may be required for individuals demonstrating impairments in gestural, spoken, and/or written modalities.

Backlog

List of all not yet implemented user stories.

Capacity (of a visual variable)

The number of perceptible steps that can be differentiated.

Category

A type of Choosable, a category containing similar Choosables.

Choosable

A semantic construct abstracting an element that can be chosen through an Option.

Choosable Option

A semantic construct. Type of Option to choose a Choosable.

Choosable Parent – Child relation

Semantic construct indicating that a Choosable is a child of a certain other construct.

Chooses relation

Semantic construct indicating that selecting a certain Option chooses a certain Choosable.

Cognitive fit

See section 7.5.2.3.9.

Cognitive integration

See section 7.5.2.3.5.

Cognitive map

A mental overview of a complex system that contextualizes its components.

Cognitive overload

The condition when more information is presented at a time than can be effectively processed by the human mind.

Cognitively effective

Refers to a UI that that succeeds in transferring intended information to the user.

Cohesion

A software module property that describes the level of functional relatedness of its elements. See section 6.3.1.

Complexity management

See section 7.5.2.3.4.

CV-analysis

Commonality/Variability analysis. See section 6.3.2.

Daily Scrum meeting

A Scrum development practice. See section 6.4.13.

Defining Category

A Category that forms a context that helps define its children's semantics.

Destination recognition

See section 7.5.2.3.5.

DIP

Dependency Inversion Principle. See section 6.3.4.

Disabled Choosable

A Choosable that cannot be chosen.

Dual encoding

See section 7.5.2.3.7.

EEG

Electroencephalography (EEG) is the recording of electrical activity along the scalp. EEG measures voltage fluctuations resulting from ionic current flows within the neurons of the brain.

EMG

Electromyography (EMG) is a technique for evaluating and recording the electrical activity produced by skeletal muscles.

Endotracheal tube

A tube through the mouth or nose into the wind pip for mechanical ventilation.

Expertise reversal effect

Phenomenon that simplification of information representation makes it more difficult to comprehend by domain experts.

Extreme Programming

A popular agile development method. See section 6.4.

Favorite Choosable

A Choosable that is frequently chosen by the user.

Focused Subchoice

A property of Subchoice indicating whether it is being made.

Focused Option

A visual property of an Option symbol that is the default or that it is Selecting.

Gaze Pointer

A semantic construct indicating the location on the screen where the user is looking.

Graphic complexity

The number of different used symbols.

Graphic economy

See section 7.5.2.3.8.

ICU

Intensive Care Unit, a special department of a hospital or healthcare facility that provides treatment for critically ill patients.

ISP

Interface Segregation Principle See section 6.3.5.

Level of measurement

Indication of the type of information: ordinal, nominal or interval

LSP

Liskov Substitution Principle. See section 6.3.3.

Mechanical ventilation

Artificial breathing, applied through either a facial mask or a tube inserted into the mouth/nose or through and incision in the neck.

Message

A type of Choosable, a message that is relayed to a listener.

Metaphor

An Extreme Programming development practice. See section 6.4.17.

Module

A type of Choosable, separate part of the program offering a certain type of functionality.

MST

Medisch Spectrum Twente, the hospital for which CommIC is built.

MVP

Minimal Viable Product. Minimal implementation of the system that is usable in practice.

MVVM

Model-View-Viewmodel design pattern. See section 7.2.2.3.

Navigation Option

A semantic construct. Type of Option to navigate the UI.

OCP

Open-Close Principle. See section 6.3.2.

Option

A semantic construct. An input option that can be chosen. Choosing an Option is the sole form of interaction.

Order (of a Choosable)

A property of an ordinal Choosable indicating how it is ordered relative to other ordinal Choosables.

Orientation

See section 7.5.2.3.5.

Page

Semantic construct, subset of Choosables in same Category that is shown simultaneously.

Pair programming

An extreme programming practice. See section 6.4.6.

Perceptual discriminability

See section 7.5.2.3.1.

Perceptual limit

The maximum amount of information the human mind can effectively process at a time.

Perceptual step

The minimum value change of a visual variable that can be perceived.

Planning game

An agile development practice. See section 6.4.12.

Product owner

A role in the agile development team. See section 6.4.1.

Redundant coding

The practice to use multiple visual variables to create visual distance.

Route choice

See section 7.5.2.3.5.

Route monitoring

See section 7.5.2.3.5.

Tracheostomy tube

A tube through an incision on the anterior part of the neck for mechanical ventilation.

Scrum

A popular agile development method. See section 6.4.

Scrum master

A role in an agile development team. See section 6.4.2.

Selecting Option

Visual property of an Option symbol that is receiving input.

Semantic construct

Elementary building block of information. See section 7.5.2.1.

Semantic transparency

See section 7.5.2.3.3.

Semantically immediate

A property of a symbol whose meaning can be deduced by novice readers based on appearance alone.

Semiotic clarity

See section 7.5.2.3.2.

Shape (of a Choosable)

A property of a Choosable indicating the 2-dimensional shape to the Choosable.

Spatial Position (of a Choosable)

A property of a Choosable specifying where it is located relative to other Choosables.

Sprint

A short agile development cycle. See section 6.4.4.

Sprint retrospective

A Scrum development practice. See section 6.4.14.

SRP

Single Responsibility Principle. See section 6.3.1.

Subchoice

Semantic construct, a set of Choosable Options that further define a Choosable.

Subchoice sequence relation

Semantic construct indicating that a subchoice follows/is followed by another.

Symbol

Graphic representation of a semantic construct. See section 7.5.2.1.

Symbol redundancy / overload / excess / deficit

See section 7.5.2.3.2.

Test-driven development

An agile development practice. See section 6.4.7.

Token

Instance of a symbol. See section 7.5.2.1.

Virtual Choosable

A Choosable with abstract semantics that need further details.

Visual appearance (of a Choosable)

A property of Choosable indicating what it looks like.

Visual distance

Measure for the number of visual variables on which symbols differ and the size of these differences.

Visual expressiveness

See section 7.5.2.3.6.

Visual notation / vocabulary / semantics / grammar / syntax

See section 7.5.2.1.

Visual saturation

Condition when all (8) visual variables are used to carry information.

Visual variable

Atomic building block of a visual representation. See section 7.5.2.1.

Visual vocabulary

See section 7.5.2.1.

User story

An agile representation of a requirement. See section 6.4.3.

XP

Extreme Programming, a popular agile development method. See section 6.4.

10 REFERENCES

A.J. Doud, J. L. (2011). *Continuous Three-Dimensional Control of a Virtual Helicopter Using a Motor Imagery Based Brain-Computer Interface.* University of Minnesota.

A.J. Rotondi, i. (2002). Patients' recollections of stressful experiences while receiving prolonged mechanical ventilation in an intensive care unit. *Critical Care Medicine*, 746-752.

Bertin, J. (1983). *Semiology of Graphics: Diagrams, Networks, Maps.* University of Wisconsin Press.

C. Jones, R. G. (2000). Disturbed memory and amnesia related to intensive care. *Memory*, 79-94.

E. Naves, L. R. (2012). *Alternative communication system for people with severe motor disabilities using myoelectric signal control.* Federal University of Uberlandia.

Eric Freeman, E. R. (2003). *Head First Design Patterns.* O'Reilly Media.

Healthline. (2005). Health Reference Library.

J. Highsmith, A. C. (2001). Agile Development: The Business of Innovation. *Computing Curricula*, 220-222.

John Hopkins Medicine. (n.d.). *ALS - Amyotrophic Lateral Sclerosis.* John Hopkins Medicine. Retrieved 05 16, 2014, from http://www.hopkinsmedicine.org/neurology_neurosurgery/centers_clinics/als/conditions/als_amyotrophic_lateral_sclerosis.html

K. Schwaber, J. S. (2013). *The Scrum Guide.* Srum.org.

Maksimovic, Z. (2012). Retrieved from Agile-code: http://www.agile-code.com/blog/mocking-with-moq/

Martin, R. C. (2003). *Agile Software development.* Upper Saddle River, New York: Pearson Education, Inc.

Microsoft. (2014). *Developer's Guide to Microsoft Prism Library 5.0 for WPF.* Microsoft.

Ministerie van Volksgezondheid, Welzijn en Sport. (1998). *Wet medisch-wetenschappelijk onderzoek met mensen.* Retrieved from Overheid.nl: http://wetten.overheid.nl/BWBR0009408/geldigheidsdatum_08-07-2014

Moody, D. (2009, 11/12). The "Physics" of Notations. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, pp. 756-779.

P Garrard, D. B. (2002). Cognitive dysfunction after isolated brain stem insult. An underdiagnosed cause of long term morbidity. *Journal of Neurology, Neurosurgery, Psychiatry, 73*, 191–194.

R. Rivera Fernandez, J. S. (1996). Validation of a quality of life questionnaire for critically ill patients. *Intensive Care Med*, 1034-1042.

Steward, S. (2009). Designing AAC Interfaces for Commercial Brain-Computer Interaction Gaming Hardware. (pp. 265-266). New York: University of Delaware.

T.M. Brown, M. B. (2002). ABC of psychological medicine: Delirium. *British Medical Journal, 325*, 644-647.

The Inference Group. (n.d.). Retrieved from The Dasher Project: http://www.inference.phy.cam.ac.uk/dasher/

Tigchelaar, M. (2013). *Haal meer uit je hersenen.* Bert Bakker.

University Of Michigan. (2000, December 13). Working Together In "War Rooms" Doubles Teams' Productivity. *ScienceDaily*. Retrieved from www.sciencedaily.com/releases/2000/12/001206144705.htm

Appendix A. COMMUNICATION IMPEDING ILLNESSES

Note that not all of these diseases are sufficient cause for ICU treatment, but all of them cause significant communication difficulties for their victims. The source of most of the information described here is the Healthline Health Reference Library (Healthline, 2005).

Guillain-Barre syndrome
Guillain-Barré syndrome is a disease that affects the nervous system. It is very rare, at one to two cases per 100,000 people annually. However, it is the most common cause of acute non-trauma-related paralysis. Paralyses occurs from the bottom up. In higher parts of the body often only the muscles weaken, with facial weakness being common. Other symptoms include drooling or difficulty swallowing and/or maintaining an open airway and respiratory difficulties. Eye movement abnormalities are not common. Most patients require hospitalization and about 30% require ventilator assistance.

Locked-In syndrome
Locked-In syndrome is not a specific medical illness, but rather the name for a condition of complete paralyses. There are many different causes for this to occur. Patients lock motor control of every voluntary muscle, except the eyes, whilst remaining fully conscious. Often only vertical eye movement is possible and also blinking. "Although patients are conscious, attention, executive function, intellectual ability, perception, and visual and verbal memory can be affected" (P Garrard, 2002). Locked-In syndrome is very rare.

Alzheimer's disease
Alzheimer's disease (AD) is a neurologically degenerative condition characterized by a progressive decline in memory, attention, problem solving, and language skill. The prevalence of AD among older adults has become a public health issue worldwide. Current reports indicate that 6–10% of all individuals over the age of 65 and 33–40% of individuals by the age of 90 years suffer from AD. AD alone is not sufficient reason for ICU treatment.

Amyotrophic Lateral Sclerosis
Amyotrophic lateral sclerosis (ALS) is a nervous system disease that attacks nerve cells called neurons in your brain and spinal cord. Estimates suggest that ALS is responsible for as many as five of every 100,000 deaths in people aged 20 or older.

These neurons transmit messages from your brain and spinal cord to your voluntary muscles. These are the muscles you can control, such as arm and leg muscles. At first, this causes mild muscle problems. Some people notice trouble walking or running, trouble writing and speech problems. Eventually, you lose your strength and cannot move. When muscles in your chest fail, you cannot breathe. A breathing machine can help, but most people with ALS die from respiratory failure. Individuals affected by the disorder may ultimately lose the ability to initiate and control all voluntary movement, although bladder and bowel sphincters and the muscles responsible for eye movement are usually, but not always, spared until the final stages of the disease. (John Hopkins Medicine)

Paraplegia & Tetraplegia
Paraplegia and Tetraplegia are paralysis caused by illness or injury that results in the partial or total loss of use of all muscle control in limbs and torso. Paraplegia does not affect the arms. A common cause is

traumatic lesion; the condition of disruption of nervous signals through the spinal cord, usually due to a fraction or dislocation of a vertebra.

Muscular dystrophy
Muscular dystrophy is a disease that causes progressive muscle weakening. In severe cases it can cause complete functional disability of the muscles. The muscles affected vary, but can be around the pelvis, shoulder, face or elsewhere

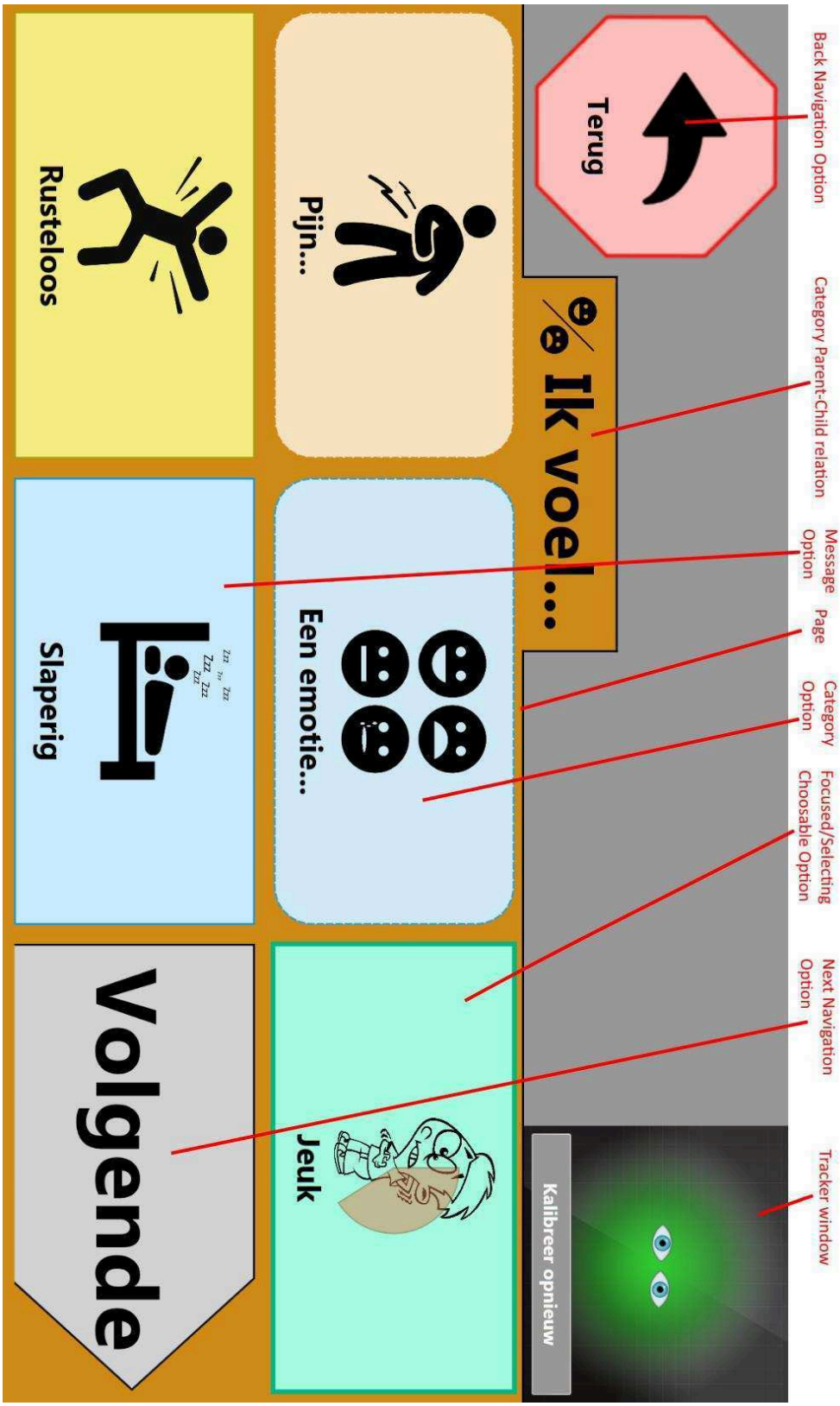
Cerebral palsy
Cerebral palsy is characterized by abnormal muscle tone, reflexes, or motor development and coordination. There can be joint and bone deformities and contractures (permanently fixed, tight muscles and joints). The classical symptoms are spasms and other involuntary movements, including facial gestures. Problems in speech are commonly caused by difficulty controlling breathing, the larynx or oral muscles.

Appendix B. MATRIX

✓	satisfied
+	mostly satisfied
–	a little satisfied
×	not satisfied
?	unknow

User stories	GoTalk Pocket	Voice	Brainfingers	SIDE	GazeTalk	The Grid 2	Comunicator	Tobii	Comment
Patient communication should be possible when...									
the patient is mechanically ventilated	✓	✓	✓	✓	✓	✓	✓		
the patient has no muscle control	×	×	✓	×	×	×	×		
the patients has weak muscles	–	×	✓	✓	✓	✓	✓		Voice: easy to accidentally touch other parts of screen
the patients has uncoordinated muscle control	–	×	–	×	–	×	–		Voice,SIDE: recovery from accidental input very cumbersome
the patient can only move his finger / toe / hand / foot / arm / leg / head / facial muscles	×	×	✓	✓	×	×	–		Tobii,GazeTalk: low-resolution pointing possible
the patient can only move his eyes	×	×	✓	×	✓	✓	✓		
the patient is disoriented	+	+	×	–	+	+	+		
the patient is exhausted	–	–	+	–	+	+	+		Using eyetraker only requires constant attention, less so with switch
the patient has a very short attention span	+	✓	–	–	+	+	+		Using eyetraker only requires constant attention, less so with switch
the patient is cannot memorize	+	✓	×	✓	✓	+	+		Grid/Tobii: difficult to use multiple inputs
the patient is confused/has no problem solving capability	+	+	×	+	+	+	+		
the patient does not speak the same language	–	–		–	–	–	–		
the patient has no experience using computers	✓	✓	–	✓	✓	✓	✓		
the patient is illiterate	+	✓	–	✓	✓	✓	✓		
the patient is deaf	+	✓	✓	✓	✓	✓	✓		
the patient can only hear loud noises / low tones	+	✓	✓	✓	✓	✓	✓		
the patient is blind	×	✓	×	×	×	×	×		
the patient is color blind	✓	✓	✓	✓	✓	✓	✓		
the patient can only see blurry	–	✓	×	+	×	–	+		Tobbii: buttons sizes customizable + auditory prompts
the patient is lethargic or slow	+	✓	?	–	–	+	+		SIDE: risk of incorrect input, GazeTalk: less visual stimuli
little can or should be demanded from the patients cognitive abilities	+	✓	×	+	–	+	+		Using eyetraker only requires constant attention, less so with switch
The system be able to handle ...									
patients having spasms	+	–	?	–	–	–	–		
patients having delirious episodes	✓	–		?	?	?	?		Can patient hardware mollen / configuratie wijzigen?
The patient should be able to...									
have a normal conversation	×	×		×	+	+	+		Tobbii/The Grid/GazeTalk can combine with Dasher for fast typing
to convey an emotion	✓	✓		✓	+	✓	✓		
to ask about their health condition	✓	✓		✓	+	✓	✓		
to request to cycle with his hands or feet	✓	✓		✓	+	✓	✓		
to ask for someone	+	+		✓	✓	✓	✓		
to tell he has an itch, where and how severe	–	✓		+	+	+	✓		
to tell that he is not comfortable / nauseous / stuffy / thirsty / hungry / sleepy	✓	+		✓	+	✓	✓		
to ask for start/stop of mechanical ventilation	✓	+		✓	+	✓	✓		
to call for help	×	×		×	+	×	+		GazeTalk/Tobii allow external alarm device to be connected
to know the time	×	✓		–	–	+	+		Clock integrated (Voice) vs indirect via computer control
to know where he is	×	✓		×	×	×	×		Voice geeft locatie naam
to familiarize himself with his surroundings	×	×		×	×	×	×		
to tell he does (not) understand	✓	+		✓	+	✓	✓		GazeTalk,Voice: through text
to ask for rest	✓	✓		✓	+	✓	✓		GazeTalk: through text
to ask for sedative	✓	✓		✓	+	✓	✓		GazeTalk: through text
be distracted from pain/stress	×	+		+	✓	+	+		Only GazeTalk integrates, else by control of comuter / externals
be entertained (video/music/(audio)book	×	–		+	✓	✓	+		Tobii: Only via computer control
contact others via e-mail/messaging/phone call	×	✓		×	✓	✓	+		Tobii: Only via computer control
Medical staff should be able to inquire...									
if the patient feels pain	✓	✓		✓	+	✓	✓		GazeTalk: through text
what kind of pain he is feeling	×	+		✓	+	✓	✓		GazeTalk,Voice: through text
where the patient feels pain	×	+		–	–	–	+		Pain pointer (inprecise), or through tekst
how much pain is felt by the patient	–	✓		+	+	+	✓		Pain pointer, or through tekst
the answer to a yes/no question	✓	✓		✓	+	✓	✓		GazeTalk: through text
what happened	×	+		✓	✓	✓	✓		Through tekst/pictures
whether the patient understood	✓	✓		✓	+	✓	✓		GazeTalk: through text
The system should...									
allow more efficient communication for more capable patients	×	–	–	×	–	+	+		Depending on offered I/O capabilities
always be save to use	✓	✓	+	✓	✓	✓	✓		Training brainwaves puts stress on cognitive abilities
always be kept clean	✓	✓	✓	✓	✓	✓	✓		Just requires appropriate hardware to run software
be very easy to setup for new patients	✓	✓	–	+	–	+	+		GazeTalk: output options need configuration
require very little time to setup for new patients	✓	✓	×	+	–	+	+		GazeTalk: output options need configuration
support adding or removing inputs		?				+	+		
recover from malfunctioning or suddenly disconnected from input or output		?	?	?	?	?	?		
automatically recover after (accidentally) being turned off	✓	+	?	?	?	?	?		
notify personal if its functioning is compromised	×	×	×	×	×	×	×		
be portable	✓	✓	–	–	–	–	–		

Appendix C. GUI DESIGN



Appendix D. Black box test cases

Test suite 10: UI

Test case 204: Navigate Option hierarchy

#	Action	Expected value
1	Prerequisite: The system is booted and the Talk module contents are shown.	
2	Select a Category	The Category's contents are displayed.
3	Select a Category with more children than fit on one Page	Some of the Category's contents are displayed. In the lower right corner a Next navigation Option is displayed.
4	Select the Next navigation Option.	The next Page slides in from the right, showing more of the Category's children. In the upper right corner of the page a Previous navigation Option is displayed
5	Select the Previous navigation Option	The previous Page slides in from the left.
6	Select the back Option	The previously selected Category's contents are displayed.
7	Select a Category	The Category's contents are displayed.
8	Select a MessageChoosable	The Choosable's message is displayed in a message window and it is pronounced. After a few seconds the window disappears. The Talk module contents are displayed.

Test case 201: Gaze-select Option

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and and perfectly calibrated	
2	Look at an Option	The option immediately receives focus (bigger, thick border). After a short timeout a filling pie shape appears. The moment the pie is full the Option is selected.
3	Look at an Option, until the pie is almost full, then look away. As soon as the pie shape disappears, look at the Option again	The pie shape reappears and is filled (almost) as far as before.
4		

Test case 202: Gaze stickyness

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and and perfectly calibrated	
2	From the top of the screen move your gaze slowly down until it enters an Option	Option selection starts when your gaze enters the Option
3	Move your gaze just outside the Option again, over a neighboring Option	selection of the first Option continues, the neighboring Option does not start selecting
4	Move you gaze further towards the center o the neighboring Option	The first Option selection stops, the neighboring Option starts selecting.

Test case 203: Simultaneous gaze en touch input

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and and calibrated	
2	Look at an Option	The Option starts selecting.
3	Tap another Option without looking at it.	That Option is selected.

Test case 205: Letters Module

#	Action	Expected value
1	Prerequisite the system is booted and the Talk Module is shown.	
2	Select the Letters Module	The Letters Module is shown, displaying all letters of the alphabet evenly distributed over Category Options visible on the Page and a punctuation Category. In the header an empty text output area is shown
3	Select a letter Category	Options are displayed for each letter in the Category (possibly equally distributed over more Categories).
4	Select a letter.	The letter appears in the text output area in the header. The Letters Module contents are shown.
5	Select the Backspace Option in the punctuation Category	The letter in the output area is removed.
6	Repeat step 5	Nothing happens.
7	Repeat step 3-4 a number of times to spell a word.	
8	Select the space Option in the puntuation Category	A space is added to the text in the output area. The word you just typed is pronounced. The Letters Module contents are displayed.
9	Repeat step 7-8.	
10	Select the Back Option	The Talk Module contents are displayed. The text output area is removed from the header.
11	Select the Letters Module.	Te Letters Module contents are shown together with the words you typed in the header text output area.

Test case 208: Touch interface & no data collection

#	Action	Expected value
1	Prerequisite: system is booted	A dialog appears, asking to enable eye tracking
2	Tap "No"	On touch down, the Option receives focus. On touch up focus is lost and the dialog closes, the main window remains unaltered, A dialog appears asking to enable data collection.
3	Tap "No"	The dialog closes.
4	Tap "I understand"	The message is pronounced.
5	Go to the log folder and open all log files from this session	Only debug.log has content

Test case 234: Play text on enter

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and and perfectly calibrated. PlayTextOnEnter config option is set to true.	
2	Look at an Option	The selection pie appears and shortly after the Option's text is pronounced quickly and with average volume, finishing amply before the selection is complete.
3	Look at another Option, halfway trough pronounciation of the text, look at another Option	The moment the first Option loses focus pronounciation is interrupted.

Test case 235: Pain pointer

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and and perfectly calibrated.	
2	Navigate to and select the "I feel pain" Option	An image of the human body appears.
3	Look at the nose of the body.	The view starts zooming in towards the nose. When its size is bug enough it becomes focused. When the nose fills most of the Page zooming slows to a halt. A window appears reporting "My nose hurts" for a few seconds.
4	Look a the left hand. When it receives focus, look at the right border region of the Page.	The image start zooming out and panning towards the right. No body part is focused. The rate of zoom and translation is about the same as when zooming in.
5	When the back appears on screen keep looking at it.	As soon as the back is no longer in the right border region the view starts zooming in and the point you look at moves to the center of the view. When the back is big enough it receives focus and the whole bag moves to the center of the view.
6	Look at a part of the Page not close to the border where no body part is drawn.	Nothing happens.

Test suite 8: Gaze tracking

Test case 200: Server malfunction

#	Action	Expected value
1	Prerequisite: The system is booted, eye tracker connected.	
2	Open the task manager and kill the EyeTribe server process.	A dialog reports that a problem has occurred. After 10 seconds the system reboots.

Test case 199: Post calibration tracker disconnect

#	Action	Expected value
1	Prerequisite: system is booted, eye tracker is connected and calibrated	

2	Disconnect the eye tracker USB cable	A dialog reports the eye tracker was disconnected.
3	Reconnect the tracker USB cable	The dialog disappears.
4	Look at an Option	A filling pie shape appears over the Option

Test case 198: Restless calibration

#	Action	Expected value
1	Start calibration	
1.1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
1.2	Make sure your eyes are positioned in the middle of the tracker window at about 40cm from the tracker and the tracker clearly detects them	tracker window background becomes green, the little eyes are displayed uninteruptedly
1.3	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
2	Keep moving your eyes randomly around the screen, don't pause.	
3	Calibration failure recovery	
3.1		A dialog reports the calibration failed.
3.2	Tap the ok button	The dialog closes. The tracker window button reads "Calibrate"
3.3	Make sure your eyes are positioned in the middle of the tracker window at about 40 cm from the tracker and the tracker clearly detects them	The tracker window background becomes green, the little eyes are displayed uninterruptedly
3.4	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
3.5	Follow the red dot with your eyes.	This calibration window fades out and the tracker window reports a good/perfect calibration quality.
3.6	Look at an Option	A filling pie shape appears over the Option

Test case 197: Tracker disconnected during calibration

#	Action	Expected value
1	Start calibration	
1.1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
1.2	Make sure your eyes are positioned in the middle of the tracker window at about 40cm from the tracker and the tracker clearly detects them	tracker window background becomes green, the little eyes are displayed uninteruptedly
1.3	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
2	Follow the red dot with your eyes. While doing so, disconnect the tracker USB cable.	A dialog reports the eye tracker was disconnected.
3	Reconnect the USB cable	The dialog closes.
4	Calibration failure recovery	
4.1		A dialog reports the calibration failed.
4.2	Tap the ok button	The dialog closes. The tracker window button reads "Calibrate"

4.3	Make sure your eyes are positioned in the middle of the tracker window at about 40 cm from the tracker and the tracker clearly detects them.	The tracker window background becomes green, the little eyes are displayed uninterruptedly
4.4	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
4.5	Follow the red dot with your eyes.	This calibration window fades out and the tracker window reports a good/perfect calibration quality.
4.6	Look at an Option	A filling pie shape appears over the Option

Test case 196: Wrong points calibration

#	Action	Expected value
1	Start calibration	
1.1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
1.2	Make sure your eyes are positioned in the middle of the tracker window at about 40cm from the tracker and the tracker clearly detects them.	tracker window background becomes green, the little eyes are displayed uninteruptedly
1.3	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
2	As the calibration points pause there movement, look a random on screen point other than the red dot.	After nine pauses of the red dot. The calibration window fades out.
3	Calibration failure recovery	
3.1		A dialog reports the calibration failed.
3.2	Tap the ok button	The dialog closes. The tracker window button reads "Calibrate"
3.3	Make sure your eyes are positioned in the middle of the tracker window at about 40 cm from the tracker and the tracker clearly detects them.	The tracker window background becomes green, the little eyes are displayed uninterruptedly
3.4	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
3.5	Follow the red dot with your eyes.	This calibration window fades out and the tracker window reports a good/perfect calibration quality.
3.6	Look at an Option	A filling pie shape appears over the Option

Test case 192: Undetected calibration

#	Action	Expected value
1	Prerequisite: system is booted on tablet, eye tracker is detecting eyes	
2	Make sure your eyes are no longer visible to the tracker	The eye tracker window background becomes red, a no-eyes-are-detected sign is displayed.
3	Tap the calibration button	The calibration windows appears, a red dot moves around the screen pauzing 15 times. The calibration window fades out.
4	Calibration failure recovery	
4.1		A dialog reports the calibration failed.
4.2	Tap the ok button	The dialog closes. The tracker window button reads "Calibrate"

4.3	Make sure your eyes are positioned in the middle of the tracker window at about 40 cm from the tracker and the tracker clearly detects them.	The tracker window background becomes green, the little eyes are displayed uninterruptedly
4.4	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
4.5	Follow the red dot with your eyes.	This calibration window fades out and the tracker window reports a good/perfect calibration quality.
4.6	Look at an Option	A filling pie shape appears over the Option

Test case 191: Imperfect detection calibration

#	Action	Expected value
1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
2	Make sure your eyes are visible, but not clearly by the eye tracker by varying the angle of the tracker and position of your face	The tracker window background becomes orange, the little eyes are displayed most of the time, but they sometimes flicker.
3	Press the calibrate button and follow the moving dot with your eyes.	The calibration window appears displaying a moving dot. After the dot has paused at nine different screen positions, it might start revisiting some previous points. Eventually, the calibration window fades out.
4		The tracker window reports a poor or moderate calibration quality.
5	Look at an Option	A filling pie shape appears over the Option

Test case 190: Perfect/good calibration

#	Action	Expected value
1	Start calibration	
1.1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
1.2	Make sure your eyes are positioned in the middle of the tracker window at about 40cm from the tracker and the tracker clearly detects them.	tracker window background becomes green, the little eyes are displayed uninteruptedly
1.3	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
2	Follow the red dot with your eyes	The red dot moves around the screen, pausing at nine locations, then the calibration screen fades out again. A dialog prompts to collect data.
3	Do not look at an option	The No option has focus and a the dialog message is followed by number counting countdown.
4	Shortly look at No and then at no option.	The countdown ends and then resumes.
5	Look at No	The countdown ends. After a short delay, a pie filling shape appear over the No Option.
6	Shortly look at Yes	The focus moves from No to Yes. After a short delay, a pie filling shape appear over the Yes Option.
7	Do no look at an option	The focus moves from Yes to No and the countdown resumes where it left off.

8	Await the countdown.	The dialog disappears. The tracker window reports a good/perfect calibration quality and the tracker window button is labeled 'Recalibrate'.
9	Look at an Option of the dialog	After a short delay, a pie filling shape appear over the Option.

Test case 183: Trackerless startup

#	Action	Expected value
1	Precondition: tablet is off, eye tracker is not connected	
2	Boot the tablet	CommIC automatically starts and asks whether to enable eye tracking
3	Tap "OK"	The tracker window appears in the right corner, reporting "Connecting". After a short while a message will appear telling the eye tracker is not connected
4	Connect the eye tracker	The message disappears, the tracker window reports that the tracker is connected
5	Position your eyes 40-70 cm directly in front of the tracker.	Two little eyes appear on a green background within the eye tracker window

Test case 182: Basic startup

#	Action	Expected value
1	Prerequisite: Tablet is turned off, eye tracker is connected	
2	Boot the tablet	CommIC starts automatically, asking whether to enable eye tracking
3	Tap "OK"	The tracker window appears in the top right corner, reporting "Connecting...".
4	Quickly tap the tracker window button, while it still says connecting.	Nothing happens
5		Within a few seconds the tracker window reports that the eye tracker is connected. It's button reads "Calibrate". Alternatively, a message appears reporting that a problem has occured and the system will restart 10 seconds later.
6	If the system reports the eye tracker is connected, position your eyes 40 - 70 cm in front of the eye tracker	Two little eyes with a green background should appear in the tracker window.

Test case 195: Unfocused calibration

#	Action	Expected value
1	Start calibration	
1.1	Prerequisite: system booted on tablet, eye tracker is detecting eyes	
1.2	Make sure your eyes are positioned in the middle of the tracker window at about 40cm from the tracker and the tracker clearly detects them	tracker window background becomes green, the little eyes are displayed uninteruptedly
1.3	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.

2	Follow the red dot on the screen, but focus on a point closer to you so you see it blurred	The calibration succeeds, reporting a less than good quality. The test is complete. Alternatively, a message window appears reporting calibration failure.
3	Calibration failure recovery	
3.1		A dialog reports the calibration failed.
3.2	Tap the ok button	The dialog closes. The tracker window button reads "Calibrate"
3.3	Make sure your eyes are positioned in the middle of the tracker window at about 40 cm from the tracker and the tracker clearly detects them	The tracker window background becomes green, the little eyes are displayed uninterruptedly
3.4	Tap the calibrate button	The calibration window fades in, instructing you to follow the red marker with your eyes.
3.5	Follow the red dot with your eyes.	This calibration window fades out and the tracker window reports a good/perfect calibration quality.
3.6	Look at an Option	A filling pie shape appears over the Option

Appendix E. Final evaluation

Requirement	P	S	Specific feedback / test procedure/ future work
Medical personal and family wants patients to be (physically) able to communicate with them.			
The system can be controlled with the eyes.	5	✓	
The system enables patients to form normal sentences.	5	+	Most patients have insufficient atention span to complete a sentence. Possible improvements are word and letter prediction and support for AAC typing software Dasher.
The system can be controlled with the brain.	1	N	
The system can be controlled via a hardware switch	4	N	
The system filters accidental input from spasms.	2	N	
The system does not require good hearing to use.	3	✓	Tested by an unexperienced user performing standard tasks without system sound.
The system does not require good eyesight to use. (blurry, colorblind, limited eye movement)	5	+	Tested by a user who is colorblind, one with blurry vision and one with limited eye movement, performing the standard tasks. Eye tracking is not possible. Icons should be replaced by ones without thin lines and black as primary color for max. contrast. Patient C indicates that blurry vision is a common problem, but within operational distance from screen vision is quite ok. Priority of requirement should be increased. Gaze traking is not usable with limited eye movement, but screen can be positioned such that patients can see it and use the touch screen.
The system is usable without eyesight.	1	N	
The system can be controllable by the touchscreen	5	✓	
Patients want to orient themselves.			
The system allows patients to know the time.	4	N	This is important to give patients some information to "hold on to".
The system can tell patients their location.	4	N	This is important to give patients some information to "hold on to".
The system can show patients a live video of their surroundings.	1	N	
The system can show patients personal photos.	2	N	
The system can act as a mirror for the patient.	3	N	
The system can tell patients the reason for their hospital admission.	3	N	
The system can show patients their treatment history.	3	N	
Patients want to be distracted from their unpleasant situation.			
The system offers entertainment in the form of video / TV / music / (audio)books / games.	3	N	
The system offers remote communication through e-mail / messaging / phone calls.	2	N	
The system offers a possibility to browse the web.	2	N	
The system offers mental exercises.	1	N	
The system allows patients to easily make small talk.	3	N	
The system allows patients to watch/listen to video/audio/books provided by their family.	3	N	
The system offers use of facebook.	2	N	

Requirement	P	S	Specific feedback / test procedure/ future work
Patients want the software system to require little physical and mental effort to use.			
The system allows patients to easily communicate common messages.	5	✓	Medical personal and patient C judge the available common messages to be complete. Patient C indicates that ICU patients will rarely request movenment excersises, since movement is experienced as unpleasant. These choosables can be removed, if screen space needs to be freed.
The system offers easier access to functionality that patients personally use more often.	3	N	
The system preserves patients' configuration across multiple usage sessions.	4	N	
The patient UI can be localized to the patient's culture.	4	✓	
Using input devices requires little effort.	5	✓	Gaze input requires minimal effort according to patient A. According to patient C manual input requires enormous effort, continuously underestimated by patients themselves. Touch interface should not be used for these patients, even when they have sufficient motor control. CommIC and the manual are modified to make this clear to medical personal/family when prompting to use eye tracking.
The system can localize output messages, independent of patient UI.	4	N	
The system's speed and complexity matches patients' varying capabilities.	5	-	Complexity and speed can be modified, but this needs to be configured manually. It should happen automatically.
Using the system does not require memorization.	5	✓	Tested by user without prior experience, performing standard tasks.
The system is usable within a short attention span.	5	✓	Tested by user with small attention span, performing standard tasks.
The system is usable for lethargic/slow patients.	5	✓	Tested by lethargic user, performing standard tasks.
Using the system requires minimal problem solving capacity.	5	+	Tested by user with low mental clarity, performing standard tasks. All tasks were sucessfully performed, except for spelling a word. This is an indication that the interface is usable with limited mental clarity. However, we currently do not know with certainty the cause behind these observations. An interview with the patient once he is recovered should be conducted.
Using the system requires minimal information processing capacity.	5	+	
Using the system does not require experience with computers.	4	?	Tested by user without prior computer experience, perform standard tasks. No suitable test subject has yet been found!
Using the system does not require literacy.	4	✓	Tested by user, performing standard tasks while all characters have been replaced by questionmarks and with audio feedback enabled
The system can dynamically switch between output message localizations.	4	N	
The system allows the communication partner to input messages in the output message language and translate them to the patient's language.	3	N	
The calibration process should be easier for the patient.	5	✓	Tested by Patient A performing calibration. He no longer had trouble completing the process.
After choosing a leaf choosable, the user should not immediately be able to choose a choosable again to prevent accidental input	5	✓	
A repeat button should be added.	3	N	
Patients and medical personal want to be able to rely on the software system's functionality.			
The system notifies medical personal of sudden loss of connection of input or output.	3	-	Only present personal is notified, but also personal outside of the room should be notified.
The system is sturdy enough to withstand patients' delirious episodes.	5	✓	This is risky to test, but medical personal en the department of medical technique judge the hardware to be sturdy enough.

Requirement	P	S	Specific feedback / test procedure/ future work
If an error occurs, the system should restart.	4	✓	
Medical personal does not want the patient's wellbeing or healthcare process to be at risk or obstructed.			
The system is always save to use.	5	✓	
The system's hardware is shaped such that it can be easily cleaned.	5	✓	
The system's hardware should never be an obstacle for medical procedures.	5	✓	If the system is used unsupervised, it should be positioned on the side of the bed opposed to the room entrance for quick access to patient. This is noted in the manual.
The system forms a minimally obstruction to the patient's field of view, such that he can see medical personal that talks to him.	4	✓	
The system's sound volume is not bothersome or cause patient privacy concerns during visiting hours.	4	✓	
The system does not produce too much or accidental sound at night or while the patient is sleeping.	4	N	
Medical personal wants the system to be easy and quick to use.			
The system continues to function after it and the patient are re-positioned many times per day.	5	-	It is unclear when the system needs to be repositioned and recalibrated. Also this takes to much time and effort from personal and patient. The new workflow, designed in sprint 7 should solve these issues.
Turning on system power will start the software.	2	✓	
The system allows patients to leave a message for later.	4	N	
Communication through the system is quick.	5	✓	
Setup of the system for new patients is quick.	4	-	Tested by nurses setting up the system with real patients. It can take substantial time to successfully calibrate the system for patients with unclear eye detection. The new workflow, designed in sprint 7 should solve this. The time to clean the arm before use is a minor problem for some nurses (not all agree).
Medical personal wants to use the system with multiple different patients.			
The system can be setup for new patients or switched to an existing patient.	4	N	
The system is portable.	3	✓	
Physiotherapists want the patient to move optimally during their ICU stay.			
The system is usable lying flat, sitting straight up or semi-straight up.	5	✓	
The system is usable outside of the bed during therapy.	4	?	No test opportunity with a real patient. Gaze tracking is usable with healthy standing user.
The system does not obstruct movement exercises of any body part and continues to function afterward.	5	✓	The system cannot be in position while the user is handcycling, but we see no improvement opportunity for this.
The system is usable for patients lying on their side.	1	×	The temporary arm does not support correctly positioning the tablet. The new arm should support this angle of freedom.
The system is usable from a chair next to the bed.	4	✓	
Developers want to remotely monitor and remedy problems and collect usage data			
The system anonymously logs user input.	4	✓	
The calibration process should be logged to gain insight where it can be improved.	4	✓	

Requirement	P	S	Specific feedback / test procedure/ future work
Add prompt to ask patients if they argree to have their usagedata stored.	5	✓	
The system publishes its debug log for remote access.	4	✓	
The system can be remotely updated.	4	+	On remote updates, an untrusted software warning appears. This is confusing for medical personal and should be solved.

Legend	
✓	satisfied
+	mostly satisfied
-	somewhat satisfied
×	not satisfied
N	not yet implemented
?	unknow
P	requirement priority
S	stakeholder satisfaction

Appendix F. User manual

CommIC gebruiken

De patiënt kan:

- Een korte opdracht uitvoeren
- De ogen open goed te houden
- Uw vinger volgen (op 40cm afstand)

of

- Een tablet bedienen met de hand

Nee → **Probeer het later opnieuw**

De patiënt moet cognitief goed bij zijn, om CommIC te kunnen kalibreren met de ogen. Daarnaast moet deze kunnen focussen op de knoppen op het scherm.

Voor de touch bediening moet de patiënt zijn hand kunnen optillen en nauwkeurig manoeuvreren naar het scherm.

Ja ↓

Maak CommIC schoon

De tablet, eye-tracker, statief en kabels kunnen allemaal met **Isopropyl-alcohol 70%** schoongemaakt worden.
- Let op dat de tablet uit staat!
- Spuit de alcohol op een doekje en niet direct op de tablet of eye-tracker

Zet de tablet aan, CommIC start automatisch

De aan/uit-knop zit links bovenop de tablet. Om de tablet aan te zetten drukt u de knop **5-10 seconden** in. Om de tablet uit te zetten drukt u de knop kort in.

Oogbesturing gebruiken?

Ja ↓ Nee

CommIC kan gebruikt worden met of zonder oogbesturing. Dit is alleen geschikt voor patienten met genoeg energie en motorische controle. Als voor Ja wordt gekozen, kan zowel de oogbesturing als het touchscreen gebruikt worden. Bij nee alleen het touchscreen.

Positioneer CommIC:

- Zodat deze niet in de weg staat *
- De tablet recht voor de patiënt
- De ogen in het midden (zie afbeelding)
- De achtergrondkleur: groen

De achtergrondkleur geeft aan of de ogen zichtbaar zijn voor CommIC. Is deze **rood**, dan kan CommIC de ogen niet zien. Is de kleur **groen** dan kan CommIC de ogen wel zien.

De grootte van de ogen geeft aan hoe ver CommIC van de patiënt staat. Zijn de oogjes klein? Zet CommIC dan dichterbij! Hierdoor zal CommIC beter reageren op de ogen.

* Positioneer het systeem aan de zijde van het bed, tegenover de ingang van de kamer. Zo kun je snel bij de patiënt in geval van nood.

Kalibreer CommIC:

- Druk op de knop “Kalibreer”, een rode stip verschijnt
- Vraag de patiënt om de stip op het scherm te volgen met de ogen

Tijdens de kalibratie zal de stip verplaatsen naar alle 9 posities op het scherm. Hoe beter de patiënt naar de stip kijkt, hoe beter het systeem daarna werkt.

Als de kalibratie te slecht is (patiënt volgt de stip niet goed) dan zal CommIC aangeven “Kalibratie mislukt”. Herpositioneer het scherm of probeer het later nog een keer.

Anoniem data verzamelen?

Direct na de kalibratie zal CommIC vragen of er anoniem data verzameld mag worden. De patiënt kan met de oogbesturing of het touchscreen voor Ja of Nee kiezen. **Op verzoek kan verzamelde data ook na de tijd nog worden verwijderd.**

Ja ↓ Nee ↓

CommIC gebruiken!

CommIC kan nu worden gebruikt!
De patiënt kan nu een knop indrukken door hier enkele seconden naar te kijken.

Feedback geven?

Om CommIC te verbeteren ontvangen wij graag feedback, van iedereen! Tips, suggesties, problemen: wij horen het graag!
Het feedbackformulier kan worden ingevuld door: artsen, A(N)IOS’s, verpleegkundigen, fysiotherapeuten en familie van de patiënt. Ook de patiënt mag het feedback formulier invullen, als deze hiertoe in staat is.

Feedback kan ook telefonisch of per mail doorgegeven worden (zie hieronder). Ook kan het feedbackformulier op de website www.commic.nl ingevuld worden.

Problemen?

Bij problemen kunt u eerst proberen CommIC opnieuw op te starten door de tablet uit en opnieuw aan te zetten. Lost dit het probleem niet op? Neem dan contact op met:

Jeffrey Benistant
Tel: 063 063 60 90
Email: j.benistant@mst.nl
Site: www.commic.nl

Appendix G. Feedback form

Feedback formulier CommIC

Gelieve onderstaande vragen in te vullen na gebruik van CommIC. Bij voorbaat dank!

Datum:

U bent:

◯ Patiënt
◯ Arts ◯ Verpleegkundige ◯ Fysiotherapeut
◯ A(N)IOS ◯ Familie ◯ Anders:.....

Hoe goed werkte CommIC tijdens deze sessie?

Uitstekend	Goed	Voldoende	Matig	Slecht
◯	◯	◯	◯	◯

Met welk doel heeft u CommIC ingezet/gebruikt?

Heeft u dit doel bereikt? (Waarom niet?)

Welke aspecten van de communicatie heeft u als frustrerend ervaren? In welke mate?

Zijn er zaken waarover u zou wilde communiceren, maar dat niet deed omdat het te onpraktisch of onmogelijk was? Welke?

Kunt u verdere functionaliteit bedenken voor het systeem die u zou kunnen helpen? Welke?

Kunt u mogelijk problemen bedenken die kunnen optreden bij gebruik van het systeem? Welke?

Voor het geval dat wij aanvullende vragen hebben, verzoeken wij u uw email achter te laten.

E-mail: