Vesna Krnjic

Integrating User-Centered Design In A Large Scale, Agile, Free Open Source Project

Master's Thesis

Graz University of Technology

Institute for Softwaretechnology
Head: Univ.-Prof.Dipl-Ing.Dr.techn. Wolfgang Slany

Supervisor: Univ.-Prof.Dipl-Ing.Dr.techn. Wolfgang Slany

Graz, July 2014

Abstract

In the last two decades, software development has changed enormously. Choosing the right development method is becoming even more crucial for the success of a project. Developers have to deal with frequently changing requirements, short releases, fast development and short times to market. Deploying classical software development approaches like the waterfall model is highly risky, often more costly and generally less efficient than newer approaches like Agile Software Development (ASD).

Usability is another key success factor of software and is one that heavily influences user acceptance. A variety of usability methods can be applied to improve the usability of a software. These include the well known User-Centered Design (UCD) process where the user stands in the middle of the development process.

Combining software development approaches like ASD together with UCD can effectively increase the success of software. ASD, however, does not provide guidelines on how to integrate usability testing in the software development process and so best practices for the integration of the two methods are discussed in the literature. In addition, agile practices are not applied exactly the same way in all projects, so the integration of UCD in practice is still challenging. Only a little is known about the combination of Free Open Source Software (FOSS), ASD and UCD. This thesis will also deal with that problem.

The first part of this thesis describes ASD, FOSS and UCD processes and the best practices recommended in the literature for combining the approaches to improve the success of a project. The second part of the thesis addresses practical work, namely the integration of UCD in Catrobat, a large-scale, ASD, FOSS project, especially in the Pocket Paint, Pocket Code and Pocket Codes' Formula Editor subprojects. The thesis describes an agile UCD cycle, the used usability methods, experienced problems and possible solutions, which were found during the process, as well as open issues. At the end of the UCD cycle the team conducted a usability test in order to evaluate the usability of the resulting software. The test method, test environment, test users and main results are discussed as well. To evaluate the UCD process used, the agile steps recommended in the literature were compared to steps integrated in Catrobats' UCD process.

Keywords Usability, User-Centered Design, Agile Software Development, Free Open Source Software

Kurzfassung

In den letzten zwei Jahrzehnten hat sich die Softwareentwicklung enorm verändert. Für den Erfolg eines Softwareprojekts wird die Wahl der passenden Softwareentwicklungsmethode immer entscheidender. Softwareentwickler müssen mit sehr schneller Entwicklung, sich häufig ändernden Anforderungen, kurzen Release Phasen und kurzem "Time-to-Market" fertig werden. Der Einsatz von klassischen Softwareentwicklungsansätzen wie zum Beispiel dem Wasserfallmodell ist äußerst riskant, sehr oft teurer und nicht so effizient wie etwa der Einsatz von neueren Ansätze wie Agiler Softwareentwicklung (AS).

Benutzerfreundlichkeit ist ein weiterer wichtiger Aspekt einer Software und beeinflusst stark die Benutzerakzeptanz. Eine Vielzahl an Usability Methoden kann eingesetzt werden, um die Benutzerfreundlichkeit einer Software zu verbessern. Dazu gehört auch der bekannte User-Center Design (UCD) Prozess, wo der Benutzer im Mittelpunkt des Entwicklungsprozesses steht.

Die Kombination von Softwareentwicklungsansätzen wie AS mit UCD kann zur wirkungsvollen Steigerung des Erfolgs einer Software beitragen. AS schreibt jedoch keine Richtlinien für die Integration von Usability in den Softwareentwicklungsprozess vor. In der Literatur werden bereits Modelle, die die Integration dieser zwei Methoden beschreiben und die sich in der Praxis bewährt haben, diskutiert. Agile Praktiken werden nicht in jedem Projekt gleich eingesetzt, deswegen ist die Integration von UCD in der Praxis noch immer herausfordernd. Über die Kombination von AS, Free Open Source Software (FOSS) und UCD ist nicht viel bekannt. Diese Arbeit geht auch auf dieses Problem ein.

Der erste Teil dieser Arbeit beschäftigt sich mit AS, UCD und FOSS Prozessen und der in der Literatur empfohlenen Kombination dieser Methoden, um zur Steigerung des Erfolgs eines Projekts zu führen. Der zweite Teil beschreibt die praktische Arbeit, konkret die Integration von UCD in Catrobat, einem agilen sehr umfangreichen FOSS Projekt und dabei besonders die Integration der oben erwähnten Methoden bei den Teilprojekten Pocket Paint, Pocket Code und Pocket Codes Formula Editor. Die Arbeit beschreibt einen agilen UCD Zyklus, ausgewählte und an die agile Kultur angepasste Usability Methoden, überwundene Probleme und mögliche Lösungsansätze, sowie noch offene Probleme. Die Benutzerfreundlichkeit der daraus resultierenden Software wurde mittels eines Usabilty Test evaluiert. Testmethode, Testumgebung, Testpersonen und die wichtigsten Ergebnisse wurden ebenfalls beschrieben. Um den eingeführten UCD Prozess zu evaluieren, werden die in der Literatur empfohlenen bewährten Praktiken mit den im Catrobat Projekt eingeführten verglichen.

Schlüsselwörter Benutzerfreundlichkeit, User-Centered Design, Agile Softwareentwicklung, Freie und offene Softwareentwicklung

Statutory Declaration

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

Graz, ____________________ ____________________

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eidesstatt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ____________________ ____________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgements

My special thanks goes to my family for their support during my studies especially Florian and our son Oliver.

I would like to express my very great appreciation to Professor Wolfgang Slany for his enthusiastic encouragement and for giving me a tremendous amount of freedom during the work on the Catrobat project.

I am very grateful for the assistance given by the Catrobat members particularly the incredible UX team.

This work would not be possible without the participation of the test users, thank you.

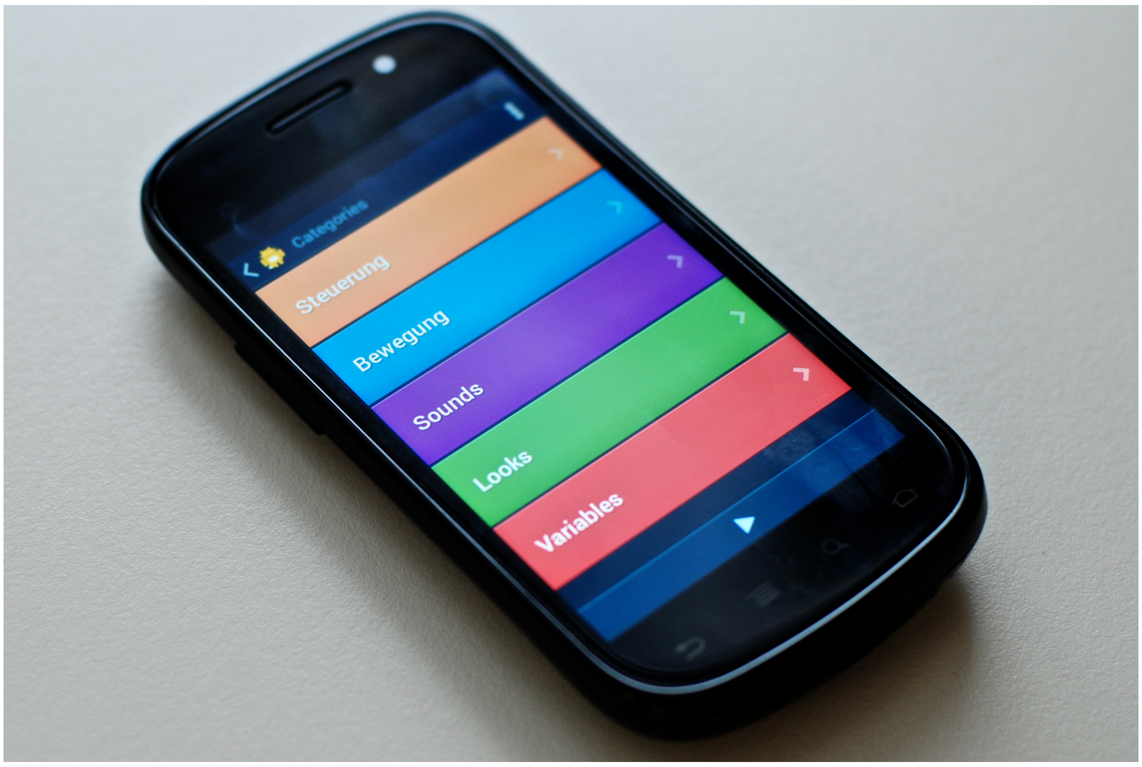
At last I want to acknowledge the help provided by Annemarie Harzl in reviewing this thesis and all other people who contributed their time and efforts to complete this work.

Contents

List of Figures

1. Introduction

"The beginning is the most important part of the work." (Plato)



In addition to the well known waterfall software model, different approaches for software development have been introduced in the last few years. In particular agile software methods have been proposed as a solution to deal with challenges like fast development, short time to market, constantly changing requirements or short release cycles. Methods such as Extreme Programming (XP), Scrum, Lean Development or Adaptive Software Development use the principles of the approach laid down in the Manifesto for Agile Software Development (ASD). All those methods share common principles like short iterations, minimal documentation, face-to-face communication and collaboration, continuous integration and so on. Agile approaches promise to produce more useful and reliable software, quicker than traditional software development methods.

Free Open Source Software (FOSS) is another movement that has been given much attention in the last years. Open source projects receive the collaboration of a large number of geographically distant developers that do not share an organisational structure. This could indicate that such projects are inferior to the use of agile methods since some basic values like face-to-face communication are not applicable. Nevertheless, FOSS communities share common principles with ASD like being ready for changes, delivering real features, working with continuous feedback and respecting collaborations. Both movements have brought great benefit to the software community. This begs the question, if the methods could be used together to improve the development process and productivity. In Chapter 3 a short overview about the differences and similarities of this two approaches will be given.

The Catrobat project, the case study presented in Chapter 6 is a large scale project which uses agile methods and develops Free Open Source Software. Catrobat team members are unpaid volunteers, most of them situated in Graz, Austria and some are even geographically distributed. Such team constitution can be rather found in FOSS teams than in agile.

Nowadays, the usage of software has shifted from technical experts to ordinary users, even young children. It cannot be presumed that the users have basic knowledge of the software so the software must be self-explanatory. Besides the appropriate software development method, usability has became an important aspect for the success of a software product. User-Centered Design (UCD) is a process which gives significant attention to the needs, wishes and limits of the end-users of a product at each stage of development, but it is not a process model for software development. Therefore, UCD needs to be combined with a software development methodology.

Combining ASD and UCD can effectively increase the success of software but agile methods do not explicitly include principles and practices for usability and user experience requirements. ASD and UCD are both iterative approaches, having sustainable differences but also similarities. While UCD concentrates on end-users and their needs and requirements in software design, agile software development focuses on software development and project management. The question to be answered is how the methods could be merged into a powerful development approach. Actually, there are no clear principles or guidelines for practitioners to achieve successful integration of these two methods. It is necessary to develop individual integration strategies for every project. The most important requirement is that usability researchers and designers understand the agile culture and in almost the same manner the agile developers learn about usability and design technologies.

The first part of this thesis deals with the question how UCD and ASD could be used together in a software development process and what agile UCD techniques are that have proven to be effective.

The first goal of this thesis was:

- To discuss ASD, FOSS and UCD, and to find out how the methods can be used together to improve the quality of a software product.

The second part of the theses describes the challenges and issues that were faced during the integration of UCD in the Catrobat project and the process followed. Furthermore, the thesis presents the most important results of usability studies conducted and gives recommendations for improvements of the project. Because Catrobat consists of more than 26 sub-projects and it was not possible to support all sub-teams, we started the integration of UCD with the Pocket Paint sub-project. Subsequently, we introduced UCD in further Catrobat projects.

The second goal was:

- To integrate UCD in the Catrobat project, especially the sub-projects Pocket Paint and Pocket Code.
- To improve the usability and find out the main issues of the Pocket Paint, Pocket Code and Pocket Codes' Formula Editor sub-projects.

This work is restricted to three sub-projects: Pocket Paint, Pocket Code and the Formula Editor contained in Pocket Code. Pocket Paint is an image editor app associated with Pocket Code that allows image creation and manipulation. Pocket Code is a free, open source mobile visual programming system for the Catrobat programming language which allows users to create and execute Catrobat programs on Android, iOS, Windows Phone 8 smartphones as well as on HTML5 capable mobile browsers. Pocket Code contains the Formula Editor which enables textual formula input and representation like the familiar pocket calculator.

The thesis is structured as follows. Chapter 2 presents a summary of ASD and describes Extreme Programming and Scrum, two most known agile methods. Subsequently in Chapter 3, FOSS is presented and a brief overview is given of the differences and similarities between ASD and FOSS methods. Chapter 4 describes the conventional UCD process and some selected methods. Following on from that Chapter 4 also highlights the differences between usability for mobile phones and large displays. Chapter 5 addresses the relationship between UCD and ASD and describes the best practices combining these two methods.

The second part of this thesis describes the experiences gained and lessons learned by integrating UCD in the Catrobat project. Agile processes have not been adopted in practice exactly the same way in every project. Consequently, Chapter 6 gives an overview about the Catrobat project itself and the agile practices used. Chapter 7 describes the initial usability test that was conducted before we applied UCD methods to the project. Afterwards, in Chapter 8 the process for the integration of UCD in Catrobat sub-projects Pocket Paint, Pocket Code and Formula Editor are sketched. To evaluate the newly designed and implemented sub-projects we conducted a usability evaluation the main results of which are presented in Chapter 9. Finally in Chapter 10, conclusions are drawn.

2. Agile Software Development

"First do it, then do it right, then do it fast." (Anonymous)

Software engineering has dramatically changed in the last few years. In todays modern software development processes it is crucial to choose a convenient development method. A short time to market is a key factor for product success. The requirements change frequently, so traditional methods like the waterfall model often do not work. Agile methodology has been a widely known trend in the software development area for more than ten years. The agile approach focuses not only on programming but also on subjects like team work and project management. Compared to conventional development methods where the development cycles take months or years agile methods concentrate on short development iterations with continuous testing and integration and flexible release dates. [2]

In 2001, 17 software developers wrote the Manifesto for Agile Software Development (ASD) [3] to define the approach of ASD. The Manifesto stated that when developing software, it is preferable to value:

- *Individuals and interactions over tools and processes,*
- *working software over comprehensive documentation,*
- *customer collaboration over contract negotiation* and
- *responding to change over following a plan* [3].

The following twelve principles of ASD were defined [3] :

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *Business people and developers must work together daily throughout the project.*
- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress.*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- *Continuous attention to technical excellence and good design enhances agility.*

- *Simplicity–the art of maximising the amount of work not done–is essential.*
- *The best architectures, requirements, and designs emerge from self-organising teams.*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.*

2.1. Different Methodologies

Several agile methodologies share common principles and practices, so different approaches for ASD are currently in use. Table 2.1 shows the most important agile methods.

Scrum
eXtreme Programming (XP)
Adaptive Software Development (ASD)
Lean Development (LD)
Crystal
Dynamic Systems Development Method (DSDM)
Rational Unified Process (RUP)
Team Software Process (TSP)
Feature Driven Development (FDD)
Capability Maturity Model Integration (CMMI)
Capability Maturity Model for Software (SW-CMM)
Personal Software Process (PSS)
Cleanroom

Table 2.1.: Most important ASD Methodologies

All these methods share common core elements like *"short, well-defined iterations that deliver real users value; tight team processes for efficient development, minimal documentation of specifications; and continual feedback from stakeholders to validate progress"* [2]. The main factors that enhance communication in agile development are the planning game, real customer involvement, stand-up meetings, simple design, collective code ownership, coding standards, the use of whiteboards, and informative workspace [4]. In the following two Subsections 2.1.1 and 2.1.2 Scrum and Extreme Programming will be described, two of the far most well known and widely used agile methods.

2.1.1. SCRUM

In 1995 Ken Schwaber and Jeff Sutherland introduced Scrum [5], a framework that can be used by teams to work together to develop a complex product. The Scrum framework is made up of teams and their associated roles, events, tools and rules. Software developed in Scrum occurs in small pieces iteratively and incrementally. A *Scrum team* consists of a *product owner*, the *developer team* and a *scrum master*. Scrum teams organise their work in teams, rather than being led by someone outside the team. They have all abilities required to finish the work without depending on someone outside the team. All team members are equally responsible for producing results.

Product Owner: The product owner is the only person, responsible for setting the priorities of the work items, maximising the value of the product and the work of the developer team. The product owner also represents the customer and is responsible for finding out what the stakeholders and the users actually need and communicating the needs to the team.

Development Team: The *development team* consists of self-organised professionals having all the skills as a team necessary to create a product. All development team members are called developers irrespective of the work being performed by the person. Scrum does not recognise sub-teams in the development team. Some team members may have specialised skills, but the team accepts the responsibility for the project as a whole.

Scrum Master: Each project team has a scrum master who manages the processes and runs the daily stand up meetings. The scrum master is for example responsible for coaching the development team in self-organisation, and helping them to create high value products.

The Sprint: Development is organised into sprints which are usually one month long or even shorter during which a "useable and potentially releasable product increment is created" [5]. Every sprint starts with the selection of user stories to implement during the sprint. It is important to choose as many user stories as can be completed by the end of the sprint. No changes are made during the sprint that would affect the sprint goal. A sprint can be canceled by the product owner if for example the sprint goal no longer remains valid.

The collaborative scrum team plans a sprint at the sprint planning session. Such sprint planning is limited to a maximum period of eight hours. The team decides which functionality will be developed during the sprint. Product backlog serves as the input. The development team must decide how many items are selected from the product backlog for the next sprint. After choosing the items the sprint goal is defined by the scrum team.

Daily Scrum: A 15 minute time slot is used for daily scrums at the same time and place. The development team synchronises and creates the plan for the next 24 hours, inspects the work done since the last daily scrum and goes through the progress toward the sprint goal.

Sprint Review: After each sprint end a sprint review is held to monitor the success and to adapt the product backlog if needed. This is an informal meeting where the scrum team and stakeholders review the work done during the sprint. The goal of the meeting is to elicit feedback and to foster collaboration. The result of a sprint review should be an updated product backlog with product backlog items for the next sprint.

Product Backlog: Product backlog is an incomplete, dynamic, sorted pool of product requirements. The product owner is responsible for the content and availability of it. The product backlog evolves together with the product and the environment used and can be updated any time by the product owner. Higher ordered items are specified in greater detail than lower ordered ones.

Sprint Backlog: Sprint backlog consists of items from the product backlog selected for the next sprint and can only be changed by the development team during a sprint. It also contains a plan for delivering the product increment, the sum of all the product backlog items completed during a sprint, and realising the goal.

2.1.1.1. How Does SCRUM Work?

The fundamental process of Scrum is simple and it consists of three primary tasks.

1. Product owners determine what needs to be built in the next 30 days or less.
2. Development team builds what is needed in 30 days or less, and then demonstrate what they have built. Based on this demonstration, the product owner determines what to build next.
3. Scrum masters ensure this process happens as smoothly as possible, and continually help improve the process, the team and product being created.

2.1.2. Extreme Programming

As defined by [6] Extreme Programming (XP) *"is a discipline of software development based on values of simplicity, communication, feedback, courage, and respect. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation."* XP was introduced in 1999 by Kent Beck [7] and is one of the oldest agile methods. The

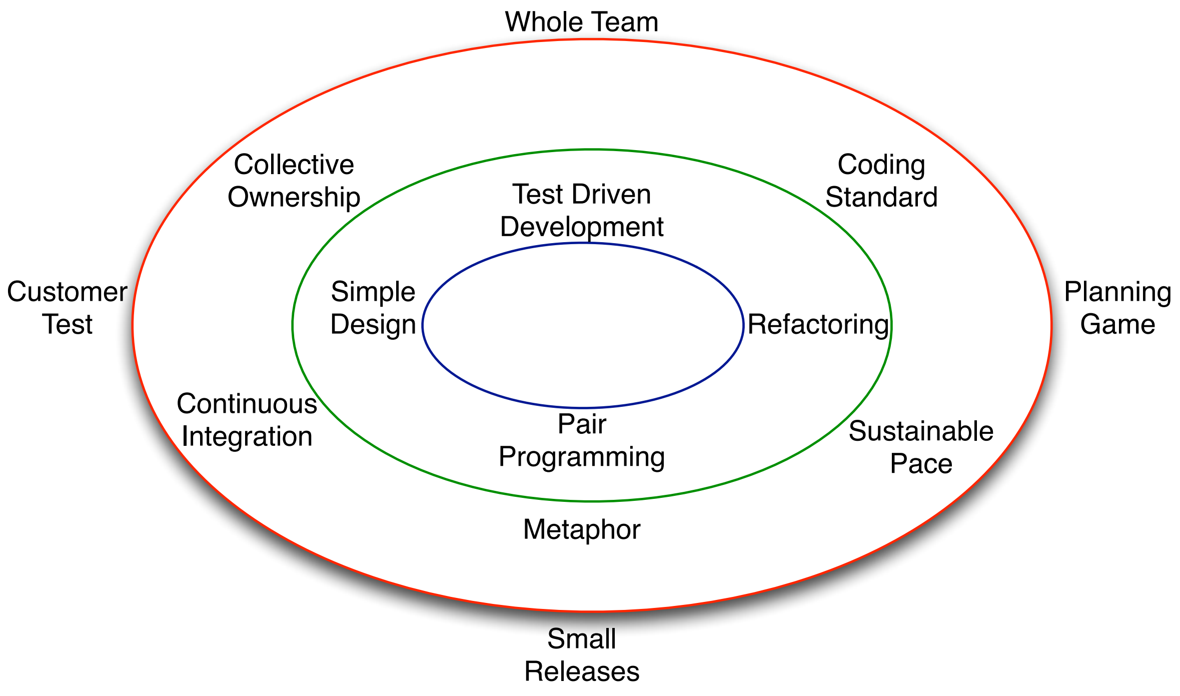


Figure 2.1.: Twelve initial practices of Extreme Programming described by [6].

methodology is based on twelve initial practices (see in figure 2.1) and four values described in Subsection 2.1.2.2 which have been extended in various ways since XP's introduction.

2.1.2.1. Core Practices

The following section describes the twelve core practices of XP shown in figure 2.1 as described by [6] and [2].

Whole Team: The XP team consists of a manager, a customer and developers. All members are equal partners and contribute to the project. The customer should be a real, experienced, end-user which provides the requirement, sets the priorities, and directs the project. The XP customer is empowered to decide what features are useful in the product. The manager provides resources, handles external communication and coordinates activities. Some of the teams include testers, who help the customer to define the customer acceptance tests. XP teams consist of contributors with special skills instead of specialists.

Planning Game: The XP planning game consists of two planning steps.

- Predicting what will be accomplished by the due date.

- Determining what to do next.

Release Planning:
In the release planning game, the XP customer presents user story cards describing the features that are needed in the next release of the product. Programmers then estimate their cost. The output of the release planning is an estimate of the release date, based on the total implementation time of the selected stories and the team's capacity.
Iteration Planning:
Each iteration starts with an iteration planning session whereby the team selects the stories to be implemented during an iteration. The team decides how many stories to select based on their capacity. For example if they can implement 50 points in an iteration, once they have reached the amount of 50 points, no more stories can be selected for that iteration. Usually XP teams program software in two-week iterations. They deliver running and tested software at the end of each iteration. After the selection, the developers break down the stories into independent tasks that can be done by different team members.

Customer Tests: To prove the implementation of all the desired features, the XP customer defines automated acceptance tests. The team creates the tests and uses them to prove that the feature is implemented correctly.

Small Releases: At the end of every iteration the XP team delivers running and tested software with the business value chosen by the customer. XP teams release to their end-users frequently. The customer represents the end-users. Some teams deploy new software into production every day.

Simple Design: XP teams always try to keep the design as simple as possible but also appropriate for the current functionality of the software. Design steps like quick design sessions and design revisions are held throughout the entire project.

Pair Programming: Two programmers work together on a single computer with the same code. Pair programming ensures that all production code is reviewed by at least one other programmer which results in better design, better testing and better code. Another advantage of pairing is the automatic knowledge exchange throughout the team. Through pairing, programmers learn specialised knowledge from other programmers and improve their skills thereby becoming more valuable to the company.

Test-Driven Development: First of all XP programmers write a test and then they implement the desired functionality. So, nearly 100 percent test coverage can be achieved. Before the implementation the test must fail. After the implementation the developers run all the tests again, but this time they must run successfully. Every time a programmer releases some piece of code all tests must run correctly. The developers get immediate feedback on how the software is doing.

Design Improvement: To deliver the business value in every iteration well-designed software is crucial. To accomplish this aim XP teams use a process of continuous design improvement called *refactoring*. During the refactoring process the XP team removes redundancy, eliminates unused functionality, and changes obsolete designs. With this procedure the team keeps the code clean and simple. According to [8] "Refactoring throughout the entire project life cycle saves time and increases quality."

Continuous Integration: XP developers should integrate and commit code to the project repository at least once a day. Every team member should work with the latest version. Continuous integration discovers compatibility problems early and avoids the need for many weeks of system integration at the end of the project.

Collective Code Ownership: Collective Ownership means that every developer is allowed to improve any code of the project. Code improvement include adding new functionality, fixing problems, improving design or refactoring the code. As described by [8] collective code ownership increases code quality and reduces defects.

Coding Standard: All XP contributors follow a common coding standard established by the team which keeps the code consistent and easy to read and easy to refactor. It is important that all the code is consistent to encourage collective code ownership.

Metaphor: The metaphor describes the program in a simple way. It is the common vision developed by the XP team of how the product works. Explaining the program design to new developers should be possible without looking at huge documentation.

Sustainable Pace: XP teams create completed, tested, integrated, production ready software each iteration. If it is not possible to finish all of the required features by iteration end the team have an iteration planning meeting and reschedule the iteration. XP teams usually should not work overtime. It is very important to find the team's velocity that will remain consistent for the entire project.

2.1.2.2. The Values Of Extreme Programming

Extreme Programming is a software development method based on values of simplicity, communication, feedback and courage. In the following section I will describe the four values in more detail.

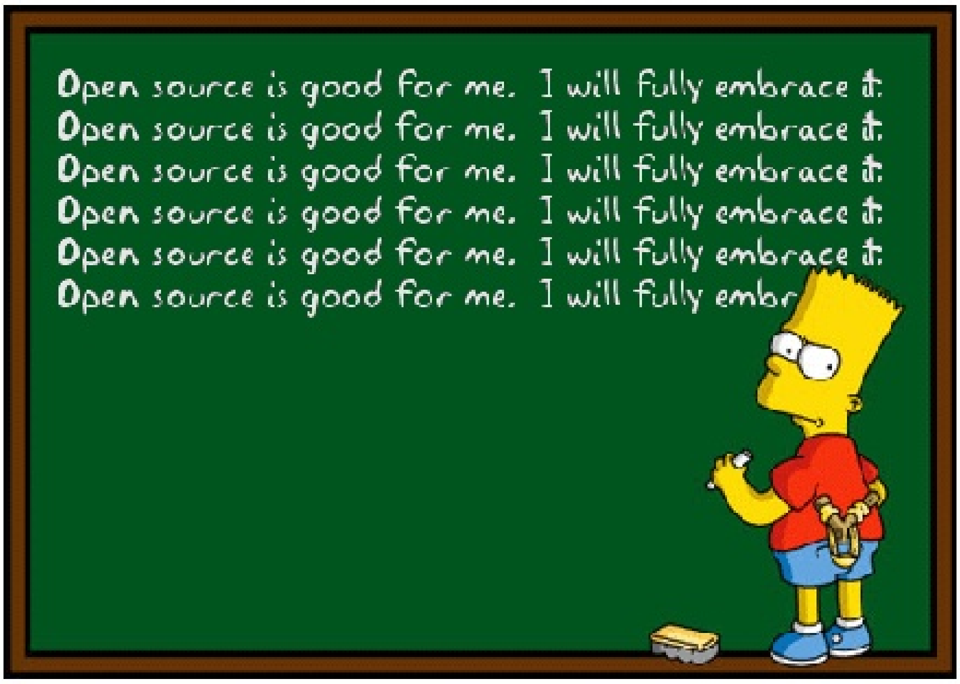
Simplicity: Develop the simplest product that meets the customer's needs.

Communication: Most projects fail because of poor communication. The XP method prefers daily face to face communication. The whole team works together on everything from requirements to code and creates the best solution to the problem that they can together.

Feedback: Early feedback from customer and developers is helpful for delivering desired and working software. The software is demonstrated early and often to get feedback and make any necessary changes. The whole process is adapted to the software and not the other way around.

Courage: Telling the truth about progress and estimate is crucial. The whole team works together and adapts to changes whenever they occur.

3. Free Open Source Software And Agile Software Development


Open source is good for me. I will fully embrace it.
Open source is good for me. I will fully embrace it.
Open source is good for me. I will fully embrace it.
Open source is good for me. I will fully embrace it.
Open source is good for me. I will fully embrace it.
Open source is good for me. I will fully embr

Source: http://www.flickr.com/photos/mywebguy/3489767568/

In the last few years ASD as well as Free Open Source Software (FOSS) have been two important movements in the software industry. In contrast to studies for each of the fields, there is hardly any research about the combination of the two areas. The authors of [9] tried to identify success and failure in the application of agile methods in teams with open source background, structure and characteristics. Agile and FOSS are widespread and well discussed software paradigms [3, 8, 7, 6, 5], [10, 11, 12]. According to the literature there are significant differences between them, but also many commonalities. Most software development methodologies are designed for commercial teams and products: a paid group of developers working on commercial products with corporate reporting, externally set milestones, and penalties for not completing on time. Open Source (OS) projects are usually completely different: unpaid volunteer developers working on products that they enjoy with minimal management, highly flexible milestones, and no penalties for not participating. People involved in OSS usually are geographically distant and do not share any organisational structure.

3.1. Free Open Source Software

According to the work of Wheeler[1] Open Source Software or Free Software programs *" are programs whose licenses give users the freedom to run the program for any purpose, to study and modify the program, and to redistribute copies of either the original or modified program (without having to pay royalties to previous developers)."* Free software is licensed with the GNU General Public License (GPL)[2], whereas OSS can use the GNU Library or General Public License (LGLP) but also some other licenses like MIT or the Common Development and Distribution License.[3]

GNU defines the difference between Open source software and Free software as follows:

> *"The fundamental difference between the two movements is in their values, their ways of looking at the world. For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, "Open source is a development methodology; free software is a social movement." For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution." [11]*

FOSS developers are very often also the end-users of the software they develop. Usually the core source code of a FOSS project is written by a small group of core developers who set the direction of the software. Software developers participate in FOSS projects and spent a lot of time programming, often without payment. Reasons why software developers contribute to such software could be for example personal reward, improvement of their own skills, fun, self-determination, peer recognition, project affiliation or identification, self-promotion and so on.

Besides the effort and the time they spend on the project they also often provide their personal computer resources and access to the Internet or host personal web sites. Furthermore software developers are free to choose their own developer methods and tools. Authors of [13] explain the reason why the developers and contributors participate in such FOSS projects as follows:

> *"It seems that one such answer must account for the belief in the freedom to access, study, modify, redistribute and share the evolving results from a FOSS development project."*

Even FOSS projects have to deal with issues like project control, cooperation and coordination between the contributors. Software version control (SVN) like CVS, Subversion or GIT are often used to coordinate and synchronise the project as well as for control over project software releases of the project. Usually, CVS repository administrators decide which parts will be checked-in and who is allowed to

[1]http://www.dwheeler.com/oss_fs_why.html
[2]https://www.gnu.org/licenses/gpl-3.0.en.html
[3]http://opensource.org/licenses

check-in. FOSS developers bring skill and knowledge in a special domain which is required to develop such complex software. As the practice shows, FOSS projects like Linux kernel[4] or GIT[5] produce high quality software which is used by an enormous number of end-users. The authors of [13] recommended FOSS development as an alternative strategy for how to develop large software projects.

3.2. Free Open Source Software And Agile Software Development

According to the literature [9] FOSS and ASD have similarities as well as differences in their processes.

The most important common team values are:

- Shared goals
- Shared understanding
- Shared beliefs
- Shared values on how to reach the goals
- Self-organisation with focus on collective code and product ownership
- Standardisation of code
- Shared management integrated in the programming activity
- Knowledge transformation supported by tools and collective reflection
- Trust into the team members

The main differences in their processes are shown in Table 3.1:

FOSS	ASD
Large number of volunteers, globally distributed, not part of the organisation	Small number of developers, usually co-located, integrated in organisations
No patterns for planning, testing and design	Definition for planning, testing and design
Collaboration asynchronous through Information and Communication Technology (ICT) Media	collaboration synchronous, co-location
Decision is a quorum consensus	Management

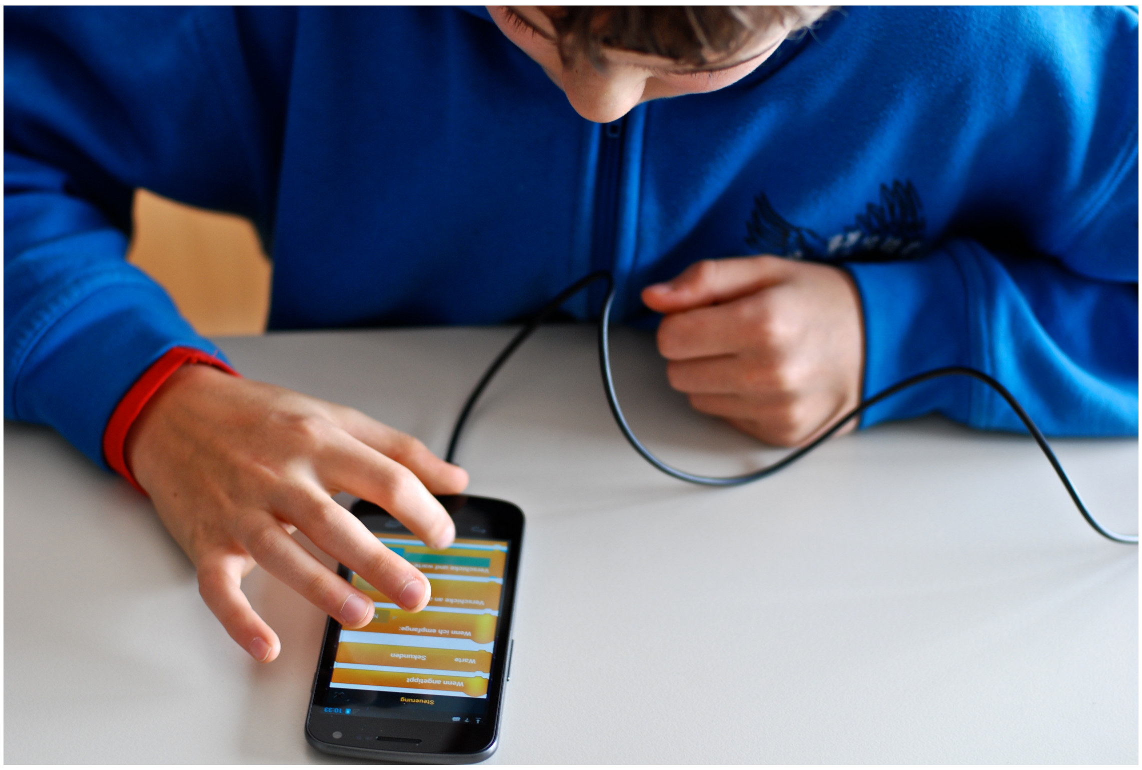
Table 3.1.: Differences between Free Open Source Software and Agile Software Development Process

[4] www.linux.org

[5] www.git-scm.com

4. User-Centered Design

"Because it's nearly impossible to design a user interface right the first time, we need to test." (Jakob Nielsen)



User-Centered Design (UCD) is a generic term to describe a variety of methods where users are involved in an iterative process and somehow influence a design of a product. Actually, UCD is not only a diversity of methods it is also a philosophy that puts the user and his or her needs, wishes and limits in the centre of the development process (see Figure 4.1).

In the literature terms like Usability Engineering, Human-Centered Design or User-Centered System Development (UCSD) are used to describe pretty much the same methodologies. In this thesis I will use UCD as a generic term to cover all these methodologies.

As far back as 1985 Gould and Lewis described in their paper *Designing for Usability: Key Principles and What Designer Think* [14] the basic principles of good user interface design and methods for achieving a high level of usability. The authors present three principles of design: early focus on users, empirical measurement using prototypes and iterative design. In 1992 Nielsen [15] modified and extended the version of Gould and Lewis. Most of today's UCD processes are based on those principles. Table 4.1 shows the steps of Nielsen's usability engineering model.

0. Consider the large context
1. Know the user
Individual user characteristics
The user's current task
Evolution of the user
2. Competitive analysis
3. Setting usability goals
4. Participatory design
5. Coordinated design of the total interface
Standards
Product identity
6. Guidelines and heuristic analysis
7. Protopyping

Table 4.1.: Elements of the usability engineering model, Jakob Nielsen (1992) [15]

There are many ways in which users are involved in UCD however the most important aspect is that users are involved in the iterative design process and evaluation. User influence can take place at different design stages. For example, users could be involved in requirements gathering, usability testing or even methods like *Design Studio* [16] where users have significant effect on the design by being part of the design team throughout the design process.

4.1. UCD Phases

UCD is an iterative process with multiple stages of product development. Usually it consists of four main phases: *Analysis, Design Phase, Implementation Phase and Validation Phase.* The following section describes the UCD activities in the development process as proposed by the Usability Professionals' Association (UPA) [1].

Usability testing is included in each phase, so providing good user experience is a continuous process.

[1]http://www.usabilityprofessionals.org/

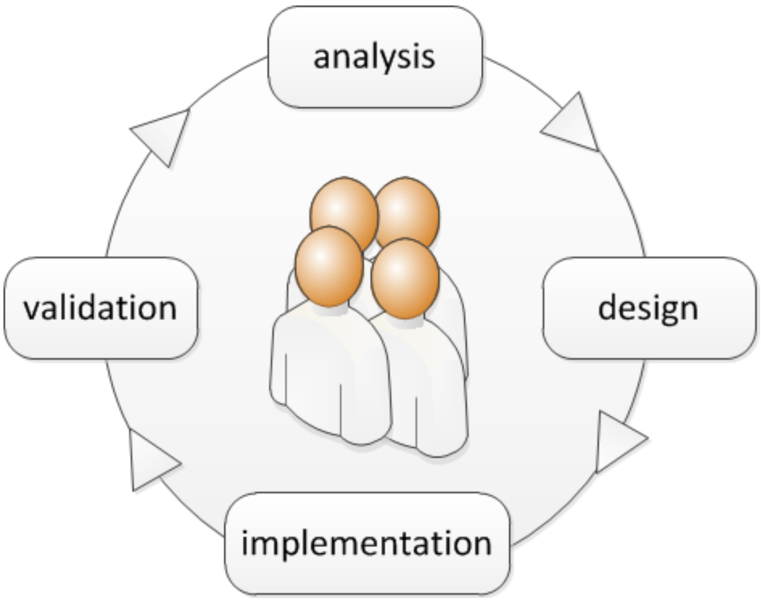


Figure 4.1.: User-Centered Design phases as defined by the standard ISO 9241-210 consists of four phases and the user is in the middle of the process.

Analysis Phase: Before the design is developed a comprehensive analysis should be conducted to ensure project and user requirements as proposed by the User Experience Professionals Association (UXPA) [17].

- *Meet with key stakeholders to set vision*
- *Include usability tasks in the project plan*
- *Assemble a multidisciplinary team to ensure complete expertise*
- *Develop usability goals and objectives*
- *Conduct field studies*
- *Look at competitive products*
- *Create user profiles*
- *Develop a task analysis*
- *Document user scenarios*
- *Document user performance requirements*

Design Phase: This phase serves for creation and analysis of prototypes where the design meets all project and user requirements. Again the UXPA [17] recommended the following steps:

- *Begin to brainstorm design concepts and metaphors*
- *Develop screen flow and navigation model*
- *Do walkthroughs of design concepts*
- *Begin design with paper and pencil*
- *Create low-fidelity prototypes*
- *Conduct usability testing on low-fidelity prototypes*
- *Create high-fidelity detailed design*

- *Do usability testing again*
- *Document standards and guidelines*
- *Create a design specification*

Implementation Phase: During the implementation phase the design prototypes are implemented into working systems [17].

- *Do ongoing heuristic evaluations*
- *Work closely with delivery team as design is implemented*
- *Conduct usability testing as soon as possible*

Validation Phase Evaluation of the implemented system and usability testing against the usability objectives should be provided [17].

- *Use surveys to get user feedback*
- *Conduct field studies to get info about actual use*
- *Check objectives using usability testing*

4.2. Analysis Tools And Methods Used For UCD

In UCD many different methods can be used for the analysis phase. The most common tools are Personas, User Scenarios and Use Cases (see Table 4.2).

Analysis Tools	*Short Description*
Persona	A persona is a user-archetype which defines a fictional user of a system and is based on the knowledge of the real users. The main reason to create personas is to set a common understanding of the end user in a team.
User Scenario	User scenarios describe situations in which users will use the system. Actually, user scenarios put the developers and researchers into the perspective of the user, it helps to understand what users goals are and how they will use the system to achieve their goals. In contrast to use cases a scenario does not contain any of the technology or system interactions and only focuses on what the user is doing.
Use Case	A use case describes an interaction between a user and the system.

Table 4.2.: Analysis Tools used for UCD

UCD Methods: Table 4.3 gives an overview about existing methods used in UCD. Most of the methods are recommended by Jakob Nielsen [18]. It is important to emphasize that the methods described below are most effective if they are combined together. In general, you should not use one of them as your sole source of evaluation data.

UCD Method	*Short Description*	*Number of Users*
Empirical Methods		
Contextual Inquiry	Usability professionals observe the users at their own workplace and analyse their workflows and activities in order to learn about users environment factors that could influence the use of software.	variable
Focus Group	Participants sit together and discuss issues relating to a system user interface under the guidance of a facilitator who maintains the group's focus. The main objective is to discover what users want from the system.	6-9
Interview	Interviewing users, stakeholders, content experts etc. face to face or remotely. During the interview the respondents are sometimes able to view the system.	variable
Paper Prototype	Users perform the tasks given using a low fidelity version of the system. Sometimes the prototypes are only hand-sketched drawings of an interface.	5-7
Survey	Data is collected through the use of qualitative, open ended questions or quantitative predefined questions. Interviews or Questionnaires can be conducted in person, online or by telephone. Surveys can be useful when we want to collect data that cannot be directly observed such as the opinion of a user about the interface of a system.	variable
Task Analysis	Observations combined with an interview are used, so usability experts and designers can identify steps required to reach user goals using the system.	5 or more

Usability Test	Users work with a prototype of the system to perform a set of predefined tasks. Researchers observe the performance to determine usability problems. One of the most popular usability methods is "Thinking Aloud" where users are asked to say everything they are thinking so the observers can understand the users' activities.	5 or more
Log File Analysis	User behaviour is analysed from logs which are produced automatically by different techniques like web servers, operation systems or databases.	none
Inspection Methods		
Expert Review	Experts explore the system and report in detail the divergence to design and usability principles based on their expertise. Several experts are needed to ensure that the main problems are found.	3 or more
Heuristic Evaluation	In a heuristic evaluation, usability experts review the system interface and compare it against usability principles (heuristics). The analysis results in a list of potential usability issues. A small team of experts is required to find main usability problems.	2 or more
Cognitive Walkthrough	Cognitive walkthrough is a task-specific method used to identify usability issues of an interactive system. An evaluator leads the user through an interactive system asking questions during the walkthrough or after the test.	variable

Table 4.3.: Methods for UCD

The following subsections describe some of the most used Human Computer Interaction (HCI) Techniques in more detail.

4.2.1. User Research

User research focuses on understanding user needs, their behaviours, and interests through feedback methodologies like observation or task analysis.

4.2.1.1. Interviews And Questionnaires

Years ago Nielsen explained in the article ”First Rule of Usability? Don’t Listen to Users”[2] that there is a big difference between what users say and what they do. He is of the opinion that the statement is just as relevant these days. Nielsen considered that interviews fail when ”you’re asking people to either remember past use or speculate on future use of a system.” [18] He recommended not to conduct interviews if you want to find something out about a specific design.

However, interviews are very useful after user testing. Therefore, he advises combining interviews with other usability methods like the thinking aloud test or formal experiment. Interviews are beneficial when you want to find out how users think about a certain problem or if you want to know what users thought of a product after using it. Interviews provide subjective data about user impressions.

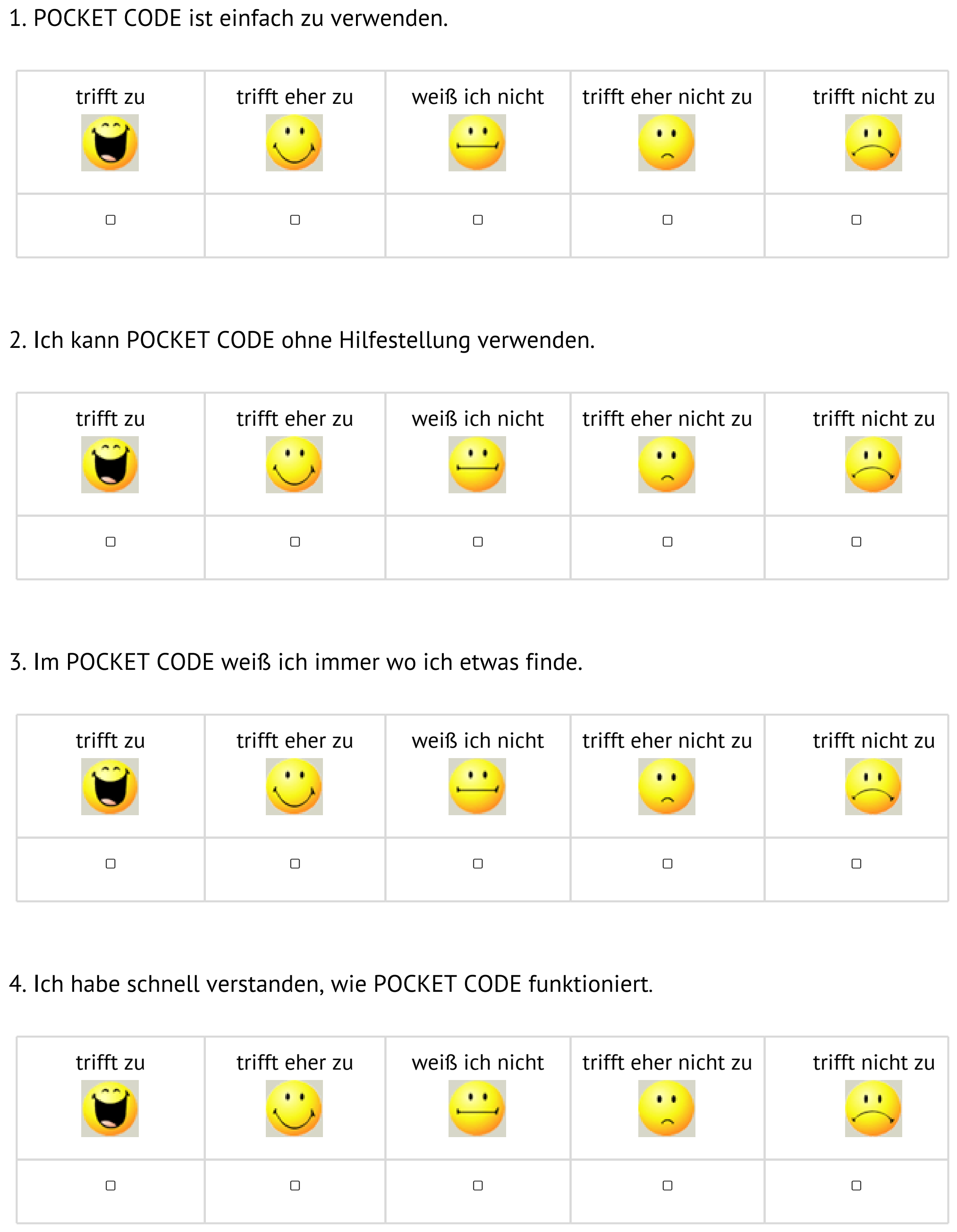
Similar to interviews, questionnaires involve asking users a set of questions and recording their answers. Usually questionnaires are printed to paper (see Figure 4.2) or conducted with the help of a computer online or offline. Questionnaires can be administrated without any assistance from further persons besides the user, whereas with interviews, an interviewer is necessary to ask and record the questions. The application of questionnaires is suitable before and after a usability test to gather qualitative and quantitative data.

4.2.1.2. Contextual Inquiry

Contextual Inquiry (CI) is a field research method to gather information about users and their specific needs. Beyer [2] describes CI as observing the user in his or her own environment and talking about what they are doing and why. The main interest herein lays upon how users perform their work, what they are trying to execute and what difficulties arise. The main focus is not on design. CI consists of two main parts, *the contextual interview* and *the interpretation session*.

The contextual interview takes place where the user is, for example an office, a home or a car. For the interviewer it is important to observe the informal notes and the auxiliary tools the user needed to perform his or her job. Contrary to usability testing, which focuses on finding usability problems, the aim of CI is to understand the whole workflow. The interpretation session is used to evaluate the field data collected. Usually the notes from the field tests are unstructured and

[2]http://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/



1. POCKET CODE ist einfach zu verwenden.

trifft zu 😄	trifft eher zu 🙂	weiß ich nicht 😐	trifft eher nicht zu 🙁	trifft nicht zu ☹️
☐	☐	☐	☐	☐

2. Ich kann POCKET CODE ohne Hilfestellung verwenden.

trifft zu 😄	trifft eher zu 🙂	weiß ich nicht 😐	trifft eher nicht zu 🙁	trifft nicht zu ☹️
☐	☐	☐	☐	☐

3. Im POCKET CODE weiß ich immer wo ich etwas finde.

trifft zu 😄	trifft eher zu 🙂	weiß ich nicht 😐	trifft eher nicht zu 🙁	trifft nicht zu ☹️
☐	☐	☐	☐	☐

4. Ich habe schnell verstanden, wie POCKET CODE funktioniert.

trifft zu 😄	trifft eher zu 🙂	weiß ich nicht 😐	trifft eher nicht zu 🙁	trifft nicht zu ☹️
☐	☐	☐	☐	☐

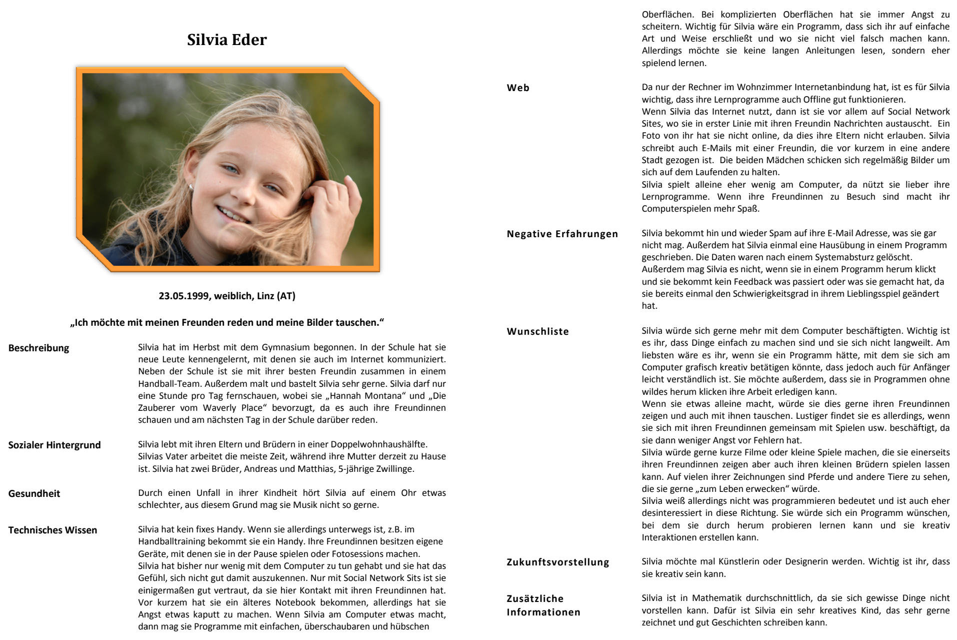
Figure 4.2.: Example of a Feedback Questionnaire for children used after a Thinking Aloud test.

hard to understand. Each set of interview notes is treated separately by the team. The interviewer repeats the story to the rest of the team which gathers the key information. At the end of a session all the relevant information from one user is written down and captured in work models. The analysis relies mostly on the capabilities of the interviewer. Data gathered from the interview is used in the early stage of a product lifecycle. Raven and Flanders [19] summarise CI as follows:

1. *Data gathering must take place in the context of the user's work.*
2. *The data gatherer and the user form a partnership to explore issues together.*
3. *The inquiry is based on a focus; that is, the inquiry is based on a clearly defined set of concerns, rather than on a list of specific questions (as in a survey).*

4.2.2. Personas

Personas, first introduced by Alan Cooper[3], define a typical hypothetical user of a system. Personas are based on the knowledge of real users and give the stakeholders insights into who the users are. By thinking about the needs of a fictional persona, developers may be better able to envision what an end user may need. In Agile Software Development specific personas can be used for writing user stories. Sometimes teams design the software for a specific persona. Figure 4.3 shows a typical user for the Catrobat Project described in Section 6.



Silvia Eder

23.05.1999, weiblich, Linz (AT)

„Ich möchte mit meinen Freunden reden und meine Bilder tauschen."

Beschreibung	Silvia hat im Herbst mit dem Gymnasium begonnen. In der Schule hat sie neue Leute kennengelernt, mit denen sie auch im Internet kommuniziert. Neben der Schule ist sie mit ihrer besten Freundin zusammen in einem Handball-Team. Außerdem malt und bastelt Silvia sehr gerne. Silvia darf nur eine Stunde pro Tag fernschauen, wobei sie „Hannah Montana" und „Die Zauberer vom Waverly Place" bevorzugt, da es auch ihre Freundinnen schauen und am nächsten Tag in der Schule darüber reden.
Sozialer Hintergrund	Silvia lebt mit ihren Eltern und Brüdern in einer Doppelwohnhaushälfte. Silvias Vater arbeitet die meiste Zeit, während ihre Mutter derzeit zu Hause ist. Silvia hat zwei Brüder, Andreas und Matthias, 5-jährige Zwillinge.
Gesundheit	Durch einen Unfall in ihrer Kindheit hört Silvia auf einem Ohr etwas schlechter, aus diesem Grund mag sie Musik nicht so gerne.
Technisches Wissen	Silvia hat kein fixes Handy. Wenn sie allerdings unterwegs ist, z.B. im Handballtraining bekommt sie ein Handy. Ihre Freundinnen besitzen eigene Geräte, mit denen sie in der Pause spielen oder Fotosessions machen. Silvia hat bisher nur wenig mit dem Computer zu tun gehabt und sie hat das Gefühl, sich nicht gut damit auszukennen. Nur mit Social Network Sits ist sie einigermaßen gut vertraut, da sie hier Kontakt mit ihren Freundinnen hat. Vor kurzem hat sie ein älteres Notebook bekommen, allerdings hat sie Angst etwas kaputt zu machen. Wenn Silvia am Computer etwas macht, dann mag sie Programme mit einfachen, überschaubaren und hübschen Oberflächen. Bei komplizierten Oberflächen hat sie immer Angst zu scheitern. Wichtig für Silvia wäre ein Programm, dass sich ihr auf einfache Art und Weise erschließt und wo sie nicht viel falsch machen kann. Allerdings möchte sie keine langen Anleitungen lesen, sondern eher spielend lernen.
Web	Da nur der Rechner im Wohnzimmer Internetanbindung hat, ist es für Silvia wichtig, dass ihre Lernprogramme auch Offline gut funktionieren. Wenn Silvia das Internet nutzt, dann ist sie vor allem auf Social Network Sites, wo sie in erster Linie mit ihren Freundin Nachrichten austauscht. Ein Foto von ihr hat sie nicht online, da dies ihre Eltern nicht erlauben. Silvia schreibt auch E-Mails mit einer Freundin, die vor kurzem in eine andere Stadt gezogen ist. Die beiden Mädchen schicken sich regelmäßig Bilder um sich auf dem Laufenden zu halten. Silvia spielt alleine eher wenig am Computer, da nützt sie lieber ihre Lernprogramme. Wenn ihre Freundinnen zu Besuch sind macht ihr Computerspielen mehr Spaß.
Negative Erfahrungen	Silvia bekommt hin und wieder Spam auf ihre E-Mail Adresse, was sie gar nicht mag. Außerdem hat Silvia einmal eine Hausübung in einem Programm geschrieben. Die Daten waren nach einem Systemabsturz gelöscht. Außerdem mag Silvia es nicht, wenn sie in einem Programm herum klickt und sie bekommt kein Feedback was passiert oder was sie gemacht hat, da sie bereits einmal den Schwierigkeitsgrad in ihrem Lieblingsspiel geändert hat.
Wunschliste	Silvia würde sich gerne mehr mit dem Computer beschäftigten. Wichtig ist es ihr, dass Dinge einfach zu machen sind und sie sich nicht langweilt. Am liebsten wäre es ihr, wenn sie ein Programm hätte, mit dem sie sich am Computer grafisch kreativ betätigen könnte, dass jedoch auch für Anfänger leicht verständlich ist. Sie möchte außerdem, dass sie in Programmen ohne wildes herum klicken ihre Arbeit erledigen kann. Wenn sie etwas alleine macht, würde sie dies gerne ihren Freundinnen zeigen und auch mit ihnen tauschen. Lustiger findet sie es allerdings, wenn sie sich mit ihren Freundinnen gemeinsam mit Spielen usw. beschäftigt, da sie dann weniger Angst vor Fehlern hat. Silvia würde gerne kurze Filme oder kleine Spiele machen, die sie einerseits ihren Freundinnen zeigen aber auch ihren kleinen Brüdern spielen lassen kann. Auf vielen ihrer Zeichnungen sind Pferde und andere Tiere zu sehen, die sie gerne „zum Leben erwecken" würde. Silvia weiß allerdings nicht was programmieren bedeutet und ist auch eher desinteressiert in diese Richtung. Sie würde sich ein Programm wünschen, bei dem sie durch herum probieren lernen kann und sie kreativ Interaktionen erstellen kann.
Zukunftsvorstellung	Silvia möchte mal Künstlerin oder Designerin werden. Wichtig ist ihr, dass sie kreativ sein kann.
Zusätzliche Informationen	Silvia ist in Mathematik durchschnittlich, da sie sich gewisse Dinge nicht vorstellen kann. Dafür ist Silvia ein sehr kreatives Kind, das sehr gerne zeichnet und gut Geschichten schreiben kann.

Figure 4.3.: Example of a Persona used for Catrobat Project.

Personas are constructed by performing user research like interviews or observations with end-users. Information gathered from different users are combined to create a single persona. Generally, personas include the following key information [4]:

- Name
- Job title and job description

[3]http://en.wikipedia.org/wiki/Alan_Cooper
[4]www.usability.gov

- Demographics like age, education and family status
- The goals and tasks they are trying to complete using the product
- Physical, social and technological environment
- Picture representing that user group
- A quote that summarise what matters most to the persona as it relates to the product

4.2.3. Prototyping

Prototyping is very important for usability evaluation in the early design stage. Building and evaluating prototypes is cheap and can be done very quickly. The author of [20] categorised the prototyping into the following four types: *Verbal Prototypes, Paper Prototypes, Interactive Sketches and Working Prototypes*. The more the design stage is in progress, the higher is the complexity of the prototypes.

Verbal Prototypes: Only a textual description of the system is available.

Paper Prototypes: Paper prototyping is a popular method used in UCD process to get usability feedback from real users in the early design stage. The method is highly recommended by usability experts like Nielsen. He summarises the method as follows:

> *"With a paper prototype, you can user test early design ideas at an extremely low cost. Doing so lets you fix usability problems before you waste money implementing something that doesn't work."* [18]

Andrews [20] distinguishes between *Low-Fidelity Paper Prototypes* and *High-Fidelity Paper Prototypes*.

Low-Fidelity Paper Prototypes are hand-drown sketches. The goal of such prototypes is to get *maximum feedback from minimum effort.*

High-Fidelity Paper Prototypes are usually created with special prototype software programs like Adobe Illustrator. Printouts are in colour and look very close to the final design. According to [20] users concentrate on things like fonts or colour rather than the flow through the application.

Interactive Sketches: Hand-drawn sketches are used to create interactive prototypes with clickable elements. Software like VVV[5] can be used to build such clickable version. The design is kept to a minimum to avoid discussion about fonts or icons.

[5]http://vvvv.org/

Working Prototypes: Implemented prototypes are used with reduced functionality and fake data. Nielsen described the simplification of the algorithms as follows:

> *"...and implementation time can sometimes be reduced by "cheating" on the algorithms to make them ignore the special cases that often take an inordinate amount of programming effort." [15]*

4.2.4. Heuristic Evaluations

Heuristic evaluation belongs to the set of inspection methods where usability experts explore a interface. Usability inspection aims find usability problems in a user interface. Nielsen and Molich define Heuristic Evaluation as follows:

> Definition: "Heuristic evaluation is a usability engineering method for finding the usability problems in a user interface design so that they can be attended to as part of an iterative design process. Heuristic evaluation involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the "heuristics")."

Table 4.4 provides the usability principles (heuristics) defined by Nielsen and Molich.

Visibility of system status	*The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.*
Match between system and the real world	*The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.*
User control and freedom	*Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.*
Consistency and standards	*Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.*
Error prevention	*Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.*

Recognition rather than recall	*Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.*
Flexibility and efficiency of use	*Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.*
Aesthetic and minimalist design	*Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.*
Help users recognize, diagnose, and recover from errors	*Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.*
Help and documentation	*Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.*

Table 4.4.: Heuristics (Nielsen and Molich, 1990; Nielsen 1994)

To apply heuristic evaluation more than one evaluator is needed because a single person is not able to find all the usability problems of an interface. Several evaluators will find more usability problems. Heuristic evaluation is performed by having each evaluator inspect the interface alone. After each evaluator has finished the inspection the findings are combined. Heuristic evaluation must not be done by experts, developers can also evaluate a software against the heuristics. However, it is known that usability professionals find more problems than developers. The evaluation can be conducted in an early stage of development, it is cheap and can be done quickly because end users are not involved in the process. [18]

4.2.5. Usability Testing

Usability is defined by Jakob Nielsen as follows:

> ”Usability is a quality attribute that assesses how easy user interfaces are to use. The word ”usability” also refers to methods for improving ease-of-use during the design process.”

Nielsen defined five quality attributes as well:

- *Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?*
- *Efficiency: Once users have learned the design, how quickly can they perform tasks?*
- *Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?*
- *Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?*
- *Satisfaction: How pleasant is it to use the design?*

There are many usability testing methods for measuring usability. In the following paragraphs, I will describe the more well-known, and widely used user testing methods: Thinking Aloud Test, Co Discovery Learning, Formal Experiment and A/B Testing.

4.2.5.1. Thinking Aloud Test

In 1993 Jakob Nielsen wrote that ”Thinking aloud may be the single most valuable usability engineering method” [18], 20 years later he is of the same opinion. He defines Thinking Aloud (TA) Testing as follows:

> ”In a thinking aloud test, you ask test participants to use the system while continuously thinking out loud - that is, simply verbalising their thoughts as they move through the user interface.”

The method can be used for almost any system. Watching the user interact with the system, usability experts can easily identify major misconceptions. A small number of users is needed to find most usability problems. Such tests can be conducted early in the development process.The TA method does not provide representative time measurements of real usage because talking during the test slows down the users. The experience of using a system depends on what users are doing so they solve a predefined set of typical usage situation tasks during the test. [21]

4.2.5.2. Co-Discovery Learning

Co-discovery Learning (CL) or Constructive Interaction is an adaptation of the thinking aloud test. In CL, users are grouped in pairs and talk aloud naturally to each other while completing a task. They are to help each other in the same manner as they would if they were working together to accomplish a common goal using the product. Both users are encouraged to explain what they are thinking about the tasks while working on them. Compared to the thinking-aloud protocol, this technique makes it more natural for the test users to verbalise their thoughts during the test. Compared with the thinking aloud test, CL requires twice as many users therefore it is most suitable for projects where it is easy to get a large number of users. It is also very well suited to testing with children because children often are not able to follow the standard instructions for an ordinary thinking aloud test. [21] [20]

4.2.5.3. Formal Experiment

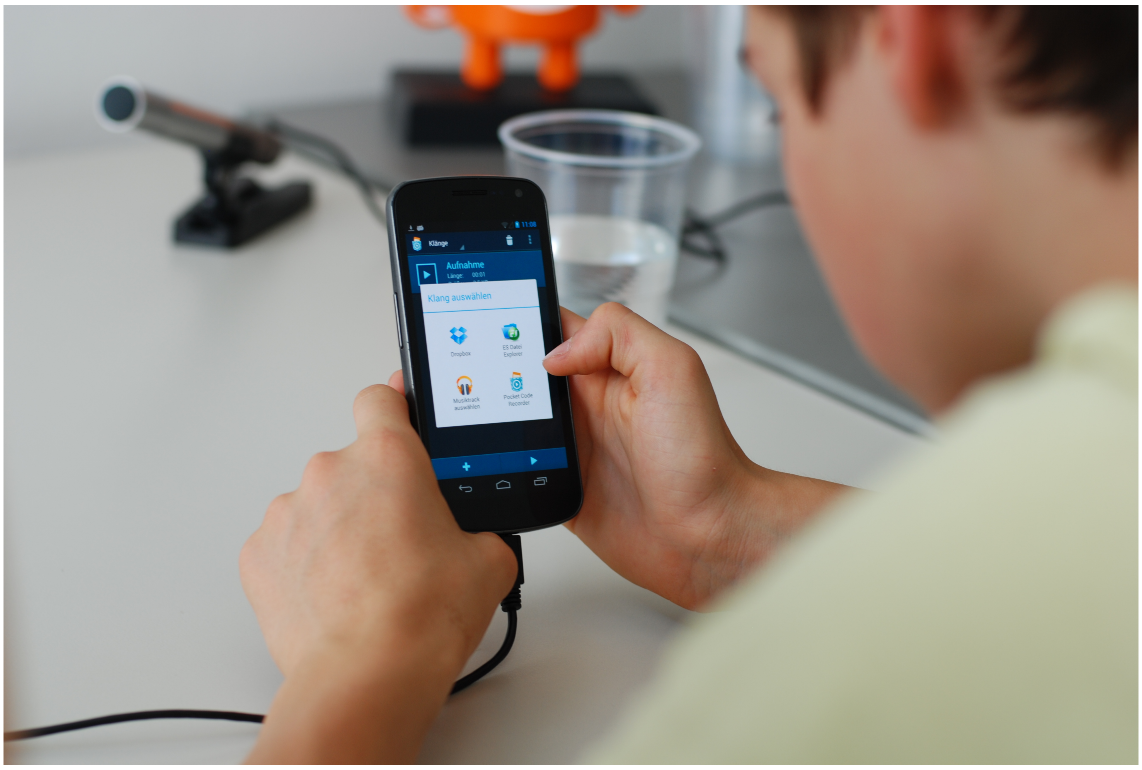
Formal Experiment is a summative evaluation of a fully implemented design. The aim of such an experiment is to conduct an objective measurement of the system's performance such as the number of errors, accuracy, time spent using help, number of commands used etc. Usually two alternative interface designs are objectively compared to discover which one is better according to predefined criterion like efficiency or error rate and so on. Formal experiments can also be used for testing the absolute performance of one interface. It is a controlled experiment that requires a large number of test users for statistical accuracy. Formal experiment is a very good method for collecting objective quantitativ data and it allows a comparison of different designs. Nevertheless such method is not always preferable because it is usable only late in development, needs a significant number of test users and requires facilitator expertise. [20]

4.2.5.4. A/B Testing

A/B testing also called *split testing* or *multivariant testing* is used to compare two or more different live traffic designs where each version is presented to real live users. The objective is to find out which version performs best. A/B testing can be used for comparable designs, competing products or old applications versus new applications. The method is widely used on the web where different versions of the website are shown to different visitors. To get some data a metric is measured for each variant. Such a metric could be click-through rate, or sales volume and so forth. The number of users of the product should be large enough, so that statistical significance can be achieved. [20][18]

4.3. Mobile Usability

"For every feature that's removed, there is one less opportunity for the UI to confuse users, which makes the remaining features easier to use." (Nielsen, Budiu)



Desktop user interface platforms differ from mobile interfaces in many aspects including interaction techniques, reading behaviour or the context of use. Further distinctions are big versus small screens, on the move versus stationary, touch versus mouse, wireless connectivity versus fast wired Internet and so on. Therefore, different designs should be provided for mobile and desktop sites.

However, mobile usability issues do not differ too much from conventional usability issues. Each guideline from desktop design is even more important for mobile. It is evident that mobile users need a different design to desktop users and vice verse.

Nielsen et all. [22] proposed in their book *Mobile Usability* to design specifically for mobile devices. In 2009 they conducted a study where the participants success rate averaged 64 percent on sites especially designed for mobile compared with 53 percent for using desktop sites on mobile. According to the study users are also more satisfied with the version especially designed for small screens. It can be assumed that when users are successful and satisfied that they are likely to come back.

They repeated the study in 2011 with the expectation that user performance would have improved over time. The success for mobile sites rose to 74 percent but also the success rate for full sites had increased to 60 percent. It follows from this that the full sites had become more mobile-friendly.

4.3.1. Full Site Designs Don't Work For Mobile Use

Obviously, mobile screens are much smaller than desktop screens or tablets. Less information can be displayed simultaneously, so users are required to rely on their short term memory. According to Nielsen et all. [22] reading on small screens makes it twice as hard to understand a text than on large computer screens. They argue that mobile users are more impatient that desktop users. Writing for mobile is different, Nielsen strongly recommended to write shorter and simpler for the smaller screens to allow for sufficient text comprehension. Further advice is to design all content for the appropriate screen size. Rescaling images is most often not enough, they should be cropped and zoomed differently for mobile and desktop. A mobile optimised site should only offer functionality that is important for mobile use. Providing a link to the full site offers the users the possibility to access the functionality that only is available on desktop sites. It is also important to keep the navigation as simple as possible to prevent user disorientation.

Progressive Disclosure: In the mobile environment interface designers have to overcome two contradictory requirements:

- Users want powerful apps with maximum functionality.
- Users want simple apps because they don't have time to learn the complexity.

In order to meet these requirements progressive disclosure can be used. The idea behind progressive disclosure is to show the user only the most important functionality initially, offering advanced features only on request. This approach has the advantage that novice users do not have to handle unnecessary complexity and with features they do not need. *"Progressive Disclosure improves learnability, efficiency of use and error rate."* [22] The challenges for interface designers are to decide between initial and secondary features and to design a clear interface to change between primary and secondary content.

Nowadays mobile screens are so small that it is imperative not to waste space. Nielsen recommended to use more standard OS icons, or at least supplement the icons with a text label and to use two interfaces or views for the data only when the views provide significantly different aspects of the data.

The content should be shorter and simpler written without limiting the product. The mobile version should contain less description and information about a product but the same items as the desktop site. Secondary content can be placed on a secondary page. On desktop devices it is often better to show all content on the first site and save user clicks. The interface for mobile devices should be enlarged so that users can select it even if they have big fingers ("fat finger problem"). [22]

Using gestures can sometimes save space normally used for buttons. Gestures should be similar to those which people use in the real world because they tend to be easier to learn and remember. To give an example: using the swipe gesture for turning a page turned out to be good because most users grasped it quickly. A tutorial at the start of the app is insufficient to teach users how to use the gestures. A better solution is to introduce tips more actively and to reveal information progressively in the right context.

The menu bar is the most common way to make commands visible for users. The disadvantage of a menu bar is the usage of precious space on the small screens. Sometimes the menu bar is temporarily hidden and shown only when needed. If so, it is important to use a simple operation to reveal the menu bar again. The benefit of using a menu bar is that commands and options are always visible to users. If mobile guidelines are followed, the menu bar always looks the same so users have to learn less and can focus on their real tasks.

4.3.2. Input Devices

Input devices for mobile phones and tablets are usually fingers whereas for desktops and laptops mouse devices and keyboards are used. The main differences between the input methods are:

- Mouse devices are more precise than fingers.
- Mouse devices have three controls: right and left button, scroll and wheel whereas smartphones have only one control.
- Mouse devices require time to switch between different input devices like keyboard and mouse.
- Mouse devices enable nonlinear relationship between the speed of moving the physical device and the onscreen pointer, which allows high-precision pointing.
- Possible signal states for mouse devices are hover, mouse-down and mouse up. Using our fingers as input three signal states are possible, finger-down, finger-up and long press.
- Mouse devices are suitable for very large desktop monitors, fingers are not due to arm fatigue.
- Mouse devices have a visible pointer or curser, fingers do not.
- Mouse devices require learning time to operate with them, fingers do not.

The above list reveals that both mouse devices and fingers have strengths and weaknesses. It is important to consider the input device when designing a user interface.

The fact that desktops have a keyboard is also very important. Typing on a keyboard is less error-prone and much faster than typing with fingers on a virtual keyboard device.

4.3.3. Responsive Design

These days users are confronted with devices with significant different screen sizes which are able to connect to Internet for example mobile phones, tablets, laptops, desktops, game consoles or even TVs (see Figure 4.4).

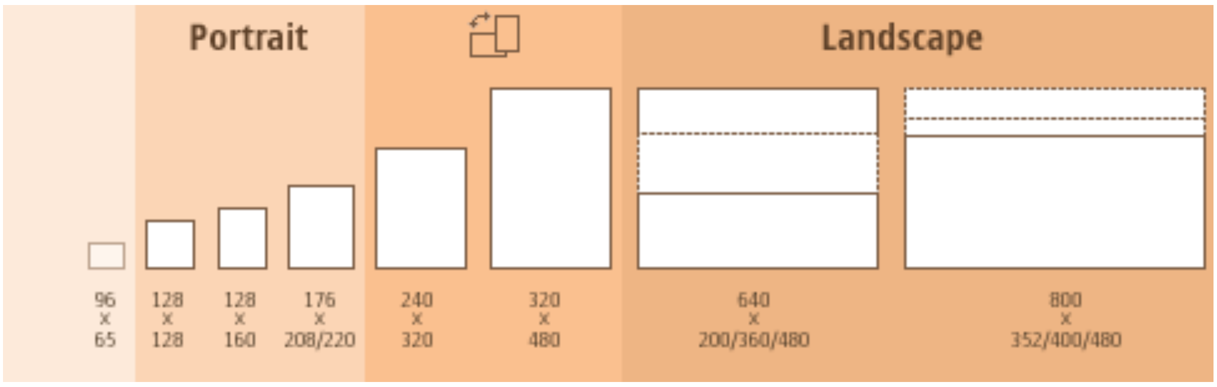


Figure 4.4.: Most significant screen sizes collected by Morten Hjerde [1] out of 400 different device models.

To deal with the number of screen sizes *responsive design* can be used. Responsive design means *"optimizing the layout of a Web page for the screen dimensions and screen orientation."* [22]

Figure 4.5 shows a site with the same content in three different screen sizes. Depending on the size the number of items used for a layout changes.

The particular benefit obtained by responsive design is that only one site is used for mobile and for desktop so the maintenance requirements are low. Moreover responsive design supports users which have Internet access only via mobile devices. However, it must be kept in mind that certain features make less sense on mobile than on a desktop.

4.3.4. Mobile Apps

According to the study conducted by Nielsen et all. [22] the success rate when users used mobile apps was 74 percent, which is much higher than 64 percent which was the success rate using mobile specific web-sites. Regardless of how well a mobile site is optimised, only limited optimisation is possible. In contrast an app can exploit specific functionalities of each device much better than a web site can running in a browser. Usually apps have a simpler design than web-sites. The biggest disadvantage compared to websites is that app content is less discoverable. It can be assumed that users are more likely to search on the Web than look up an app in an application store. An alternative to mobile apps are mobile Web apps. Such Web apps are very similar to native apps, they work in a browser but look like native apps.

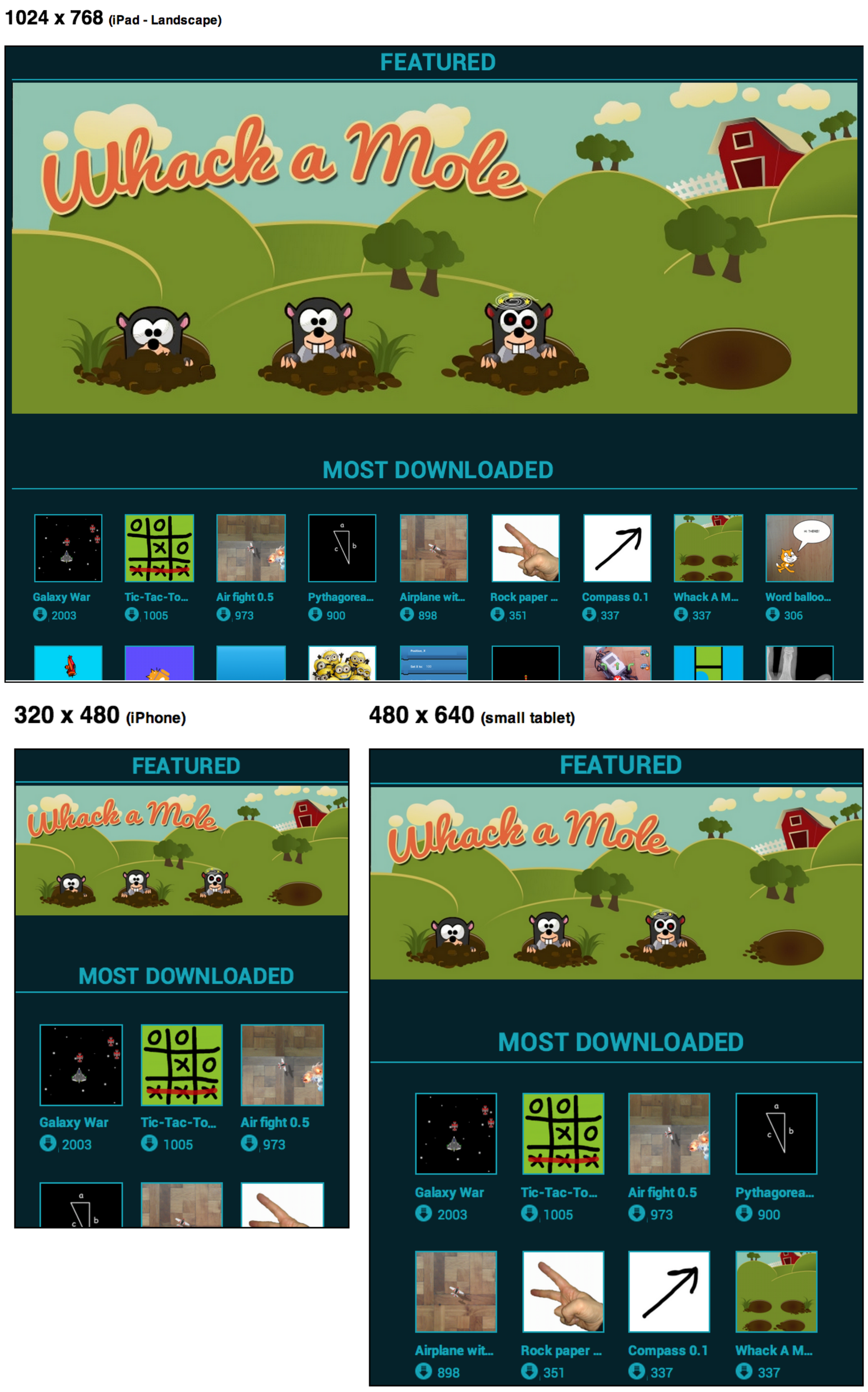


Figure 4.5.: Pocket Code's community site changes the layout according to the screen sizes.

In 2000 Nielsen et. all [22] conducted a usability study with two simple tasks (see Table 4.5). They used WAP Phones for the experiment. The same study was repeated 9 years later with the expectation that the performance would be much better. The results were quite surprising. Users spent 38 percent more time on the tasks in 2009 than they did in 2000.

Task	WAP Phones (2000)	Modern Phones (2009)
Find the local weather for tonight	164 sec.	247 sec.
Find what's on BBC TV 1 tonight at 8 p.m.	159 sec.	199 sec.

Table 4.5.: Comparing mobile usability studies form year 2000 and 2009 (Nielsen)

In 2000 users had only one option to look for the required information because there were only a few selected services. So they got the information very quickly with only few key presses. Nine years later the alternative ways to get the information were widely spread.

In the last four years mobile usability increased at a very fast rate because of the new touch-screen platforms like iPhone or Android. *"The usability of using high-end phones in 2012 was probably similar to that of using desktop websites in 2004."* [22]

5. User-Centered Design And Agile Software Development

It is important to emphasise that UCD is not a process model for software development since it covers only user research, design and usability testing. It needs to be combined with a software development methodology so that they handle the whole life-cycle of a project together. Combining UCD and ASD can effectively increase the success of a project. However, conducting usability studies and doing the necessary analysis of the data afterwards as described in Section 4 can be time-consuming, especially with respect to development cycles as short as those in ASD. To perform successfully in such environments usability experts and user interface designers need to learn and understand the agile culture.

The integration of agile methods with UCD have been tried at the early stage of the agile methods since 2002 when Kent Beck and Alan Cooper discussed the integration of Extreme Programming with Interaction Design [23].

Hodgetts describes in his paper *Experiences Integrating Sophisticated User Experience Design Practices into Agile Processes* [24] the collaboration of the whole team and the specialisation of team members, two important organisational and cultural attributes that are common among many UCD groups. He is of the opinion that these attributes are also key challenges while combining agile culture and the UCD approach. One of the key questions is the internal structure of the team. Should the UCD team be part of the core team or should it be organised separately? One goal of agile processes is to be flexible and work with people without specialisation. Agile developers are able to take different tasks each iteration. So teams can be more productive and efficient with fewer resources. UCD practitioners are usually specialists, for example only designers or usability experts. The author describes that teams with specialists have often problems finishing the planned work during an iteration:

> "Specialization forces the team to pre-assign tasks to individuals, thus hindering the team's ability to dynamically allocate tasks on a day-to-day basis as new circumstances emerge. The pre-allocation of tasks also requires additional planning time that often extends the length of each iteration's initial planning activities."[24]

Fitting UCD activities into ASD and the short development iterations is usually not a trivial task.

In this chapter the relationship between UCD and ASD will be discussed. I will start with some agile culture values that are important for the integration of UCD in an agile team. Then, I will describe some existing methods for the integration of the two disciplines as they currently exist. Finally, I will present best practices for integration of UCD with ASD.

5.1. Agile Culture

According to Beyer [2] the agile culture values provided below can have an effect on the collaboration of UCD professionals and software developers. Understanding those values and its aims can be useful for UCD professionals to promote UCD within an agile team.

5.1.1. Agile Culture Values

There Is Only One Team: For an agile team, every single task is the responsibility of the team, instead of individuals, so the entire team shares the success or the failure of the project they are working on. All team members should work tightly together in a single room, and help each other. Agile teams avoid specialists, which means that every member of the team is able to work on any part of the project. However, some special skills required by a team, like UCD, take time to develop and are highly specialised. Such skills are often provided by one or only a few team members. Agile teams often do not treat UCD professionals as ordinary team member but rather as some kind of customer or supporter of product owner. Few teams treat UCD professionals as full team members and do expect them to code. The *"one-team"* expectation can be a benefit for UCD professionals. As part of a team UCD experts are able to promote UCD values to all team members. If there is not enough time to complete the UCD task within the envisaged time, the whole team is responsible for that. Therefore, simple tasks can be done by other team members under the supervision of UCD professionals. [2]

The User Is On The Team: Agile teams expect that someone who represents the user belongs to the team. Usually, they do not distinguish between a user, an internal stakeholder or a client. Most of the developers do not care about end-users' satisfaction; if they fulfil the requirements defined by internal stakeholders the job is regarded as done.

> "Agile methods tend to overlook the real issues getting in the way of collecting valid end-user feedback. They tend to assume that users can say what they want if asked; that users can articulate their tasks, motives, and goals; and that users can devote extensive time to guiding developers - none of which are true."[2]

Usability professionals with special skills can help in such situations and moderate between developers and end-users. Here the special skills of UCD professionals are valuable. Providing usability methods and expertise in order to collect user data and usability issues of the software is crucial. With the knowledge of a usability expert the team is able to work even without a real end-user as team member.

Planning Is A Waste Of Time: From an agile point of view too much up-front planning and documentation is pointless since requirements and management goals are permanently changing. Spending a long time on detailed designs is risky because it is very likely the entire design will never be implemented because of requirement changes. Therefore, it is better to break down the work into small parts and develop them one after another, rather than spend a huge amount of time on a design that will never be realised.
Agile development methods have no specific techniques to support the design of user interface (UI) and usability.

> " So "pure" agile methods have no way to develop a complete understanding of users and their needs; no way to invent and structure a coherent solution; and no way to design a consistent user experience, interaction paradigm, and appearance across the product." [2]

To address this problem several experts [2], [25] suggest introducing a *"Phase 0"*. During Phase 0 UCD experts do all the work which should happen before development starts, including user studies and core design of the proposed system so that user stories can be written.

Face-to-face Communication Is Better Than Documentation: As mentioned before, agile culture does not rely on documentation because it takes too long to write. Instead of documentation, ASD prefers face-to-face communication. Program code is used as documentation, so it can immediately be tested with end-users instead of paper prototypes. End-users can only rarely be part of the team, so face-to-face communication is not possible. In such situations usability experts are very helpful. If users are involved in the development, they sometimes cannot articulate their needs to the developers without any visual help or design proposals. In this respect again, UCD professionals can help out because they have the skills to analyse and understand user needs and to explain it to developers. A close working relationship between UCD experts and programmers is suggested because usability professionals are able to represent the end-user. UCD professionals may also need to reduce their own documentation and work more with mockups rather than complete high-fidelity UI designs. Less documentations fits better in ASD culture because the developers are not used to read such documents.

Short Sprints Are Good: Agile teams work in small well defined sprints. After each sprint, a running, tested and a real customer value version of the product is delivered. Functionality is broken down into small parts so one part can be implemented in one sprint. UCD professionals should apply the same technique to their work. At the end of a sprint the user interface should be bug-free and usable. If the work takes too long it should be split into smaller parts.

Continual Feedback: Agile methods expect that short sprints bring a great deal of advantages, for example, provide customer value at an early stage of development or get feedback after each iteration. As I stated earlier, agile developers also take the assumption that the user is on the team so they believe that all questions about UI can be clarified during the development. Testing has different meaning for an agile developer than for a UCD expert. For a developer testing usually means checking if the code is written correctly whereas UCD experts try to find out if the end-user understands and is able to use the software. Agile developers present the results only to the stakeholders and get feedback if something was miscommunicated. It is up to the UCD team to get real user feedback.

5.2. Best Practices For Integrating UCD With Agile

Beyer [2] explained in his book that the integration of *"UCD with agile development is not only possible, it is critical to the success of agile methods"*. In this Section I will describe best practices for the integration of UCD with ASD.

Get User Feedback From Real Users In Context: Getting useful feedback from end-users is not an easy task. Most developers, even senior agile developers, do not have the ability to understand the end-users and their needs. For that reason, UCD professionals with special skills are needed. End-users are often not able to clarify what they need or want from a product. When asking them for feedback they usually want to be helpful, so they respond to the system they were given. Such reports are usually on some minor issues of the system rather than essential problems.

Another problem is that users are not available as team members. In practice, we often find the end-user represented by an undefined customer or product owner role, which does not result in satisfactory solution. Using marketing methods such as surveys or focus groups to replace end-users is also not advisable. Maybe such methods are good to collect sales points or market requirements, but are not useful for collecting design data.

> "So, any effective user-centered agile process must include real user research: finding out who the end-users are and how they work; analyzing the tasks they do and the strategies they use to achieve those tasks; getting quick feedback on design ideas and on system base levels to determine whether the project is on track;"[2]

Introduce A Phase 0 (Sprint 0): Commonly, agile communities avoid any form of up-front planning. They believe that designing a product, including the end-user in iterations and designing until the product works is the best way to proceed. Beyer [2] observed that only about 20% of a product can be changed during one iteration. Getting rework time for refactoring is not always possible for the development team. Even if there is time for rework, the question arises how often will reworking on the same design be tolerated in an agile project. A better solution would be to introduce a *Phase 0* or sometimes also called *Sprint 0* for some hight-level, up-front user research and design, tested with end-users because it would reduce the probability of design reworks and the time spent on them.

UI Design Should Be Done One Iteration Ahead: Developing a good UI design is a lot of work which usually takes more than one sprint to be properly done. One way to cope with this problem is to start designing and user testing one or more sprints ahead of coding.

Validation Should Be Done One Iteration Behind: The best time to conduct usability studies with end users is one sprint behind the implementation. Beyer [2] describes in his book, that such a practice also supports large-scale agile projects.

Parallel UCD Stream: To make sure that the UCD team has enough time to focus on the overall consistency of the project and to ensure compliance with company standards and guidelines, a separate autonomous UCD stream is created. The UCD stream runs in parallel to agile development streams and is synchronised at special key points. Problems discovered can be added to the agile development stream as new stories.

Programmer/Designer Holiday: Programmers' holiday uses an iteration to cleanup code, refactor functionality, fix bugs and upgrade software or hardware. During such a programmers' holiday the customer can focus on and plan the next steps for the project. Programmers' holiday was defined by Martin et all as "An iteration of technical tasks so that the customer can have some time to think-ahead" [26]. Such holiday can be useful for the UCD team as well. UCD teams can look at the overall coherence of the user interface during this time.

Usability Experts As Full Team Members: UCD team members should be part of the team and co-located with programmers. All team members are responsible for the whole product and its usability. UCD experts should support the team wherever they are needed and have the skills to do so.

5.3. User-Centered Agile Process

In this section two agile UCD processes based on best practices for integrating UCD with ASD will be described. Both processes recommend starting a project with a Phase 0. During that phase all team members try to get the "Big Picture" of the product, organise themselves and conduct user research, gather requirements and prepare some high-level designs. After that phase the agile team should be able to write good user stories. Contextual Inquiry (CI) as described in Section 4 is an appropriate method for gathering user data, so both processes suggest using that method. With some training such interviews are not restricted to usability professionals and can be performed by, or at least assisted by, every team member. In contrast to usability tests which concentrate on finding problems, the goal of CI is to understand the whole working process of users. The new product solution should improve and change the user's tasks rather than only fix existing problems. After the interviews all team members analyse the data collected. The UCD team identifies the content which has design relevance.

5.3.1. Parallel Work Streams

Parallel work streams are one way of making UCD agile. Figure 5.1 shows a basic structure of a project process introduced by [2]. It consists of a Phase 0 where the team gathers user data, a release planning phase and sprints. The customer team uses the information gathered in Phase 0 to better understand the end-user. At the same time we can conduct user interviews to gather data for the creation of *personas* as described in Chapter 4 to represent the project's user population. Despite the fact that personas with the extensive user research do not fit in the agile culture, agile teams use them to ground communication throughout development and write user stories.

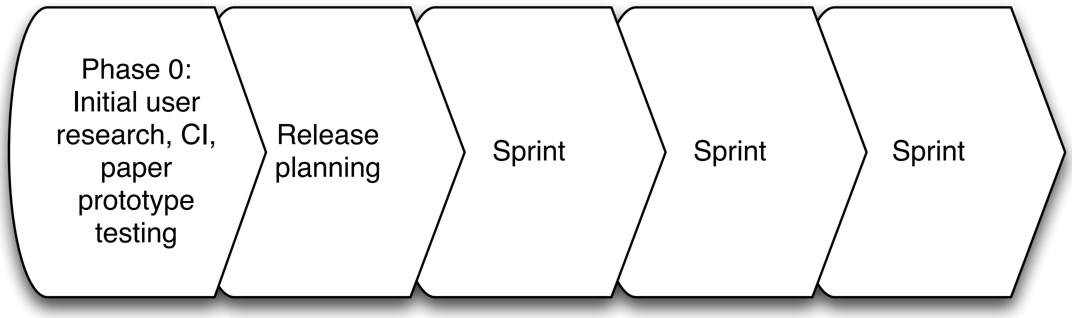


Figure 5.1.: Basic structure of UCD agile development introduced by [2]. Phase 0 is added to the usual agile development process.

Before writing user stories the team should create paper prototypes of the system and conduct some usability tests with real end-users. Paper prototypes are essential because it is easier for users to see the system and interact with it rather than to hear a description and have to imagine it. They can focus on basic workflow problems, rather than things like icons or colours. Another advantage of a paper prototype is that its creation is fast and cheap. Paper prototypes can also be easily modified by end-users and designers during the test. Prototypes are useful in Phase 0 for validating the vision and for making assumptions. Paper prototypes should be used only as high-level descriptions. The detailed design should be created one sprint ahead of the implementation so the UCD team has enough time to iterate the design with end-users before development starts.

After Phase 0 the release planning session is held where the team writes user stories which represent the key elements of the release. All team members who participated in user research during Phase 0 should take the role of the customer in order to support the product owner. During Phase 0 team members especially usability experts gather knowledge of end-users.

As described in Section 2 user stories are written on cards and contain exactly one high-level description of a feature or behaviour. The cards are written from the user's point of view and should provide value to the user. It is important that each story card is fully implemented within a single sprint. UI design is often very complex and too big for one sprint. Therefore, complex functionality is divided into smaller parts and across several cards.

Additionally, each story card contains an estimated cost in "ideal programmer days" or just "points" without a unit and no real-world reference. New teams usually use programmer days to estimate the cost whereas experienced teams often use points. A story estimated at two points can be implemented in half the time required for a story with four points and so on. Agile teams measure their velocity which is the number of story points the team implement within a sprint.

The next step is to organise the stories into sprints. Stories with high-priority or those with fundamental functionality should be implemented in early sprints. Story points assigned to a sprint must be 20 percent less than the team velocity to allow the team a buffer time for fixing bugs and doing the necessary rework collected from user feedback. The first two sprints should be planned carefully whereas the rest can be organised more roughly, because in an agile process changes are welcome. Detailed planning is inappropriate, as too much might change before later sprints are started. Each sprint takes a predefined time and a known number of user stories. Hence, the team can calculate the number of sprints needed for the release and calculate the release date. It is allowed to move stories between releases, and to adjust the release date. Changing story priorities, rewriting stories or introducing new stories can be done during sprint planning sessions.

After the release planning the team meets for a sprint planning session to think over whether the plan is still valid. The team should address the following issues: Has anything changed since the project start? Do they have to add new stories to the release, is there rework to do or bugs to fix? If there are any changes the stories should be written, estimated and prioritised into the schedule, especially for the current and the next sprint. Some development teams write task cards for the stories in the current implementation. The UCD team needs task cards for the stories in the previous, the current and the next sprint.

Each Sprint consists of a set of stories. Until now only mockups and paper prototypes have been created. The final visual design will not be created until it is needed.

As demonstrated in Table 5.1 the UCD team should work one sprint ahead of developers and test one sprint behind. This approach gives the UCD team the necessary time to develop UIs and also to test them with end-users. The pattern shown in Table 5.1 is repeated again and again until the software release ends.

During the sprints the UX team supports the developers on the implementation where real decisions are made and the detailed behaviour is discussed. Coordination between the developers and the UCD team members should be intense and on a regular daily basis. Furthermore, the UCD team also conducts interviews combined with CI and paper prototyping to bring real user feedback into the development process.

	UCD Team	Development Team
Sprint 1	Design UI for story A; prototype and iterate with users	Put development system in place; implement low-UI stories
Sprint 2	Design and iterate UI for story B; Consult on implementation of story A	Implement story A
Sprint3	Test implementation of story A with end-users; Consult on implementation of story B; Design and iterate UI for story C	Implement story B

Table 5.1.: Design is done one sprint ahead of implementation and user testing one sprint behind.

Usability testing during the sprints is very important. Usually interviews with end-users are conducted. First, the UCD team identifies which parts of the UI done in the previous sprints need user feedback. In practice not all parts of the UI will be tested because of limited resources. Parts of the UI which are completely new or stories added at a later date which are not covered by the initial research should be tested with end-users. Because the product is not finished the usability expert should tell the user which elements they want to test and especially what tasks the prototype supports. How the interview is conducted depends on the available prototype. If it is a paper prototype the interviewer should walk users through real-world examples and discuss how the prototype supports the user in doing the task. If problems occur the user and the interviewer are able to modify the paper prototype on the fly. Another option is to test with online prototypes which are less flexible. During the interview issues are discussed with the end user and ways of fixing them are generated. They can sketch solutions together. Sometimes it is also necessary to test the running code from previous iterations with real interaction.

Finally, the team interprets the data collected from the interviews. They evaluate the results of the interpretation session, discuss proposed designs, produce a revised prototype and write story cards to represent that fix. In the next sprint the story cards must be prioritised. It is not advisable to treat UI problems like software bugs, because bug fix processes tend to hide UI problems.

To make the user feedback possible the agile team should have someone responsible for recruiting users and planning regular weekly test sessions.

Ongoing Project: In practice, it is more likely that UCD experts join an already ongoing agile process rather than be part of the team from the beginning. Under such circumstances, it is not possible to do all the work intended for Phase 0. The task of the UCD team member is to support the customer and to integrate the design and usability work. First, find out whether the team is working with real end-users and start user visits in their real working area. Plan the interview so that you can use the data to design UIs that will be needed in this sprint or the next. Furthermore, start working on the design for the stories planned in the next sprint, create mockups and get some user feedback before the sprint starts. Initial parts of the interviews could be used for CI to capture work practice data and the remaining time for feedback on the design or implementation. After a few sprints the team should collect enough work practice data to evaluate the overall direction of the project with regard to the needs of the end-users. During programming holidays try to catch up the work normally done in Phase 0.

System Extension: For system extensions with new features Phase 0 is used to learn about the new tasks, their structure and the new issues. Contextual interviews can be applied to gather all that data. It is possible that new functionality needs a new user interface, so the interface is designed and tested using paper prototypes. After a few rounds of testing prototypes user stories are written.

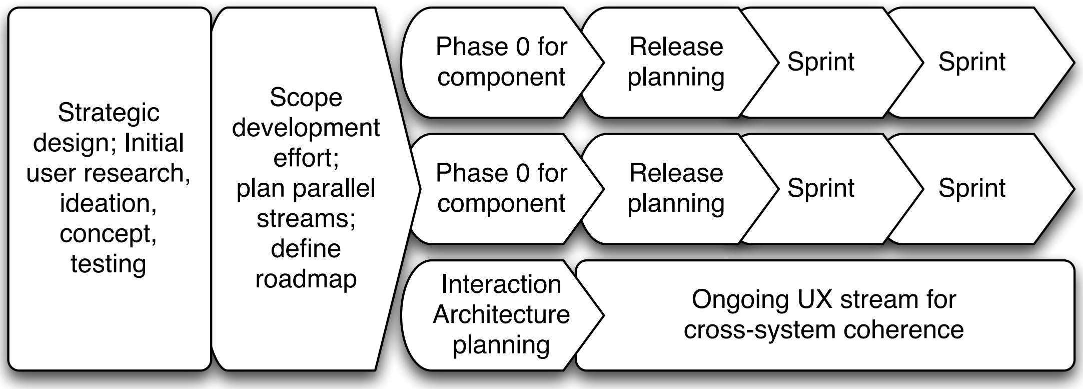

Strategic design; Initial user research, ideation, concept, testing
Scope development effort; plan parallel streams; define roadmap
Phase 0 for component
Release planning
Sprint
Sprint
Phase 0 for component
Release planning
Sprint
Sprint
Interaction Architecture planning
Ongoing UX stream for cross-system coherence

Figure 5.2.: Structure of a project where strategic design and planning sets direction and defines parallel work streams. [2]

Major New Release Of A Large Project: Beyer [2] strongly recommended to start a new release of a project that consists of significant new functions *"with a strategic design to understand the domain and envision an overall solution"*. Significant new functionality which influences big parts of the system requires more user research than possible in the scope of Phase 0 in order to understand the whole problem domain. Therefore, Beyer suggests doing robust user research like Contextual Design (CD), a method developed by the author and Holtzblatt [27]. CD process involves the following steps: Contextual Inquiry, Interpretation, Data Consolidation, Visioning, Storyboarding, User Environment Design and Prototyping. Figure 5.2 gives an overview of the project structure. After finishing the user research work and the prototyping individual working streams can be defined. Each stream is an agile project and starts the work with Phase 0 to identify the details of the component moving on to release planning to define the user stories and the sprints for implementation and validation with end-users. Beyer [2] is of the opinion that a large project needs more processes than a simple one.

> *"Agile methods excel at taking a well-defined product of limited scope and producing useful working software quickly. But when the scope is very large, research, design and planning work needs to precede agile development to create a coherent system"*

5.3.2. Parallel Streams With Iteration

Figure 5.3 illustrates, parallel streams with iteration, a further approach for an agile UCD process described by Sy [28]. It consists of two parallel tracks where design and development iterate separately but simultaneously. Usability tests should be conducted before the coding begins. But, as described in Chapter 2 agile teams code from the outset of the project. Separating design iteration and development iteration is a good way to deal with this problem. Two parallel tracks are used, one for the UX team and one for the developers. As in the model previously described the

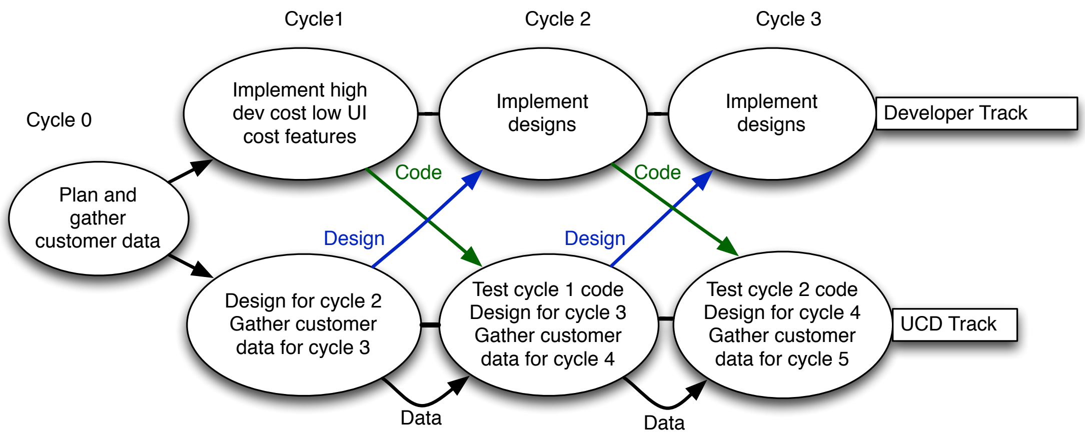

Cycle1
Cycle 2
Cycle 3
Cycle 0
Implement high dev cost low UI cost features
Implement designs
Implement designs
Developer Track
Plan and gather customer data
Code
Code
Design
Design
Design for cycle 2 Gather customer data for cycle 3
Test cycle 1 code Design for cycle 3 Gather customer data for cycle 4
Test cycle 2 code Design for cycle 4 Gather customer data for cycle 5
UCD Track
Data
Data

Figure 5.3.: Two parallel tracks one for the usability experts team and one for developers. Track separation allows iteration on designs [28].

project starts with a Phase 0 where the team gathers the requirements. A distinction is made between completely new products and the next release of an existing project. For a new product the first step is to analyse how users are performing their work. Methods like interviews or CI can be used to gather that data. From that data high-level prototypes should be designed and overall design guidelines should be defined. For an ongoing release the UCD team should analyse previous CI and usability tests. Based on that data the team should clarify the design goals through the upcoming iterations. All that work should be done in weeks instead of months and requires adjustments from usability experts, who are often used to longer periods from non agile UCD processes.

The main idea illustrated in Figure 5.3 is to design one cycle ahead of developers and to gather requirements two cycles ahead until the software is released.

In Cycle 1 usability activities should involve designing prototypes for Cycle 2 and conducting usability tests to enhance the designs. Furthermore, CI and interviews should be performed to examine the design for Cycle 3. At the beginning of the project UX experts usually need a few cycles time to conduct these tests, so developers should work on parts of the system which require no user interface design like coding the software architecture or implementing functionality with minor user interface.

In Cycle 2 developers can start coding the design done in Cycle 1. Close communication between the UX team and developers is recommended. Moreover in Cycle 2 prototypes based on data gathered in Cycle 1 should be created. Conducting usability tests of the implementation version from Cycle 1 is essential to include user feedback in the design. This cycle also includes CI to investigate the design for cycle 4.

Following the model the UX team is able to iterate designs before the implementation begins. However, it is not always possible to finish a complex design in such a short time. It is therefore necessary to split large designs into smaller pieces. A possible solution could be to create *mini-designs* that incrementally build on each other and add elements to the fundamental design. [28] In this case small pieces of design prototypes are created and usability testing is done in the separate UCD Track until the design is finished. Only if the overall design is well defined at the beginning of the project it can then be broken down into small pieces. To follow the agile culture where components build on one another, a UCD team starts with a fundamental design on which the small increments can be added on top of it.

6. Case Study - Catrobat

The practical part of this thesis was to improve usability in the Catrobat project, especially the Pocket Paint and Pocket Code sub-projects. After a short period of time it became apparent that we had to implement a process which puts the user in the middle of the whole development cycle like UCD, in order to achieve our goal. Furthermore we found out, that it is insufficient to integrate UCD in a project without paying attention to the software development method used. In order to succeed, the usability process needs to be adapted to the software development process.

Step by step I will describe in the following Chapters how the UCD was introduced, which usability methods we chose for our project and the problems that we were facing during the integration. But first, I will give readers an overview of the Catrobat project, the sub-projects Pocket Code and Pocket Paint and the ASD method used.

6.1. Catrobat

Catrobat[1] is a mobile visual programming language, having integrated development environments (IDEs) and interpreters for Android, iOS, Windows Phone, and HTML5 browsers. Catrobat and the software developed by the Catrobat team is a large-scale Free Open Source Software. The source code of the project is available on Github[2]. There are more than 25 sub projects, one for example on an Android stand alone application (apk) builder, another one on a Wi-Fi interface to Parrot's AR.Drone. According to Ohloh[3] statistics, 208 developers have spent 137 man years of effort on Catrobat as of January 2014. Catrobat

CATROBAT

[1]`http://catrobat.org/`

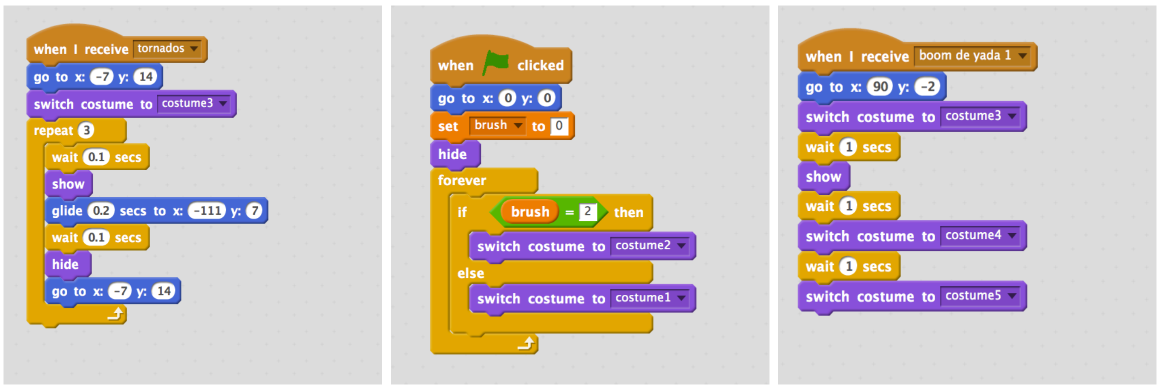
[2]https://github.com/Catrobat/

[3]`http://www.ohloh.net/p/catrobat`

is inspired by the Scratch[4] programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab [29]. Similar to Scratch, the aim of Catrobat is to enable children and teenagers to creatively develop and share their own software.

6.2. Scratch

Scratch is a visual programming environment developed by MIT that allows children to create interactive programs like animated stories, games or music videos. The main aim of Scratch is to introduce programming to users between the ages of 8 and 16 years with no previous programming skills. Programmers have to put together command blocks as shown in Figure 6.1 which control the graphical objects called sprites. The background where the sprites move is called the stage. To provide more flexibility sounds and images can be imported or created using a built-in paint tool and sound recorder. Scratch projects can be saved to the file system or shared on a website. For many users the first contact with the programming language is to explore code from existing projects provided by other users.



```
when I receive tornados
go to x: -7 y: 14
switch costume to costume3
repeat 3
    wait 0.1 secs
    show
    glide 0.2 secs to x: -111 y: 7
    wait 0.1 secs
    hide
    go to x: -7 y: 14
```

```
when clicked
go to x: 0 y: 0
set brush to 0
hide
forever
    if brush = 2 then
        switch costume to costume2
    else
        switch costume to costume1
```

```
when I receive boom de yada 1
go to x: 90 y: -2
switch costume to costume3
wait 1 secs
show
wait 1 secs
switch costume to costume4
wait 1 secs
switch costume to costume5
```

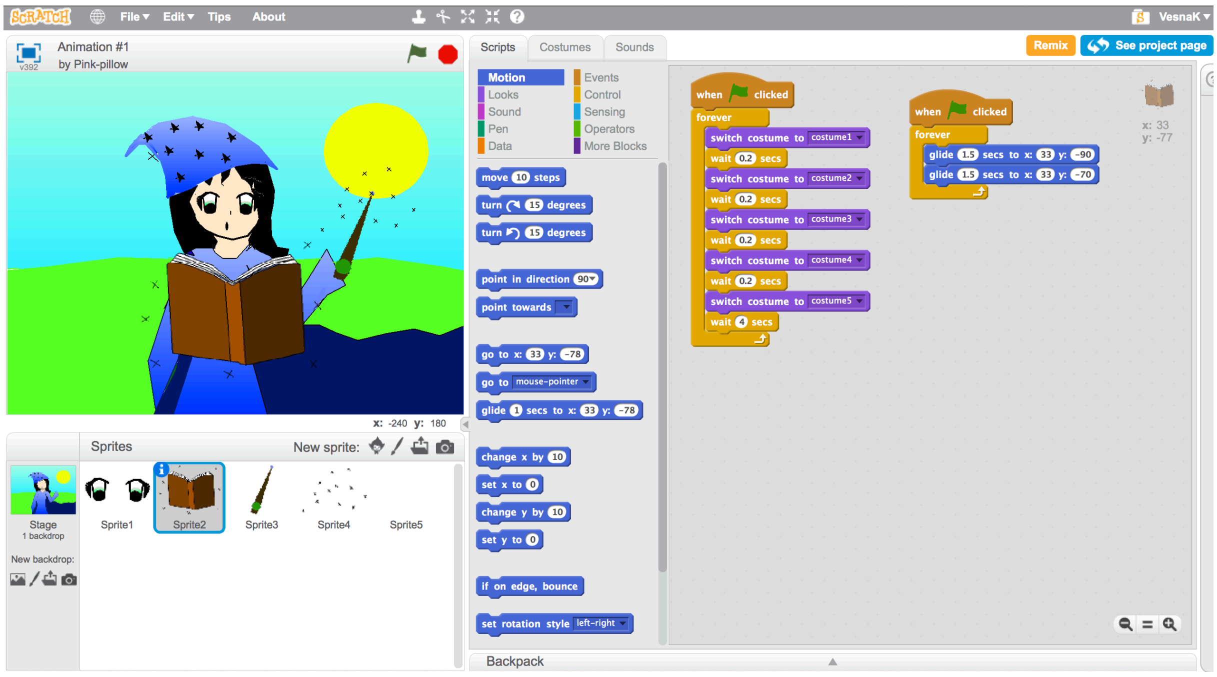
Figure 6.1.: Code example of the Scratch programming language.

Scratch consists of a single-window, multi-pane design which ensures that all key components are always visible. Figure 6.2 shows the programming environment with its four main panes, the scripts for the current sprite on the right, the command palette in the middle, the stage on the upper left and the pane with the thumbnails of all sprites used in the current project.

6.2.1. Functionality Of Scratch

Program fragment changes are immediately visible on the stage. Actually, users can change parameters or add additional blocks to a script while it is running.The programming language does not need any compilation steps. Maloney et all. [30]

[4]http://scratch.mit.edu/



```
when [flag] clicked
forever
    switch costume to costume1
    wait 0.2 secs
    switch costume to costume2
    wait 0.2 secs
    switch costume to costume3
    wait 0.2 secs
    switch costume to costume4
    wait 0.2 secs
    switch costume to costume5
    wait 4 secs

when [flag] clicked
forever
    glide 1.5 secs to x: 33 y: -90
    glide 1.5 secs to x: 33 y: -70
```

Figure 6.2.: Scratch: Single-window, multi-pane.

described Scratch in the paper "The Scratch Programming Language and Environment" as *tinkerable* which means that users can experiment with the programming language the same way as with mechanical or electronic components. In [30] the authors describe the advantage of *tinkerability* as "it encourages hands-on learning and supports a bottom-up approach to writing scripts where small chunks of code are assembled and tested, then combined into larger units."

The Main Differences Between Catrobat And Scratch Are:

- Support and integration of multi-touch mobile devices.
- Use of mobile device's special hardware (e.g., acceleration, compass, inclination).
- No need for a traditional PC.
- Availability on all major mobile platforms as well as HTML5 capable browsers.

6.3. Pocket Code

Pocket Code is a free, open source mobile visual programming system for the Catrobat programming language which allows users to create and execute Catrobat programs on Android, iOS, Windows Phone 8 smartphones as well as on HTML5 capable mobile browsers. It allows kids, starting from the age of eight to create storytelling and music animations or to develop their own games. Pocket Code is an IDE for a visual programming language, therefore it is a highly user-interface centric application (see Figure 6.3). Similar to Scratch programmers put command blocks, also called *bricks*, together to control graphical objects. Images and sounds can be imported or created using Pocket Paint, a built-in paint tool, and the integrated sound recorder.

Users can share Catrobat programs on the Pocket Code community website[5].

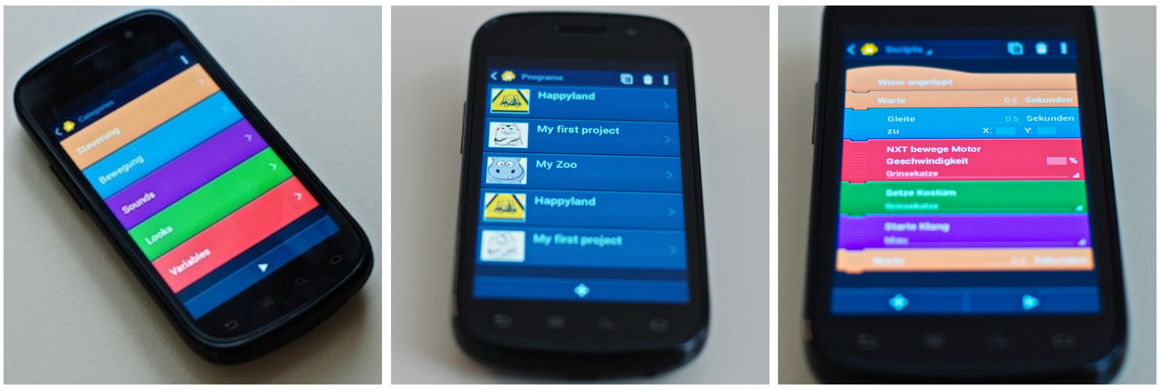


Figure 6.3.: Pocket Code for Android.

6.3.1. Formula Editor

Formula Editor is a very important part of Pocket Code where users are able to assign formulas, variables, sensor values, mathematic functions and so on, to the bricks. As I described before, the Catrobat programming language was mostly inspired by Scratch. Visual languages like Scratch use the visual approach, where even formulas are created with pre-defined graphical blocks. When complex formulas are involved, this latter approach can become unwieldy and even confusing. Programming languages like TouchDevelop[6], which are mainly textual, follow an approach

[5]`https://pocketcode.org/`
[6]`https://www.touchdevelop.com/`

where code and formulas are entered via an interface that looks like a pocket calculator. The authors of the paper *Comparing Purely Visual with Hybrid Visual/Textual Manipulation of Complex Formula on Smartphones* (see Appendix A) introduced Pocket Code with its Formula Editor, a new approach with visual programming but textual formula representation, which combines the ease of visual programming with the effectiveness and clarity of textual formula display. One of the reasons for introducing this new combination is that the Catrobat language is optimised for use on smartphones with touch screens and small display sizes, where Scratch-like blocks cannot easily be accommodated due to the restrictions of the screens and the difficulty to drag and drop blocks closely nested together with one's fingers compared to using a mouse.

6.4. Pocket Paint

Pocket Paint is an image editor application associated with Pocket Code that allows image creation and manipulation, setting parts of pictures to transparent and zooming in up to pixel level. It is integrated into Pocket Code but can also be used on its own.

6.5. Agile Process Used By Catrobat Team

Understanding agile values and its aims can be useful for usability professionals to collaborate with software developers and to promote usability within an agile team [2]. The following Section gives readers a brief overview of how ASD is adapted and used in the Catrobat Project.

The Catrobat team is using a customised version of the Extreme Programming (XP) method combined with the Kanban[7] system as shown in Figure 6.4.

I will describe the process by taking a closer look at the Pocket Code team working on the sub project for the Catrobat Android application. All other sub-teams are employing the following XP practices and values as well. The team consists of one coordinator and many developers. The coordinator manages the team and is responsible for planning games, weekly meetings and that the correct story cards are worked on, the planning games are coordinated, the weekly meetings are conducted, new members are assigned to a senior developer as a pair programming partner.

[7]`http://en.wikipedia.org/wiki/Kanban`

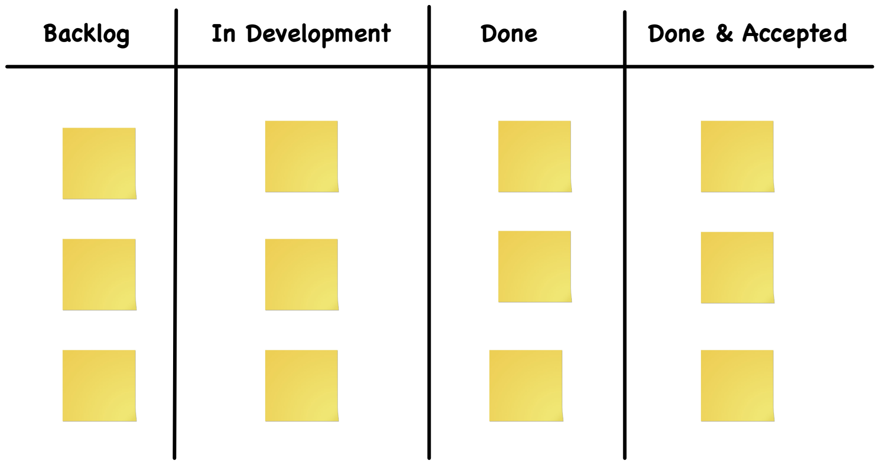


Figure 6.4.: Kanban system used by the Catrobat team.

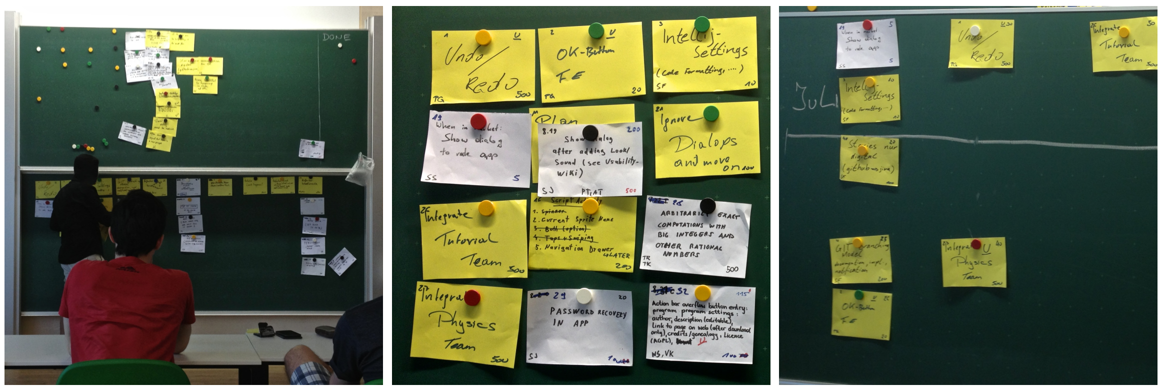


Figure 6.5.: Pocket Code's Planning Game.

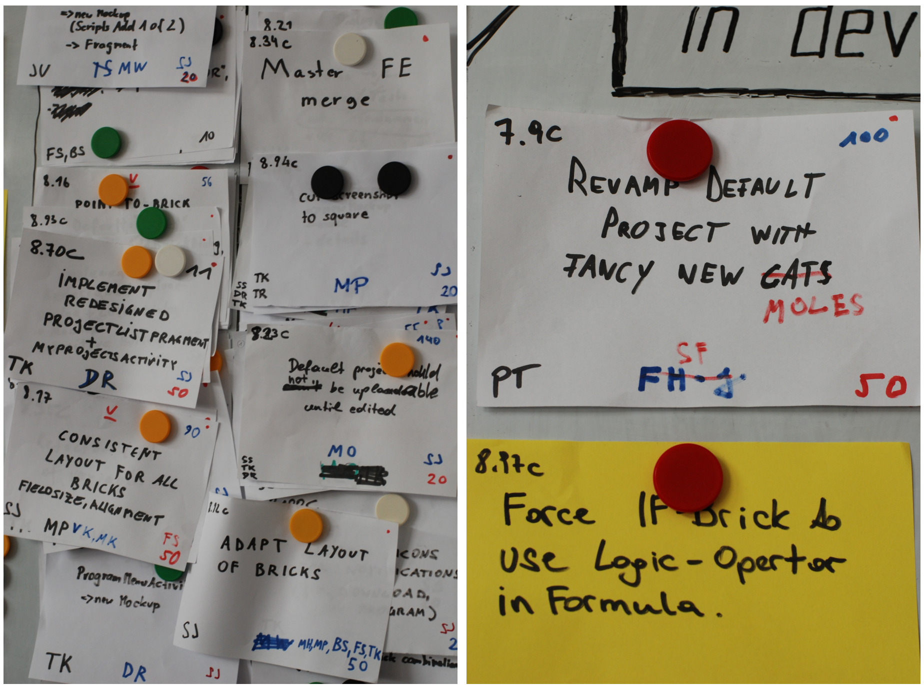
The Pocket Code team is able to sit together in a room, where all necessary infrastructure is available for the development the software. This infrastructure includes various whiteboards with story cards attached to them and the large charts, reflecting the current state of the project. The visible story cards sometimes lead to instant discussions among developers during development thus clearing any misunderstandings instantly and resulting in more collaboration among the team members. As the team sits together, face-to-face communication has resolved many difficulties in the project. The agile practices being used by the project teams are explained in the following subsections.

Planning Game: The planning game is held at the beginning of each release iteration. All Catrobat team members collaborate in the planning game rather than just the Pocket Code sub-team members. They discuss future ideas and the features to be implemented. Each feature is written on a story card, as shown in Figure 6.5 and 6.6. During the planning game, the team chooses and prioritises the story cards that will be implemented in the next release.

Story Cards: Story cards are written by developers, if possible with the support of the usability team. A story card (see Figures 6.6 and 6.7) must contain the following information:

- Release version and a story card number
- U: Represents an usability card. So the usability team has to accept it.
- Priority number (Lower numbers are higher priority).
- Description (What is the Story Card about?)
- Initials of the story card creator.
- Initials of the story card developer.
- Estimate amount of work. (*"roulette coins"* units, not time units)

The stories are then estimated by the developers and prioritised by the whole team. The story cards are essential for determining the project velocity. The completed story cards from the last iteration help to plan the next one in the planning game. We plan to use JIRA, a software tool for the visualisation of story cards so we have customised JIRA. As a result the story cards look exactly the same as they were before on paper.



8.34c
Master FE
merge
8.70c
Implement redesigned ProjectListFragment + MyProjectsActivity
TK DR
SJ 50
Consistent layout for all bricks
Fieldsize, alignment
MP VK,MK
FS 50
Adapt layout of bricks
SJ
MP
SJ 20
in dev
7.9c
100
Revamp default project with fancy new ~~cats~~ moles
PT
SF
~~FH~~
50
8.17c
Force IF-Brick to use Logic-Operator in Formula.

Figure 6.6.: Story Cards.

Meetings: We have weekly team meetings where the current status of the project is discussed, the next possible story tasks are reviewed, bugs are analysed and new members are introduced to the team.

In addition to weekly meetings there are also **bi-weekly meetings** where the coordinators from each team sit together and discuss current status of all the sub teams and their current progress, hence information is exchanged which could affect other teams like changes in the Catrobat-Language-XML.

Test Driven Development: All Catrobat teams use Test Driven Development (TDD) not just in the manner that tests are written before code is implemented but also the tests are considered as the documentation of the code and tests also specify the functionality of the software. Our rules concerning test cases are:

- You make a test before you start to implement a new function or fix a bug.
- You never commit something to the common repository which has not been tested and accepted.
- The test cases and the code of the main program you made have to be accepted by another developer (Core-Member).
- If code you implemented is not tested or is not cleanly written it will not be accepted.

Continuous Integration: In addition to TDD Pocket Code also uses continuous integration with the help of Jenkins[8]. There are three Google Nexus S devices and 5 emulators running 24/7 checking out the master branch and performing two hours worth of tests. These tests consist of user interface tests and unit tests written with the help of Robotium[9] and JUnit. Pocket Code uses a GitHub[10] repository where a single code base is maintained.

Coding Standards: For the coding standards and the general guidelines we follow our motto: we are the *"CodingMonks and CodingNuns of the Catrobat Church of Clean Code" (CoCoCCoCC).*
We:

- *Refactor and clean up the code for readability and understandability.*
- *Use no abbreviations.*
- *Use as few comments as possible (why? cannot easily be kept up-to-date)*
- *Use good, pertinent names of variables, methods, objects, etc.*
- *Try to make the code crystal-clear, self-explanatory, and we should be annoyingly thorough in ensuring this.*
- *Eliminate duplicate code.*
- *Leave no compiler warnings.*
- *Follow these guidelines not only for the main code but also for the test code.*
- *Cleanly use known design patterns.*
- *Eliminate all German language in the code.*

[8]`http://jenkins-ci.org/`
[9]urlhttps://code.google.com/p/robotium/
[10]https://github.com/Catrobat/Catroid

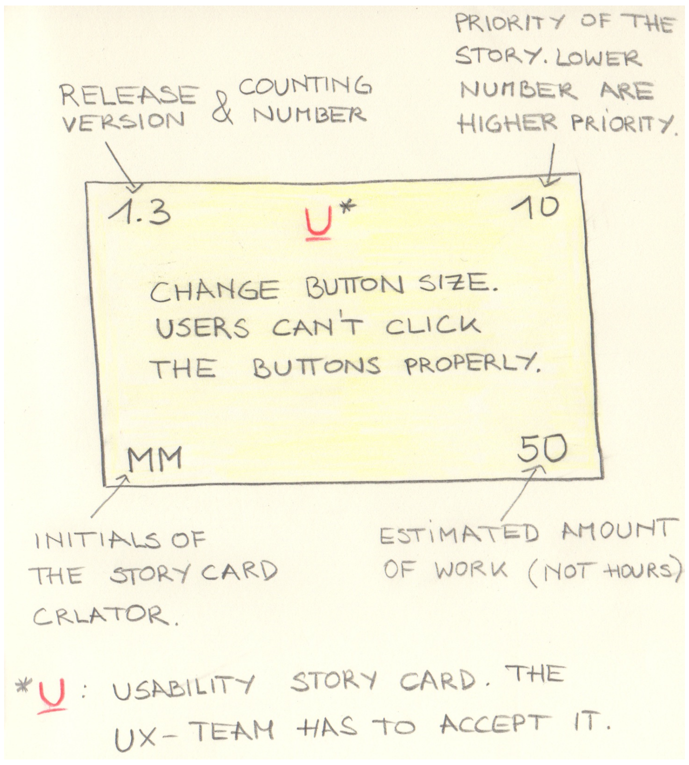

RELEASE VERSION & COUNTING NUMBER
PRIORITY OF THE STORY. LOWER NUMBER ARE HIGHER PRIORITY.
1.3
U*
10
CHANGE BUTTON SIZE. USERS CAN'T CLICK THE BUTTONS PROPERLY.
MM
50
INITIALS OF THE STORY CARD CRLATOR.
ESTIMATED AMOUNT OF WORK (NOT HOURS)
*U : USABILITY STORY CARD. THE UX-TEAM HAS TO ACCEPT IT.

Figure 6.7.: Story Card.

- *Add unit, regression, and functional tests so that the code and functionality is 100% covered.*
- *If something seems to be untestable, discuss it first with others.*
- *Have all the code (including test code) reviewed by independent team members (other than by those members who made a specific change in that part of the code.*

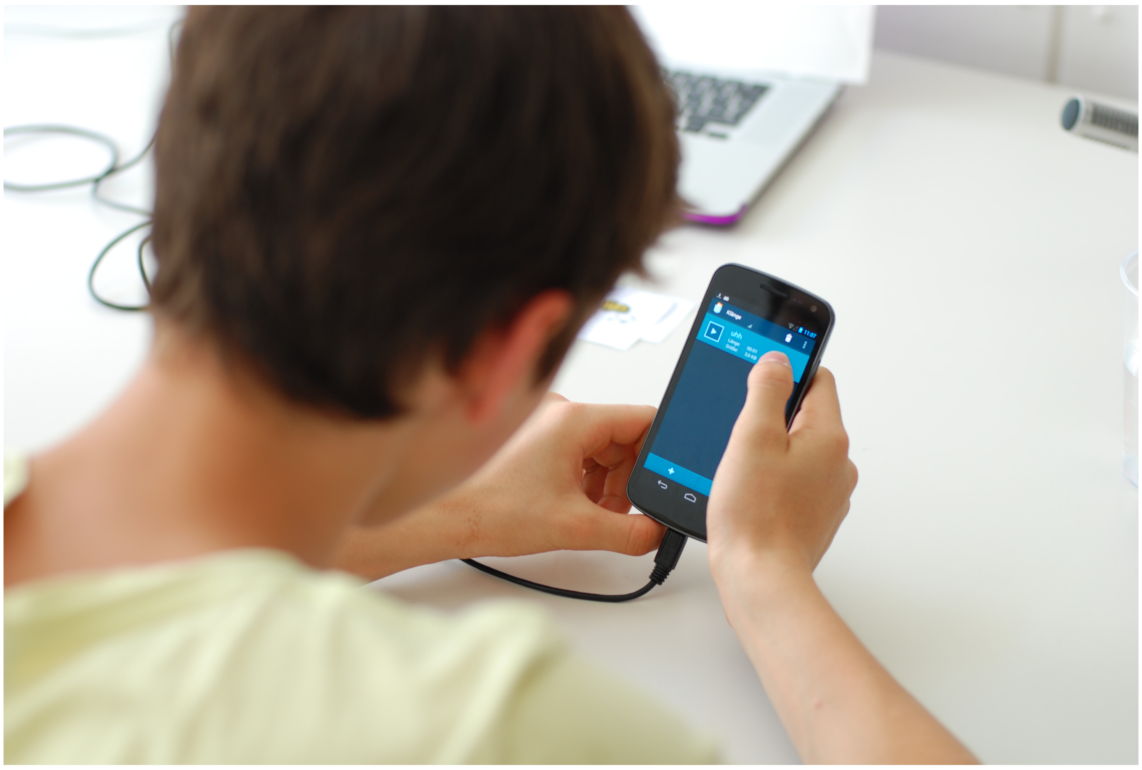
Collective Ownership: The code is on GitHub so that all team members can change every line of code, improve performance, fix bugs, refactor and so on whenever they get a chance to do so.

Refactoring: Usually, we refactor the code on the fly. There have also been refactoring releases, where the planning game was just about creating refactoring story cards.

Communication: Our team prefers face-to-face communication during development. For coordination the main developer communication runs over IRC. Wiki, Email and Google Calendar are other tools used for coordination. The Community communication is currently running over a Google group.

Retrospective/Reflections Meetings: Occasionally, we conduct a retrospective meeting to reflect on how things are working in our process and in the project as there is always a chance to improve. These meetings are held to listen to any suggestions for improvements to find out why things went wrong and what should be continued.

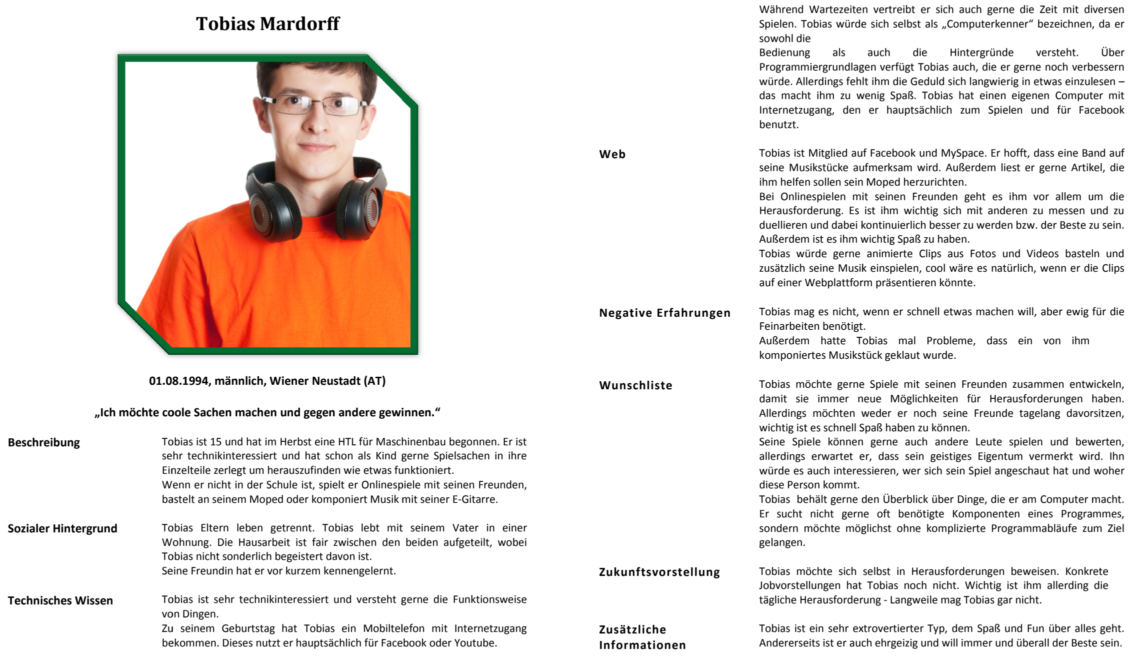
7. First Steps Toward Integration Of Usability In The Catrobat Project



7.1. Initial Situation

From the beginning the Catrobat project has been using the ASD method described in Section 6, without a focus on usability. The main goals of the project were to develop the Catrobat programming language, implement a programming environment for the language and background functionality without special focus on UI and the user needs.

Initially, some user research was done where three personas were created: Silvia a teenage girl, Tobias a teenage boy and one for a mother named Angelika. The purpose of personas is to help agile teams to get insights into the real end-users as



Tobias Mardorff

01.08.1994, männlich, Wiener Neustadt (AT)

„Ich möchte coole Sachen machen und gegen andere gewinnen."

Beschreibung	Tobias ist 15 und hat im Herbst eine HTL für Maschinenbau begonnen. Er ist sehr technikinteressiert und hat schon als Kind gerne Spielsachen in ihre Einzelteile zerlegt um herauszufinden wie etwas funktioniert. Wenn er nicht in der Schule ist, spielt er Onlinespiele mit seinen Freunden, bastelt an seinem Moped oder komponiert Musik mit seiner E-Gitarre.
Sozialer Hintergrund	Tobias Eltern leben getrennt. Tobias lebt mit seinem Vater in einer Wohnung. Die Hausarbeit ist fair zwischen den beiden aufgeteilt, wobei Tobias nicht sonderlich begeistert davon ist. Seine Freundin hat er vor kurzem kennengelernt.
Technisches Wissen	Tobias ist sehr technikinteressiert und versteht gerne die Funktionsweise von Dingen. Zu seinem Geburtstag hat Tobias ein Mobiltelefon mit Internetzugang bekommen. Dieses nutzt er hauptsächlich für Facebook oder Youtube.
	Während Wartezeiten vertreibt er sich auch gerne die Zeit mit diversen Spielen. Tobias würde sich selbst als „Computerkenner" bezeichnen, da er sowohl die Bedienung als auch die Hintergründe versteht. Über Programmiergrundlagen verfügt Tobias auch, die er gerne noch verbessern würde. Allerdings fehlt ihm die Geduld sich langwierig in etwas einzulesen – das macht ihm zu wenig Spaß. Tobias hat einen eigenen Computer mit Internetzugang, den er hauptsächlich zum Spielen und für Facebook benutzt.
Web	Tobias ist Mitglied auf Facebook und MySpace. Er hofft, dass eine Band auf seine Musikstücke aufmerksam wird. Außerdem liest er gerne Artikel, die ihm helfen sollen sein Moped herzurichten. Bei Onlinespielen mit seinen Freunden geht es ihm vor allem um die Herausforderung. Es ist ihm wichtig sich mit anderen zu messen und zu duellieren und dabei kontinuierlich besser zu werden bzw. der Beste zu sein. Außerdem ist es ihm wichtig Spaß zu haben. Tobias würde gerne animierte Clips aus Fotos und Videos basteln und zusätzlich seine Musik einspielen, cool wäre es natürlich, wenn er die Clips auf einer Webplattform präsentieren könnte.
Negative Erfahrungen	Tobias mag es nicht, wenn er schnell etwas machen will, aber ewig für die Feinarbeiten benötigt. Außerdem hatte Tobias mal Probleme, dass ein von ihm komponiertes Musikstück geklaut wurde.
Wunschliste	Tobias möchte gerne Spiele mit seinen Freunden zusammen entwickeln, damit sie immer neue Möglichkeiten für Herausforderungen haben. Allerdings möchten weder er noch seine Freunde tagelang davorsitzen, wichtig ist es schnell Spaß haben zu können. Seine Spiele können gerne auch andere Leute spielen und bewerten, allerdings erwartet er, dass sein geistiges Eigentum vermerkt wird. Ihn würde es auch interessieren, wer sich sein Spiel angeschaut hat und woher diese Person kommt. Tobias behält gerne den Überblick über Dinge, die er am Computer macht. Er sucht nicht gerne oft benötigte Komponenten eines Programmes, sondern möchte möglichst ohne komplizierte Programmabläufe zum Ziel gelangen.
Zukunftsvorstellung	Tobias möchte sich selbst in Herausforderungen beweisen. Konkrete Jobvorstellungen hat Tobias noch nicht. Wichtig ist ihm allerding die tägliche Herausforderung - Langweile mag Tobias gar nicht.
Zusätzliche Informationen	Tobias ist ein sehr extrovertierter Typ, dem Spaß und Fun über alles geht. Andererseits ist er auch ehrgeizig und will immer und überall der Beste sein.

Figure 7.1.: Personas.

described in Chapter 4. In our project, the personas slowly disappeared from the minds of developers, as no team member was responsible for keeping them alive.

At the time I joined the project only one team member was responsible for usability research. Each developer was designing the user interface fragment he or she was programming. The developers were not following any usability or design guidelines. The main focus of the project was to integrate new functionality instead of improving the existing user interface. Figures 7.2 and 7.3 show the initial UI of Pocket Code, the Catrobat Android application, and Pocket Paint.

Our first initiative towards the integration of usability into the Catrobat project was to put together a sub-team which mainly focused on usability research. So, we built a usability team consisting of two usability members.

Up to that point, only one usability test with ten end-users had been performed. The usability test conducted was a combination of the TA method with active intervention where two children work together on a task during the test. The participants were between eight and ten years old and the distribution between boys and girls was equal. For the test setup they used one external camera to capture the screen and the inputs on the touch screen and a second camera to record the participant's face. Additionally, the usability software Morae[1] was used to record device screen, the test users faces and sound together. The test device for the usability test was a Samsung Galaxy 10" tablet. During the usability test many serious usability

[1]http://www.techsmith.com/morae.html

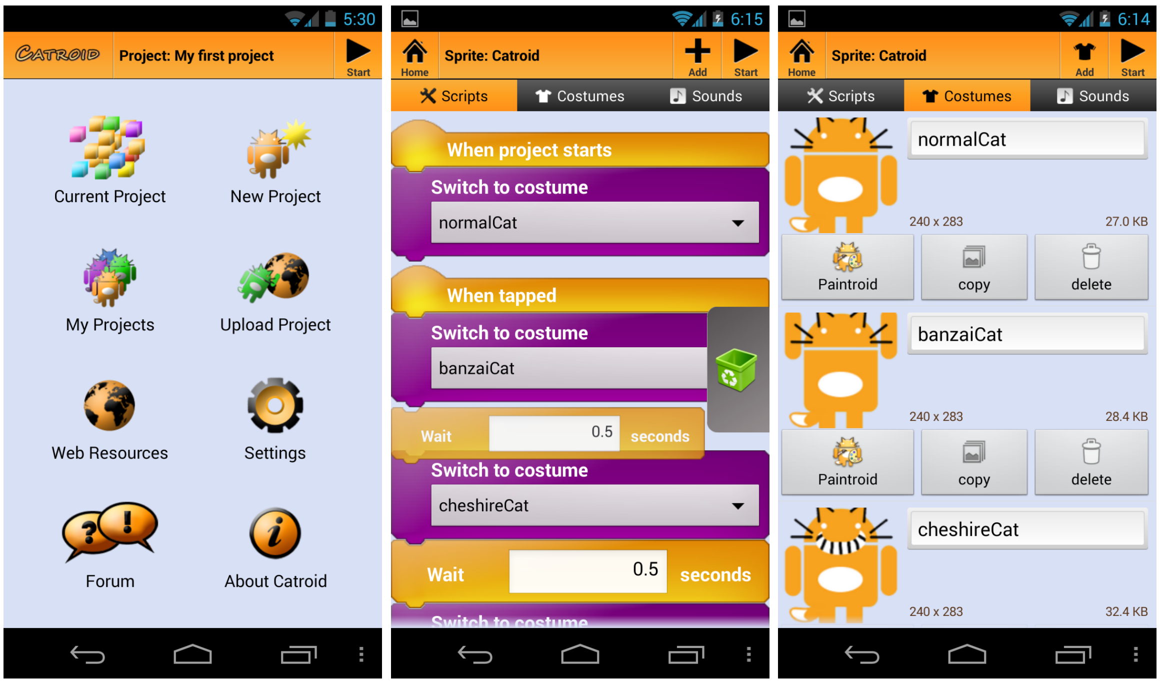


Figure 7.2.: Old UI of Pocket Code: start screen, scripts and costumes.

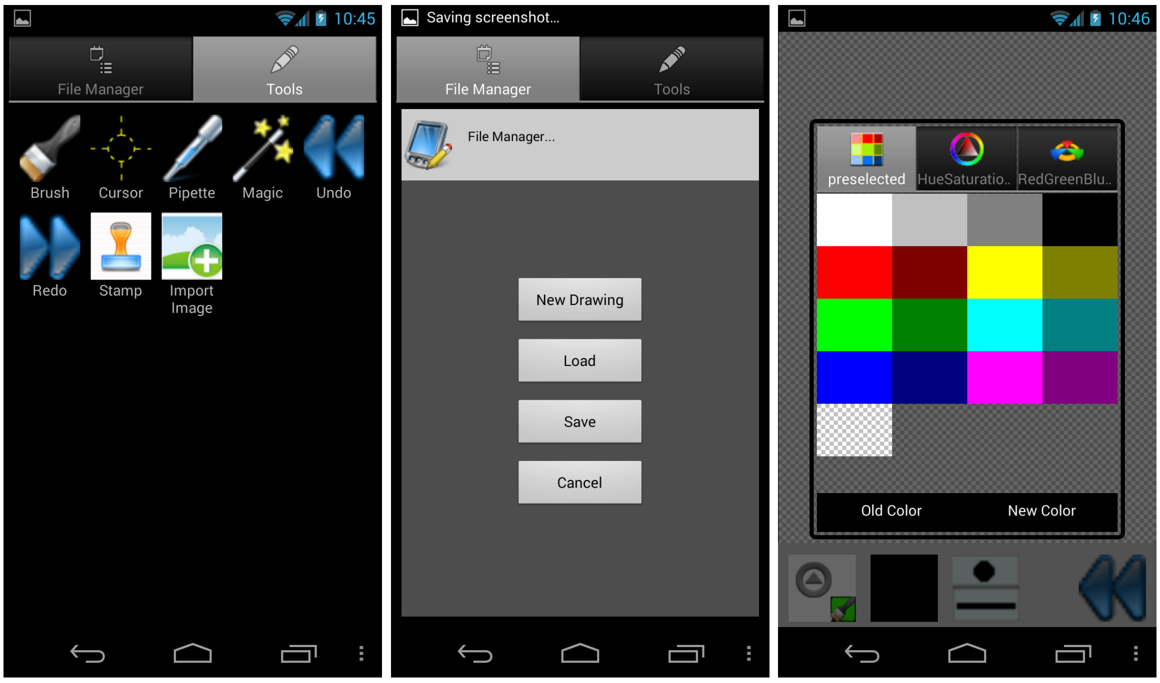


Figure 7.3.: Old UI of Pocket Paint: file management and tools screen, change colour screen.

problems were identified. [31] We were not sure if the problems were due to the age of our end-users or the app itself. Therefore we conducted a second test with teenagers.

Pocket Code is a programming system for children and teenagers between eight and seventeen years. We wanted to find out if end-users older than ten, would have the same problems with the application as the younger ones, so we decided to repeat the test with teenagers and used a smartphone with a smaller screen size as the input device because the software is not designed for a tablet.

7.2. First Usability Test Steps

To be sure that the test results described by [31] were also applicable for older users and not only for very young children, we conducted a further TA test. This Section gives an overview about the test methodology used and the main findings.

7.2.1. Test Method

The usability test described below differed in three important aspects from the first test [31]. First of all we simplified the test setup and attempted to test with less effort. We tried to create a test setup with less equipment but still good enough to gather data for later evaluation. Therefore, we decided to connect a smartphone device directly to a computer. We used Camtasia[2] to record the smartphone screen together with the face impressions of our users. The test setup used is shown in Figure 7.4.

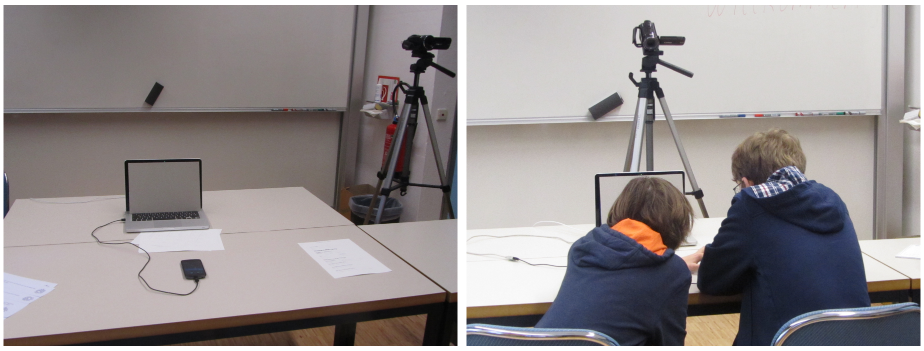


Figure 7.4.: *"Quick Test Setup"*

The second big difference in our test was the age of the test participants. This time we conducted the test with teenagers. The author of [31] strongly recommended the

[2]urlhttp://www.techsmith.com/camtasia.html

co-discovery method so we followed his recommendation and applied that method for our tests.

The last big difference was the device used for the test. The tested software was developed for smaller screens. So we decided to perform the tests also on a small smartphone screen instead of a tablet.

7.2.2. Test Goal

We tested the same Android applications with different tasks. The test goal was to observe if the same usability problems occur with teenagers as with younger children.

7.2.3. Test Setup And Equipment

The test was conducted at Graz University of Technology. We used one laptop and one camera to capture the face expressions and the smartphone screen (see Figure 7.4).

Table 7.1.: Hardware

Hardware:
Google Samsung Galaxy s4 5 inches screen size
Mac Book Pro 13”
Video camera

The smartphone was connected to the laptop with a USB cable and droidAtScreen[3] software, so the mobile phone screen was mirrored for the facilitator and captured with Camtasia, a screen capture software, for later analysis of the tests. The Mac Book camera recorded the user’s face during the test to get the participants’ facial expressions. Additionally an external camera recoded the whole test, including the interviews before and after the tests.

7.2.4. Tasks

The first task was an introductory one. Test users had five minutes to look around the application and to find out what they could do with that application. After finishing the task we asked them if they had any idea what the app is for. For the

[3]http://droid-at-screen.ribomation.com/

second task the participants had to browse to a specified game and play it. After that test the users had to download a program and to find out what was wrong with the program. Finally, they had to correct the error. For the last task test participants had to program their own game without any instructions. The tasks are summarised in Table 7.2.

Task	Description
T1	Schau dir Pocket Code an und finde heraus was du damit alles machen kannst. Have a look at the Pocket Code application. What is the app for?
T2	Spiel das Moorhuhn Spiel! Wer erreicht mehr Punkte? Play the ″Moorhuhn″ game. Who scores the most points?
T3	Gehe auf die Pocket Code Internetseite und lade dir das Programm ″Instrumenten Party - aber eines spielt falsch!″ herunter. Schau dich im Programm um. Finde heraus welches Instrument falsch spielt und korrigiere den Fehler. Go to the Pocket Code Website and download the program ″Instrument Party- but one is playing wrong!″ Have a look at the program. Find out which instrument is playing wrong and correct the error.
T4	Programmiere dein eigenes beliebiges Spiel. Hier kannst du machen was du willst. Program your own game. You can create whatever you want.

Table 7.2.: Tasks for the Thinking Aloud Test

7.2.5. Participants

We conducted the test with four teenage boys and two teenage girls, aged from 13 to 16. All test participants were familiar with Android smartphones. Four out of the six test users had their own Android smartphone, the other two were allowed to use their parents devices. Most of them used their smartphones for playing games, writing SMS watching YouTube and for Facebook.

7.2.6. Test Procedure

Before starting the test we clarified all the formalities like obtaining the parents' signed consent. For each test we followed the same test procedure as described below:

- Check all technical equipment. Start recording the test with both cameras. Check if the screen saver is off.
- Welcome the participants, introduce the test team and show them the test laboratory.
- Inform the participants about the test procedure and clarify that the software is tested not the users.
- Fill in the background questionnaire with the participants.
- Explain the test equipment.
- Explain what thinking aloud is.
- Start with the first introductory task which is always very simple like exploring the software.
- Start with the real test.
- Final interview.
- Feedback interview and questionnaire.
- Thank the participants and stop recording the test.

7.3. Main Results And Conclusions

Basically, all test participants liked the application. They were very surprised how easy it was to program their own games in a very short period of time. They liked the idea of creating something on their own.

Nevertheless, after only three user tests we found out that the applications' UI was too complicated and often hard to understand even for the older end-users. Test participants also did not like the design. So, a complete new design was necessary because of the test observations and because fixing problem is sometimes more time consuming than redesigning the whole application. It was a hard decision to make but in our opinion reasonable because of the test observations. We detected three different types of issues: usability issues, design problems and functional bugs.

I will not enumerate all usability issues detected since this would go beyond the scope of this thesis. The next two subsections describe only the main findings. Possible solutions and improvements for the important problems and issues will be discussed in Chapter 8.

Moreover, we realised that we should include more usability methods like paper prototyping, observation and heuristic evaluation in the project to improve the quality of the application. As we know from literature (see Chapter 5) UCD and ASD are two methods that work very well together. So, we decided to use UCD to redesign the Pocket Code and Pocket Paint applications.

7.3.1. Main Problems With Pocket Paint

- Participants did not understand the undo button. They assumed that it was a back button. (see Figure 7.5)
- Change colour button not understandable. (see Figure 7.5)
- Change tools button not understandable. It was not obvious for the participants, that they could access file management functions through this button. (see Figure 7.5)
- Change size and form of brush button not understandable. (see Figure 7.5)
- Participants did not understand the functionality of the stamp tool.
- It was not clear how to get back to Pocket Code.
- Participants had difficulties with saving the current file.
- Functionalities like load image were difficult to find.
- Icons did not fit to the functionality.
- Participants did not like the design of the UI.

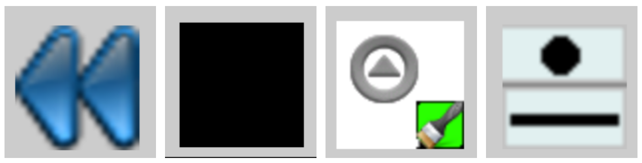


Figure 7.5.: Pocket Paint: back button, change colour button, select tools button, change brush size button.

7.3.2. Main Problems With Pocket Code

- Inconsistent Layout.
- As shown in Figure 7.6, the buttons for adding new scripts, objects and backgrounds (first one from the left) looked different to the buttons for new costume and new sound.
- Instead of using the add buttons users tried to add new items by clicking on the empty space below (see Figure 7.7).
- Connection between Script, Sound and Costume not understandable for almost all test users.
- How to add a costume/sound to an object is not understandable. Some users used the script ”set costume” to add a new costume to an object. Others added a costume in the correct way, after adding it they were surprised why the costume was not successfully added to the object.
- Some main functionalities can be reached through a long press in Pocket Code. Test participants did not often use ”long press”.
- Bricks cannot be selected everywhere.

- How to play an existing program was not intuitive.
- How to delete a brick was not clear.
- How to play a project was not clear.
- "When" bricks cannot be moved around like other bricks. This was irritating and frustrating for the test users.
- Participants did not know what "Pocket Paint" is.
- Not enough feedback after an action. Long press on a script should trigger a clearly visible action.

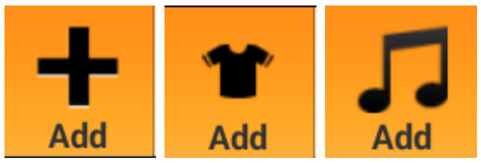


Figure 7.6.: Pocket Code: add scripts, background and sprites; add costumes; add sound.

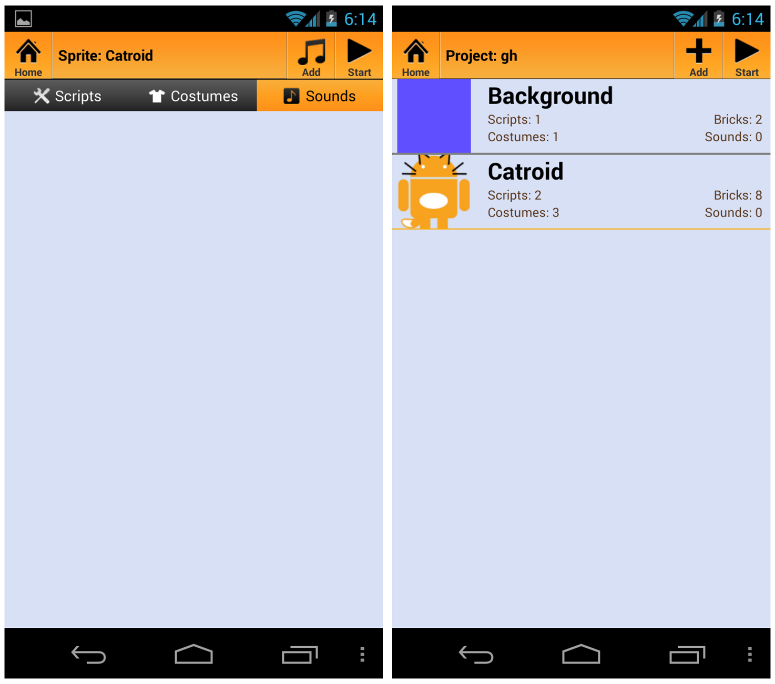


Figure 7.7.: How to add a new sound, background or sprite was not clear for the test users.

8. Agile User-Centered Design Process Approach

Due to the test results described in Chapter 7 the usability team started with the rework of Pocket Paint and Pocket Code by using UCD methodologies. The goal was to bring the user in the center of the software development process and to focus on usability. This Chapter describes the UCD process we followed. During the collaboration with the Catrobat Pocket Paint and Pocket Code sub-teams we started to adapt the UCD process used. It turns out that working on an agile project also allows the usability team to learn about agile culture and use to use agile techniques.

The main focus of my practical work was to adapt UCD processes to our project. To achieve this I had to choose proper HCI methods and had to conduct usability tests to get user feedback in order to be able to improve the applications acceptance and usability.

I was responsible for creating the main flows, menu navigations, low-fidelity paper prototypes and icons for Pocket Code and Pocket Paint. High-fidelity prototypes and icons have been created by designers, with the exception of Formula Editor which I designed.

This Chapter is structured as follows. First, I will describe the collaboration with the Pocket Paint team followed by Pocket Code and Formula Editor. I will highlight some of the adapted UCD steps and explain why they were necessary. For every sub-project I will discuss the challenges we faced and the solutions we found for the challenges.

8.1. Pocket Paint

Pocket Paint was the first sub-project to be supported by the usability team. Our aim was to redesign the application and improve its usability. From the very beginning we were able to communicate face to face with all developers, because the team consisted of only three active developers. Very soon after the beginning of our collaboration the usability team started to hold weekly meetings in the project room. Later, we held our weekly-meetings together with the Pocket Paint team to discuss the development steps of the project.

The following subsections describe one UCD cycle with several iterations. We started with the analysis phase, followed by design and implementation phases and finally the validation phase. Throughout the whole process we focused on end-users, introduced appropriate HCI methods, provided frequent usability tests and worked very closely together with developers and other project stakeholders. We also adapted the UCD process to fit into the ASD used by the Catrobat project.

8.1.1. Analysis Phase

According to the user feedback and test results, Pocket Paint needed a new design. The UCD team first met with key stakeholders including the project head, line manager, coordinators, senior developers and developers in order to discuss the project vision and the usability test results outlined in Chapter 7.

As the project did not take usability into account from the beginning, the developers were not used to involving the end-user in any design decisions. So, it was very important to give them an understanding of usability and to explain why we needed to improve Pocket Paints' usability and design a new UI for it. Once, we had demonstrated them what the biggest problems were for the end-users, we developed usability goals and objectives together, wrote new story cards and marked any which had to do with usability or design as *usability cards* with a red U (see Figure 6.7). We decided that all *usability cards* must additionally be accepted by a usability team member. This decision was applied to all Catrobat sub-teams.

Within the results of the conducted usability test described in Chapter 7 we identified different groups of problems, usability issues, design problems and functional bugs. Implementing background functionality without user interaction did not need support from the usability team, therefore developers started working on such story cards, while the usability team conducted a short user and task analysis.

We did not test Pocket Paint as a stand alone application. We assumed that most end-users would primarily use it together with Pocket Code and that their main objective would be drawing or manipulating objects and backgrounds for their programmed games, storytelling or music animations.

User Groups: We analysed the end-user profiles and checked if the existing personas were still valid. We observed from usability tests and from an analysis of Scratch programs that girls usually program different things than boys. Girls tend to create stories or animations using people and animals for their programs while boys mostly program action games using vehicles, explosions and action figures. Therefore, we concluded that at least the media data provided by Pocket Paint should be different for boys and girls, which could probably help with the loss of interest of girls in computer science. [32], [33] found out that girls tend to loose interest in math and science related disciplines, including computer science during middle school. We assume that girls have different needs to boys while programming. Moreover we think that younger kids from the age of 8 to 11 need also different UIs than older ones because of their lack of familiarity with different concepts (e.g. pocket calculators) and that to much complexity would be discouraging for them. We categorised our potential end-users in the following four groups:

- Girls between 8 and 11 years.
- Girls between 12 and 17 years.
- Boys between 8 and 11 years.
- Boys between 12 and 17 years.

8.1.2. Design And Implementation Phase

Generally, design and implementation are two separate phases in UCD. We were also aware that agile teams do not spend too much time on planning or designing software; they code from the outset. So, designing the entire application in advance was not possible. We had to split the complex design in smaller pieces, so that the usability team and the developers could work in parallel in short iterative cycles. First of all we did some brainstorming on different design concepts together with the developers.

Low-Fidelity Paper Prototypes: After the brainstorming session, the usability team created first low-level prototypes with paper and pencil as shown in Figure 8.1. We developed first screen flows and a possible navigation model (see Figure 8.1 and 8.2).

To get feedback for further development as early as possible we performed expert reviews and did in-house usability test with students. We used the feedback from our colleagues to modify the low-level prototypes. Afterwards, we tested the adjusted prototypes with real end-users. Our paper prototype test sessions were very short, not recorded and were not held in usability test laboratories due to lack of time, because we needed quick results.

We showed the test participants our prototypes and asked them to answer questions such as:

"Which button would you click if you want to change the brush size?"
"What do you think the top bar represents?"
"What would you do if you want to go back to Pocket Code?"

Sometimes we modified the prototypes during the usability tests together with the participants. At the weekly meetings, we discussed new insights from user feedback with developers and wrote new story cards if necessary.

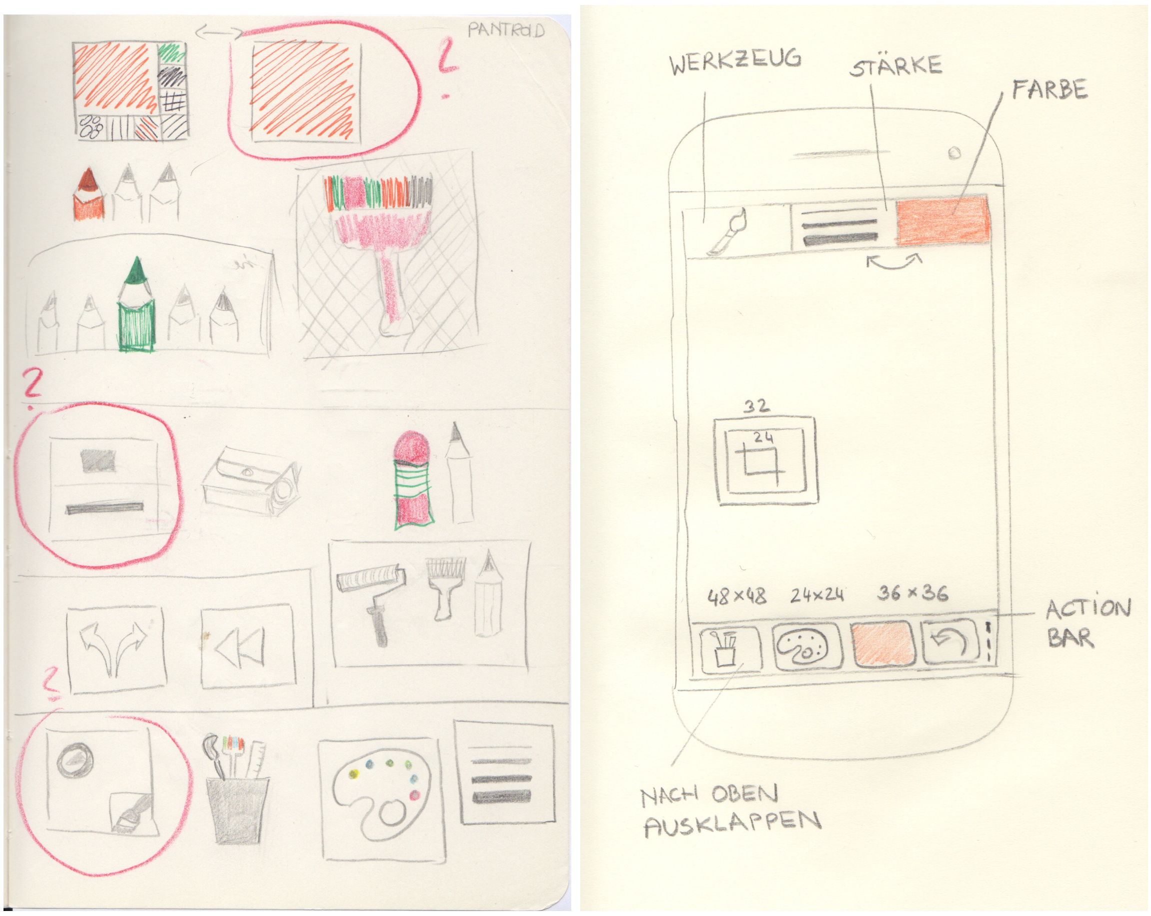


Figure 8.1.: Pocket Paint first low-level prototypes created with paper and pencil.

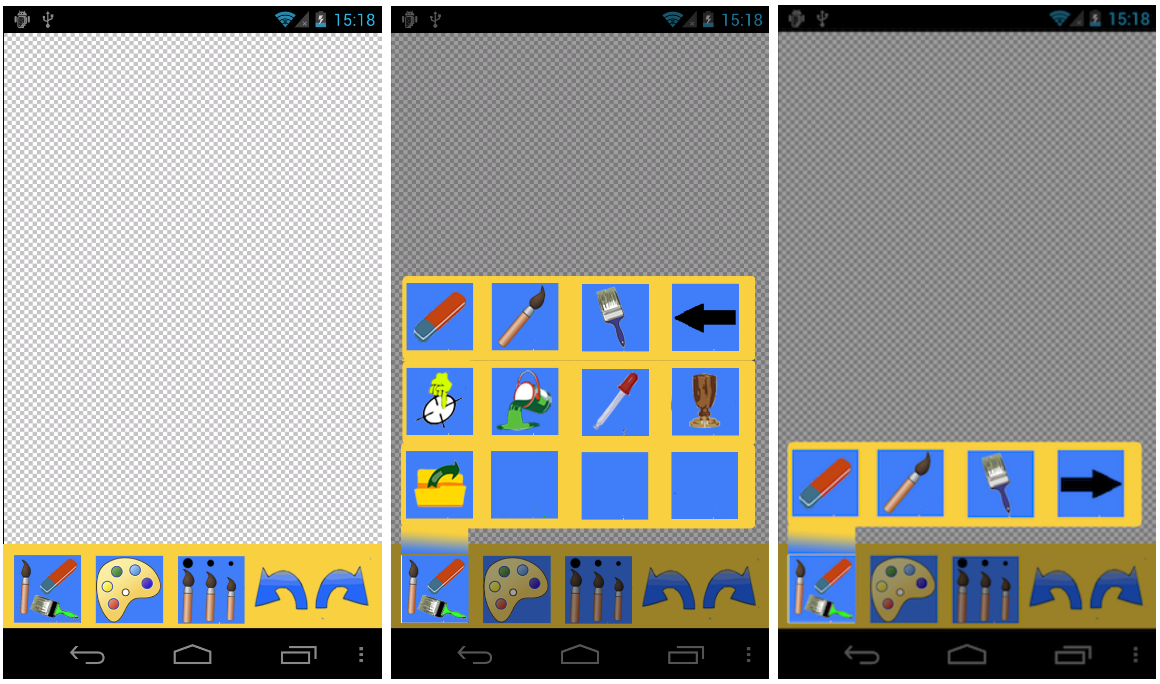


Figure 8.2.: Pocket Paint first sketches Overview.

Requirement Changes: As described in Chapter 2 changes are welcome in an ASD project are. In the middle of the design phase the key stakeholders decided to develop separate applications for all user groups identified. The first application had to be adjusted for boys between 12 and 17 years. Therefore, we had to adapt the prototypes to the new user group.

Guidelines: For the new design we defined some fundamental usability guidelines. We especially used the usability guidelines for mobile applications recommended by Nielsen et all.[22].

We followed the YAGNI[1] (You aren't gonna need it) and KISS[2] (Keep it simple stupid) principles. From paper prototypes to high-fidelity design, we wanted to keep the design as simple as possible and implement only those features which were necessary and had top priority.

We studied the Android specific guidelines[3] and looked closer at features like *Action bar* containing *App icon, View control, Action buttons* and *Action overflow*.

In order to find out if the end-users were familiar with Android specific features we applied a modified version of CI and observed 13 boys between 13 and 17 years. We visited the boys in their own environments like parks, youth centres or homes where they spent their free time. All of them had their own Android smartphones. First, we asked them what they mostly did on their smartphones to find out which applications they were used to. Afterwards, we asked them to use some of the applications they knew well on their own Android mobile phones.

The goal of this observation was to gain insights in the way the participants used applications designed for Android mobile phones with specific design solutions like Androids' action bar or the functionality of the back button. We found out, that almost all test users were able to use the back button or knew the meaning of an action overflow. So, we decided to follow some Android guidelines and used the action bar with its features and followed the guidelines for dialogs and button sizes.

UX Team: In the Catrobat project there existed an inactive design team with only one designer. We started to collaborate with the designer and involved him in all our design decisions. After a while the design and usability team became one, so we renamed our usability team to UX team. We started to apply the same team structure as all other Catrobat teams had, so we appointed one team member as usability coordinator and one team member as design coordinator. Some time later our team rose to 4 team members, two designers and two usability team members.

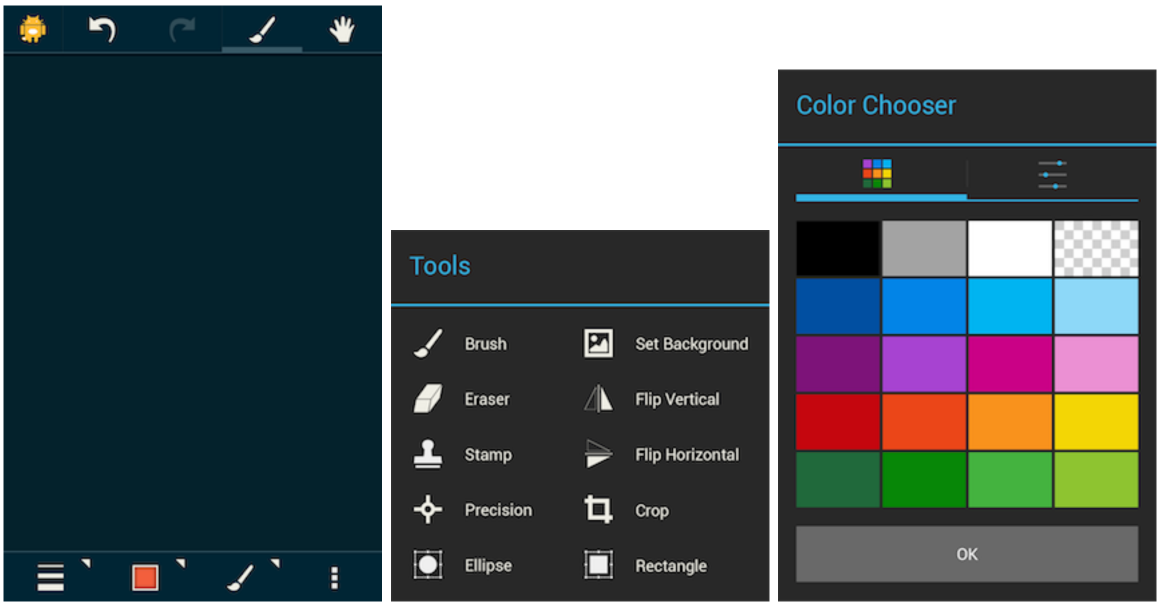


Figure 8.3.: Final Pocket Paint high-fidelity design.

High-Fidelity Design And Implementation: The next step in our process was to create high-fidelity detailed designs (see Figure 8.3). To evaluate the high-fidelity mockups and first implementations we applied expert reviews and heuristic evaluation as described in Section 4. All usability and functionality issues detected during the usability tests were discussed with developers. Again we wrote new story cards together and defined usability cards.

Communication And Coordination: We preferred face-to-face communication during the whole release cycle. We used Dropbox and Google Docs to exchange prototypes and documents. The main coordination communication tool was Email. Throughout the design and implementation phase usability team members, designers and developers worked closely together.

8.1.3. Validation Phase

The implemented features were again evaluated by the end-users to get their acceptance and further feedback. In the course of an intermediate TA test of Pocket Code we had the opportunity to test the implemented version of Pocket Paint with five end-users. The main goal of the usability test was to get user feedback and to detect usability issues. A further aim was to find out how well the two applications worked together. Figure 8.4 shows the final version of Pocket Paint.

[1] http://en.wikipedia.org/wiki/You_aren't_gonna_need_it
[2] http://en.wikipedia.org/wiki/KISS_principle
[3] http://developer.android.com/design/index.html

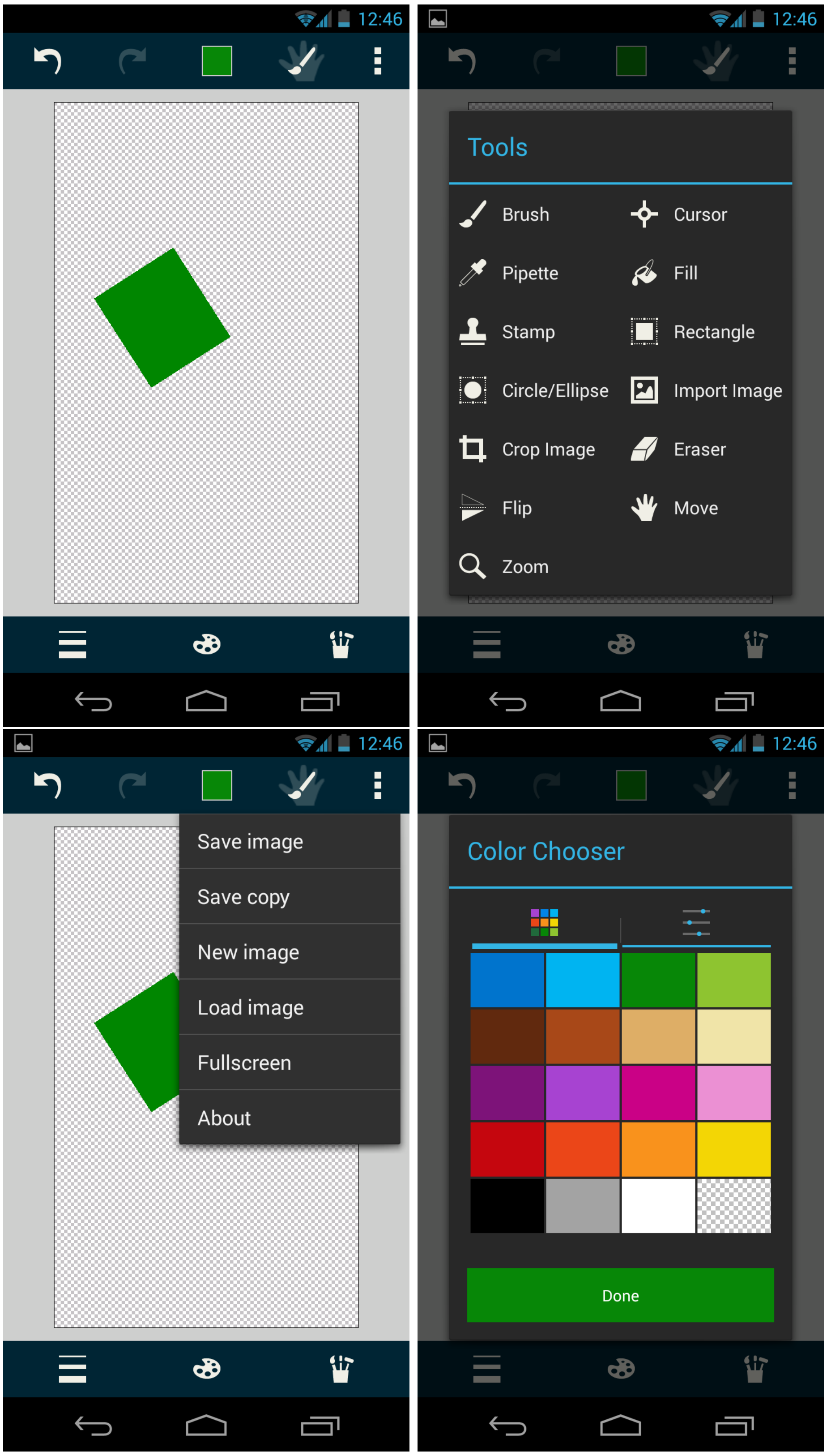


Figure 8.4.: New design of Pocket Paint.

Main findings of the test were:

- Some icons were still ambiguous for the end-users.
- Tools needed an additional textual description.
- Save image, import image, not clear enough.
- Connection between Pocket Code and Pocket Paint not clear enough.

After the TA test we started a new iteration cycle and began with the creation of new low-level paper prototypes.

The final release validation of Pocket Paint[4] will be discussed in Chapter 9.

8.2. Pocket Code

The usability team simultaneously worked on Pocket Paint and Pocked Code for Android, Catrobat's main sub-project. Redesigning Pocket Code was a big challenge for all team members because it is a huge, complex, free open source, large-scale project. The developer team consisted of at least 15 developers, so face-to-face communication was complicated. We experienced problems like lack of communication and collaboration or lack of visibility of our work. To address these conflicts we tried to apply best practices described in Section 5 for the collaboration between usability experts and agile teams. Some of the recommended practices worked very well, but some of them could not be applied to our project. The usability team learned a lot about ASD while working together with Pocket Code team. The following subsections describe our approach for the collaboration between the usability team members and the developers. I will again highlight the UCD methods we introduced and the way we adapted them to become more agile.

8.2.1. Analysis Phase

Pocket Code is the programming environment associated with Pocket Paint and has the same end-users, so additional user analysis or user studies were not necessary. Rapidly, we started working on the core application design. The process flow and main navigation was much more complicated than for Pocket Paint, so we needed much more time for initial designs. The issue list detected by previous usability tests was very long and we had to keep different aspects like different user groups, simplicity or for example expandability of the software in mind during the redesign of the application. Again we met with all team members who were interested in the discussion about the redesign of the application. We discussed the opportunity to include some standard Android functionalities like the action bar, standard lists and dialogs in our new design. We planed the software release, and decided together

[4]https://play.google.com/store/apps/details?id=org.catrobat.paintroid

with all stakeholders to stop implementing further functionality until the new design had been developed. During the design phase developers were able to do code refactoring and bug fixing.

After that meeting the UX team started to brainstorm about the high-level system concept, the app navigation and main screens. One important step was having a look at and analysing other Android applications to get an overview about the behaviour of Android specific applications.

8.2.2. Design Phase And Implementation

Right before we started to develop the new system the Catrobat core team decided to create separate apps for every user-group as described in the previous section 8.1. The first version of the app should be especially designed for teenage boys. So, we were able to concentrate on the proper user-group from the beginning of our design process.

Low-Fidelity Prototypes: The first low-fidelity prototypes we created did not contain icon design or colour selection (see Figure 8.5). We tried to focus on usability guidelines, especially usability guidelines for mobile apps, to make the software more comprehensible and usable.

Again, we tested our first low-fidelity prototypes with our colleagues from other sub-teams in order to detect significant problems. The feedback from other students was always very useful, because they had knowledge about programming languages and were able to detect logical and conceptual issues. But, we did not want to rely solely on our opinions or the opinions of our pees. Therefore, we conducted observations with teenage boys as well. We found out that participants without any previous knowledge about our software or at least knowledge about Scratch were overwhelmed by the prototypes. We got the best feedback from participants who were familiar with older versions of our software.

Communication And Collaboration: The first team conflict arose because the UX team members did not communicate enough with developers and did not involve them in the design process. The work of the UX team was not visible for the Pocket Code team because our collaboration was not close enough. We had been so busy creating the new design, that we forgot all about agile culture what we had learned before. Only the project head and some senior developers were involved in all our design decisions because we often brainstormed together. To deal with this problem we started to share our prototypes on Dropbox and via Google Docs. As it turned out later, neither Dropbox nor Google Docs were the right technologies to work

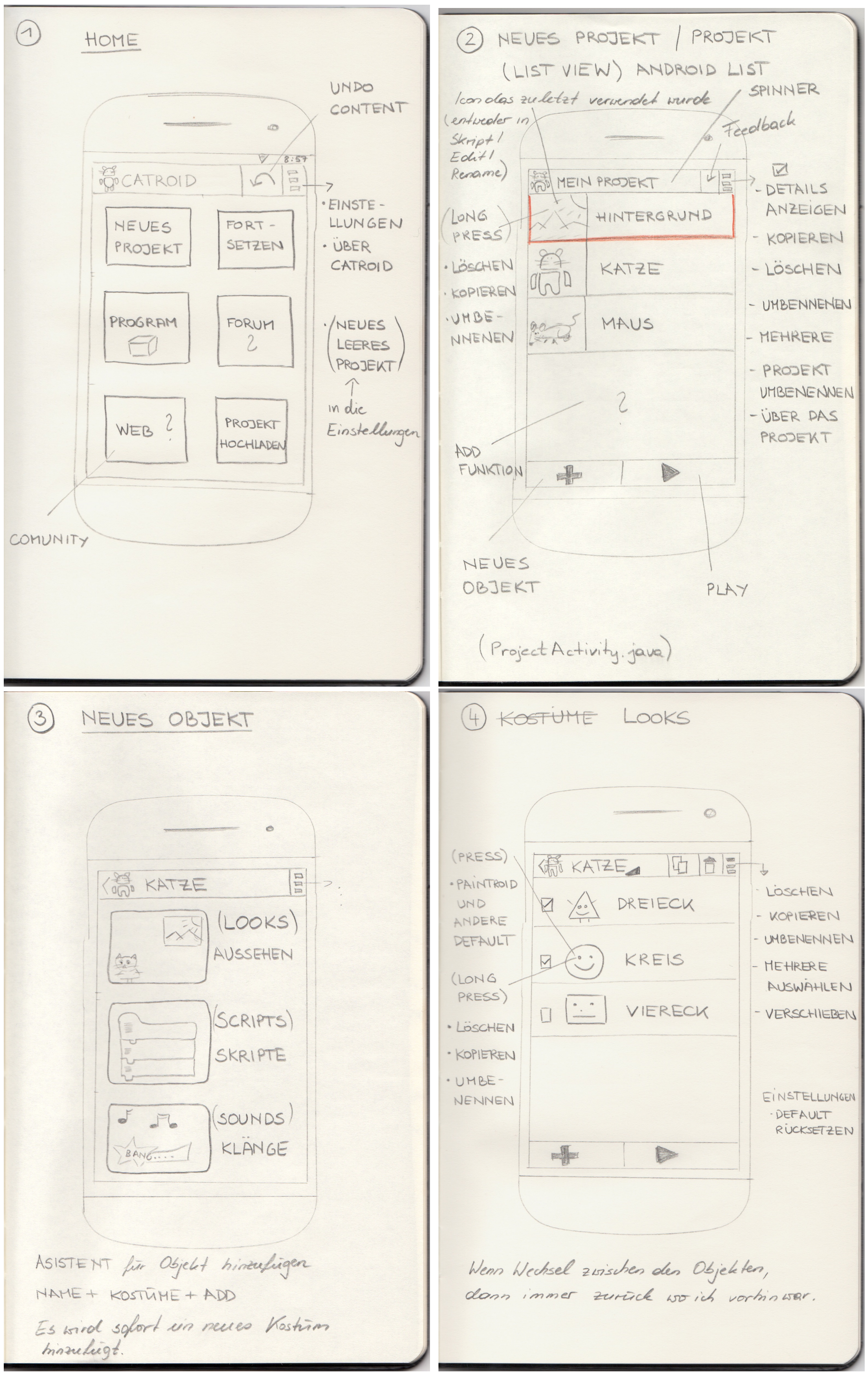


Figure 8.5.: Pocket Code: first sketches for main menu, new project screen, new object screen and looks.

with, so later on we tried to talk directly to all team members and worked in the project room at regular intervals. For design exchange we used GitHub[5].

UX Whiteboard: To show the developers what we were currently working on, the UX team started to use a whiteboard with its own design and usability story cards. We did not have enough space in the project room to use a separate whiteboard for each sub-team, so we used only one whiteboard for all sub-projects. UX team story cards look the same as the developer story cards (see Section 6). We also introduced a modified version of a *"burn down chart"*[6] where we tracked our progress against the release plan. We summed up the estimated amount of work of all story cards planned for the release and updated the sum once a week. It was a good way for the team to clearly see what was happening and how fast progress was being made.

Prototypes For Developers: The prototypes we created next (see Figure 8.6), were mainly intended for developers and internal team members. The main goal was to exactly define which features we were about to use, so that developers could start with the implementation of background functionalities. The design was again kept to a minimum to avoid discussion about fonts or icons.

To remove ambiguity, we met all stakeholders, developers, senior developers, the line-manager and project head and presented the prototypes to them (see Figure 8.7). We went through all mockup screens and got feedback from them, discussed potential problems and improvements. Furthermore, we discussed which functionalities should be customised and where we could use standard Android functions.

We did not test the prototypes with end-users because they were very similar to the low-level mockups created before with paper and pencil. The only method we applied to evaluate the prototypes was an expert review.

High-Fidelity Prototypes: The step from low-fidelity prototypes without design focus to high-fidelity prototypes as shown in Figure 8.8 was very time consuming. The whole UX team worked on those tasks. The team adapted designs according to the user feedback, created icons, chose colours, drew the prototypes with appropriate graphic programs and so on.

At this design stage we focused solely on the application design and disregarded the agile culture or the agile team we worked with. Some of our new UX team members did not know anything about ASD or agile UCD, but we did not have the time to deal with this problem, because the release date was drawing nearer. We did not hold our weekly meetings frequently in the project room which resulted in exclusion and lack of collaboration with the developers. We did not communicate frequently and also did not show them interim design results. Again, only the

[5] `https://github.com/Catrobat/DesignAndUsability`

[6] `http://en.wikipedia.org/wiki/Burn_down_chart`

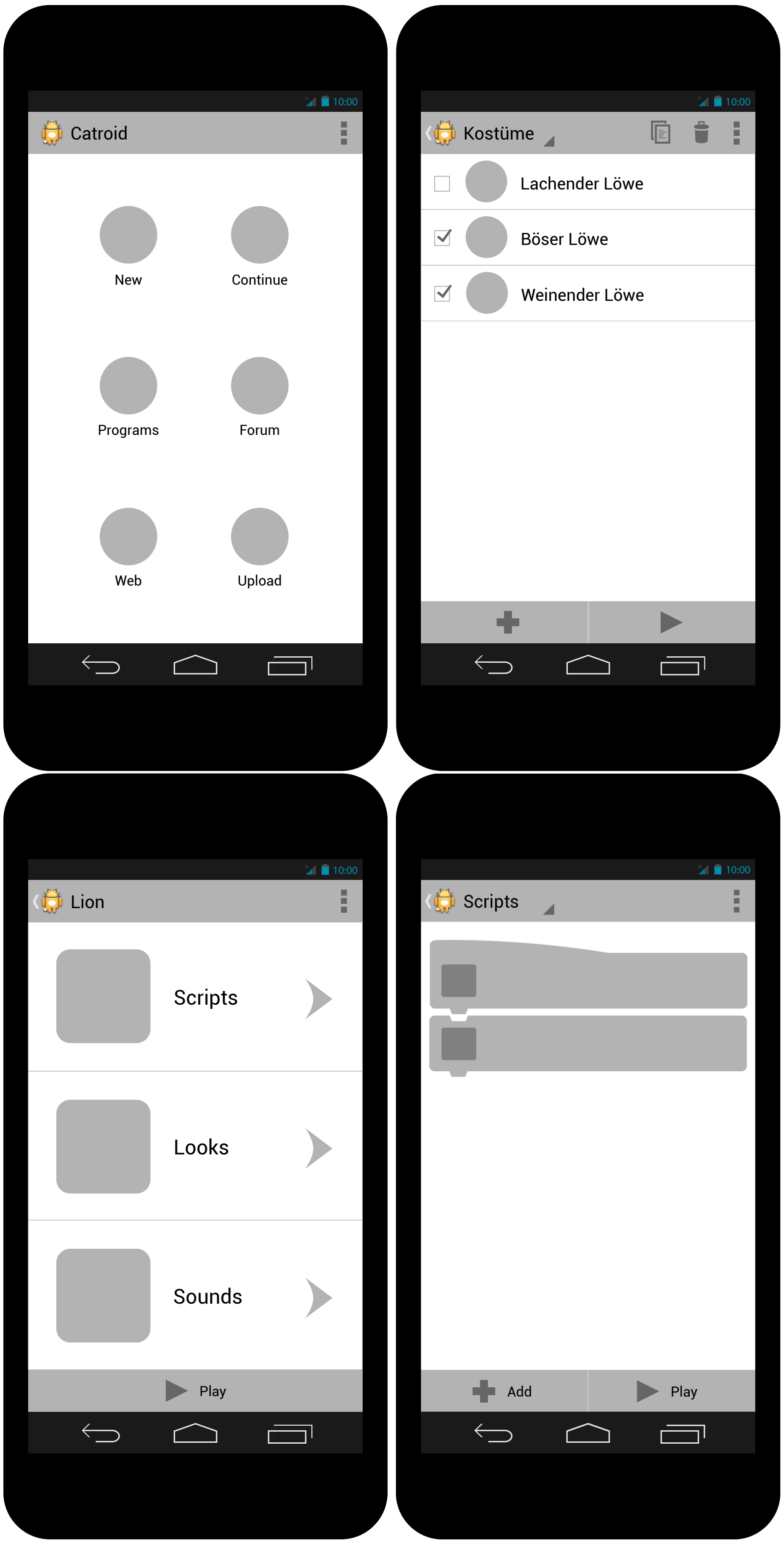


Figure 8.6.: Pocket Code prototypes created especially for developers.

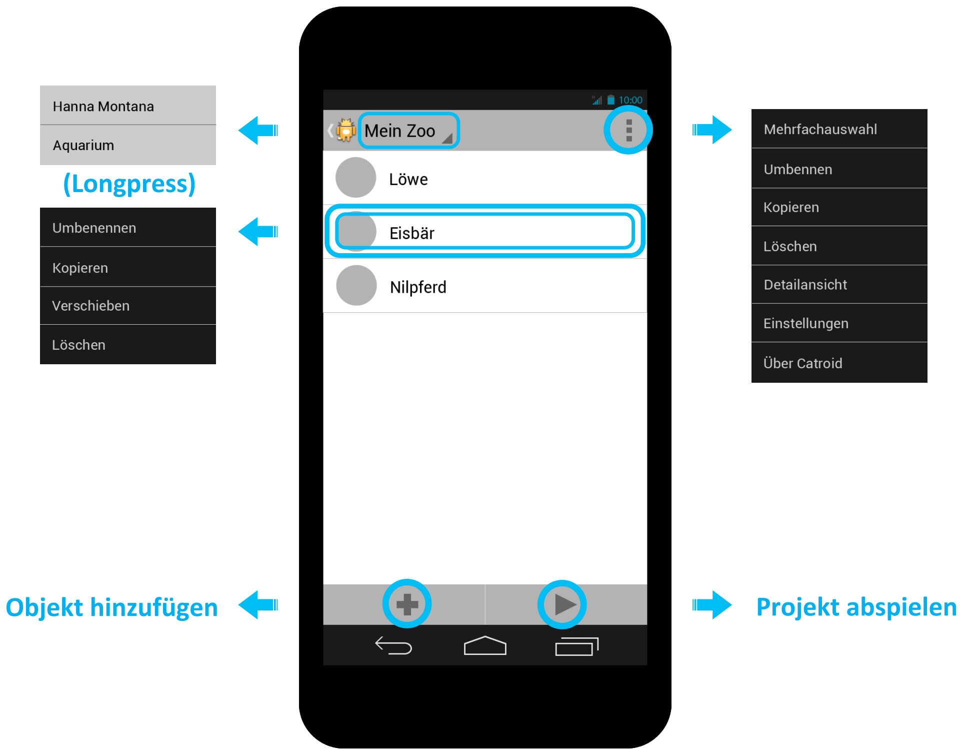


Figure 8.7.: Pocket Code: Mockups.

project head and some of the senior developers were involved in our designs, which lead to misunderstandings. ASD favours light-weight just-in-time work instead of extensive up-front design, so the developers did not accept our long design cycles. While the UX team wanted to create a finished design and detailed tested mockups, the developers expected rapid prototyping and face-to-face communication during the implementation, so conflicts were inevitable.

We learned from this experience and tried to improve our communication and collaboration by making our work more visible. We shared our designs regularly with developers, even if it meant that they had to reimplement things twice when something changed and supported them during the implementation.

We decided to use Dropbox for internal communication between the UX team members, to share ideas, sketches, mockups and so on, and GitHub[7] for the exchange with developers because Dropbox had proven to be the wrong decision for interacting with developers as mentioned before.

Design Wall: To be even more visible in the project room, we created a design wall with actual mockups, sketches, design proposals and so on (see Figure 8.9). It was very important to keep the design wall up to date, and to update it frequently in order to demonstrate the ongoing design progress.

[7]`https://github.com/Catrobat/DesignAndUsability`

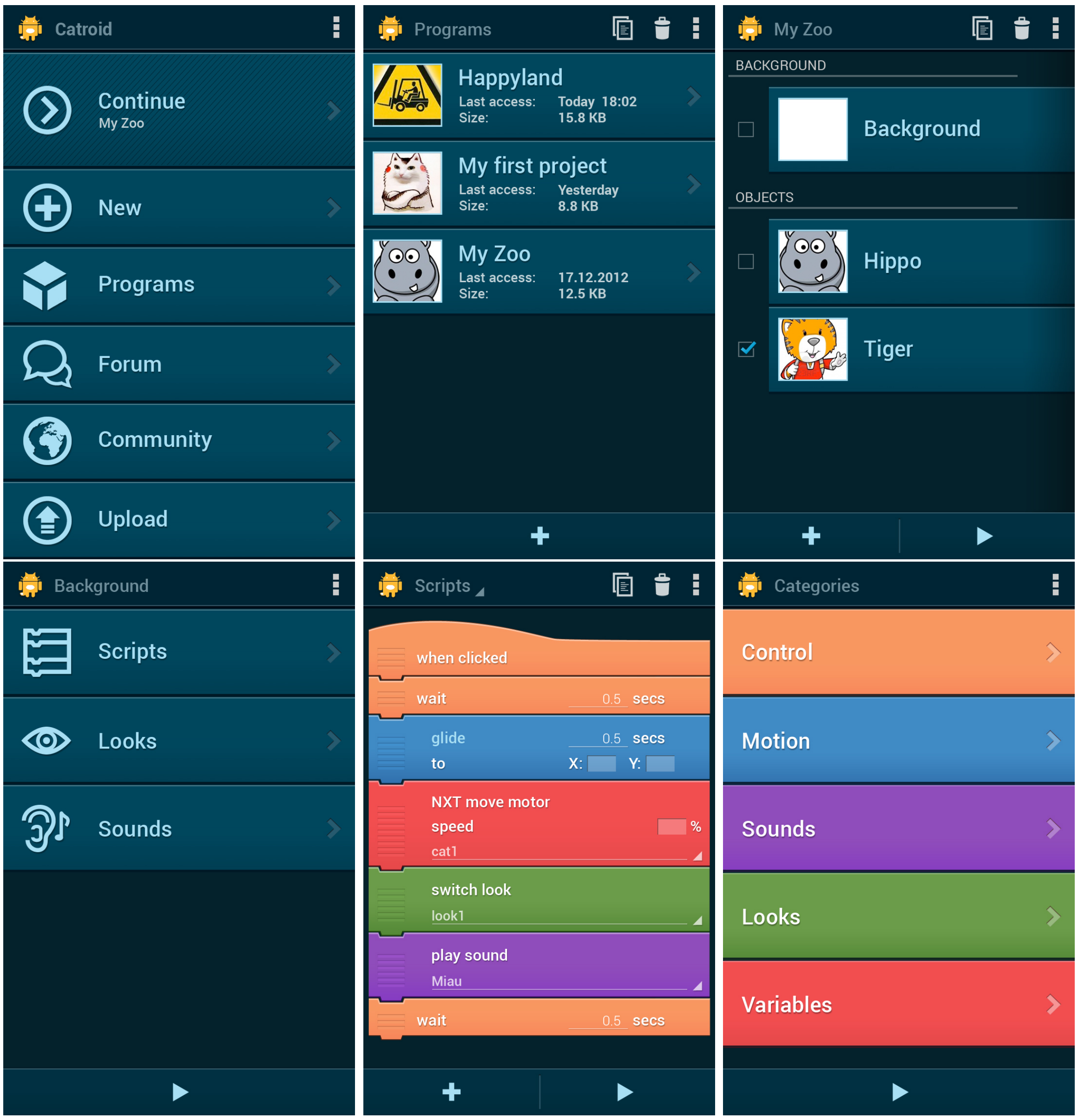


Figure 8.8.: High-fidelity prototypes of Pocket Code.

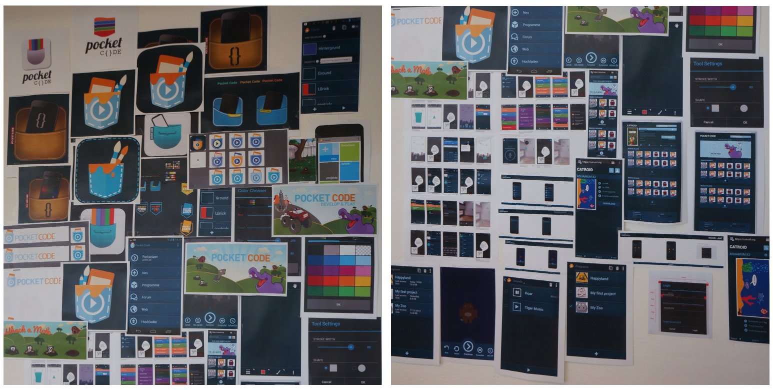


Figure 8.9.: Design Wall.

Formative Usability Evaluation: To make the user interface more usable we conducted formative usability evaluations. We used Heuristic Evaluation (HE) to get a list of potential problems. The UX team inspected the interface continually using a small checklist of principles and provided feedback. The mockups were updated according to the feedback and finally the developers implemented it.

Quick Usability Test Setup: Recruiting children and teenagers as test participants is much more complicated than recruiting adults. We had to deal with additional challenges like declaration of consent from parents and so on. Whenever an opportunity arose to test with children, we had to take it. So, it was very important to create a simple test setup that could be used every time and everywhere. We were not able to provide complicated usability test setups and methods on demand. Another reason for introducing a very simple test setup was that we could not spend to much time on preparing the test setup. Usually, we needed the user feedback immediately so we did not have time to do extensive preparation of the tests or analysis of the recorded tests in significant detail. Our idea to save some preparation time was to use an action camera to record the tests. Depending on the situation and test participant we could use the headgear (see Figure 8.10 on the left) to put the camera on foreheads, or an a chest-strap to put it on the chest (see Figure 8.10 on the right). The GoPro[8] action camera with its wide-angel was good enough for recording such test. Even if we placed the camera on the table (see Figure table 8.10 in the middle) the range of the wide-angle was able to record the screen of the test device.

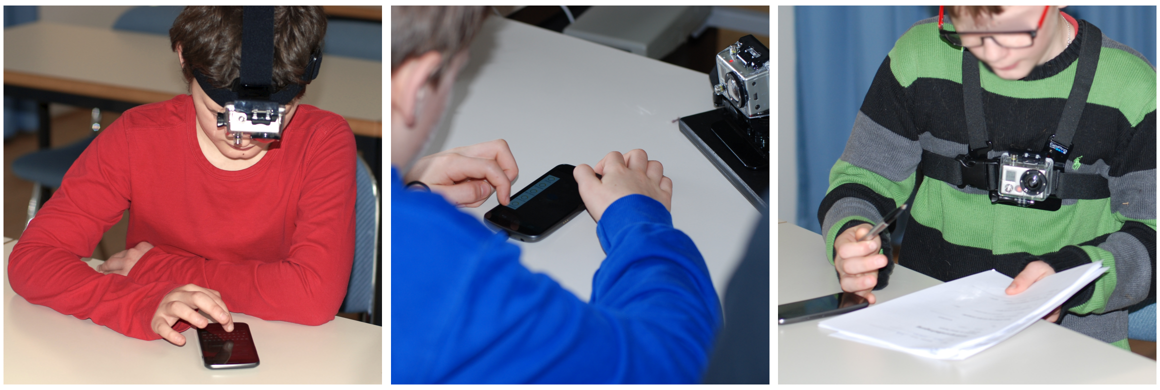


Figure 8.10.: Quick usability tests using GoPro (`gopro.com`) action camera.

8.2.3. Validation Phase

Before we released a beta version of Pocket Code to Google Play[9] we conducted a summative usability test to find out the current usability of the application. The

[8]`gopro.com`
[9]`https://play.google.com/store/apps/details?id=org.catrobat.catroid`

main test results will be described in Chapter 9. Further evaluation like field studies and formal experiments are planned for the future.

8.3. Formula Editor

After working extensively on Pocket Code we found out that a release of Pocket Code without the Formula Editor made no sense. So we changed the release plan and decided to release Formula Editor together with Pocket Code and Pocket Paint as well. Without the Formula Editor programmers can only enter a simple number to a brick without further manipulation like variables or mathematical functions (see Figure 8.11), therefore the integration of Formula Editor into Pocket Code was a very important step.

The following subsections give an overview of the process we followed during the redesign and development of Formula Editor.

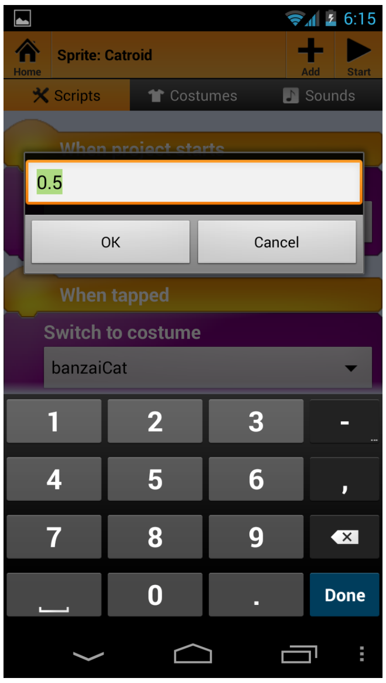


Figure 8.11.: Pocket Code without Formula Editor.

8.3.1. Analysis Phase

Before the UX team supported the Formula Editor sub-project, the developers had been focused on background functionality only. Nobody was responsible for developing a concept for the user interface. We started our work with an analysis of Formula Editors' functionality. We had to find out which functionality it already contained and what were the most important and most used features. Figure 8.12 shows the UI of the Formula Editor at that time. Users needed to swipe lift or right on the display to get to the next screen, which we assumed not to be intuitive. By testing the version with our colleagues and end-users we found out that more than 50% did not use functions stored on second or third screens because they did not know how to reach them. Once we gave them a hint how to reach the functions, participants were very curious and used for example sensor functions like X-Acceleration even if they did not know exactly what their purpose was. This lack of accessibility of these function made a redesign necessary so that users could easily access all possible functions and sensors.

Again, we met with the project head and developers to discuss the next project steps. Together we brainstormed possible new navigational methods for Formula Editor.

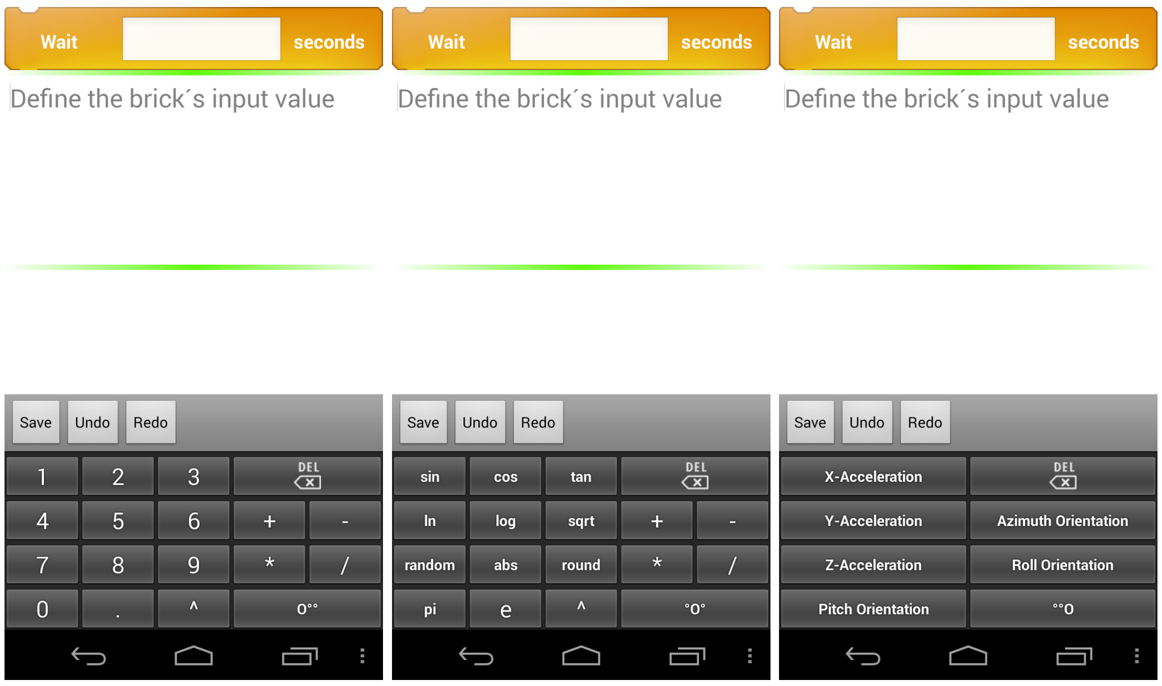


Figure 8.12.: Formula Editor old version.

8.3.2. Design And Implementation Phase

Based on the analysis we did before, the UX team started to draw first mockups as shown in Figure 8.13.

Functions like numbers or basic mathematic operations, which we assumed would be used the most, were placed on the first screen. All other functionality was grouped together in categories like sensors, logical operations and math, so that users just needed to click a button on the first screen to reach all other functions.

Since Formula Editor will be completely integrated into Pocket Code it was obvious to use the same design and guidelines. On the one hand we tried to hide the complexity from beginners but on the other hand we wanted to provide advanced users with all the complex functionality already implemented.

Communication And Collaboration: The collaboration between the UX team and the Formula Editor sub-team was very good. One reason for this was that, the developers were busy working on implementing background functionalities and fixing bugs, so we had enough time to create and test our mockups. The developer team consisted of only two team members so we communicated face-to-face. We decided to share all prototypes immediately on Dropbox, so the functionality could be implemented simultaneously. Figure 8.14 (on the left) demonstrates first implementations of the low-level prototypes with background functionality without applying the design. On the right side of Figure 8.14 we can see the unfinished high-fidelity prototypes of Formula Editor.

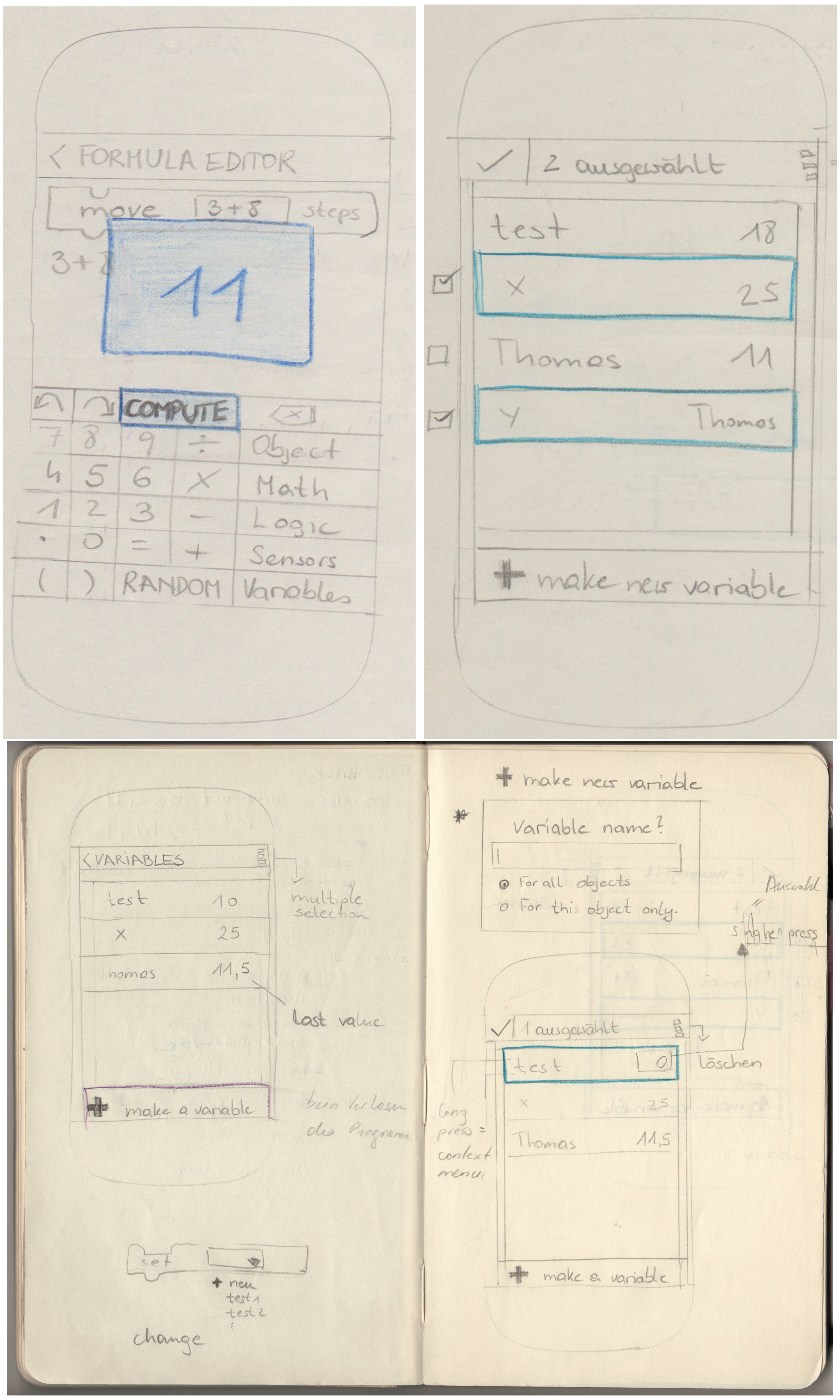


Figure 8.13.: Formula Editor first paper mockups.

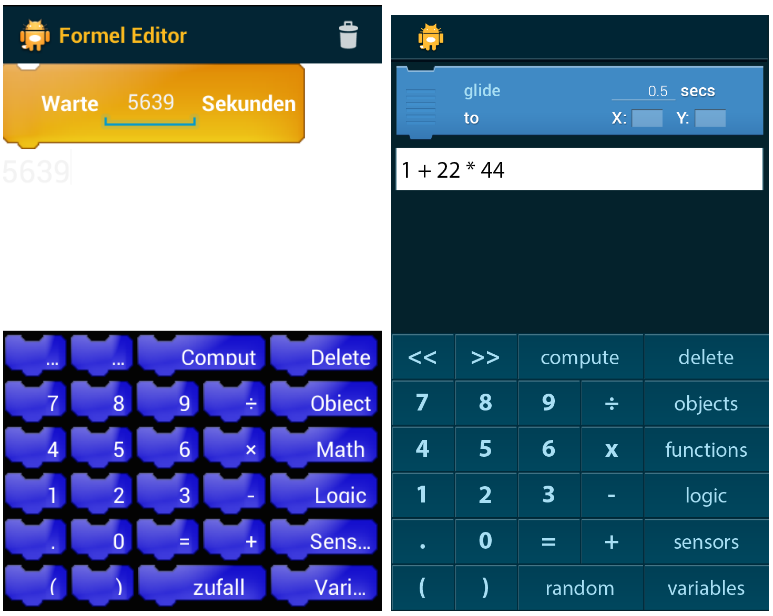


Figure 8.14.: On the left first Formula Editor implementation, on the right first High-Fidelity prototypes.

8.3.3. Validation Phase

Basic functionality of Formula Editor was tested as part of the TA test of Pocket Code and Pocket Paint, before we released the applications. The test results will be presented in Chapter 9. A special usability test for Formula Editor has been postponed to future work. We planned a formal usability experiment in order to evaluate Formula Editor and to compare it with other related formula manipulation approaches like the one of Scratch. The planned test and the methodology is described in the attached paper *Comparing Purely Visual with Hybrid Visual/Textual Manipulation of Complex Formula on Smartphones* Appendix A.

9. Evaluation

As described in Section 4.1, a UCD cycle should end with a validation phase in order to get user feedback about the actual use of the software. At the end of such a cycle it is essential to check if the objectives defined in the analysis phase have been achieved using usability testing.

The first part of this Chapter describes the usability evaluation of redesigned Pocket Code, the main usability issues found, possible solutions and recommendations.

The second part of the evaluation consists of an analysis of the whole agile UCD process applied in the Catrobat sub-projects Pocket Paint and Pocket Code. I will look at agile steps we have already adapted and the steps that should be introduced in the future. Moreover, I will compare our adapted UCD process with the best practice processes recommended in Chapter 5.

9.1. Usability Evaluation

9.1.1. Test Method

In order to evaluate the redesign of the application we conduct a formative evaluation in form of a TA test. We were still interested in finding and eliminating problems rather than gathering statistics of participants behaviour like task success or time for task completion. The TA test was conducted with one pilot user and six test users. The basic test run was identical for all test users and consisted of the following steps.
Step 1: Welcome
Step 2: Background questionnaire
Step 3: Execution of tasks
Step 4: Feedback questionnaire and concluding interview
The TA test method was the same as already described in Section 7, so I will not describe the method again.

9.1.2. Research Questions

The main objective of the test was to evaluate the usability of the redesigned Pocket Code application, to find out main usability issues and UI parts which should be improved in the next release. We defined the tasks in the way that users had to use main features of Pocket Code and some basic functions of Pocket Paint and Pocket Codes Formula Editor. Therefore we were able to evaluate Pocket Paint and Formula Editor as well, up to a certain point. A separate Formula Editor test is planned for the future.

9.1.3. Test Setup

The usability test was conducted at Graz University of Technology. Figure 9.1 shows the *quick test setup* used as already described in Chapter 7, which was developed by our UX team. To improve the audio recordings we additionally used an external microphone.

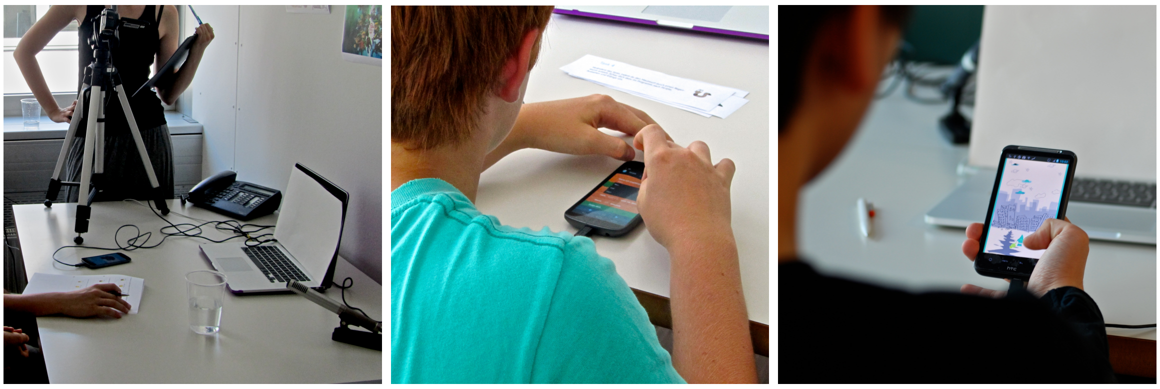


Figure 9.1.: Test Setup, Thinking Aloud Test.

9.1.4. Tasks

We defined eight tasks (see Table 9.1), so that the test participants had to use a wide range of Pocket Code's features. The first three tasks were very easy and served as the introduction to the basic programming concepts of Pocket Code. The task difficulty increased continuously.

Task	Description
T1	Starte Pocket Code und finde heraus was du damit alles machen kannst.. Start the Pocket Code application. What is the app for?
T2	Öffne das Programm "Galaxy Wars" und spiele kurz damit. Open and play the game "Galaxy Wars".

T3	Suche das Spiel ”Whack-a-mole” und starte es. Find the game ”Whack-a-mole” and start it.
T4	Verändere das Spiel, indem du den Maulwurf durch einen Regenwurm ersetzt. Schau dich dazu im Programm nach Skripte, Aussehen und Klänge um. Modify the game, replace the mole with an earthworm. Look at the programs Scripts, Looks and Sounds.
T5	Lass den Regenwurm nun lachen oder ein anderes Geräusch machen. Dazu wirst du einen neuen Klang im Programm hinzufügen müssen. The earthworm should smile or make another sound. You have to add new sound to your program.
T6	Erstelle ein neues Programm mit einem Hintergrund und einem Objekt. Ändere dazu den schon vorgegebenen Hintergrund und füge auch ein neues Objekt hinzu (z.B. aus der Galerie). Create a new program with a background and an object. Change the default background and add a new object (e.g. from the Gallery).
T7	Lass das Objekt erst nach 5 Sekunden an der Stelle X=200, Y=300 auftauchen. The object should appear after 5 seconds at the position X=200, Y=300.
T8	Bei Berührung soll das Objekt verschwinden. Nach 5 Sekunden soll es wieder auftauchen und 2 Sekunden lang zur Stelle X=100 und Y=100 gleiten. When touching the object it should disappear. After 5 seconds it should appear again and slide for 2 seconds long to the position X=100 and Y=100.

Table 9.1.: Tasks for the final Thinking Aloud Test.

9.1.5. Test Users

For the TA test we recruited six high-school boys between 13 and 17 years. All of them had their own smartphone and were using it on a daily basis. Three of them used iPhones and the other three Android smartphones. All test participants were allowed to use the Internet without their parents' supervision. None of them had previous programming experience. Table 9.2 summarises the most important user data.

test persons	**age**	**own smart-phone**	**OS**	**favourite app**	**smartphone most often used for**
Pilot "Bobby"	15	yes	Android	Youtube	phone calls, SMS
TP1 "Thomas"	13	yes	iOs	Facebook, Youtube, Games	Games, Internet, Facebook
TP2 "Patrick"	14	yes	Android	several games, Facebook	Games, Facebook
TP3 "Martin"	14	yes	Android	WhatsApp	Games, phone calls, SMS
TP4 "Alex"	14	yes	iOS	not YouTube	Games, phone calls, SMS, Internet, Facebook
TP5 "Florian"	16	yes	Android	Facebook	Games, Photography, phone calls, SMS, Internet, Facebook
TP6 "Anton"	17	yes	iOS	WhatsApp	Internet, Facebook

Table 9.2.: Some background information of the test participants.

9.1.6. Main Findings

The data from the TA tests which related to usability events were marked with time stamps after the test and analysed by the UX team. The main findings of this analysis will be listed below and some recommendations for improvement will be presented as well.

9.1.6.1. Pocket Code

Test participants liked the idea of creating their own games on their smartphones very much. After the test, 5 out of 6 test participants wanted to install the Pocket Code application on their own smartphones, because they wanted to program later at home. Most of them mentioned that Pocket Code was fun to use, and that they were interested in learning more about programming. From the questionnaires and interviews we could conclude, that all but one of the participants liked the design of Pocket Code.

However, Pocket Code still contained usability issues, design problems and bugs which we discovered during the usability tests. The main findings of the analysis are listed below.

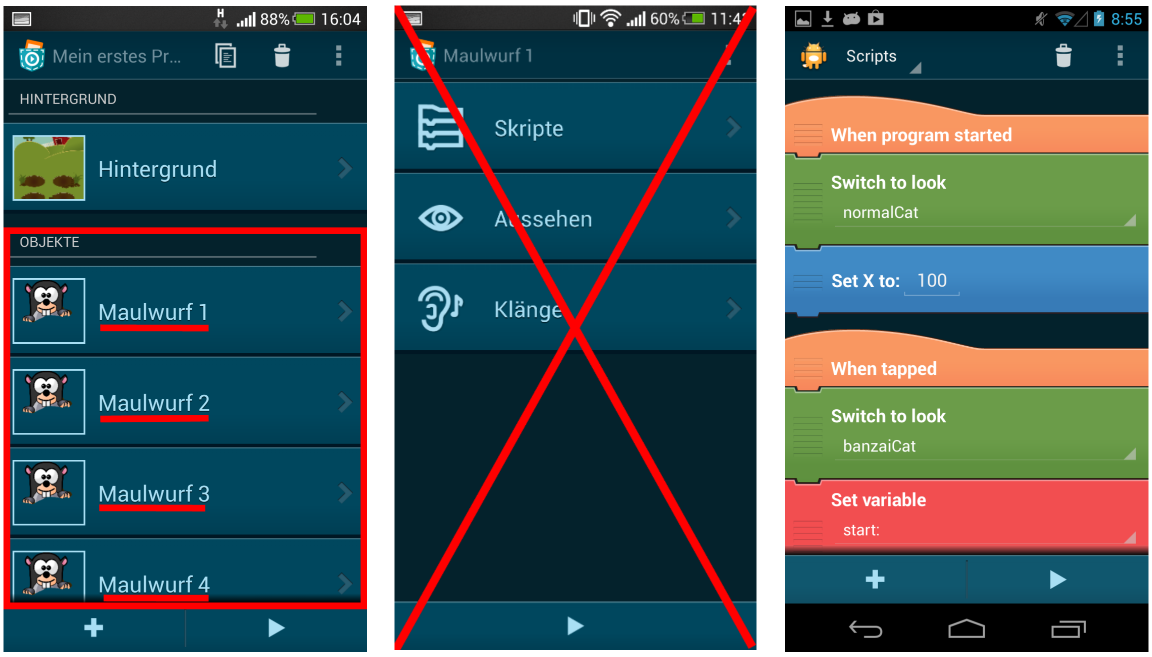
Play Icon: Test participants did not understand the meaning of the play icon in the button bar (see Subfigure 9.2(b)). Some users assumed that they would move one step forward by pressing the button. A redesign of the play button is essential.

My First Program: In our sample program objects had the same names as images. Therefore, test participants could not always differ between objects and images. We should use different names for objects and images in the sample program.

Undo Button: Test participants often wanted to undo an action so they searched for an undo button. Adding an undo button in the future will improve the user satisfaction.

Scripts, Looks, Sounds: Test participants still did not understand the relation between Scripts, Looks and Sounds. When clicking on an object (see Subfigure 9.2(a)), users proceed to the next screen *"Scripts, Looks, Sounds"* (see Subfigure 9.2(b)), where they can decide what to do next. If *Scripts*, is clicked, they get to the visual command blocks (bricks), which control the graphical object and define the logical program (see Subfigure 9.2(c)). Test participants were often overwhelmed by the visual representation of the scripts, because at first they did not know what scripts are for, so they spent only a few seconds looking at the screen and left.

For instance, to add a new Look to an object, users first had to choose Looks from the screen *Scripts, Looks, Sounds*, then use Pocket Paint to draw the new Look or import it from the Gallery. Observations from the usability test showed that creating and saving a new Look was not a problem, but the next step, which should be getting



(a) Objects. (b) Scripts, Sounds and Looks screen. (c) Scripts.

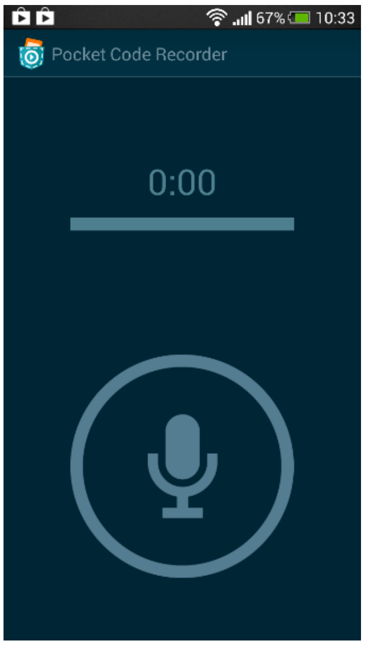
Figure 9.2.: Screens, Looks, Sounds.

back to the Scripts and choosing the new Look from the drop down menu of the proper brick, was not intuitive for the test participants. Most of them wondered, why the new Look had not been added to the object after they had saved it.

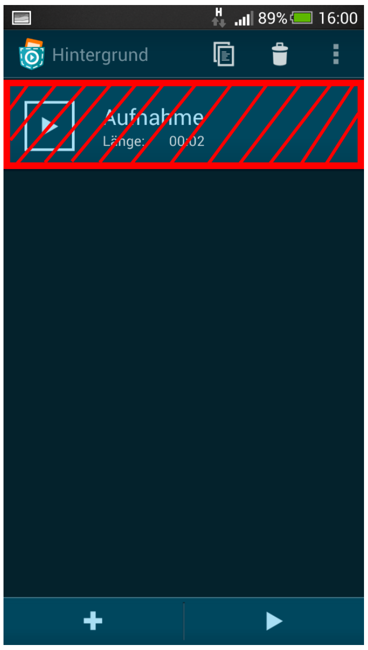
Such an important functionality must be more more straightforward. A possible solution to that problem could be removing the *Scripts, Looks, Sounds* screen. If an object is selected the first screen should always be the Scripts screen. Adding Looks and Sounds should be possible by choosing *New...* from the brick drop down menu. So, for example, the brick menu for the *Switch to look* brick should be redesigned. If a Look already exists, *Choose...* should be the first selection in the menu per default. If no Look is available, *New...* should be displayed, so users can create and add a new Look without switching to another screen.

Sound Recorder: Subfigure 9.3(a), shows another issue with the new UI. Here, participants did not know how to start recording a sound. We needed to improve the record icon.

Play Sound: Participants did not exactly know how to play sound. Some of them did not tap the button on the left, but rather the caption on the right. A simple solution would be to play the sound if the user taps anywhere on the sound object (see Figure 9.3(b)).



(a) How to record a sound is not clear for the test participants.



(b) Where to click to play a sound?

Figure 9.3.: Sound, Play Sound.

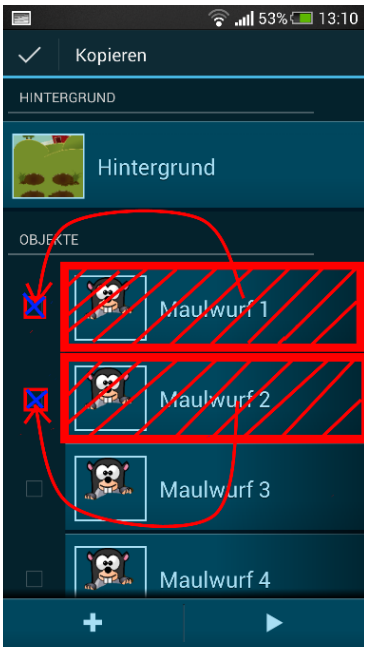
Checkbox Selection: If users wanted to select an object they had to select the checkbox behind the object icon and name (see Figure 9.4(a)). Test participants often wanted to select the object by clicking on the image or even the textual description of the object. We should provide the users with the option to select the checkbox by clicking anywhere on the object and its name.

Delete Brick: Test participants wanted to delete a brick via drag and drop (see Figure 9.4(b)). We had to discuss if we wanted to provide the drag and drop functionality to delete bricks additionally to the current method or if we should implement some other options, like a slide to the left.

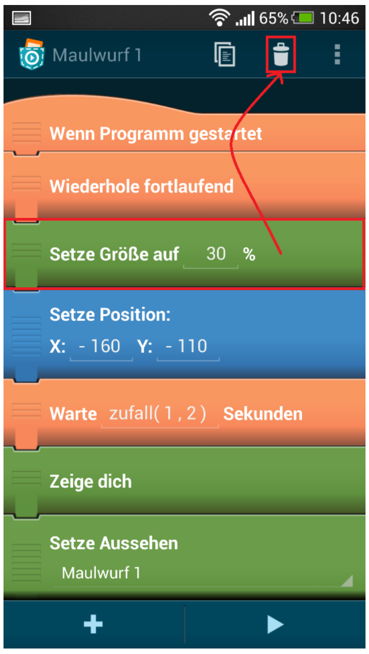
Additional problems occurred when deleting a brick because users wanted to click on the textual description *delete*, rather than on the selection box on the right side of the text (see Figure 9.4(c)).

Another problem with selecting an item occurred when users wanted to select a brick. They tried to select a brick directly instead of using the checkbox on the right side of a brick. This feature should be implemented as well.

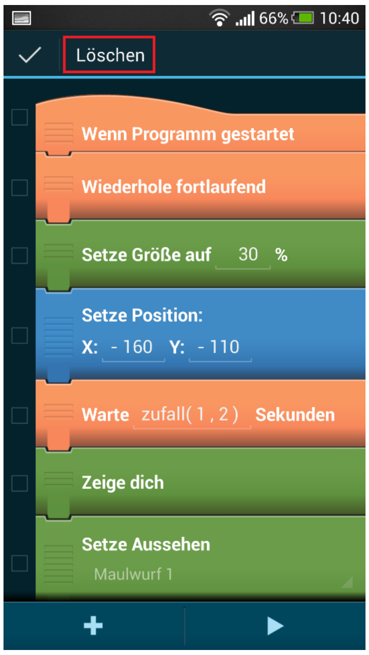
Rename: The renaming process of an object is not intuitive either because users can choose more than one object (see Subfigure 9.5(a)). The solution was to restrict



(a) How to select a checkbox not clear.



(b) Drag and Drop for deleting a brick not possible.



(c) Where to click to confirm the selection?

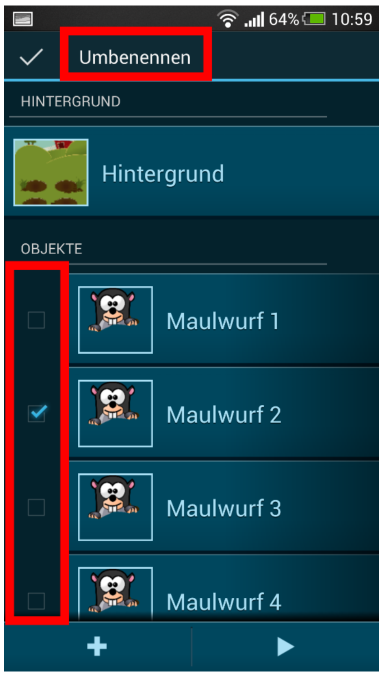
Figure 9.4.: Checkboxes, Delete Bricks.

renaming to only one object. When attempting to rename an object users experienced the same issues as before. Users wanted to click on the textual description instead of the checkbox left of the text.

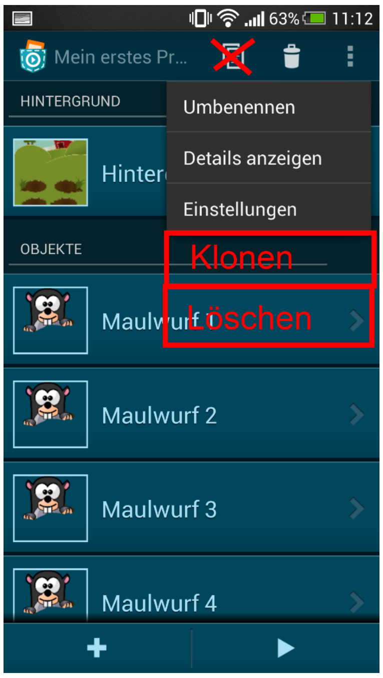
Copy: Test participants did not recognise the copy icon. We should place it in the overflow menu with a textual description (see Subfigure 9.5(b)).

Bricks: Generally when a brick is tapped the menu for copy, move or delete brick appears. But there is one exception: if the small area where attribute values can be entered or manipulated is tapped, the according window/menu opens (see Subfigure 9.6(a)). Some test participants did not expect that, and were surprised that a menu list appeared when they wanted to manipulate the attribute values. They believed that the menu list appeared randomly. One possible solution could be to put the attribute value manipulation in the menu header. When tapping on the brick the menu appears (see Subfigure 9.6(b) in the middle), so users always get the same action when tapping a brick.

Set Background Brick: If no background image is available users can chose to add a new background, but if an image already exists the first one will be displayed in the first menu entry. It would be better if *Choose...* is the first entry in the menu, to



(a) Renaming more than one object at the same time not logical.



(b) Copy icon not understandable for test participants.

Figure 9.5.: Rename and Copy.

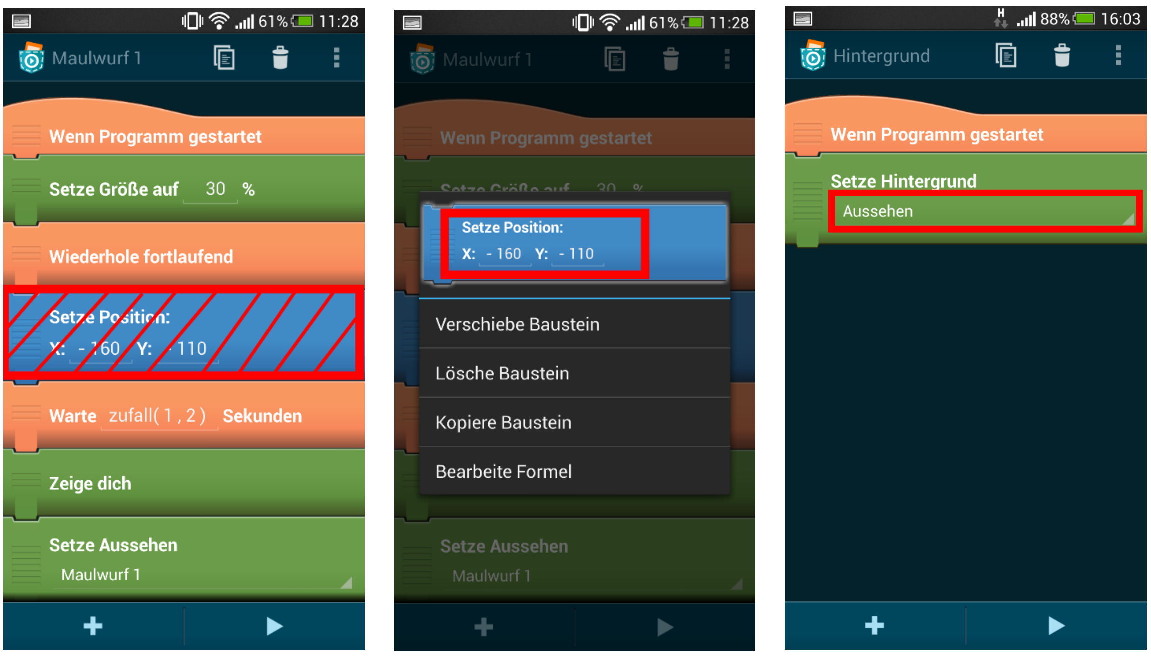
indicate to the users that there is more than one background image to choose (see Subfigure 9.6(c)).

When Bricks: If a user deletes a *"When brick"* all bricks within the control block the control brick will be deleted too. Test participants did not expect that behaviour and were very frustrated if that happened during the test. A possible solution to this problem is to let the user decide if all bricks should be deleted or only the control brick. If the user decides to delete only the control brick, the block within could be grouped and marked as inactive. The block could then be moved around as a group. Inactive blocks without a control brick would not be executed by the program until a new control brick is added.

9.1.6.2. Pocket Paint

Delete Image: It was not obvious for the test participants how to delete an image. A possible solution often used by painting programs is to delete the old one by adding a new one.

Save Image: Pocket Paint does not allow the user to save images with a space character in the file name. Participants did not understand the error message why



(a) The selection for the menu should be enlarged to the whole brick. (b) Suggestion for new menu. (c) Menu selection for *Set background* brick.

Figure 9.6.: Bricks and Set Background Brick.

the image could not be saved. We should at least allow the saving of images with white spaces.

Dialogbox: Some of the dialogs of Pocket Paint were wrong, or not clearly enough phrased. If a user created a new image and went back to Pocket Code using the back button a dialog box appeared with the message *"Image will be overwritten"*. This message was not clearly understandable, we should revise the dialog messages in order to make them more understandable.

9.1.6.3. Formula Editor

We did not perform a special test for Formula Editor, therefore test participants used only the basic functions. In order to get better insights into the usability of Formula Editor, a further test will be conducted.

Confirm Button: Participants did not know how to confirm the input value. Most of them were searching for an OK button. Using the Android back button to confirm an input, was not intuitive for test users. So, we should implement an OK button to confirm the input and get back to the visual script representation of Pocket Code.

9.1.7. Feedback Questionnaire

At the end of the usability test we asked the participants to answer seven questions about Pocket Code in order to collect subjective feedback. Four out of six test participants stated that they were able to use Pocket Code without any assistance. Although only half of our test users thought that Pocket Code was easy to understand quickly, five out of six participants answered that Pocket Code was easy to use.

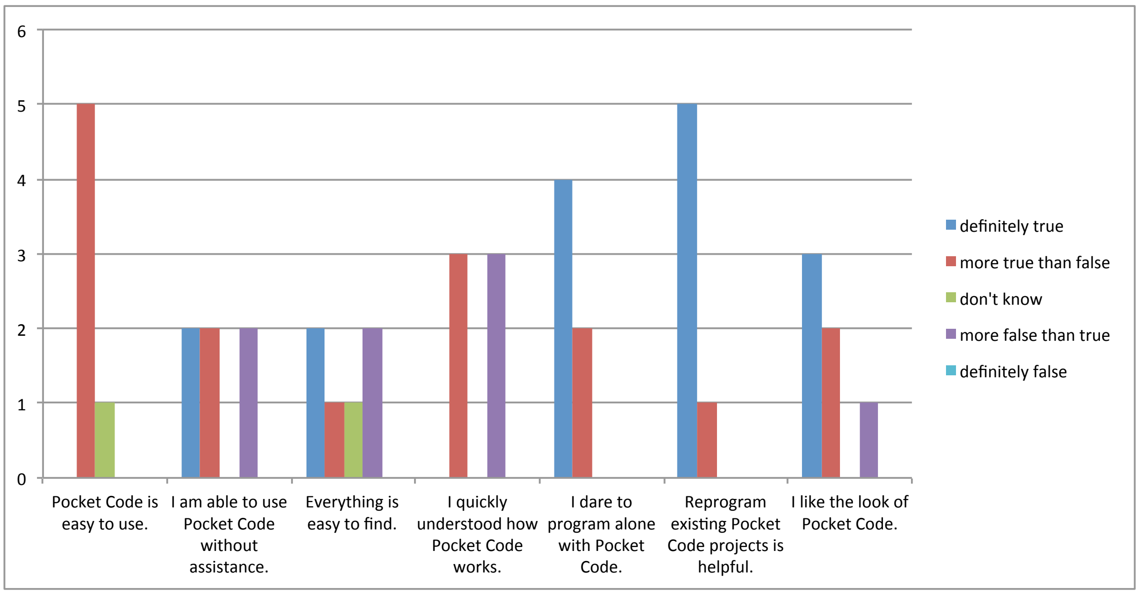


Figure 9.7.: Overview of the Feedback Questioner answers.

The analysis for special user groups like iOS users versus Android users, or younger versus older users did not deliver new insights. Figure 9.8 shows for example the results for the question how easy Pocket Code is to use. The iOS user group considered the application easier to use than Android users did. This result is surprising since we used many standard Android functions and guidelines. We supposed that Android users would be more used to that functions than iOS users.

9.2. Evaluation of UCD Process

Integrating UCD in a large-scale, FOSS, agile project was a challenge for all team members. It is very hard to apply the proposed best practice processes described in Chapter 5, because of the Catrobat project composition and complexity. The core Catrobat team consists only of volunteering students. We do not have fixed working times and nobody is being paid for the work. We are not able to plan sprints in our agile process, because we do not know exactly how many hours the developers will be available. Moreover, it is not obvious how long the students will work on

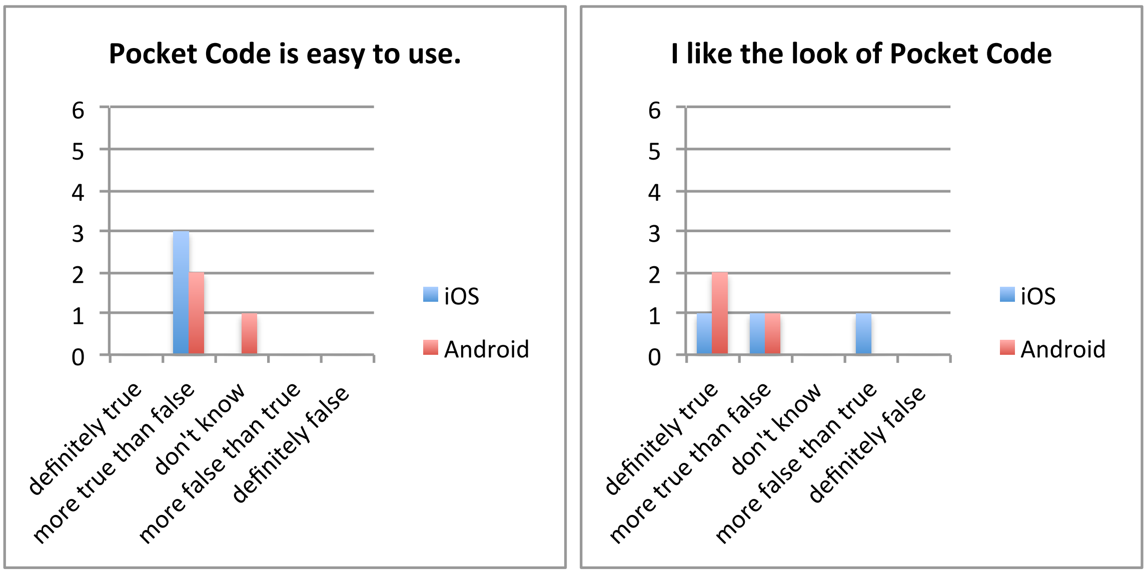


Figure 9.8.: Android user-group versus iOS user-group.

the project. Several team members have been working for weeks or months on the projects and others for years. This also applies to the UX team members.

Experience has shown that communication and collaboration between the team members is even more important than in paid agile projects. Usability experts and designers are not used to agile methods. Learning about agile culture was very important for all UX team members to understand other team members and project management better. Actually, the UX team adapted UCD methods to become more agile. We started to produce low-level prototypes, developed quick usability test setups, conducted usability tests without long preparation time, analysed tests immediately, provided feedback as quickly as possible, got used to requirement changes, created our own whiteboard and design wall to make our work more visible, started to write story cards, used GitHub and so on.

In turn, developers had to learn about usability methods as well as about the importance of including end-users in the development process.

In the position statement, *Between Pragmatism and Infeasibility Agile Usability for Children and Teens in a Very Large Multi-Year FOSS Project about Several Complex Mobile Apps and Mobile Websites* (see Appendix B), we identified the challenges of integrating UCD in an ASD process. Especially, we pointed out problems and challenges the developer team faces in a FOSS project.

Despite these difficult circumstances, I analysed best practices for the integration of UCD in agile projects and tried to adapt and integrate it in our project. The list below shows agile steps already integrated in the Catrobat project and potential steps to do in the future.

Agile steps already adapted:

- Understand the agile culture.
- Train the team in ASD practices.
- Communicate as often as possible with developers.
- Adapt UX practices to become more agile.
- Improve communication with the team.
- Make the work visible.
- Be up to date, update the whiteboard and designer wall as often as possible.
- Communicate with the developers which designs we are working on.
- Approximate when developers should expect to receive the designs.
- Step back and consider how the practices could be improved.
- Move much more quickly toward design solutions with a fewer number of usability tests within a release.
- Understand and respect across the disciplines.
- Take every chance to observe end-users using the system.
- Promote UX activities across the team.
- Do not design the software that we personally like and would prefer to use.
- Do not plan activities as whole from beginning to the end. Instead break them into parts that can be completed quickly.
- If end-users are not available, have students or other team members look over new interface versions and functions.
- Conduct informal evaluation when more complex usability studies are not accomplishable.
- Generate Usability Story Cards, usability is part of the acceptance criteria and must be accepted by UX team members.
- Have a up to date UX whiteboard.
- Update the Design Wall frequently.
- Replace requirements documents with planning sessions if possible.
- Perform in-house usability tests with low-level prototypes.

Steps to Do:

- Start UX pairing.
- Find out which UX activities require lead time, schedule them one iteration ahead.
- Keep ahead of development.
- Work with developers to finalise the implementation order.
- Present the design in person to the developers who will implement it.
- Learn form agile development, develop UX standards.
- Start within a sub-project: UX team members work as part of the core team.
- Include at least one UX specialist in all sub-teams
- Integrate usability into day-to-day process.
- Analyse data after each participant, or at least after each day of testing.

- If the final prototype is a high-fidelity one pass the data to the developers as part of the specification.
- Report information verbally gathered from usability testing to the developers (agile developers usually do not read reports)
- Create Design Cards without implementation time estimates, present them on the planning boards, use different colours to be able to distinguish from story cards
- Discuss issue cards with developers immediately
- Provide enough time to do UX work before programming.
- Do UX incrementally, break it down and estimate by feature in the same way as programming tasks

10. Conclusions and Future Work

The main focus of this thesis was to improve usability in Catrobat, a large-scale, Free Open Source, agile project, mainly in the Pocket Paint and Pocket Code sub-projects and in Pocket Code's Formula Editor. In order to reach the goal, it was insufficient just to conduct usability tests and write usability reports, because agile teams are not used to reading much documentation. Moreover, the UX team had to work very closely with developers and all other stakeholders during the development and implementation of the software to be sure that the issues detected were implemented correctly. We introduced special usability story cards which had to be accepted by UX team members. We worked on a User-Centered Design process and developed more flexible, agile methods for usability evaluations. To work successfully with agile teams, it was necessary to learn and understand the agile culture.

The work started with an overview about Agile Software Development. In the last decade, software development has radically changed. Traditional models like the waterfall model often could not keep up with the fast development, extremely short time to market and frequently changing requirements. New promising software development processes like ASD has been introduced.

All agile methods share common values like collaboration, short iterations, minimal documentation or continual feedback. Scrum and Extreme Programming are the two most popular agile methods. FOSS projects work on products with minimal management and contain usually unpaid volunteer developers, who are often geographically distant. As discussed in the thesis, FOSS and ASD have different approaches but also many similarities like shared goals, shared values on how to reach the goals, standardisation of code, trust in team members and so on.

Integrating UCD in an agile, FOSS project was a challenge for all team members. The main focus of UCD is to involve end-users in the software development process. It is an iterative process with multiple stages of product development and consists of analysis phase, design phase, implementation phase and validation phase. Different tools and methods are used in UCD processes. Researchers have introduced and defined many approaches for the integration of UCD into ASD. In the literature, we can find best practices for integrating UCD with agile. Nevertheless, in practice agile teams have their own understanding of agile and use their own adapted agile methods. So, the integration of UCD always depends on the agile culture of the team.

Catrobat is a large-scale FOSS project which uses ASD. The approach, how to combine ASD, FOSS and UCD is not yet discussed in the literature. Therefore, we were not able to follow already known processes. This thesis described, using the example of Pocket Code and Pocket Paint projects, a possible integration approach of ASD, FOSS and UCD, the challenges we had to overcome, possible solutions and still open issues.

When we started with the integration of usability we first tried to use classical UCD methods. Only a short time after, we experienced that the overall speed of development was much faster than the usability evaluation. It took too long between the usability evaluation and the time we provided feedback to the developers. We have observed that a high number of small usability tests, focused on parts of the application resulted in a better level of usability than a small number of large tests. Constant communication and collaboration between UX team members and developers was the key success factor in our projects. Agile teams quickly produce functional software. That was a big benefit for usability team members because we could often test with working software instead of paper prototypes.

During the last year, we have done a complete redesign, used different methods to evaluate it and identified many usability issues, functional bugs and design problems. Most of the problems have already been fixed in the current version or will be implemented in the next release.

Nevertheless, there is still much work to do for the future UX team. This thesis can be taken as a basis for further work. As soon as a tutorial or user hints are implemented in Pocket Code, a formal experiment should be conducted to gather summative data and get an objective measurement of the system performance. With respect to the Catrobat project, the most important step in the future is to provide UX support to all sub projects.

From my own experience, I can recommend future UX team members not to be afraid to test new ideas with real end-users, even if the number of test participants is small. Furthermore, all UX team members should spend time with the Catrobat programming language and program their own games and animations. This helped enormously in finding many major usability problems on our own, just by using the application. Being up to date is another important aspect, which means constantly looking for new usability methods and technologies the UX team could try out. As, Catrobat project developers, including UX team members, change frequently, a minimal design specification and basic usability guidelines should be defined for all future team members.

Bibliography

[1] M. Hjerde, "http://sender11.typepad.com/sender11/2008/04/mobile-screen-s.html," last visited, May 2014.

[2] H. Beyer, *User-Centered Agile Methods*. Morgan & Claypool Publishers, 1st ed., 2010.

[3] "Manifesto for agile software development, http://agilemanifesto.org/," July 2013.

[4] Z. Hussain, M. Lechner, H. Milchrahm, S. Shahzad, W. Slany, M. Umgeher, and P. Wolkerstorfer, "Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application," in *USAB 2008*, vol. 5298/2008 of *LNCS*, pp. 313–330, Springer Berlin / Heidelberg, November 2008.

[5] J. S. Ken Schwaber, "https://www.scrum.org/scrum-guides, the scrum guide - the definitive guide to scrum: The rules of the game," last visited November, 2013.

[6] R. Jeffries, "http://xprogramming.com/what-is-extreme-programming/," last visited November, 2013.

[7] K. Beck, *Extreme Programming Explained*. Addison-Wesley Professional, 1st ed., 1999.

[8] D. Wells, "The rules of extreme programming," last visited November, 2013.

[9] P. Tsirakidis, F. Koebler, and H. Krcmar, "Identification of success and failure factors of two agile software development teams in an open source organization.," in *ICGSE*, pp. 295–296, IEEE, 2009.

[10] "Free open source software, http://freeopensourcesoftware.org," July 2013.

[11] "The free software definition, http://www.gnu.org/philosophy/," February 2014.

[12] Y. Jing and W. Jiang, "Review on free and open source software," *Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference*, vol. 1, pp. 1044 – 1049, 2008.

[13] W. Scacchi, "Free/open source software development: Recent research results and emerging opportunities," in *The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*, ESEC-FSE companion '07, (New York, NY, USA), pp. 459–468, ACM, 2007.

[14] J. Gould and C. Lewis, "Designing for usability: key principles and what designers think," *Communications of the ACM*, vol. 28, no. 3, pp. 300–311, 1985.

[15] J. Nielsen, "The usability engineering life cycle," *Computer*, vol. 25, no. 3, pp. 12–22, 1992.

[16] J. Ungar, "The design studio: Interface design for agile teams," in *Agile, 2008. AGILE '08. Conference*, pp. 519–524, 2008.

[17] "The user experience professionals association, http://www.usabilityprofessionals.org/," last visited, Mai 2014.

[18] J. Nielsen, D. Norman, and B. Tognazzini, "http://www.nngroup.com/," last visited, December 2013.

[19] M. E. Raven and A. Flanders, "Using contextual inquiry to learn about your audiences," *SIGDOC Asterisk J. Comput. Doc.*, vol. 20, pp. 1–13, Feb. 1996.

[20] K. Andrews, "Human-computer interaction course notes," Version of 28 May 2013.

[21] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[22] J. Nielsen and R. Budiu, *Mobile Usability*. Berkeley, CA 94710, 1249 Eighth Street: New Riders, 1st ed., 2013.

[23] E. Nelson, "Extreme programming vs. interaction design," 2002. FTP Online.

[24] P. Hodgetts, "Experiences integrating sophisticated user experience design practices into agile processes.," in *AGILE*, pp. 235–242, IEEE Computer Society, 2005.

[25] D. D. Brown, *Agile User Experience Design: A Practitioner's Guide to Making It Work*. Elsevier Science, 2012.

[26] A. Martin, R. Biddle, and J. Noble, "Xp customer practices: A grounded theory," in *Proceedings of the 2009 Agile Conference*, AGILE '09, (Washington, DC, USA), pp. 33–40, IEEE Computer Society, 2009.

[27] H. Beyer and K. Holtzblatt, "Contextual design," *interactions*, vol. 6, pp. 32–42, Jan. 1999.

[28] D. Sy, "Adapting usability investigations for agile user-centered design," *Journal of Usability Studies*, 2007.

[29] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: programming for all," *Commun. ACM*, vol. 52, pp. 60–67, Nov. 2009.

[30] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *Trans. Comput. Educ.*, vol. 10, pp. 16:1–16:15, Nov. 2010.

[31] F. R. Knapitsch-Scarpatetti-Unterwegen, "Usability testing of mobile applications for children," 2012.

[32] C. Kelleher, *Motivating Programming: Using Storytelling to Make Computer Programming Attractive to Middle School Girls*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 2006. AAI3248491.

[33] W. Slany, "Catroid: A mobile visual programming system for children," in *Proceedings of the 11th International Conference on Interaction Design and Children*, IDC '12, (New York, NY, USA), pp. 300–303, ACM, 2012.

[34] T. Green and M. Petre, "When visual programs are harder to read than textual programs," in *Proceedings of the Sixth European Conference on Cognitive Ergonomics (ECCE 6)*, pp. 167–180, 1992.

[35] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, and Y. B. Kafai, "Scratch: programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[36] D. Garcia, L. Segars, and J. Paley, "Snap! (build your own blocks): tutorial presentation," *J. Comput. Sci. Coll.*, vol. 27, pp. 120–121, Apr. 2012.

[37] W. Slany, "A mobile visual programming system for Android smartphones and tablets.," in *VL/HCC* (M. Erwig, G. Stapleton, and G. Costagliola, eds.), pp. 265–266, IEEE, 2012.

[38] E. van Teijlingen and V. Hundley, "The importance of pilot studies," *Nurs Stand*, vol. 16, no. 40, pp. 33–6, 2002.

[39] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, and K. E. Emam, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, pp. 721–734, 2002.

[40] Z. Hussain, M. Lechner, H. Milchrahm, S. Shahzad, W. Slany, M. Umgeher, T. Vlk, C. Koeffel, M. Tscheligi, and P. Wolkerstorfer, "Practical usability in xp software development processes," in *The Fifth International Conference on Advances in Computer-Human Interactions*, (Valencia, Spain), p. 208 to 217, Copyright (c) IARIA, 2012, ACHI 2012 : The Fifth International Conference on Advances in Computer-Human Interactions, January 2012.

[41] J. Lee and D. Scott McCrickard, "Towards extreme(ly) usable software: Exploring tensions between usability and agile software development," in *Agile Conference (AGILE), 2007*, pp. 59–71, 2007.

[42] O. Sohaib and K. Khan, "Integrating usability engineering and agile software development: A literature review," in *Computer Design and Applications (ICCDA), 2010 International Conference on*, vol. 2, pp. V2–32–V2–38, 2010.

[43] Z. Hussain, H. Milchrahm, S. Shahzad, W. Slany, M. Tscheligi, and P. Wolkerstorfer, "Integration of extreme programming and user-centered design: Lessons learned," in *Agile Processes in Software Engineering and Extreme Programming* (P. Abrahamsson, M. Marchesi, and F. Maurer, eds.), vol. 31 of *Lecture Notes in Business Information Processing*, pp. 174–179, Springer Berlin Heidelberg, 2009.

Appendix

Appendix A.

Comparing Purely Visual with Hybrid Visual/Textual Manipulation of Complex Formula on Smartphones

A.1. Introduction

Visual programming languages (VPL) are important for end user programming. It empowers end users with little, no, or only casual programming experience to develop programs for their individual use. The ability to write applications, to create one's own games, or to automate small tasks can be easily learned. Due to the features of VPL they are often used when it comes to children. Especially for younger children it seems to be easier to drag & drop bricks together like in Scratch[1] than to learn a textual programming language. They can learn the basic principles of programming without bothering with the sometimes restrictive syntax of a textual programming language. There are other important benefits of visual languages as well, e.g., being able to see what command blocks are available and thus might be employed, thereby suggesting their use without the user having to know or even remember them. Regarding programming and the writing of formulas there are two clearly distinct approaches: the textual and the visual approach. With traditional textual programming languages like C or Java, developers enter statements on a standard keyboard, though modern IDEs to some degree simplify the entering of statements through context sensitive statement completion. Visual languages like Scratch use the visual approach, where even formulas are created with pre-defined graphical blocks. When complex formulas are involved, this latter approach can become unwieldy and even confusing. Some programming languages like TouchDevelop[2], which is mainly textual, pursue an approach where code and formulas are entered via an interface that looks like a pocket calculator: Statements and operators are chosen from a set of visually differentiated alternatives, but the actual visualization of the resulting statements and formulas is done in a purely

[1] `http://scratch.mit.edu/`

[2] `https://www.touchdevelop.com/`

textual way. In our paper we introduce Pocket Code[3], a new approach with visual programming, but textual formula representation, which combines the ease of visual programming with the effectiveness and clarity of textual formula display. One of the reasons for introducing this new combination is that the presented visual programming language is optimized for use on smartphones with their touch screens and small display sizes, where Scratch-like blocks cannot easily be accommodated due to the narrowness of the screens and the difficulty to drag and drop blocks closely nested together with one's fingers compared to when using a mouse pointer. Furthermore we will discuss different programming language approaches for editing and manipulating formulas and present a proposal on how these approaches could be compared and evaluated regarding their efficiency, effectiveness, and user satisfaction.

A.2. Related Work

Green et al. [34] argue that dataflow visual programming languages are not consistently superior to text languages. Their study shows that some visual notations, for example the gate notation, are in fact worse than equivalent textual notations.

When creating formulas with end user programming languages there are three main approaches:

- The purely textual approach like in Microsoft's Excel, where formulas are created and displayed textually. Excel is a spreadsheet application, designed mainly for adults and use on traditional computers, where users can calculate values with formulas entered in Excel.
- The purely visual approach, like in Scratch [35], Snap![4] [36], and Blockly[5] where pre-defined visual segments are used to create and to display a formula.
- The "hybrid" approach like in Microsoft's TouchDevelop and Pocket Code's formula editor, where formulas are created using visual elements similar to a pocket calculator, but are displayed textually.

In the course of our paper we will focus on the latter two approaches. Table A.1 gives an overview of different approaches for formula creation and visualization in different programming environments.

Snap! is an extension to Scratch, was designed for traditional computers (large screen, keyboard, mouse), but works on smartphones as well. In addition to Scratch's approach (see Figure A.1) Snap! highlights different nesting levels of formulas in a so called "zebra"-mode: Parts of the formula are alternatingly lighter and darker colored (see Figure A.3).

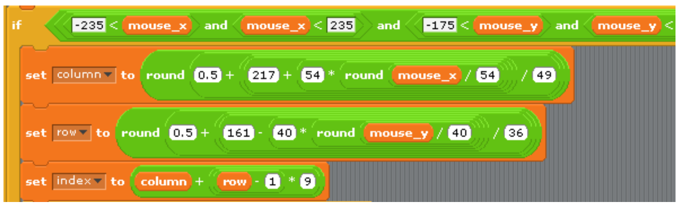
[3]http://www.pocketcode.org/

[4]http://snap.berkeley.edu/

[5]https://code.google.com/p/blockly/

Table A.1.: Characteristics of considered programming system allowing to create formulas

Programming system	formula creation	main target group	screen and input
Excel	textual	adults	large screen, mouse, keyboard
Scratch	visual	children	large screen, mouse, keyboard
Snap!	visual	children & adults	large screen and mobile, mouse, keyboard
TouchDevelop	hybrid	adults	smartphone, touch screen
Pocket Code	hybrid	children	smartphone, touch screen
Blockly	textual or visual	adults	large screen and mobile, mouse, keyboard



```
if -235 < mouse_x and mouse_x < 235 and -175 < mouse_y and mouse_y <
  set column to round 0.5 + 217 + 54 * round mouse_x / 54 / 49
  set row to round 0.5 + 161 - 40 * round mouse_y / 40 / 36
  set index to column + row - 1 * 9
```

Figure A.1.: Visual formula editing in Scratch

Blockly[6] was also mainly designed for traditional computers (large screen, keyboard, mouse), but works on smartphones as well. It allows seamless switching between a purely visual approach similar to Scratch, and purely textual ones (alternatively JavaScript, Python, and XML) and back (see Figure A.2).

TouchDevelop is an application creation environment intended particularly for students or hobbyist programmers. It is intended to be used primarily on smartphones. The programming language is text-based but uses a few non-ASCII graphical characters to represent of the syntax, for example arrows, recycling symbols, etc. It resembles a traditional text-based programming language, though with a specialized editor (see Figure A.4) and use of annotation of program text and automated reformating. Formulas are entered visually but displayed textually, similar to Pocket Code's formula editor (see Figure A.5). In this particular example the user entered a syntactically wrong formula (two multiplication signs one after the other). Af-

[6]`http://blockly-demo.appspot.com/static/apps/code/en.html`

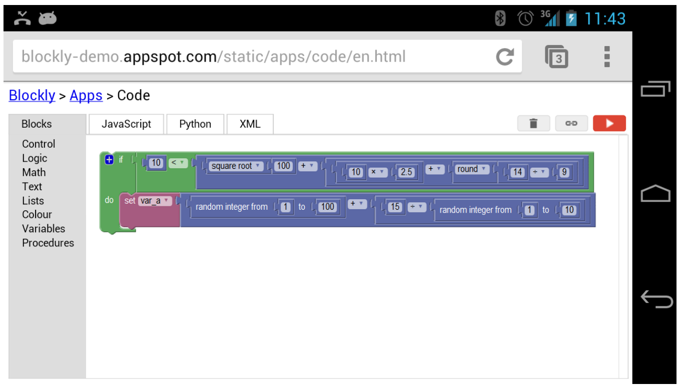


Figure A.2.: Blockly in landscape mode.

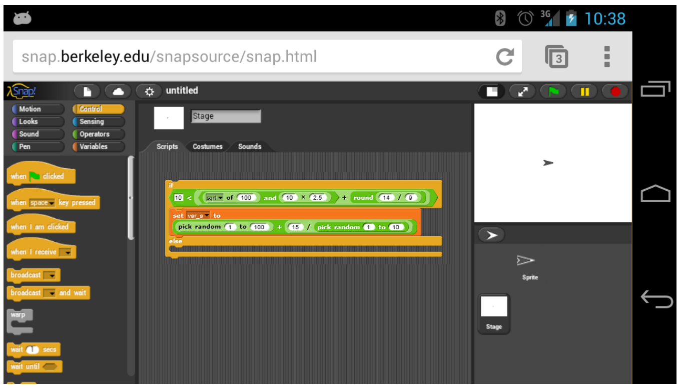


Figure A.3.: Visual formula editing in Snap! in landscape mode.

ter pressing the BACK-button an error message appears and the syntax error gets highlighted in red.

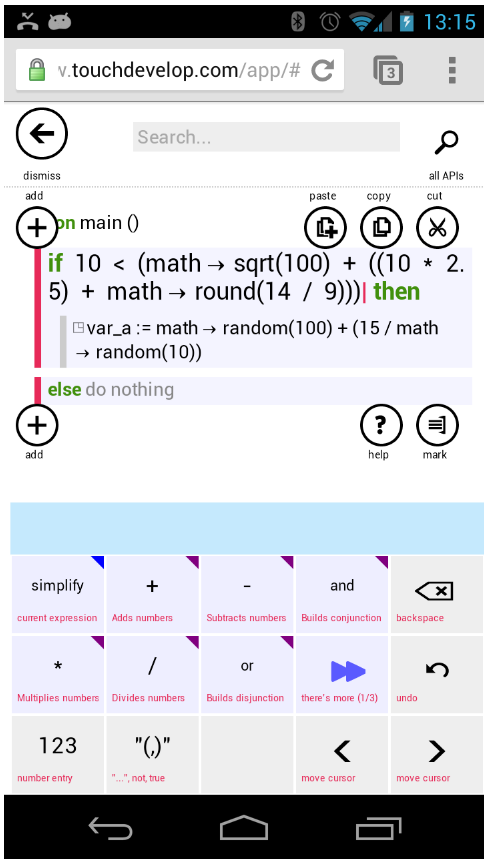
A.3. Pocket Code

Pocket Code is a free and open source mobile visual programming system for the Catrobat language[7]. It allows users, starting from the age of eight, to develop games and animations with their smartphones. To program, the children use their Android phone, iPhone, Windows Phone, or other smartphone with an HTML5 browser. No notebook or desktop computer is needed [37].

Pocket Code is inspired by, but distinct from, the Scratch programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab [35]. Similar to Scratch, its aim is to enable children and teenagers to creatively develop and share their own software. The main differences between Pocket Code and Scratch are:

1. Support and integration of multi-touch mobile devices

[7]http://catrobat.org/



```
on main ()
if 10 < (math → sqrt(100) + ((10 * 2.5) + math → round(14 / 9))) then
    var_a := math → random(100) + (15 / math → random(10))
else do nothing
```

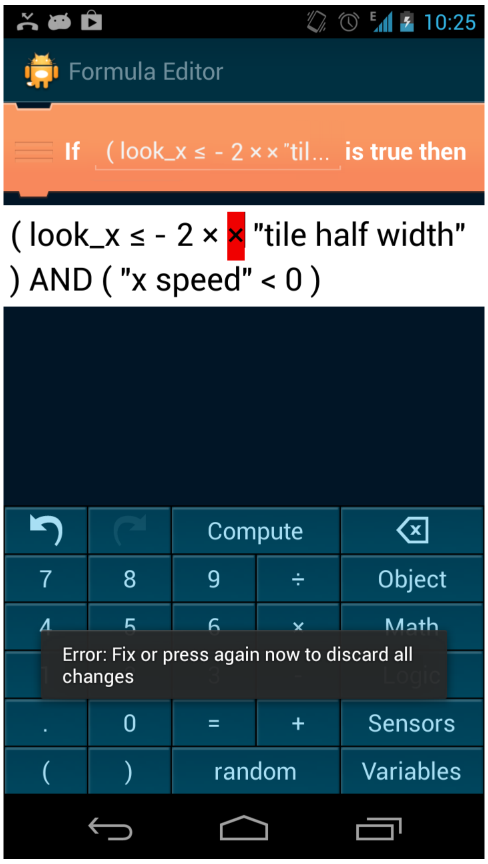
Figure A.4.: Textual formula editing in TouchDevelop.

2. Use of mobile device's special hardware (e.g., acceleration, compass, inclination)
3. No need for a traditional PC

Additionally, there are more than 30 ongoing subprojects mostly aiming at extending Pocket Code's functionality, e.g., a 2D physics engine that will make the programming of games similar to the popular Angry Birds type of games very easy. Another example an extension allowing a user to very easily record the screen as well as sound during execution of a program and to upload it to YouTube, the high definition video being created on our server and uploaded from there to avoid high costs and lengthy file transmissions for the kids.

Here a succinct overview of the design intentions of Pocket Code's formula editor is given:

- Show the user which statements operators, variables, messages are possible, thus simplifying discoverability for the user.
- Make it easy to edit on a small touch screen ("use only one's thumb to enter a whole program or formula").
- Use text based formula: easier to display on narrow screen (text wrapping) and at the same time well known from typical pocket calculators, calculator


10:25
Formula Editor
If (look_x ≤ - 2 × × "til... is true then
(look_x ≤ - 2 × × "tile half width") AND ("x speed" < 0)
Compute
7 8 9 ÷ Object
4 5 6 × Math
Logic
. 0 = + Sensors
( ) random Variables

Error: Fix or press again now to discard all changes

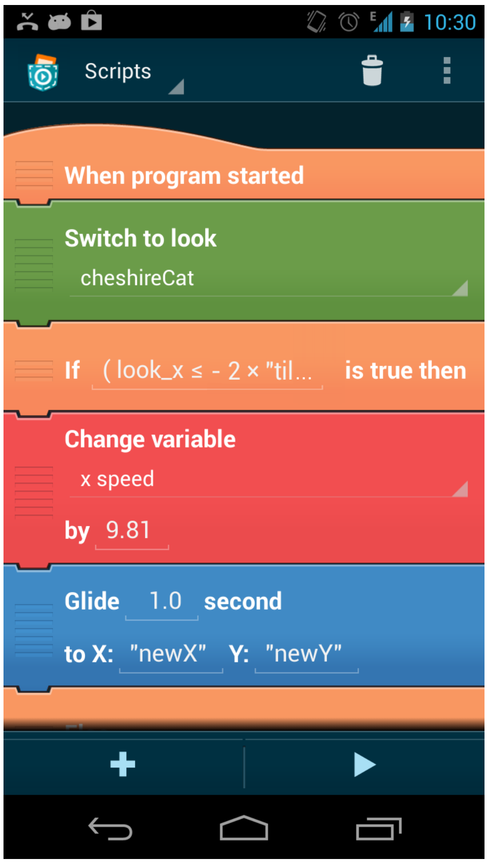
Figure A.5.: Textual formula editing in Pocket Code.

apps, but also from spreadsheets and even from math classes at school.

- Make it easy to get a preview of current value of a formula through the "compute" button.
- Give an overview of current variable values.
- Make it easy to position the cursor at any place in the formula, and to select larger parts of a formula.
- Visualize "matching" parentheses in complicated nested expressions.
- Eliminate some syntax errors preemptively.
- Highlight other syntax errors when user tries to use an "unfinished" formula (see Figure A.5).
- Scrolling of long formulas.
- Allow copy/cut/paste of parts of formulas.
- Allow for easy undo/redo.

A.4. Formula manipulation approaches under study

In this section two main ways of editing and displaying formulas are discussed (the third main approach, textual creation and textual visualization of formulas, was


Scripts

When program started

Switch to look
cheshireCat

If (look_x ≤ - 2 × "til... is true then

Change variable
x speed
by 9.81

Glide 1.0 second
to X: "newX" Y: "newY"

Figure A.6.: Visual script of a program written using Pocket Code that contains the If-brick with the formula that was edited in Figure A.5.

mentioned in Section A.2, but will not be discussed here):

1. Visual creation and visual representation of formulas like in Scratch and Blockly. Scratch (see Figure A.1) and other visual programming languages use purely visual formula editing. Different segments have to be nested within each other to compose a formula.
2. Visual creation and textual representation of formulas, like in TouchDevelop and Pocket Code. In TouchDevelop and Pocket Code's formula editor, formulas are created via a pocket calculator-like interface (see Figures A.4 and A.5). Statements and operators are selected visually, but the actual formula representation is purely textual. This should help save screen space and provide users with a better overview over their formula.

Visual composition of formulas can become a tedious task, because numerous visual components have to be nested within each other for more complex formulas. This is especially true for the small screens of smartphones. The screen limitations of mobile phones and the common knowledge of how to use a calculator led to the decision to display formulas textually in Pocket Code's formula editor. Most teenage and adult users know how to operate a pocket calculator and should therefore experience no problems with Pocket Code's formula editor. Future usability studies will determine whether the textual or visual approach works better for smaller children, who have not use a calculator before. Displaying formulas textually may be faster and more easily understandable for users who are familiar with pocket calculators. In order to evaluate the two different approaches we will conduct a formal experiment as described in the following section.

A.5. Evaluation of Pocket Code's formula editor

To assess the usability of Pocket Code's formula editor we followed the main objectives of User-Centered Design (UCD) methods defined by ISO 9241-210:2010[8] including user research, interface design, and usability testing during the implementation cycles. According to the agile principles used by the software development team, the UCD methods applied followed the agile methods as well, such as inspection, heuristic evaluation, paper mockups, and thinking aloud tests [25]. Additional to previously done formative testing we are planning to conduct a formal experiment in order to gain a summative assessment of the formula editor. This section describes the methodology that will be applied to evaluate the usability of Pocket Code's formula editor and compare it with three different programming language approaches.

A.5.1. Methodology

The purpose of the planned experiment is to provide scientific evidence to support or revoke the assertions described below. The following hypotheses are stated:

- Null hypothesis: For the manipulation of complex formulas, the calculator metaphor (the hybrid textual/visual approach) is more effective and efficient than the pure visual programming language approach.
- Alternative hypothesis: The contrary of the above null hypothesis: For the manipulation of complex formulas, the calculator metaphor (the hybrid textual/visual approach) is only as good or less effective and efficient than the pure visual programming language approach.

A complex formula in this context will be clearly defined, for example something like a logical formula composed of nested expressions at least 4 levels deep, with 12 parentheses, 3 variables, 1 sensor value, 6 constants, 4 logical operators, 6 numerical constants, and 8 operators.

Users will be allowed to use the phones in portrait and landscape modus. We will also experiment with different screen sizes and resolutions.

Before running the experiment a pilot test will be conducted to discover errors and to obtain extra practice for the research team [38]. We will evaluate several aspects

[8] `http://www.iso.org/iso/catalogue_detail.htm?csnumber=52075`

during the pilot test, like the reactions of participants, discovery of errors in the test setup, and the procedure for data processing and analysis.

After the pilot test and resulting adaptations to the test procedure, the real test with the null hypothesis presented before will be conducted. The following subsection describes the experimental design and specifies the test metrics.

A.5.2. Research Questions

The aim of the study is to compare four different programming language environments described in Section A.2. We want to find out what type of formula creation and visualization works faster, is more understandable, and preferred by the participants, when using and manipulating complex formulas: a purely visual programming language like Snap! and Blockly, or a hybrid programming language like TouchDevelop and Pocket Code's formula editor. To answer these questions, we will conduct a comprehensive formal experiment. Details of the planned test method and the experimental setup are provided in the following section.

A.5.3. Experimental Design

To evaluate the hybrid programming environment approach to formula editing, we are going to conduct a counterbalanced formal experiment with repeated measures. For the evaluation, we will randomly select 32 participants age 16. The participants will be recruited from schools in and around Graz. None of them will have any previous programming experience. The participants will be randomly distributed in four groups (A, B, C, and D). In each case, the participants will spend two hours in the experiment, first learning the basics of the system from a tutorial, and then trying to accomplish a series of tasks. First, they will be asked to create a very simple program in order to become familiar with the programming environment. The order of the tasks will be counterbalanced (see Table A.2) between the groups to avoid learning bias [39]. After each task the participants will be asked to fill out a feedback questionnaire for the purpose of collecting subjective qualitative data. The dependent variables that will be measured are

1. time spent on solving each task while using different programming environments
2. the number of successfully finished tasks
3. the number of tasks finished with help and
4. the number of errors occurred
5. quality of the programs created by the participants, rated by the test team.

Different applications, tasks, and time are going to be the dependent variables. In addition to the quantitative data, we will collect qualitative data as well. After each task, we are going to initiate a discussion with the participants and try to

Table A.2.: Tasks and Groups – counterbalanced formal experiment

	Pocket Code	TouchDevelop	Snap!	Blockly
A	1	2	3	4
B	4	1	2	3
C	3	4	1	2
D	2	3	4	1

get as much information as possible about their subjective experiences, information about what they liked and disliked, and what would have made the programming language more compelling, more useful, or easier to use.

A.5.4. Tasks

We are going to define tasks in such a way that the answers to the predefined research questions can be derived easily from the collected data. All participants will be asked to perform the following tasks:

T1 Programming some very simple introductory task to get familiar with the environment. During the first task participants will not have to use formulas.
T2 Extending an existing formula of a program.
T3 Writing some new formulas with the programming environment.

A.5.5. Experimental Setup

The experiment will take place in a laboratory at Graz University of Technology. We will use the same test setup for all of the tests, only the smartphone for TouchDevelop will be a Windows Phone. All other programming environments will run on Android devices. Hardware:

- Windows Phone or Android smartphone with maximum screen size of 5 inches.
- Laptop
- Video camera

The smartphone will be connected to the laptop, where the screen will be mirrored for the facilitator and captured with Morae[9], a usability software, for later reexamination of the tests. The laptop camera will record the user's face during the test to get the participants' facial expressions. Additionally an external camera will record the interviews and the whole test.

[9] `http://www.techsmith.com/morae.html`

A.5.6. Data Collection

The data will be collected from three different sources. The most relevant data will be compiled during the task execution. After each task, users will fill out a feedback questionnaire to provide some subjective feedback from the participants.

Appendix B.

Between Pragmatism and Infeasibility Agile Usability for Children and Teens in a Very Large Multi-Year FOSS Project about Several Complex Mobile Apps and Mobile Websites

The position statement and the resulted issues presented here was the motivation for this theses.

B.1. Abstract

In this position statement the challenges of integrating User-Centered Design (UCD) for children in an Agile Software Development (ASD) process are identified. Especially, the problems and challenges the developer team is facing in a Free Open Source Software (FOSS) project are pointed out.

B.2. Introduction

Unfortunately, Agile Software Development (ASD) initially did not take Usability and User Experience (UX) into consideration, and even nowadays the latter two are typically considered from the point of view of programmers only even though the end user plays a central role in all agile processes. Evidently, there are distinct differences between agile and UX approaches which bring up challenges to combine the two important software development areas. Most software is programmed for end users, and it would therefore seem to be self-evident to involve usability experts in the software development process. FOSS is widespread and a well established software paradigm [10]. However, little is known about the composition of

ASD Methods and FOSS movements, especially the composition of them with the integration of UXD.

B.3. Usability Lifecycle

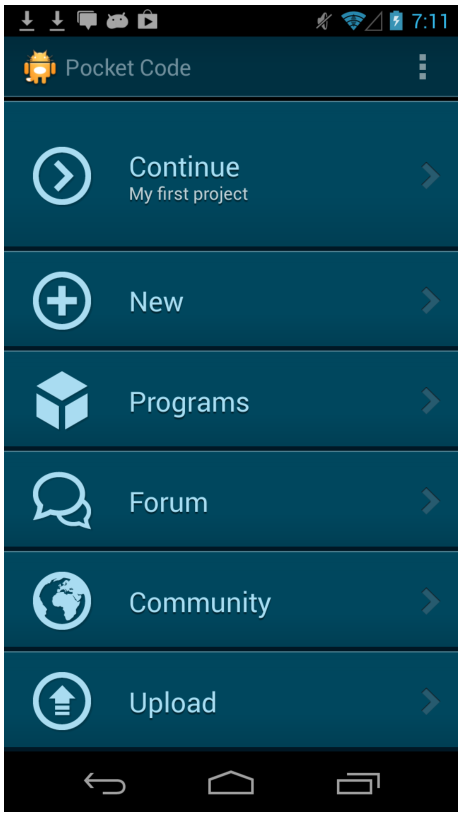
The Usability Lifecycle defined by Nielsen [15] consists of following eleven stages: know the use, competitive analysis, setting usability goals, parallel design, participatory design, coordinated design of the total interface, apply guidelines and heuristic analysis, prototyping, empirical testing, iterative design and collect feedback from the field user. The life cycle requires use of large number of usability methods. Usually the amount of time needed to follow the whole set of recommendations is insufficient, especially when it comes to Agile Usability, but the minimum set of recommendations given by Nielsen still applies in today's modern UX Design methods: "Visit customer locations before the start of the project, do iterative and participatory design, and use prototyping and empirical tests with real users" [15]. On the one hand Agile software development processes need ad-hoc usability because of the short release cycles, but on the other hand usability inputs in practice take longer periods. To overcome this problem Agile UXD was introduced by several authors [40][41][42][43].

B.4. Background on the Catrobat project

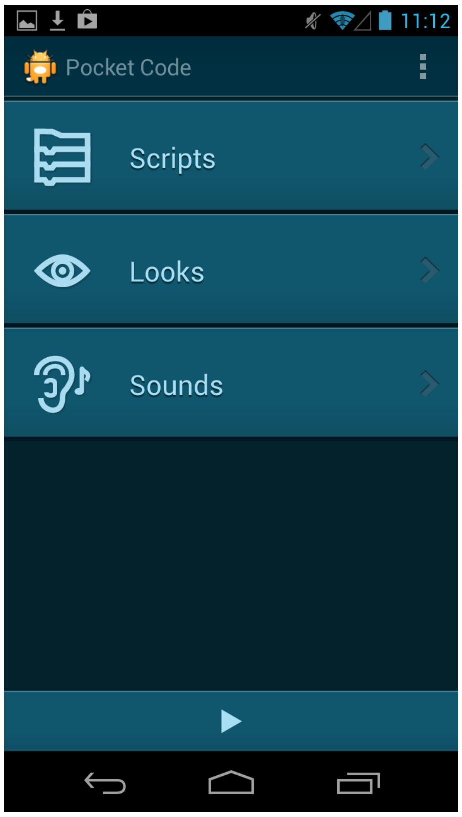
Catrobat (`http://catrobat.org/`) is a purely mobile visual programming system. IDEs and interpreters for Android, iOS, Windows Phone, and HTML5 browsers are developed by a large FOSS team. There are more than 30 subprojects, one for example on a 2D physics engine for Catrobat, another one on a Wi-Fi interface to Parrot's popular AR.Drone. According to Ohloh statistics, 162 developers have so far spent 117 man years of effort on Catrobat as of May 2013. Catrobat is inspired by but distinct from the Scratch programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab [29]. Similar to Scratch, the aim of the Catrobat project is to enable children and teenagers to creatively develop and share their own software.

The main differences between Catrobat and Scratch are:

- Support and integration of multi-touch mobile devices
- Use of mobile device's special hardware (e.g., acceleration, compass, inclination)
- No need for a traditional PC
- Availability on all major mobile platforms as well as HTML5 capable browsers



(a) Main screen of Pocket Code.

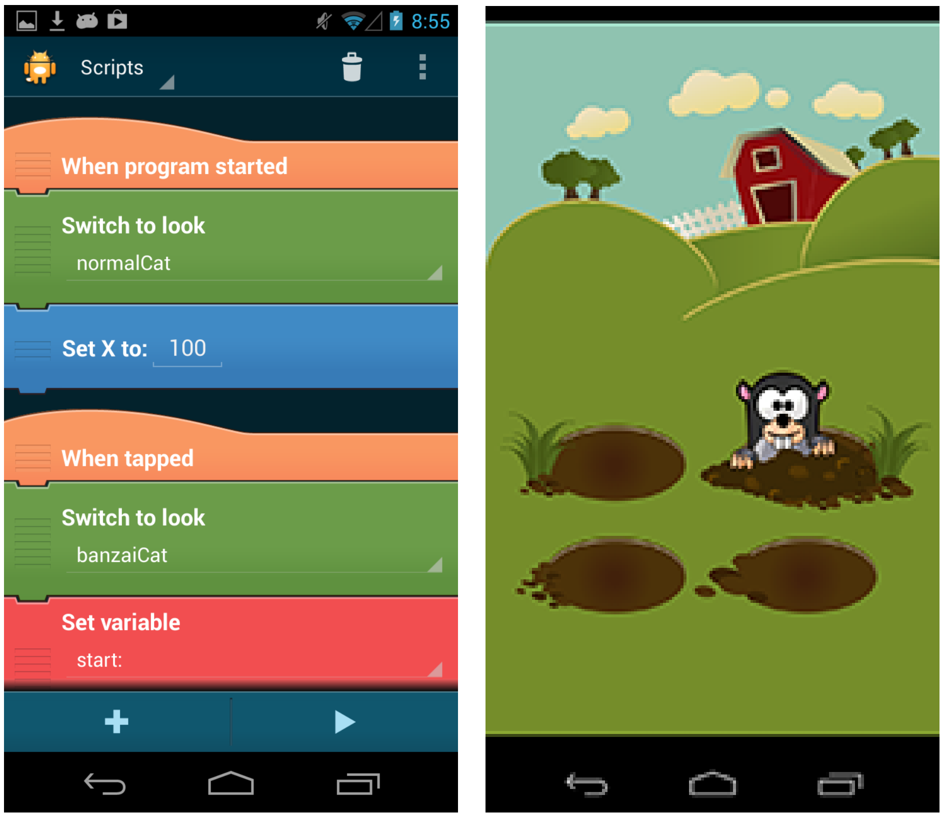


(b) Scripts, Looks, and Sounds.

Figure B.1.: Pocket Code.

B.5. Challenges - Case Study

Pocket Code (see Figure B.1(a),B.1(b)) is a free and open source mobile visual programming system for the Catrobat programming language mentioned above. It allows kids, starting from the age of eight to create storytelling and music animations or to develop their own games (Figure B.2(b)). Pocket Code is an IDE for a visual programming language, therefore it is a highly user-interface (UI) centric application (see Figure B.2(a)). For that reason the need for UX involvement occurred very early.



(a) Parts of "Whack a Mole" program. (b) "Whack a Mole" game.

Figure B.2.: Pocket Code.

Integrating usability in an agile test-driven software development was very challenging for various reasons. The software development initially started without UX integration, so usability had to be applied post-hoc. It turned out that a lack of knowledge about agile UX methods was one of the biggest challenges. The usability team did not really know much about Agile software development and Agile UX Design. Therefore, the UX team applied classical design and usability methods to ensure usability.

B.5.1. Major Challenges

The problems that arose with that approach were:

- Lack of rapid prototyping.
- Designs were too detailed, with more need for mockups.
- Too much documentation and lack of communication.
- The UX team was always behind the programming team.
- Too long testing phases.
- Complicated usability test setups and methods.
- Providing feedback on usability issues took too long.
- Children and teenagers are the end users, but it was difficult to find appropriate test persons on demand.
- The UX team had difficulties dealing with the constantly changing requirements even in the late development phase.

General problems with our FOSS project:

- many people are involved in the development
- developers change all the time and are volunteers
- developers are from all around the world (e.g., Google Summer of Code 2013)
- no constantly working, paid developers

B.5.2. Outlook

In order to integrate User Center Design in our FOSS Agile Software Development project, the next step will be to survey existing apposite User-Centered Agile Methods and adapt them so that they fit into the Catrobat project.