

Practices of agile development methods in development and improvement of an open-source community website project

A comparison of methods and strategies using agile software-development practices for the continuous development and improvement of the Catrobat community website.

Master's Thesis

Institute of Software Technology
Graz University of Technology

Alexander Gütlér
alexander.guetler@tugraz.at

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

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTÄTTLICHE ERKLÄRUNG

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

Graz, am

.....
(Unterschrift)

Englische Fassung:

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.

.....
date

.....
(signature)

Abstract

Catrobat is a visual programming language using graphical elements for building programs, games or animations. It is based on the idea of Scratch – a graphical programming language using a Lego®-brick metaphor – that makes it easy for kids to build their own programs without prior knowledge of any programming language. The Catrobat community website provides a platform for kids and teenagers to share their work with others all over the world by the use of their smartphones and tablets.

The Catrobat community website has been developed and improved since 2010, using agile software development methods like test-driven development, pair programming, self-organised teams, welcoming changing requirements and many more. Depending on the structure of the teams and the software to be developed, finding the best combination of methods is obviously not an easy process and will need adaptation over time, as not all of the agile principles published in the Agile Manifesto can be used in everyday software development projects.

This master's thesis is about development and improvement of the Catrobat community website, and highlights the use of agile methods in an open-source software-development project with the challenge of frequently changing teams and the continuous introduction of new team members. It also describes a new Git branching model for the use in a large open source software development project consisting of various sub-teams and many different features to be merged together in one release. Experiences over about three years of development and improvement of the Catrobat community website are reported. Currently still missing features for a successful online community are described.

Kurzfassung

Catrobat ist eine grafische Programmiersprache die es aus grafischen Elementen ermöglicht, Programme, Spiele oder Animationen zu erstellen. Basierend auf der Idee von Scratch – einer grafischen Programmiersprache, angelehnt an die Verwendung von Lego®-Steinen – die es Kindern einfach macht, Programme ohne Programmierkenntnisse zu erstellen. Die Catrobat Homepage ermöglicht Kindern und Jugendlichen ihre auf Smartphones oder Tablets erstellten Projekte weltweit mit anderen interessierten zu teilen.

Die Entwicklung der Catrobat Homepage begann 2010 und wurde seit damals stetig erweitert. Dabei wurden zahlreiche agile Entwicklungsmethoden wie Testgetriebene-Entwicklung, Paar-Programmierung, selbst organisierte Teams, die Bereitschaft auf Änderungen positiv zu reagieren und viele mehr angewandt. Unter Berücksichtigung der Team-Strukturen und der zu entwickelnden Software sind die Kombination sowie die Auswahl der besten Methoden nicht immer einfach. Diese müssen im Laufe der Entwicklung an die auftretenden Bedürfnisse angepasst werden, da nicht immer alle im „Agilen Manifest“ genannten Prinzipien in der alltäglichen Softwareentwicklung angewandt werden können.

Diese Masterarbeit beschäftigt sich mit der Entwicklung und Verbesserung der Catrobat Homepage und beschreibt die Verwendung von agilen Methoden in der Entwicklung eines großen Open-Source Softwareentwicklungs-Projekts. Einige der größten Herausforderungen sind die häufig wechselnden Teams und stets neue hinzukommende Entwickler. Des Weiteren wird ein neues „Git Branching Modell“ zur Verwendung in großen Open-Source Softwareprojekten bestehend aus zahlreichen unterschiedlichen Teams und unzähligen neuen Funktionen und Möglichkeiten vorgestellt, um diese in einem gemeinsamen Paket (Release) zu veröffentlichen. Erfahrungen aus über drei Jahren in der Entwicklung und Verbesserung der Catrobat Homepage werden vorgestellt und fehlende Funktionen und Möglichkeiten für eine erfolgreiche Online-Plattform werden beschrieben.

Contents

1. Introduction and Motivation for Research.....	12
1.1. About Catroid and the Catrobat Project	12
1.1.1. History of Catroid-Project	14
1.1.2. About Catrobat Mobile Application Development	15
1.1.3. Catrobat-Subprojects	15
1.1.4. Current Website Development	17
1.2. About Scratch.....	18
1.3. Motivation for Improving the Pocket Code community website	19
1.4. Agile-Development Methods	20
1.5. Online Communities	21
1.6. Thesis Overview	21
2. Related Work.....	23
2.1. Kodu.....	24
2.2. GameMaker.....	25
2.3. Flipnote Hatena and HatenaBlog	26
2.4. Wario Ware D.I.Y.	26
2.5. Little Big Planet	27
2.6. Lego ® Mindstorm.....	27
2.7. Some other Online Communities	28
2.8. Scratch.....	28
3. Theoretical Background	32
3.1. The Agile Manifesto	32
3.2. The Agile Principles.....	33
3.3. Agile Software Development Methods	35
3.4. Extreme Programming	36
3.4.1. Values of XP	36
3.4.2. Principles of XP	36
3.4.3. Practices of XP	37
3.5. Test-Driven-Development.....	38
3.5.1. Methods and Principles used in Test-Driven Development	39
3.5.2. Quality of Tests	39
3.6. Testing Methods.....	40
3.7. Planning	40
3.8. Kanban	42
3.9. TDD tools.....	44

3.9.1.	PHPUnit	44
3.9.2.	Selenium.....	46
3.9.3.	Watir.....	46
3.10.	Source code management	47
3.11.	Continuous Integration	47
4.	Current Research and Studies on Test-Driven Development.....	49
4.1.	Understanding test-driven development.....	50
4.2.	Software- and Design-Quality in test-driven development.....	52
4.3.	Test-driven Development in professional software development.....	53
4.4.	Effectiveness of TDD.....	55
4.5.	TDD of web-based applications and relational databases.....	59
4.6.	Debugging strategies.....	60
4.7.	Common mistakes in TDD practices	61
4.8.	Conclusions on current research topics.....	62
5.	Agile Website Development.....	64
5.1.	Planning Games for Releases – agile estimation and planning.....	64
5.2.	User-Stories.....	65
5.2.1.	Story-cards	65
5.3.	Using Kanban.....	65
5.4.	Pair-Programming	66
5.4.1.	Collective Code Ownership	67
5.4.2.	Code Reviews.....	67
5.5.	Testing.....	68
5.5.1.	Unit Tests with PHPUnit.....	69
5.5.2.	Database-Testing.....	71
5.5.1.	GUI-Testing with Selenium	72
5.5.2.	Test-Data-Generation	72
5.5.3.	Programming for Testability	73
5.6.	Daily Standup-Meetings.....	73
5.7.	Development-Environment	73
5.8.	Test-Automation and Deployment Process.....	74
5.9.	Continuous Integration.....	75
5.10.	Usability Issues and Screen Design	75
5.11.	Version control	76
6.	Git Branching Model	77
6.1.	Git and GitHub.....	77
6.2.	Some best practice Git branching models.....	77

6.3.	A successful Git branching model	79
6.4.	Another Git branching model.....	82
6.5.	A new Git branching model	84
6.6.	Results and Conclusion	85
6.6.1.	Feature development	86
6.6.2.	Bug fixes in release branch	87
7.	Website Improvements and newly added Features	88
7.1.	Uploading and browsing	89
7.1.1.	Automatic registration process on project upload.....	90
7.1.2.	Automatic bad-words filter check on project upload	91
7.1.2.1.	Black-word lists	91
7.1.2.2.	White-words list.....	91
7.1.3.	Lost username or password.....	92
7.1.4.	Editing a Project's Title and Description	92
7.1.5.	Browsing projects on website by category	93
7.1.6.	Remixing projects	93
7.2.	Project details.....	94
7.2.1.	Adding tags to projects.....	94
7.2.2.	Recommendation of similar projects.....	95
7.3.	HTML-5 player for projects.....	95
7.4.	Internationalization	95
7.5.	Website design changes	96
7.6.	Community tools.....	97
7.6.1.	Report projects as inappropriate.....	97
7.6.2.	Discussion board	99
7.6.3.	Help-pages and Tutorials	99
7.7.	Backend functionality	100
7.7.1.	Managing users	100
7.7.1.	Blocking users	100
7.7.2.	Managing projects.....	101
7.7.3.	Check reported projects.....	101
7.7.4.	Bad-words filter and contents	101
7.8.	Work in progress.....	102
8.	Results and Conclusions	103
8.1.	Selenium Tests	103
8.2.	Kanban Board and Stories.....	103
8.3.	Development environment.....	104

8.4.	Pair programming	104
8.5.	New team-members	105
8.6.	Test-Driven Development.....	105
8.7.	Standup-Meetings	106
9.	Future work.....	107
9.1.	Joint venture with scratch.....	107
9.2.	Registration	107
9.3.	Online tutorials and videos.....	108
9.4.	Responsive web-design.....	108
9.5.	Project management and feature-lists	108
9.6.	Improvement of the community-parts of the website	109
9.6.1.	Discussion board (forum).....	109
9.6.2.	Remixing of projects	109
9.6.3.	Commenting on projects and automatic bad-words filter	110
9.6.4.	Reviewing projects and user control	110
9.6.5.	Organizational issues.....	111
9.7.	Implement short development-cycles for continuous release of features	111
9.8.	Help-pages: getting started, exploring, guides	111
9.9.	Test-driven-development exercises.....	112
10.	List of Figures.....	114
11.	References	115

1. Introduction and Motivation for Research

Playing computer games is a good entertainment and a great experience for kids of all ages and genders. Within the last years, more and more games were being developed for mobile and handheld devices instead of traditional platforms like Desktops, Laptops or Gaming-Consoles (e.g. Sony Playstation3¹, Microsoft Xbox 360², Nintendo Wii³). Many popular games are now available for the Google-Android and the Apple iOS platforms like Google Nexus 4 phone, Google Nexus 7 tablet, Samsung Galaxy 4 phone, Apple iPhone or Apple iPad tablet, just to mention some of them.

This thesis is about a community website as a place for young “programmers” to share their projects, ideas, techniques and interests about programming on a handheld device. It will provide a very deep insight of development and improvement of the Catroid-community-Website over 18 months, the challenges of constantly changing team-members, using test-driven-development methods, automated unit- and GUI-testing using PHPUnit and Selenium, the usage of a Kanban board for coordination of development, the change in the projects name and a complete redesign of the website itself. It will also resolve some issues about website- and integration testing (done mostly with PHPUnit and Selenium), finding the right branching model for Git version control and using Jenkins for daily builds as explained in more detail in *“Introduction of a Continuous Integration Process in an Open Source Project”* [Bur12].

1.1. About Catroid and the Catrobat Project

“Catrobat is a purely mobile visual programming system. IDEs and interpreters for Android, iOS, Windows Phone, and HTML5 browsers are developed by a large FOSS⁴ team.” [Krn13]. In 2010, when development of Catroid started, there were only three small development groups, working on an *“on-device graphical programming language that runs on Android devices and that is intended for children”* [Gri11]. Based on the idea of Scratch⁵, developed at the MIT in

¹ Sony PlayStation: <http://www.playstation.com>

² Microsoft Xbox 360: <http://www.xbox.com/en-US/xbox-360>

³ Nintendo Wii: <http://www.nintendo.com/wiiu>

⁴ FOSS - Free and Open Source Software

⁵ Scratch Programming Website: <http://scratch.mit.edu>

Boston, Catroid programs are written in a graphical LEGO-style, that intuitively shows the kids (from an age of 8) which bricks will fit together, like shown in figure 1:

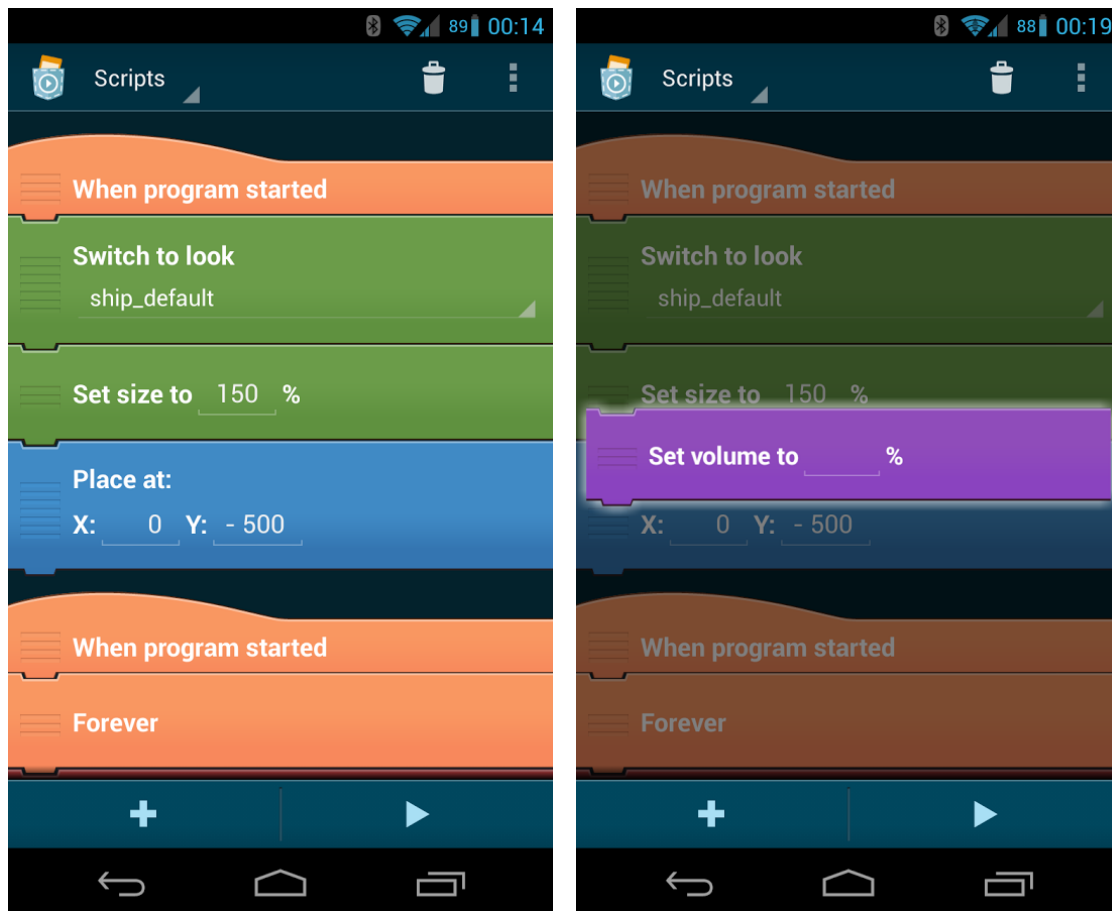


Figure 1: Bricks used for graphical programming and how they can be put together [Cat13]

With the use of the same colours as in Scratch, everyone that knows how to work with Scratch can immediately start using Pocket Code. We are currently working on moving these two programming platforms for kids closer together. There have already been some changes in Pocket Code's semantics, to better fit within the world of Scratch; as before, new objects have been invisible by default, but in Scratch, new objects (called sprites) are visible. To make it easier for Scratch users creating programs with Pocket Code, these semantic changes have been applied in August 2013.

Catroid now is a part of the Catrobat-Project, consisting of more than 22 subprojects to support integration of other platforms (iOS, Windows-Phone), supporting the LEGO MindStorm Bricks, controlling Parrot's AR.Drone 2.0, a HTML5-Player of the uploaded programs and many more. The App was renamed to "Pocket Code" in summer 2013. As we changed the name from Catroid to Pocket Code, the community's website domain changed from catrobat.org to pocketcode.org. This change was initiated by a new design of our project's logo

and some usability research done together with FH Joanneum Graz. Paintroid, another application developed by our teams is an easy to use paint-editor for Android devices, supporting easy-to-use transparency operations on images. Paintroid was renamed to “*Pocket Paint*”, to promote the “pocket”-allegory we already used for “*Pocket Code*”.



Figure 2: New Pocket Code (left), new Pocket Paint logo (middle) and old version (right)

We are still using the working titles for these two apps, "Catroid" for the Pocket Code version and "Paintroid" for the Pocket Paint version. This is historical, while in the beginning of the Catroid-Project, the only supported mobile platform was Android – that is also why all our first project names ended with “*roid*”, out of respect for Google’s Android⁶.

1.1.1. History of Catroid-Project

As mentioned in the previous section, the Catroid-Project has started with only three subprojects in 2010 – the Catroid-IDE, Paintroid and the Catroid-Website. When programming mobile devices got more and more interesting for students at our University, the team of the Catroid-Project grew rapidly to more than 100 students during one semester. Handling that many team-members was quite a challenge, especially when these students only attended the project for about 5-10 hours a week, for the duration of 12 weeks in total. Then we decided to start with the development of the Catroid-IDE on other platforms.

To get the new team-members ready for developing and testing, there was a need of some tutorials, the introduction of a so called “*buddy-system*”⁷ and the preparation of Virtual-Machines⁸ for development, that helped us to skip the cumbersome process of setting up the development-environment on different hardware (from Windows XP to Windows 7, Mac OS

⁶ <http://www.android.com>

⁷ A new team-member was mentored for the first two to three weeks by a senior developer, introducing coding-standards, test-driven-development our kanban-system, to give them a better start in our team.

⁸ Oracle’s VirtualBox was used to provide the development-environment for the Catroid-Website; see <http://www.virtualbox.org>

and all kind of Linux distributions). The problems and benefits of these Virtual-Machine-Images will be discussed later in more detail.

After the projects' first year and in participation of Google Summer of Code⁹, 2011 the project was renamed Catrobat-Project¹⁰, an umbrella organisation for all the subprojects, including Catroid-IDE, Paintroid, Catroid-Website and many more.

In 2012 the design was renewed and the Catroid-Application was renamed to Pocket Code as well as the Paintroid Application (for drawing on mobile phones) was renamed to Pocket Paint. Pocket Paint is available in Google Play since June 2013.

1.1.2. About Catrobat Mobile Application Development

At the beginning of the Catrobat Project in 2010 - the name of the project was initially called Catroid, a name to show the connection with the Android-operating-system - the applications being developed were mainly the two Apps for Android Smartphones – Catroid Interpreter (a Scratch like visual programming environment for kids) and Paintroid, a simple but powerful and easy to use Painting App with some really good features e.g. handling transparency. For version control we were first using Mercurial hosted on Google-Code, until mid-2012 and then switching to GitHub for several reasons described later in detail in chapter 3. With the project becoming larger every year and supporting even iOS and Windows-Phone applications, the whole project was renamed to Catrobat, an umbrella organization, now supporting around 28 different subprojects, most of them still in an alpha development phase. The two apps from the very beginning of our project were renamed to Pocket Code, which is currently available as a public-beta release at Google Play-Store and Pocket Paint, which is already available at the Google Play-Store. We are also hosting a community website (pocketcode.org) as a platform for sharing, downloading and remixing programs created with Pocket Code.

1.1.3. Catrobat-Subprojects

The following subprojects are a part of the Catrobat Project:

- Pocket Code for Android (“Catroid”)

⁹ Google Summer of Code is a global program that offers post-secondary student developers ages 18 and older stipends to write code for various open source software projects, <http://www.google-melange.com>

¹⁰ <http://developer.catrobat.org>

- Pocket Paint for Android (“Paintroid”)
- Pocket Code community website
- Pocket Code for iOS (“Catty”)

Currently under development but still in alpha or beta stage is:

- Tutorial game
- HTML5/JavaScript edition
- iOS edition (beta)
- Windows Phone edition
- Android stand-alone apk builder
- Android live wallpaper builder
- Arduino I/O via Bluetooth for Catroid
- Parrot AR.Drone via Wi-Fi for Catroid, with OpenCV support
- Computer vision (similar to Scratch 2.0)
- Translators support system (pootle)
- Lego Mindstorms sensor support for Catroid
- Physics engine for Catroid based on Box2D
- Sony Xperia Play (PlayStation certified) key support
- YouTube recording of stage for Catroid
- Musicdroid that allows to enter musical notation by singing
- Catroid as a Wireless Human Interface Device (gamepad, mouse, keyboard) for PCs and Xbox, Wii, PlayStation etc.
- Transcode Scratch programs into Catrobat programs
- Tablet Integration
- Drag & Drop in Pre-Stage
- URL shortening service for Catrobat
- Young kids version (ages 3 to 7): story-telling only version
- A website dedicated to educators using Catrobat
- Near Field Communication (NFC) for multiplayer coordination
- Multilingual wiki and forum for Catroid users for small screens
- Support for the Albert Robot of SK Telecom
- Textual language version of Catrobat
- 3D version

1.1.4. Current Website Development

As proposed in [Gri11], we added some more features to assist the community building process of the website. Those features are:

- Remixing of projects
- Adding tags to projects¹¹
- Adding a recommender system
- Commenting on projects
- Adding “like it” or “love it” button

The main design has completely changed since the beginning of the project, because there was no display of our numerous projects on the main page of our community website as well as no “featured projects” section. What’s more, the reduced screen size, challenges the navigating through a great number of projects. In our first version of the Pocket Code website, the projects were moved up and down by a button at top and bottom of the page, making it impossible to navigate through a few hundred projects. We have changed the welcome page of the website to show three groups of projects: most downloaded, most viewed and newest projects. Still to be implemented is a content-management system to support the creation of a featured projects section.

Some parts of the Pocket Code website had to be changed after running a number of usability tests with kids, aged 9 to 11 in June 2012 [Kna12].

“To help Catrobat become more popular and to foster the collaboration of kids when making games or animations with Pocket Code, the website still needs some improvement regarding community issues”, [Gri11, p.61]. At the current state of development, there are still some relevant features missing:

- adding comments to one’s own project
- tagging not only one’s own but also someone else’s project
- comment with “like” or “love it” on someone’s project [Res09]
- navigation (browsing) through projects [Res09]
- newest uploaded projects localized [Gri11, p.52f]

¹¹ The projects’ tags and recommender system is part of our Google Summer of Code participation in 2013.

- featured projects (e.g. what the community is loving, what is remixed, etc.) [Res09]

1.2. About Scratch

„Scratch is a programming language and online community where you can create your own interactive stories, games, and animations“ and “Scratch is a project of the Lifelong Kindergarten Group at the MIT Media Lab. It is provided free of charge“ [Mit14]

Its programming language consists of colourful graphical Lego-style objects, that fit together only in some ways, making it easier to find the correct combinations for one's program.

Scratch is developed by Mitchel Resnick and his team at the Boston Massachusetts Institute of Technology (MIT). Until version 1.4 of the Scratch software, they had relied on Java-Applet Technology, but starting from version 2, they are using Adobe's Flash Shockwave player technology. A major change in Scratch was, that from version 2.0 on, there was no need to install the software locally, instead it could be used right from within a flash-player¹² enabled web-browser. Since August 2013, an offline editor for Scratch exists for downloading and installing on Mac or Windows computers. The fact that Scratch uses Adobe's flash-technology to run the projects in a web-browser, limits its use to desktop and laptop computers (including netbooks or e.g. Google's Chromebook). It cannot be used with Apple iOS devices or current tablet computers. There is a really fantastic and well supported community website available with lots of materials for children and teachers, information for parents and thousands of projects to download, remix and learn from. While writing this thesis, more than 5.085.321 projects became available from their website, with a total number of 2.980.490 users and a total of 24.928.608 comments¹³.

¹² Adobe Flash Player: <http://get.adobe.com/flashplayer/>

¹³ Scratch website statistics: <http://scratch.mit.edu/statistics/> [Lif13]

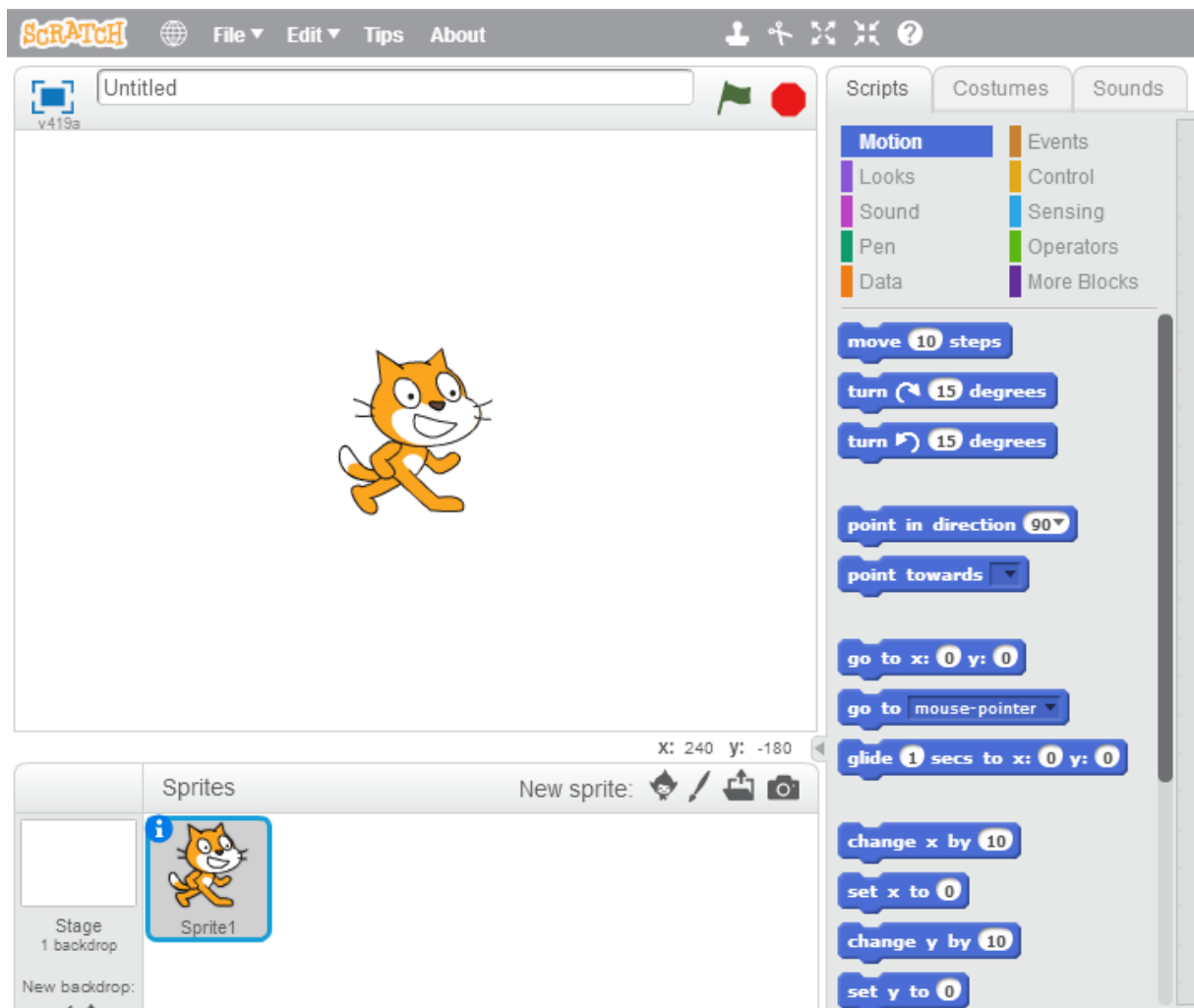


Figure 3: Scratch – new project [Mit13]

1.3. Motivation for Improving the Pocket Code community website

There are still some improvements to be made for the many users of our community website. Users shall be able to upload their own picture or select an avatar for their personal profile page on the website. Adding some information about e.g. one's country, one's interests and ideas should also be possible. Another idea of scratch – to have groups of users creating projects together in studios – would be nice to implement into the Pocket Code-website. Giving credits to users in their own words when remixing one's projects is crucial for the success of the website and should be possible when uploading a remixed version of a project. Remixing of projects and attribution has been evaluated by the Scratch-team in [Mon11] and [Hil10].

On our Pocket Code-website¹⁴ there are still some missing features like commenting on someone else's projects, presentation of work and ideas of individual users, a more detailed description of the projects and tags to get a quick overview, what the uploaded project is about. Currently there are only four sections on the website to present projects like featured, most downloaded, most viewed and newest. Adding e.g. a "like this" or similar button to help users find good projects is also essential for the success of our project. A good example of presenting projects in different groups can be found at the Scratch-website¹⁵, e.g. "*what the community is remixing*", "*what the community is loving*" and featuring projects by individual contributors. Since the number of projects and users from all over the world is growing daily, there is a need of improvements in localization of projects. Newest projects can be shown by region (like the user's country) or by the user's language.

Another aspect of improvement was changing the currently used self-developed PHP-framework for running our community website. Developing this website has been done using test-driven-development and the framework was supporting all features needed in the beginning; but with more features to add, such as presenting individual projects with screenshots and more text, the currently used PHP-framework will have to be adapted or replaced by a more powerful one.

1.4. Agile-Development Methods

Thanks to our University, we got a large project room (with about 16-20 seats) where most of the programming, planning, testing and stand-up-meetings have been taken place. Many agile-methods and practices have been used throughout all our teams. "Planning game" meetings have been held to create new features for the next release/program version. Large whiteboards with story-cards have been used as Kanban-boards. Pair programming was done most of the time and this particularly was a very good way to get new members of the team to build up know-how and understand all test- and program-code. By writing tests first and having a nearly complete unit-test coverage, there was no need for comments in our source-code. Comments in tests were allowed when absolutely needed. Daily stand-up-meetings were held, so everyone in the project room could get a quick overview of the other team's work and ideas.

¹⁴ <https://pocketcode.org>

¹⁵ <http://scratch.mit.edu>

One of the differences of agile software development in our open-source project compared to commercial projects was that we neither had a customer nor a hard deadline. The customer's role was "played" by our professor, Wolfgang Slany. Features and releases were planned, sometimes with only little coordination between sub-teams (e.g. Catroid and Web-team). This led to a long delay of the final 1.0 version of Pocket Code and the need of a new Git branching model presented in Chapter 6. Collective code ownership made it easy for us to proceed with many team member changes on a high frequency basis.

1.5. Online Communities

Compared to other online communities, the Catrobat website should become a platform for young people (from the age of 8) to be creative, express their ideas with the ease of a LEGO-style visual programming language based on the idea of Scratch. Creating, sharing, remixing and communicating one's work with other young innovators and friends should be easy and intuitive and become possible without the need of a Desktop- or Laptop-Computer.

Regarding parental concerns, bad or inappropriate content can be reported to the Catrobat team with a link from the Project's website. Reporting inappropriate content needs an explanation, and regarding to the concerns, the project will be checked and – if necessary – removed from the Pocket Code website. We have additionally implemented a bad-words-filter to check the username, the project's title and description when uploading a project. If any bad words are found within these fields, the project will not be visible on the website and the user will receive some information why the project will not be published. Unknown words to the bad-words-filter are stored in a database and the Catrobat team will be informed of these new words for approval or rejection.

As we learned from the Scratch Community website, to keep our young users motivated, there will be no "dislike" button available, and to avoid derogatory comments on projects, a user shall be able to deactivate the comments feature for it.

1.6. Thesis Overview

In Chapter 2, I will provide a short overview of other programming tools for kids and teenagers and their community projects. In Chapter 3 the theoretical background of agile software

development methods and the ideas behind test-driven development (TDD) will be presented. Chapter 4 gives an overview of the current research on test-driven development and will highlight some issues like code- and test-quality, TDD in professional software development and the effectiveness of TDD on web-application- and relational-database-development.

The agile development methods used for development and improvement of the Pocket Code community website will be discussed in Chapter 5, the new Git branching model used in our project is introduced in Chapter 6, whereas in Chapter 7 the new features and improvements of the Pocket Code-website are presented. Results and conclusions are given in Chapter 8, future work will be discussed in Chapter 9.

2. Related Work

Catrobat is based on the idea of Scratch, bringing programming to everyone on a handheld device, without the need to learn a programming language.

The Scratch-team has developed a new version of their programming environment and released it in June 2013 with the name “Scratch 2.0” after several years of prototyping, starting in 2010 with an early version of the new, browser embedded, development environment.

Mitchel Resnick describes one of the main changes of this new platform as followed: *“Scratch 2.0 joins the programming environment with the online community so it’s much easier to create and share”*.¹⁶

By integrating both, the programming environment and the website, it is more easy for kids to use Scratch, as no installation or special hardware is needed. They just have to register for an account and can start designing, programming, remixing and sharing. New projects are automatically uploaded to the cloud-storage, so one’s work can be continued on a different PC or Laptop with some internet-connection.

There are still some other programming environments with a community website oriented towards children like Microsoft’s Kodu, YoYo Games’ GameMaker, Nintendo’s Wario Ware D.I.Y. and Hatena Flipnote, which was closed on May 30, 2013 [Gri11] [Nin13]. With many games developed for mobile devices like Apple’s iPad, Game consoles like Sony PlayStation 3 and Microsoft Xbox 360, there are some communities arising within the manufacturers stores, to create and share modified levels or characters of their games, like Sony’s “Little Big Planet 2”, “Minecraft” (available for all platforms) – but all these products are commercial.

Hatena Flipnote was closed and is now continued with Hatenblog coming with a free and a paid blog plan, with its main focus lying on users from Japan. Compared to Google’s blogger.com, it nearly has the same features.

¹⁶ <http://citilab.eu/en/resnick/scb/interview>

2.1. Kodu

With *Kodu*¹⁷, developed by Microsoft Research FuseLabs¹⁸, kids can create games on a PC and Xbox with a simple and easy to use visual programming language suitable for everyone from the age of 8. It can be used for teaching creativity, problem solving, storytelling as well as programming. It can be taught by every teacher, because no previous programming knowledge or experience is needed. *Kodu* is available for free to be used with a PC. The Xbox version is commercial and only available in the USA. The *Kodu* community website has a discussion board, a “worlds” section to share games with others users and many tutorial videos on how to use *Kodu*. There is also a classroom kit available, supporting educators using the software in their classrooms.

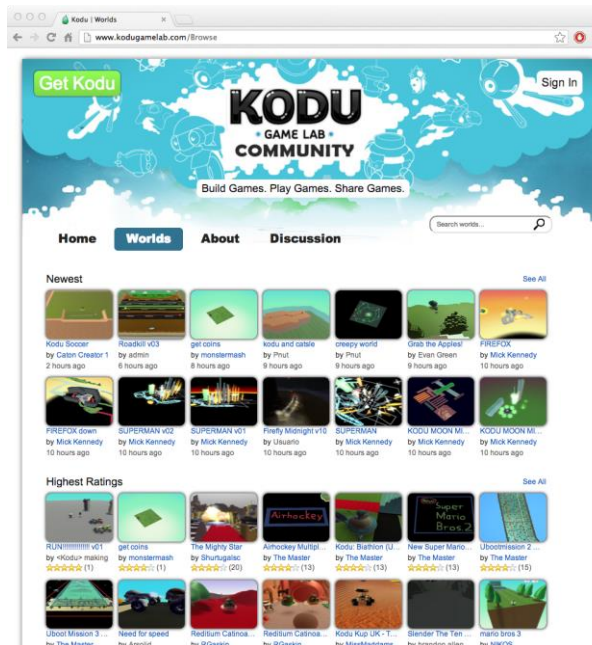


Figure 4: Microsoft Kodu GameLab community website: project overview [Kod13]

¹⁷ <http://www.kodugamelab.com>

¹⁸ <http://fuse.microsoft.com>

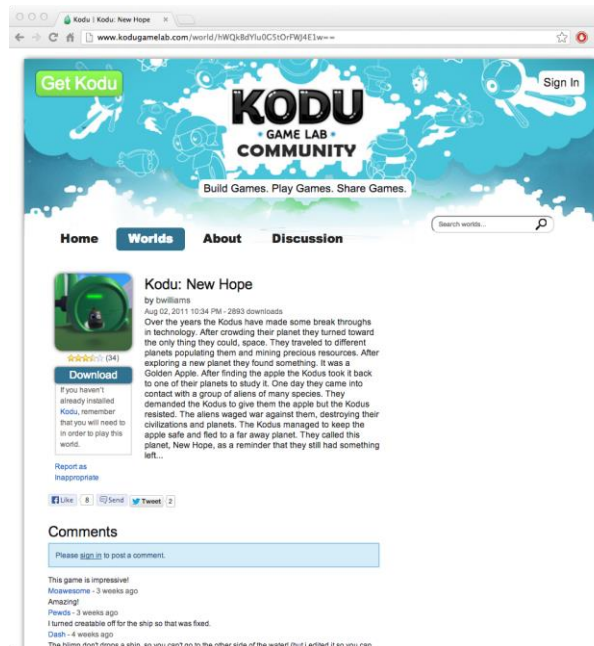


Figure 5: Microsoft Kodu GameLab community website: project details page [Kod13]

People share approximately 350 new projects every week (08/2013), and the community website presents the worlds in three categories: “Newest”, “Highest Ratings” and “Most Popular”. Some of the most popular worlds have more than 2500 downloads. Projects are shown with a thumbnail representation, the project title, the author’s name and the date of creation. On the details page a short description is shown. Adding a “star” to one’s project, writing and reading comments and reporting content as inappropriate can only be done when logged in, using login-data from either Windows Live ID, Facebook or twitter [Mic13].

2.2. GameMaker

GameMaker by YoYo Games is a professional gaming studio software, e.g. “*GameMaker: Studio Family*” for building applications and mostly games for nearly all platforms including (Windows PC, Windows 8, Mac OS X, Android, HTML5, iOS, Ubuntu and Windows Phone 8). Users can share their games in a “sandbox”, review games and write comments.

Developing games with the use of rapid prototyping is really fast as the GameMaker-Studio has everything needed to create a game inside its IDE and also use the build-in scripting language. Compared to Scratch and Kodu, GameMaker focuses more on users with programming or game design experience. Besides a free version of the GameMaker-Studio software, many more commercial products starting from \$49 up to \$499 exist. Official tutorials, games in progress, beta games, a wiki and various resources are available from their community website¹⁹.

¹⁹ <http://sandbox.yoyogames.com/make/tutorials>

Resources include backgrounds, sprites, sounds, music, scripts and other useful utilities for game-development. Shared projects are presented in the following categories: featured, top rated, most played and most recent, with a total of 421.132 members, 144.112 games created and 47.630 of them reviewed by other members of the community (08/2013). To use most of the community website features one must be logged in. There is no “report as inappropriate” feature and for the registration process, a name, an e-mail address and the date of birth is needed. Some users use a real-life picture of themselves [YoY13].

2.3. Flipnote Hatena and HatenaBlog

Decreasing popularity and usage of the Nintendo DSi, which was used for creating flipbook-like animations by the Flipnote Studio, a pre-installed software on Nintendo’s DSi handheld gaming device, the Flipnote Hatena website and Flipnote Hatena for DSi were closed by end of May 2013. A community service is continued in a different way by HatenaBlog, “*a modern blog service for people who love to write.*”.²⁰ Nintendo is currently working on a new Application for the Nintendo 3DS device and will launch the new software *Flipnote Studio 3D* in summer 2013 [Nin13]. Flipnote Studio 3D includes two community services, *Flipnote Gallery: Friends* as a free service and *Flipnote Gallery: World*, as a paid service (\$0.99 per month). Within the first service-plan, projects can only be shared with friends, within the second service-plan projects can be shared with users from all over the world.

To foster sharing of projects and activities of users in *Flipnote Gallery: World*, Nintendo provides a bonus system to get free access to the *Flipnote Gallery: World*, so called Coin-Points and Collector-Points [Nin13a].

2.4. Wario Ware D.I.Y.

As Nintendo’s game Wario Ware D.I.Y. is not available for the new Nintendo 3DS device, the micro game-development is no longer available.

Created games could be played on Nintendo Wii as well as shared online [Gri11].

²⁰ <http://blog.hatena.ne.jp/>

2.5. Little Big Planet

Sony has shipped their Playstation3 gaming console with the famous puzzle platform game Little Big Planet (short LBP)²¹, focusing on user-generated content and the principles *play, create and share*.

The experience of it is best described by:

“If you were to stand on LittleBigPlanet and try to imagine a more astounding, fantastic and creative place, full of enthralling adventure, uncanny characters, and brilliant things to do... you couldn't. All imagination is here, and what you do with it all is entirely up to you.” and *“The possibilities are endless with LittleBigPlanet. So, what do you say? Let's get out there and play!”* [Son13].

Currently there are more than 8.000.000 games (=levels) available on the community website, easy to use for children with tags categorizing each game. The levels can be starred, liked, put together on lists and downloaded to play. The community website as well as the tags are available in many different languages.

2.6. Lego ® Mindstorm

With the new programming environment in Lego® Mindstorm EV3²², it gets really easy for kids starting to program their own functioning robot. At first, an app can be downloaded on an iOS or Android device to have an instantly available remote-control for the Lego® robot out of the box. After some first experiments with controlling one's robot (that is really easy to build within about 15-30 minutes by kids aged 12 or older), a graphical programming development environment, powered by LabVIEW²³ can be downloaded to either a Windows or Mac computer. Installation is really simple and the programming environment comes with some video-tutorials (from one up to five minutes), showing the basic principles of programming (variables, conditions and loops). The sensor-blocks of the system are also demonstrated – and again –programming can be done within only a few minutes. After uploading the new program to the robot via USB, Bluetooth or WLAN, kids can present their work to their parents. The system also provides the possibility to create own program blocks with more complex functionality inside. Building-blocks are organized in colors, where each group has different functions like loops, variables, sensors, arithmetic and others.

²¹ <http://lbp.me>

²² <http://www.lego.com/en-gb/mindstorms/?domainredir=mindstorms.lego.com>

²³ LabVIEW: <http://www.ni.com/labview/>

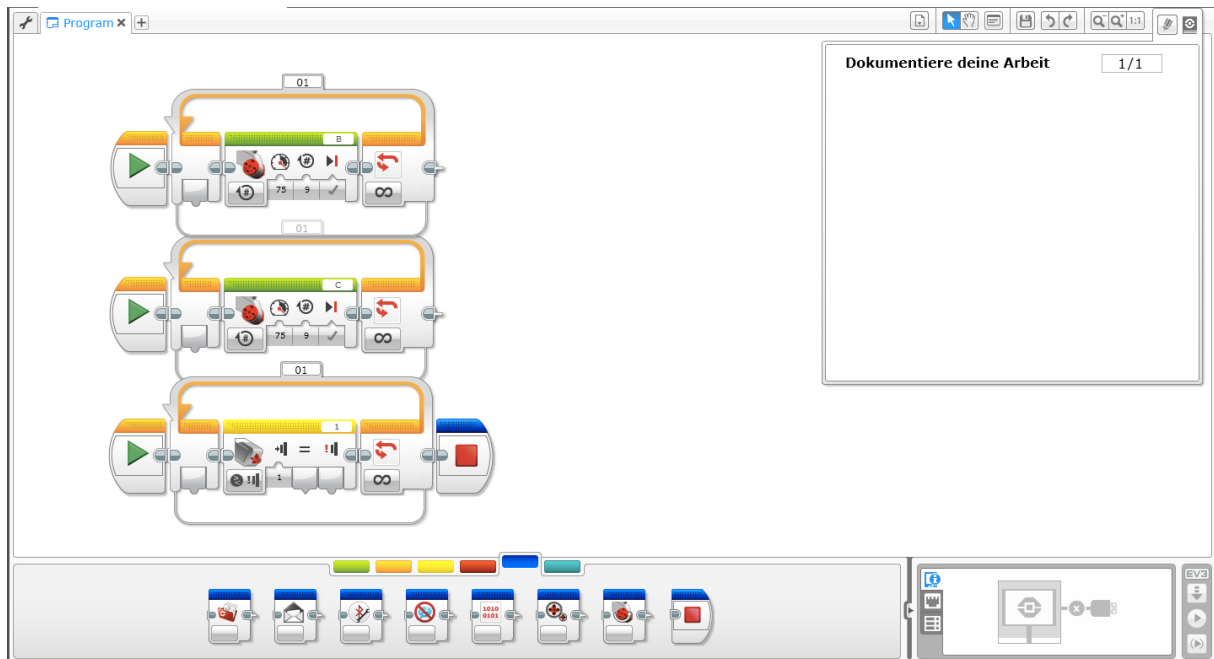


Figure 6: Lego Mindstorm Programming environment [Leg13]

2.7. Some other Online Communities

Today, there exists a great number of community-websites for interaction. One can find these systems useful or useless, depending on the benefits of these systems. In Facebook e.g., there are a lot of games, chat-possibilities, sharing tools and other nice things to have inside a digital community.

2.8. Scratch

Scratch was created and developed by the Lifelong Kindergarten Group at MIT by Mitchel Resnick and his team. As programming was taught in schools by using hard to learn programming languages - and because at the time (2002), there were no easy ways to learn programming languages available for kids – they started to develop a new, visual programming language to support the way kids where thinking and make programming easier and more intuitive. *“Scratch uses a building-block metaphor where blocks only fit together in syntactical correct ways, so the users will not have to care about syntax-errors, that are one of the biggest issues when teaching kids how to program.”* [Res02]

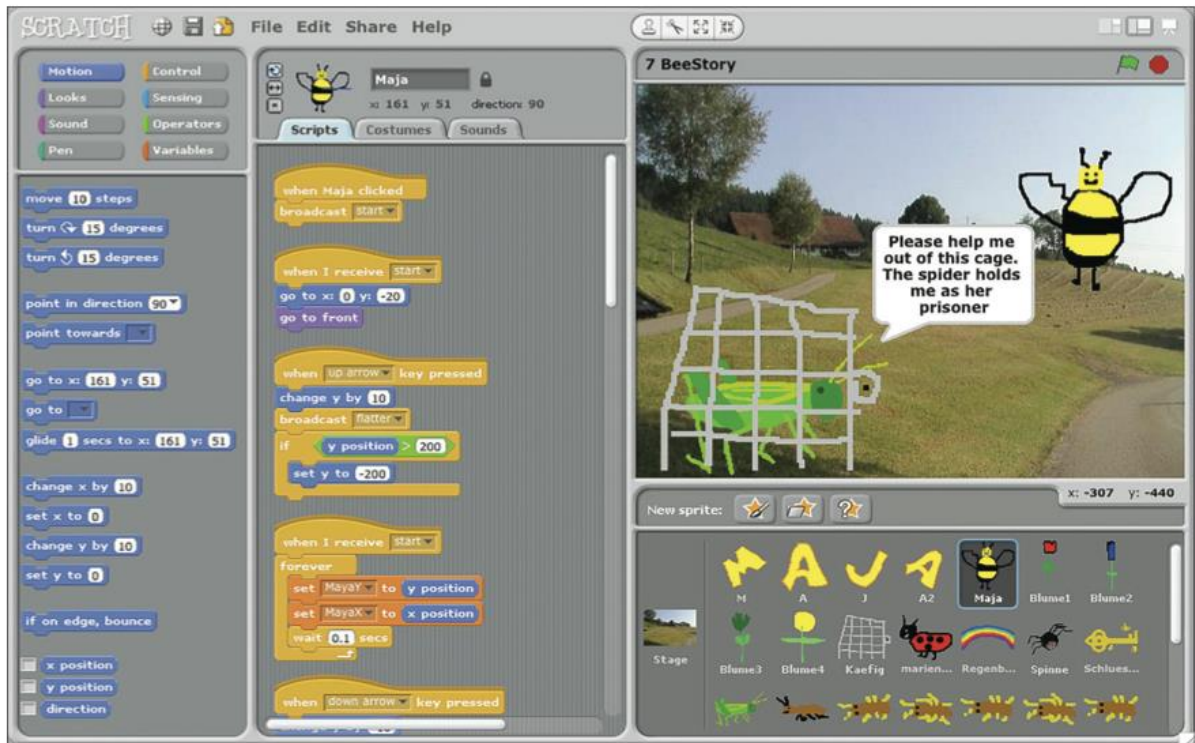


Figure 7: Scratch 1.4 programming editor window [Res09]

Scratch has developed these blocks to easily fitting together in a way like Lego blocks do - so users can choose any blocks needed for the next steps of their program. Blocks can be dragged into the scripts area and even lay there unused, until they are needed. This way of programming fosters young users to play around with the bricks and try out or run small parts of their program instead of first building a complete version of it.

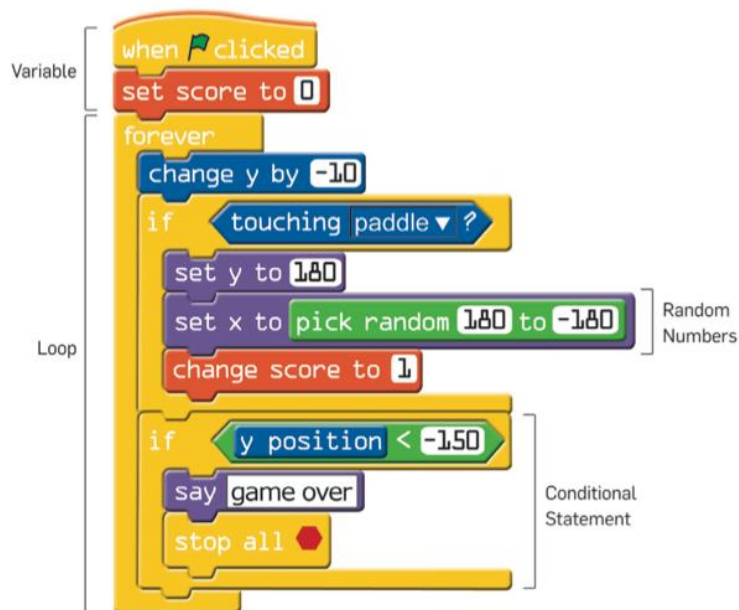


Figure 8: Scratch programming blocks [Res07]

The three core design principles of Scratch are: *“make it more tinkerable, more meaningful, and more social than other programming languages”* [Res09].

Users can create a various number of different programs with Scratch, like stories, games, animations and simulations. Programs can be personalized by adding own pictures, recorded voices and graphics.

Since Scratch was launched in May 2007, the Scratch website became an online community with mostly young people sharing and remixing projects. The biggest invention of Scratch was the combination of the programming environment with an online community. In that community kids could “show” their work to friends and download programs from other users to see how they did implement special features in their programs (that were mostly games, stories and presentations). Users can also add comments to other users work and rate it by “loving” it. To keep kids motivated and prevent them from “bashing” each other’s work, only positive ratings could be given. A “don’t love” button is not available. Users can also give credit when remixing another user’s project. By giving credit to another user’s work, kids can use the ideas of others like graphics and sounds within their own projects and by doing so, they learn to respect others work by naming the authors and giving credit [Res09][Res02].

To help other “Scratchers” with their questions and to make sharing of one’s programs easier, a community website was brought up together with the Scratch programming language. Users can upload their projects, comment on projects of other users and find support and help. Registration is very easy and no personal data, except statistical information about year of birth, country, gender, nickname and e-mail address is stored. As by 2009, about 15% of uploaded projects were remixes. As a future goal of Scratch, *“the primary focus lies on lowering the floor and widening the wall, but not raising the ceiling”* [Res09].

MIT has launched a new Version of Scratch (2.0) in summer 2013 with many new features and some technological changes. The program editor can now be loaded directly within the browser window, projects can be stored in the “cloud” and even variables can be made available to other projects via cloud-storage technology. While the 1.4 version of Scratch was using a Java-applet player, the new 2.0 version uses shockwave/flash for their player. This limits the use of scratch on mobile devices, as both, Google on Android- and Apple on iOS-platforms do not support Flash.

Programming with Scratch 2.0 can easily be done by just visiting the Scratch website²⁴ and clicking on “create”. There is no need to register first when wanting to just try it out. Scratch is available in many different languages, has a large online community and supports teachers and parents with online tutorials and teaching materials like tutorial cards. Projects can be downloaded from the Scratch website, remixed and shared again by giving credits to their creators.

²⁴ <http://scratch.mit.edu>

3. Theoretical Background

Since the publication of the “Agile Manifesto” in February 2001 by seventeen people in a Ski resort in Utah, USA, the style of programming has changed persistently. Introduction of the agile methods for software development, also known as Extreme Programming (XP) has changed the daily work for many software developers [Agi01], [Bec05].

Agile Development can now be found in all commercial software projects from small to large companies as well as in open-source development and university-studies.

In the next subsections I will give a theoretical background of the XP-methods and in Chapter 5 a detailed overview what practices were used for development and improvement of the Catrobat community website (pocketcode.org) as well as the benefits and disadvantages we have faced with agile development methods and practices. I will outline, what methods worked best for us, where and why we needed to customize them to fit our team structure and development process, and which of the methods didn’t work at all. As social issues are a major success factor in building teams in a large FOSS development project this work will examine their influence in the context of extreme programming.

3.1. The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- 1 - Individuals and interactions over processes and tools*
- 2 - Working software over comprehensive documentation*
- 3 - Customer collaboration over contract negotiation*
- 4 - Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

[Agm01]

“Individuals and interactions over processes and tools” - It is more important to have a good, motivated team with time to talk about your product and features, instead of sticking to a strict

process and tools when developing software. Sometimes it can still be useful to have a strict process-guide – but most of the time, it is not!

“Working software over comprehensive documentation” - Shipping software that works is more important than spending large amounts of time writing documentation. Tests are, when easy to read and execute, a good and sufficient documentation. Documentation of how the system works and how it can be configured can of course be provided, and sometimes must be provided for the users of the system, but the details of its implementation shouldn't need extra documentation when there are tests for each feature.

“Customer collaboration over contract negotiation” - Communication and collaboration with the customer should always be preferred over arguing about contract-details. In fact, a written contract is important to list the responsibilities for both parties. It also provides a good understanding of what is being developed and will be delivered.

“Responding to change over following a plan” - Having a plan in the beginning is a good point where to start. But as plans will change over time (they will!), it should be possible to make changes to it, whenever needed. Of course these changes have to be confirmed and committed by both parties. The concepts of these four statements on the right side should be valued, but the ideas on the left hand side should be valued even more [Agm01] [Amb13].

But what does agility mean? Andy Hunt gives a good explanation in [Hun09, p.4], within only one sentence: *“Agile development uses feedback to make constant adjustments in a highly collaborative environment.”*

3.2. The Agile Principles

The twelve principles of agile software development were given by the authors of the agile manifesto to give a better understanding of their philosophy [Agm01], [Abm13], [Bec03]:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers must work together daily throughout the project.*

5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7. *Working software is the primary measure of progress.*
8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9. *Continuous attention to technical excellence and good design enhances agility.*
10. *Simplicity - the art of maximizing the amount of work not done - is essential.*
11. *The best architectures, requirements, and designs emerge from self-organizing teams.*
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.*

There exist several methods that will help to achieve the benefits of the agile software development principles mentioned above, which are discussed in the next sections. These methods are no “silver bullet” or the “one-and-only solution to build great software”, but by following these principles of agile software development both developers and customers will be more satisfied with the results they get – high quality software and a reliable product.

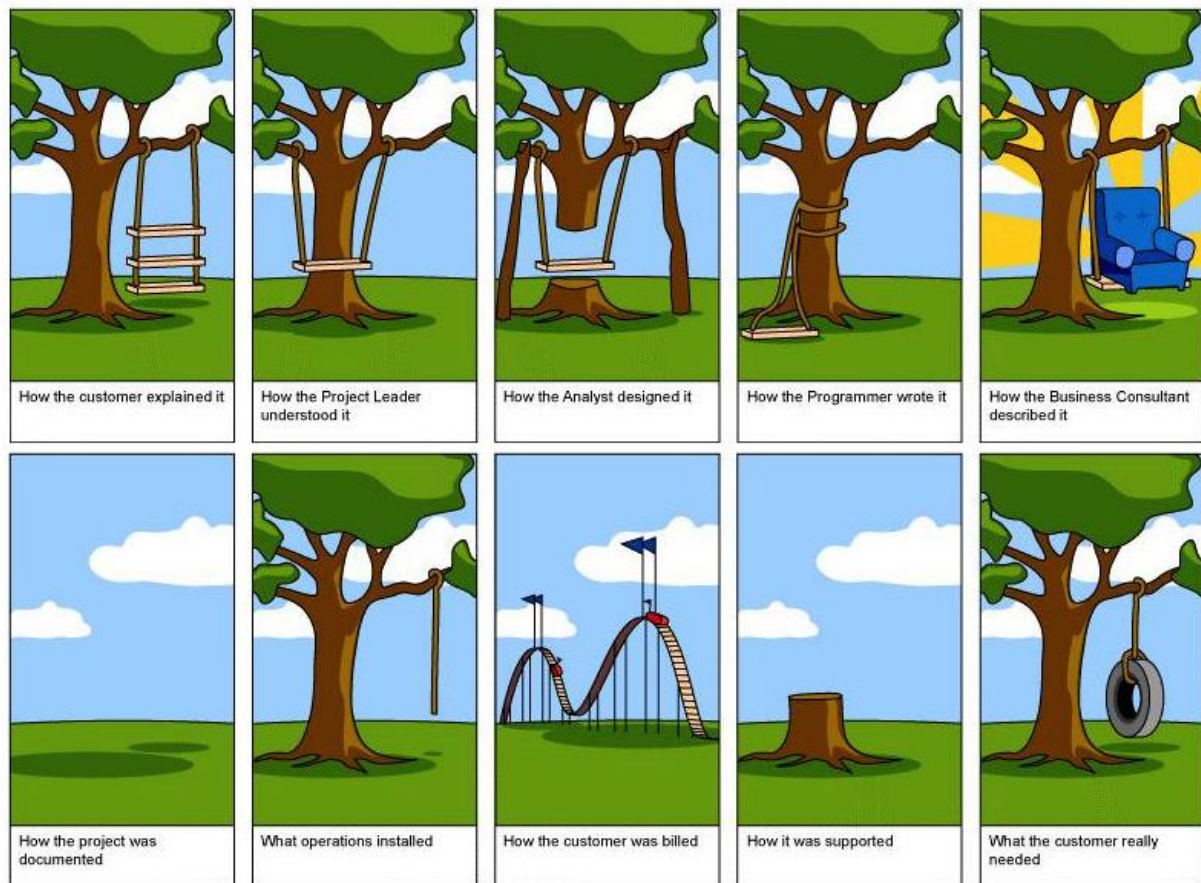


Figure 9: Projects being released in today's software development²⁵ [Ssw13]

3.3. Agile Software Development Methods

Within agile software development, many methods have evolved and new combinations of existing ones can be found, some of the most popular of them are: Extreme Programming (XP), Kanban, Test-Driven Development (TDD) and Continuous Integration (CI). There also exist some derivations of test-driven development like feature-driven development (FDD) and behaviour-driven development (BDD). The word “agility” was defined by Andy Hunt in [Hun09, p.1] as follows: *“being able to quickly adapt to an unfolding situation”*. The agile approach to software development *“emphasizes people, collaboration, responsiveness and working software”* [Hun09, p.1].

²⁵ <http://rules.ssw.com.au/Management/RulesToBetterScrumUsingTFS/Pages/default.aspx>

3.4. Extreme Programming

Extreme Programming (XP) *“is about writing great code that is really good for business”*, [Bec05]. As with every new style of programming, the XP practices result in controversial reactions. With the introduction of XP programming practices by Kent Beck et. al, a new way of software development came up. What exactly is XP? *“XP is a style of software development focusing on excellent application programming techniques, clear communication, and teamwork which allows us to accomplish things we previously could not even imagine”* [Bec05].

There is no right way to do extreme programming. As XP consists of many practices and methods, they can be combined as they fit best in a development team. The values, principles and practices of XP are presented in the next subsections.

3.4.1. Values of XP

There exist five values of XP: communication, simplicity, feedback, courage and respect. Communication is very important in software development. For many problems there already exists a solution - some team-member will know. Simplicity is defined by Kent Beck in [Bec05, p. 7]: *“what is the simplest thing, that could possibly work?”*. Feedback is essential when there is some need of change. The sooner the team gets some feedback, the easier it is to implement the changes for the next release. Courage is to face a problem and do something about it, because sometimes it takes some time for the problem to evolve and to become clear. The last of the five values of XP is respect. The team should care about the project and care about the team members, so everyone in the team should be respected [Bec05], [Hun09].

3.4.2. Principles of XP

Some of the basic principles that guide XP are: humanity, economics, mutual benefit, improvement, diversity, reflexion and quality. There should be a good balance between programming and other human needs, like recreation, socialization or some exercises. Highest priority tasks should be completed first, to maximize the business value for the customer and make the software more valuable. Mutual benefit can best be described with not writing internal documentation. When there are tests for all parts of the code, refactored regularly and following a good and meaningful naming convention, there is no need for documentation.

The saved time can be spent for implementing new tests and new features, again. Improvement is best described by Kent Beck in [Bec05, p.28]: *“Put improvement to work by not waiting for perfection. Find a starting place, get started, and improve from there”*. Diversity is an opportunity and not a problem. Having more than one solution for a problem can make the best out of it. Reflection should come after action, to think about how and why the team is working. Quality should not be used as a variable for control, developers should be proud of the system they are developing.

3.4.3. Practices of XP

Some of the XP practices stated by Kent Beck [Bec05] are given below:

- Sit together: it is important to find a suitable workspace for the team that fosters direct communication
- Whole team: means all the skills you need should be in the team
- Informative workspace: put story cards on a wall, e.g.
- Energized work: don't burn yourself – *“it's easy to remove value from a software project; but when you're tired, it's hard to recognize that you're removing value.”* [Bec05, p. 41]
- Pair programming: program in pairs, by using only one monitor, keyboard and mouse and regularly change the typing-role; change pairs regularly; it can be compared to driving a car: one is the navigator, the other stays on the road.
- Stories: write story cards defining the task with an estimation of time
- Weekly cycles: work should be planned weekly
- Ten-minute-build: the process for building the whole system and running all the tests should be done within ten minutes
- Continuous integration: changes should be integrated and tested every few hours to get immediate feedback on time
- Test-first programming: work in small cycles - write a test, write the code, refactor the code – so it is always clear what has to be done next: fix the broken test or add a new one
- Incremental design: implement only things that are needed today, do not think of future features
- Onsite customer: it is good, to have one of the customers sit next to team, if not possible, the team should have regular meetings and keep the customer well informed

[Bec05], [Sho08]

3.5. Test-Driven-Development

The idea of test-driven development (TDD) with only one sentence was best described by Koskela [Kos08, p.4]: “*Only ever write code to fix a failing test*”. As this is quite hard to believe, it keeps us on track to only build features we will need, because there was a test written before. Following these small steps by writing a test, adding some code, and so on, we will get a clean, lean design, a (nearly) full test-coverage – as there are some tests that cannot be written or shouldn’t – and as a result, code that just works.

Software changes over time, new features have to be added, existing ones need to be removed. Without having tests for all the modules and classes that need to be extended or changed, the risk of breaking existing functionality is high. But it’s not only the isolated unit-test, that gives us confidence of not breaking existing features, there are much more tests around the system like integration and acceptance tests. By running all the tests we can be sure that no side effects are going to sneak into the existing codebase.

One cannot expect a quality boost from a post-development testing phase after the code is complete. Testers in traditional software development processes wait for features to be finished and spent their time reading requirement documents and creating test plans based on these documents [Cri09].

The five main steps of TDD are manifested as [Bec03]:

- *write a small test*
- *run all the tests and see the newly added test fail*
- *make the simplest possible change in your code to make the test pass*
- *run all the tests and see them all pass*
- *refactor your code to remove duplication*

These five steps shall be iterated until all tests needed for the production code are in the system. To achieve a high quality of tests, the focus should not be on writing a large number of tests, when only a few will suffice. One must always keep in mind that tests will also need refactoring over time when new features are added – and as Whittaker writes in [Whi10], sometimes it is better to trash old tests if the underlying logic changes too much and old tests would need to be entirely rewritten. Test-driven development is also about writing and designing code to be testable.

3.5.1. Methods and Principles used in Test-Driven Development

The principle of writing small (unit) tests, testing the right things and keeping source-code and tests clean by refactoring after each change (when needed) and removing duplication should always be kept in mind during development. Tests should aim for quality rather than quantity and should maximize speed. With mock- or stub- objects, database or file-system access can be simulated, to test for speed variables. Granting everyone in the team access to the whole code gives all members the opportunity to correct failing tests caused by broken code. This collective code ownership spreads the knowledge around the team and should minimize the so-called “truck-factor”²⁶.

The following principles and methods are used with TDD:

- Always write the tests first
- Write tests and code, not documentation
- Keep refactoring and remove duplicate code
- Make the tests run automatically, e.g. after every commit to the master-branch.
- Often integrate and run integration and acceptance tests
- Keep collaborating and communicating – use pair programming and regular stand-up meetings to exchange solutions of problems and discuss current state of development and testing

[Bec03], [Bec05], [Kos08]

3.5.2. Quality of Tests

The quality of tests can be measured with statement coverage or defect insertion. Statement coverage, it should be 100 percent, is not the best metric for measuring test quality, but it is quite useful to gain a quick overview. Besides there are a number of tools available²⁷. The other method is defect insertion: by changing only one line of code the tests should fail. Here it depends on the line that has been changed, while a different return value of a constant might not break the tests at all [Bec05].

²⁶ Truck-factor or bus-factor: „is a measurement of the concentration of information in individual team members“
(http://en.wikipedia.org/wiki/Bus_factor)

²⁷ Code coverage tools: <http://c2.com/cgi/wiki?CodeCoverageTools>

3.6. Testing Methods

There are many different methods for testing, depending on the purpose of the tests, the context and the knowledge of the system. The first question asked is: what shall be tested - the behavior of the system, response time of a module or the correct return value to some given input? Based on these questions, the following testing-methods are frequently used:

- White Box Test: the behavior of the system and all classes and interfaces are known, the tests focus on detailed system knowledge
- Black Box Test: only public interfaces of the classes are known, the tests focus only on the return values
- Grey Box Test: only some behavior and interfaces and classes of the system are known
- Unit Test: smallest part of a test – each test is used to test only one part of the system; a group of unit tests can be combined in a test-suite.
- Integration Test: by adding new features to a running system the integration is verified by running all tests and some extra tests with their only focus on the integration of the new features
- System Test
- Acceptance Test: tests of all user stories
- Regression Test: adding new features or changes in configuration can lead to some side effects that can be discovered by regression tests
- Usability Test: tests based on usability issues, normally only on graphical interfaces (GUIs)
- Mutation Test: by changing or removing small parts of classes or functions, some tests should fail; if they don't, this will show that some tests are still missing.
- Performance Test
- Security Test

Depending on the purpose of the software system, these tests shall be combined to reduce the number of defects prior to the release of the system.

3.7. Planning

For a project, we have four variables that directly affect its outcome: cost, quality, time and scope. As we normally cannot influence cost and time (these are fixed by the customer), the only two remaining factors for a project are quality and scope. By delivering poor quality we

will dissatisfy our customer, so now the only remaining factor is scope. By asking the customer to prioritize all required features, we will get most of the information we need for planning [Bec01], [Bec05].

Planning is essential for all kind of software projects and should be refined, whenever facing changes in the customer's requirements or the project itself. *“Sticking to yesterday's plan despite a change in circumstances is a recipe for disaster”* [Hun09, p.43].

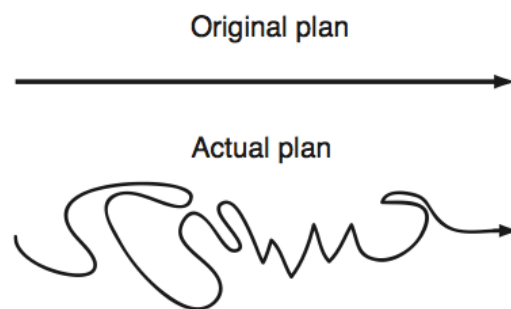


Figure 10: sticking to a plan [Ras10, p. 5]

And, *“Plans are not predictions of the future. At best, they express everything you know today about what might happen tomorrow”* [Bec05, p.92], gives a good argument, why we should plan now and maybe change the plan tomorrow, based on our experiences we had last week. A good way to get a common feeling of the project's scope and the requirements and priorities of our customers is to have a “planning-game” meeting.

In the “planning game”, a common understanding of all stories, their effort and their priority should become evident to all project members, developers, customers and project-managers. Stories are written on index cards and put on a table or a wall. These stories will be discussed, estimated and prioritized.

The planning game consists of the following four steps:

1. Create a new story or select an unplanned story – the story should contain a short description and the creator of the story, to answer questions during implementation
2. Estimate the story – this can be done either by the programmers, who will later go to implement the story, or by the team. Discussions on different efforts are welcome. In some variations, it is suggested to remove the highest and lowest estimation, others repeat the estimation until most of the estimations are equal. The possible values of

these estimations should be limited to some specific numbers, e.g. 1,5,10,25,50 and 100. These numbers shouldn't stand for development hours, they should reflect the complexity of the story.

3. Customers prioritize the story – this can and should later change when new stories come up. Stories depending on each other should be marked.
4. Repeat steps 1 to 3 until all stories are estimated.

Communication is essential when estimating and prioritizing stories. While customers won't understand, why a feature is so expensive in development and vice versa, programmers won't know what is really important to their customers, so this will lead to features not valuable to them. Discussion of stories should lead to a common understanding.

[Sho08], [Bec01], [Bec05], [Coh10]

In the Catrobat-Project we have used coins from a poker game for estimating the effort of each story. The only available numbers are: 1, 5, 10, 20, 50, 100 and 200. These “costs” stand only for the complexity and effort guessed, when discussing the story within the planning-game for a new release. We didn't track the stories development effort (in time), so no conclusion to the planning accuracy could be made.

3.8. Kanban

The Japanese word “kanban” can be translated to “signal card”, first used in the Toyota factory environment, to show the factory workers, that there is upcoming work to do. Unless the workers aren't shown a Kanban-card, they have to wait for the next steps of the work process [And10]. Kanban is also used in software development, to visualize current workload, backlog and things that are ready for release. *“The Kanban-System is a pull system, because new work is pulled into the system when there is capacity to handle it, rather than being pushed into the system based on demand”* [And10]. In software development, the Kanban-cards represent user-stories or simple tasks. A Kanban-Systems helps track the work in progress and gets a fast overview of what is currently being developed, which stories are waiting to be done next or are already finished. The cards used on the Kanban-board are small A5-sized sheets of paper with the following contents: priority of story, creator of the card (to be asked, when there is a questions regarding the story), the story (=task) itself and the complexity of the story as well as references of stories, depending on other story cards.



Figure 11: Kanban-story board of Catrobat's web-team

The Kanban-board is also a good place where the daily standup-meetings about the current development and upcoming features can happen. The only disadvantage of Kanban for distributed teams is, that the board has to be synchronized daily either by an electronic version of the Kanban-board (e.g. Atlassian's Jira²⁸) or a simple and shared spreadsheet, updated regularly.

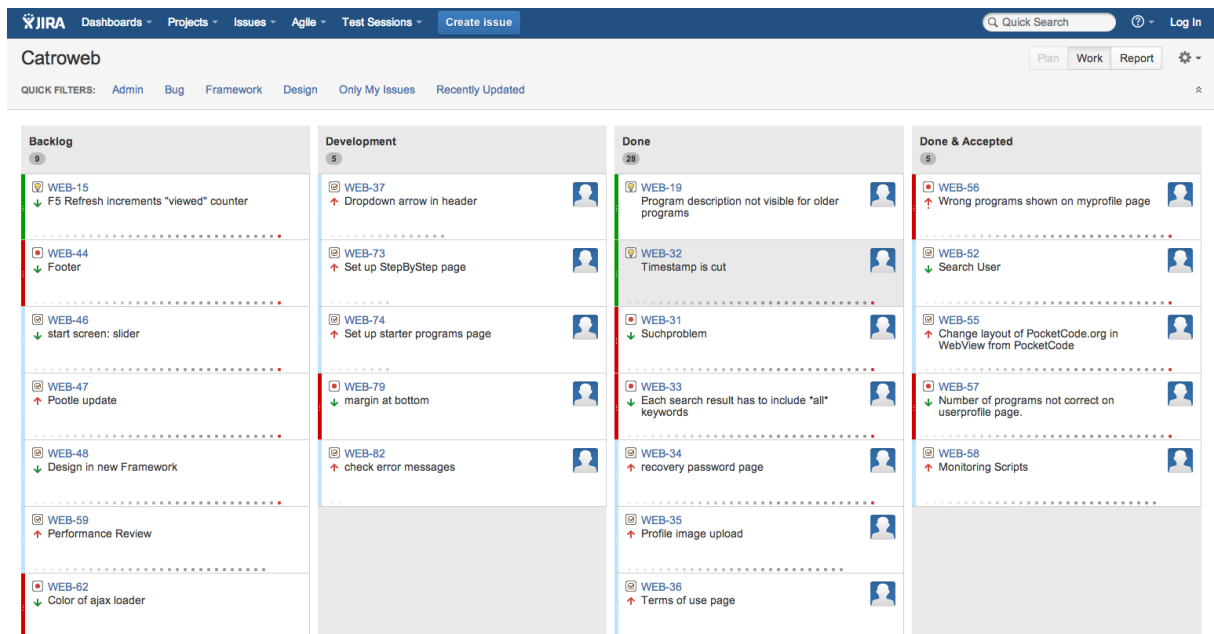


Figure 12: Virtual Kanban-story board of Catrobat's web-team using Atlassian's Jira

At the beginning of the website development for the Catrobat-project we took weekly pictures of the Kanban-board and put them on our projects' internal Wiki-website to have it available

²⁸ <https://www.atlassian.com/software/jira>

for all members of our team. The Kanban-System can help agile development teams to keep track of all important features that need to be implemented and gives a quick overview of a team's workload and development velocity. [And10]

3.9. TDD tools

There are a number of frameworks that support test-driven development, writing of tests and running them within integrated test-suites or built-in directly to the programming IDE, e.g. the JUnit plugin for Eclipse or the Selenium-IDE as a simple to use add-on for the Firefox web browser.

Important features of these TDD-tools are some test-setup and test-teardown functions, making it easy to run all tests with the same pre-conditions and do a full cleanup, after the tests are finished. These Frameworks also provide some kind of mocking tools, to keep the tests running isolated and fast. This is of great importance when running tests depending on databases and their related data or using data over network connections, that will not be fast enough to run all tests within a short time [Sho08] [Sel13].

In the next two subsections I will provide a short overview of PHPUnit and Selenium, relevant for test-driven website development in the Catrobat-Project as well as some limitations of these frameworks.

3.9.1. PHPUnit

PHPUnit is a framework to support writing and execution of unit tests for the PHP-programming language based on the original JUnit idea. The framework provides most of the needed assertions like *assertEquals()*, *assertNotEquals()*, *assertTrue()*, *assertFalse()*, *assertNull()* and so on. All assertions should be used "as they are" and should not be inverted for a better readability and understanding. For this reason one should prefer to use *assertNotEquals()* instead of *not assertEquals()*.

PHPUnit will only report failing tests. Running a successful test and feedback on tests still running is visualized by a dot on the console, while an error will produce an "E" output on the console. The execution of a running test-block will stop immediately after a single test fails. Some details of the failing test will be printed out on the console (see figure 14).

```
catroweb@webbox:~/Workspace/Catroweb$ make run-phpunit-catroid-tests
```

```

Run PHPUnit Catroid Tests...
PHPUnit 3.6.12 by Sebastian Bergmann.
.....E.....
Time: 1 second, Memory: 9.25Mb

There was 1 error:
detailsTest::testGetTags Missing argument 1 for detailsTest::testGetTags()

/home/catroweb/Workspace/Catroweb/tests/phpunit/catroid/detailsTest.php:139
/usr/bin/phpunit:46
FAILURES!
Tests: 31, Assertions: 47, Errors: 1.
Generating code coverage report in HTML format ... done
make: *** [run-phpunit-catroid-tests] Error 255
catroweb@webbox:~/Workspace/Catroweb$

```

Figure 13: PHPUnit test-results

Automation of PHPUnit tests can easily be done by using Ant- or Python-scripts e.g. Using the setup- and teardown-functions after each test-block, the test-framework provides an equal environment (e.g. individual test-data records) for running all the tests with the same user defined pre-conditions. For each failing test, a detailed test-report will be generated by the test-framework, listing the test-class, the line-number where an assertion failed, as well as the expected and actual values of the assertion. Christian Johansen [Joh11, p.36] describes it as following:

“It’s a lot easier to spot what a test is targeting if it compares two values with `assertEqual(expected, actual)` rather than with `assert(expected == actual)`. Although `assert` is all we really need to get the job done, more specific assertions make test code easier to read, easier to maintain and easier to debug.”

PHPUnit has also some code coverage features to calculate the actual test-code coverage while recording which functions where called during the test-runs. It can explicitly list the missing functions to achieve higher or complete code coverage. PHPUnit has built-in Selenium 2.x server support, and provides tools for test-doubles like mock-ups and stubs. Moreover, the framework can generate some automated documentation from a tests’ class name like *“testBalwanceIsInitiallyZero()”* and will be listed in the report as *“Balance is initially zero”* [Php13].

3.9.2. Selenium

Selenium is an open-source web browser test and automation framework. There are currently four active Selenium projects. I will give a short overview of the basic principles of each of these versions: Selenium IDE, Selenium Remote Control (RC, 1.x), Selenium WebDriver (2.0) and Selenium Grid.

First, Selenium IDE is an easy to use Firefox add-on that can record user actions and playback tests in a web browser session. It can also be used for generating code to run tests with Selenium Remote Control. Selenium Remote Control can be used to control web browsers on the local computer as well as on other remote computers. Tests can be written in any programming language to control the Selenium Remote Control Server. With Selenium WebDriver, some limitations of Selenium Remote Control could be solved. It provides a compact, object-oriented API written in Java and can handle file up- and download, pop-ups and modal webpage dialogs. Selenium Grid distributes running tests on many servers at the same time, so it can be used to test multiple operating systems or browsers and reduces execution time by parallel test execution. For the use of each of the versions, the web browser needs to have JavaScript enabled. The framework's test-suite can process web pages based on their document-object-model (DOM). Assertions like if an element with a given ID exists or if a button with a specific name is visible and clickable on the website can be defined for example [Sel13].

3.9.3. Watir

Like Selenium, Watir is an open-source test and automation framework that has a large library of Ruby scripts to automate and test web browsers. The Watir-WebDriver supports all current browsers, like Chrome, Firefox, Safari, Internet Explorer and Opera. It also has built-in support for HTMLUnit [Wat13] and runs on Windows, Mac OS X and Linux. *“Watir is pronounced water, and it stands for Web Application Testing in Ruby.”* [Zel12, p.1]

As Watir is written in Ruby, all of the Ruby features can be used for browser test-automation, like database access, file-system operations, XML processing and many more. Nearly all user interactions within a browser can be controlled using Watir: clicking a link, checking if some link with given ID is available or visible, and if some element on the webpage exists. But browser plugins like Adobe Flash, Java applets or Microsoft Silverlight cannot be controlled using Watir. [Zel12]

3.10. Source code management

Source code management (SCM), version control systems (VCS), source control or revision control systems (RCS) are all similar mechanisms to keep track of all files in a software development project. These files can be program-code, images, configuration-files, libraries, style-sheets and any other type used in the project. With SCM every version of each file can be stored and retrieved. Tags help development teams to logical mark special versions of their files, and branches can be used to separate development of features, that can later be merged back into the master (sometimes also called master branch, main-branch or mainline). It also helps to track development speed, the number of commits, the number of resolved issues and newly added features, as well as the programmer who is responsible for the changes committed. Changes should be regularly checked in to be available immediately for everyone in the team. SCM-systems also provide an automatically created overview of the latest commits with its changes. Each commit should have a short and characteristic commit-message that makes it easy for other team-members to classify them, e.g. saying that a new feature was added [Hum11] [Bur12] [Cha09].

Version control in the beginning was mostly done by RCS, CVS or Subversion. In today's software development departments, most of the teams use Git (e.g. on github.com) or Mercurial (e.g. code.google.com) for source code management.

In the Catrobat-Project we used Mercurial²⁹ as our first source code management. Starting out, we hosted the version control system on our own servers. Later in 2011 we moved our project – this was also our first year of participating in Google's Summer of Code – to code.google.com³⁰, and switched to GitHub³¹ in late 2012 because GitHub provides powerful integration features that work well together with Atlassians's Jira³², which we started to use for our project- and issue-tracking.

3.11. Continuous Integration

In traditional software development the integration part of the development process started when development was finished. Continuous Integration (CI) is practiced from the beginning

²⁹ <http://mercurial.selenic.com/>

³⁰ <https://code.google.com/p/catroid/>

³¹ <https://github.com/>

³² <https://www.atlassian.com/software/jira>

of a software development project, running e.g. nightly builds for the whole system developed so far and creating a build from the current state and executing all tests after a commit to the projects' mainline. Commits should be done more often and the whole CI-system should run automatically. Very important for a successful CI is the commitment of the development teams. Builds should be finished within a short time and all tests should run through fast. Without these requirements, programmers will not be able to get their feedback in time. When a build fails, programmers should take responsibility for fixing it, even if the error occurs in a different part of the source-code committed [Hum11].

As described in [Bur12, p.11], *“Since the traditional integration process often leads to project delays, continuous integration (CI) deals with this problem in a completely different way.”* continuous integration helps to build the whole project and run all the tests, to see if the latest changes committed to the mainline of the project still keep the build green and the program works. All this is done automatically without the need of the programmer's interaction. Within the Catrobat-Project, continuous integration was used for the core development teams of the programming IDE named Catroid and the drawing tools for mobile devices named Paintroid. These subprojects used Jenkins³³ for test-automation (integration testing and automated builds). The Catrobat-Website project was integrated in February 2013 to run automated builds after each commit to the master-branch.

³³ <http://jenkins-ci.org/>

4. Current Research and Studies on Test-Driven Development

Many studies, reports, experiments and papers focus on test-driven development and test-first-design. Until today there are only a few papers available that address the effectiveness of test-driven development (TDD) and list some negative aspects about this concept. Most of the experiments have been held in a small-scale programming environment or in computer science classes of university lectures.

While Kollanus is questioning, whether TDD is still a promising approach in [Kol10], Kollanus is further investigating critical issues on TDD in [Kol11] and presents two experiments of understanding TDD in an academic experiment [Kol08].

Others focus on the quality of testing in TDD [Cau12], finding bugs [Wir09] and the resulting code-, software- and design-quality [Jan08] when applying TDD in commercial software development [Ren08].

TDD is also compared in professional software development [Mar07] and the handling of refactoring an existing application with lots of legacy code [Ber12]. A short comparison of how some of the currently biggest IT companies - Microsoft and Google - test their software is given [Whi12] [Pag09], and some best-practice examples from the Google Testing Blog³⁴ are presented.

Further research is focused on the effectiveness of TDD [Tur10] [Erd05] and unit test automation [Wil09]. In [Abr11], Abrantes gives an overview of common agile practices in software processes, whereas Goldman presents a new model of pair programming in [Gol10].

Some methods of testing web-based applications are presented in [Luc06], strategies for debugging in [Mur08] and test-driven development of relational databases (TDDD) in [Amb07].

At last some common mistakes in TDD practices are shown from Aniche and Gerosa in [Ani10].

³⁴ <http://googletesting.blogspot.com>

4.1. Understanding test-driven development

In [Kol08] the difficulties of students using test-driven development (TDD) and some aspects of teaching TDD have been analysed. TDD causes “*high cognitive load, even for advanced students*”, and for this, requires teaching its fundamental basics in more detail.

Kollanus let his students do the bowling game kata³⁵ example to get the basic principles of TDD by self-learning.

Bowling is quite familiar for students in the United States. In central Europe the rules of bowling are not obvious to everyone, so a different kata with better-known problems should be used, e.g. the prime-factors kata³⁶ or the string-calculation kata³⁷. Doing a programming-kata before starting your daily work will improve your skills. “Kata” is a Japanese word and means “form”. As in martial-arts, where a kata is repeated for many times, it lets you learn and remember the movements. The same can be done in software development to enhance your skills and remember the patterns used [Pro12].

Another way of training TDD in teams is a coding-dojō, where a small group of programmers come together to work on a coding challenge to improve their skills and to have fun [Cod12].

Kollanus’ experiment in [Kol08] consisted of students having to do some assignments using TDD in winter term 2006. After the assignments, students have been asked several questions on their experience using TDD, with most of them not having any experience in TDD. Students had to rate, how hard the use of TDD was for them, whether they will be going to use TDD as well after the assignments and the aspects, which they found the most difficult in the assignment.

Some didn’t know exactly what kind of tests they should write, others had problems using the given tools like JUnit. Kollanus concluded that teaching TDD is essential, as most students weren’t familiar in how to write a test, how it should look like and how to use JUnit assertions. Students also didn’t understand why trivial functions also needed to be tested. In a second experiment the results reported by students were “*better trust in the code*”, motivation and TDD was kind of a design-method. Like in the first experiments, the need for teaching TDD first in more detail and showing how to write tests is essential for the understanding of it.

³⁵ <http://www.butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata>

³⁶ <http://www.butunclebob.com/ArticleS.UncleBob.ThePrimeFactorsKata>

³⁷ <http://osherove.com/tdd-kata-1/>

The critical aspects and viewpoints of TDD have been researched by Kollanus in [Kol11]. He focused on quality factors like internal code quality, external quality and productivity. Kollanus found different results when comparing controlled experiments with case studies and questioned the quality improvements. As quality gets higher with unit-testing, it may not be obviously related to TDD, as unit-testing can be done anytime while programming. One of the main difficulties in measuring and comparing results of studies and experiments is the use of different metrics (e.g. number of tests, number of lines in tests). Some results showed as well that code created using TDD-methods seemed harder to maintain. Productivity measurement factors reported are: *“development effort, lines of code per hour or number of implemented user stories”*, and as a result, their effort on problem solving did increase when using TDD.

Other critical aspects of TDD are a lack of design in the beginning. With minimal or no design done at the start, a higher effort is needed, when developing test-driven with focus on databases e.g. large code size of tests make it impossible to always run all of the tests after the implementation of a small change. Another aspect is maintenance of tests, because tests need refactoring as well when requirements are changing. One of the biggest problems reported was the difficulty of writing good and meaningful test-cases [Kol11].

TDD has been used as plain development method, part of the agile methods, and not a testing method. Kollanus [Kol10] asked the research question *“Is there empirical evidence on the suggested benefits of TDD?”* - After performing literature research on the keywords “TDD” and “test-driven development”, he found the following three main aspects on effectiveness of TDD: external quality, internal code quality and productivity. The results of literature research were quite diverging. Some found that TDD lead to an improvement of external quality and productivity, others concluded more quality but also an increasing development time, and in some papers even no changes were reported. Limited by the size of the assignment in an experiment it seems impossible to be compared to real world implementations.

Kollanus [Kol10] concludes, *“TDD may improve external quality, but in the same time increases development effort.”* and *“this increased development effort is often regarded as a good investment that finally saves the total costs”*.

[Kol08], [Kol10], [Kol11]

4.2. Software- and Design-Quality in test-driven development

“Using TDD in real world re-implementing a mobile web portal for T-Mobile³⁸ with high load running 24 hours a day helped to reach the desired goal, in time” [Ren08]. The so-called Web’n’walk-3 homepage could be personalized by users using widgets, showing the latest news or personal e-mails. Rendell listed the most common mistakes when TDD was used by new programmers that were not familiar with writing tests. They made one or more of the following mistakes when using TDD:

- Spent too much time writing tests
- Refactoring was taking longer than expected, tests had also to be refactored
- Commented out test-code that didn’t work
- Stopped writing tests or reduced amount of tests

Additional functionality was more often implemented when writing infrastructure libraries, possibly needed for future features or modules. By using TDD the YAGNI-principle became more obvious to the programmers when only user goals were to be met. Tests of the system have been done manually by using a state-machine written in UML on many devices [Ren08].

Causevic, Punnekkat and Sundmark [Cau12] found no differences when comparing traditional test-last with test-first programming. The average quality of both techniques was the same. Seven limiting factors of TDD were identified, whereas the most prominent one was the ability or inability of a programmer to write efficient and automated test cases. Using two internal quality factors, code coverage and mutation score, they found external quality depends on all source code, no matter whether developed using test-first or test-last methods.

Positive and negative test cases were measured, while positive test cases were the ones found in requirements and negative test cases were an interpretation of program execution. In their results a total amount of 65% of overall testing was done with negative test cases.

They concluded: *“Test-driven development is a development methodology and not a test design technique. That is why test cases are driving a developer towards implementing required functionality in a constructive rather than destructive way”* [Cal12].

Software must be designed for test – by using e.g. a non-deterministic hash-function for unit-tests, it will always fail. Creating these hashed values must be simulated while in a testing environment and later on changed, when used in production. Repeated behaviour is essential

³⁸ <http://www.t-mobile.com>

for running tests. Like in compilers, the code generated by them should be identical for same source-files, not just equivalent. A good balance between test and production code is essential [Wir09].

Janzen studied the effects of TDD on software quality, and discovered interesting misconceptions when asked for the benefits of TDD: TDD is interpreted as writing automated tests only by programmers, some writing all tests first instead of using the short iteration cycles as proposed rhythm of test-driven development by Kent Beck [Bec03]:

1. *Quickly add a test.*
2. *Run all tests and see the new one fail.*
3. *Make a little change.*
4. *Run all tests and see them all succeed.*
5. *Refactor to remove duplication.*

One of the most important things when using TDD is design, as TDD is also said to be “test-driven design” – improving it with every small single step. To have a test-suite that can be run instantly within a few seconds is a great thing as well, but the overall benefit of TDD is creating a solution with a good design that just works.

In their studies, Janzen and Saiedian [Jan08] found that there was an impact on code-size as “*test-first programmers wrote smaller modules than their test-last counterparts*”. There was an impact as well on complexity, as the test-first groups wrote classes with a lower number of branches, fewer methods and as a result of this, had a reduced complexity inside these classes/modules. So TDD supports a high test-coverage, improves cohesion and lower coupling, produces smaller modules and improves design changes throughout the development process [Jan08].

4.3. Test-driven Development in professional software development

Agile development methods easily help to implement new functionality as requirements change. But what about the code and features that are no longer needed? Bergman researched this topic and called it “*the big book of dead code*” [Ber12]. By asking the questions “*What happens to things we no longer need?*” and “*Do we know dead code when we see it?*” he tried to determine why and when useless features are left in the code. In his case study, a software development team was reducing code size, made tests run faster and speed up application runtime by eliminating “*dead code*”. As the current application was buggy, slow and expensive

to support, they decided to re-write the whole application from scratch: putting only features in the backlog, that were needed by all customers, resulting in one single codebase. While development took place over a long period with late user feedback, many things had to be rewritten again and should have been thrown away, but instead, all code was kept. After a management change the teams switched to agile development – beginning to remove dead code or abandoned features, that didn't make it into production. One of the biggest problems had been multiple implementations of common objects like “*Group, GroupNew, Groups, Group2*”, with only one of them being used at the end.

Deleting code was easy for the teams, as most of the former developers were gone, so no “code-owners” had to be asked before removing or rewriting old and unused code. Talking about simple design and YAGNI, developers started to write “*good, clean code*”. And by putting all the effort of programming into the book of dead code, before deletion, this became a good visualization of the efforts put in development, even if it was removed in a later release [Ber12].

“Professional software developers should produce clean, flexible code that works and ship it on time. Many of them don't – and instead ship late, buggy, messy and bloated code.”, said Robert C. Martin, one of the “creators” of the Agile Manifesto³⁹, in [Mar07], and *“TDD is an attitude as well as a discipline”*. There is no silver bullet that can transform a programmer to be a rockstar professional software developer, but TDD plays a central role on the way to professionalism. Defining three laws for TDD to use with software development doesn't always make sense⁴⁰:

- *You may not write production code unless you've first written a failing unit test.*
- *You may not write more of a unit test than is sufficient to fail.*
- *You may not write more production code than is sufficient to make the failing unit test pass.*

Strictly following the law is not always the best solution: sometimes it takes more time than two minutes to write a test or to refactor, another time more production code has to be written at once. The goal is to alternately keep writing test- and production-code. One of the biggest benefits of TDD is that tests will cover almost all of the production code, making it easy for developers to clean up code without being afraid of breaking it, even if the code they are cleaning-up has not been written by themselves.

³⁹ <http://agilemanifesto.org>

⁴⁰ <http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd>

“Tests should be kept in one place and should be easy to run. Executing them shouldn’t take more than a few minutes.” [Mar07]. These points were incorporated when developing an application called FitNesse⁴¹. With about 45.000 lines of Java code, 20.000 of them being unit tests that will execute within 30 seconds, these benefits exceed simple software verification in many ways. Most important with test code is to keep it at the same quality level as production code since tests are the best documentation of code itself. Another benefit of TDD is the time saved when debugging. By running all tests after a change was made, a newly introduced bug will be easy to find, as it was just added with the latest change done only seconds ago [Mar07]. [Mar07], [Pag09], [Whi12], [Ber12]

4.4. Effectiveness of TDD

In [Erd05] the test-first approach of test-driven development has been investigated and resulted in a formal investigation of the strengths and weaknesses of TDD. Two groups of undergraduate students were used - one using TDD, the other traditional software development writing tests afterwards. Both groups were adding features on a one-by-one basis, preceded respectively followed by the tests. Erdogmus et. al [Erd05] mentioned, that practitioners still consider test-first as an overhead in programming and tests should only be written and executed by a separate quality-assurance team.

These ideas can also be found in bigger companies like Google or Microsoft ([Whi12], [Par09]) as mentioned in the previous section, and even in a software development project⁴² at Graz University of Technology⁴³ (where I am working since seven years), these traditional ideas of test-last and separating testing from development by a Q&A department can still be found in all development teams.

As Erdogmus et al. [Erd05] compared some previous studies, they got several results: TDD is subjectively said to be the best technique of an extreme programming course by students, but in an academic research there was no conclusion of leading to better programs nor higher quality. Only enhanced program understanding was reported. In an industrial case study there was a higher code quality (by 40-50 percent) and *“no significant impact on productivity”*. In an experiment with professional programmers the produced code had fewer defects (45%), a higher product-quality (18%) but a lower productivity (14%). They also mentioned, that giving

⁴¹ <http://www.fitnessse.org>

⁴² <http://www.campusonline.at>

⁴³ <http://www.tugraz.at>

the groups in a test-first vs. test-last experiment a very tight skeleton for their work (assignments) decreased the flexibility for problem solving and lead to nearly no differences between the test-first and the test-last groups. In their experiments they have solved this limitations by giving the groups user-stories (high-level requirements). Another experiment was mentioned, in which a group using the waterfall-development method, tests were omitted at the end of the assignment. *“Testing and programming are tightly integrated activities”*, so the best environment for TDD is an incremental development process, building a prototype, asking the customer, getting feedback and changing current or implementing new features as a follow up of the customer’s feedback [Erd05].

Conclusions on the productivity, quality and a possibly lower defect rate can be made on mid- to long-term assignments, as with short-term assignments, the overhead of writing tests first (in an isolated domain) will lead to an overhead with no benefits for having a high test-coverage. Some results Erdogmus et. al. [Erd05] concluded are that test-first development improves the understanding of code, leads to a higher level of productivity and reduces debugging and rework after completion of a task.

Another aspect on effectiveness of testing was investigated by Williams, Kudrjavets and Nagappan [Wil09], presenting an overview on test-automation of unit-tests at Microsoft. At first, tests were written at the end of functionality coding, every few days. As TDD required more development time it lead to more accurate and specific tests. As a result, it could improve code quality. Unit tests were still written at the end of programming tasks, by the same programmers. Developers at Microsoft being asked about unit-testing thought more positively about reading legacy code with the help of automated unit-tests to verify code-changes or during debugging when trying to find a bug. In their studies, the TDD teams wrote more lines of test-code compared to production-code resulting in a higher test-coverage by incremental development. Some aspects in commercial software development when moving from traditional waterfall to test-driven development found by the study at Microsoft are [Wil09]:

- Management must support and be committed to unit-testing
- Same tools for testing should be used in all teams
- Writing unit tests takes more time and must be considered in release planning
- Unit-tests should be part of production code
- The system must be designed for testability
- Test-coverage should be measured
- Anyone should be allowed to add/write tests

- Tests should be easy to execute and run fast (time to run tests at Microsoft was less than ten minutes, for about 1.000 KLOC⁴⁴)

These ten minutes is quite a long time for running unit tests that should be run after every small change is made in the code. As reported by Robert C. Martin [Mar07], his framework FitNesse⁴⁵ has about 45.000 lines of code (LOC), with about 20.000 LOC for unit-tests with more than 1.100 test cases, all running in less than 30 seconds. This was only possible with the use of mock-objects for database and network resources to save execution time.

The effectiveness of pair programming in test-driven development was investigated by Goldman [Gol10a] [Gol10b]. Pair programming relies on the driver and navigator metaphor. One person typing code - the other watching him and keeping the focus on the strategic target. Goldman takes the focus on parallelizing pair programming with one programmer writing the code while the other one is writing tests. Swapping roles is also allowed in this settlement, so the main activities become a passing of unit-tests to each other.

Merging pair-programming and side-by-side programming to a parallelized method of test-driven development showed to be effective and supporting continuous code synchronisation between developers.

Common agile practices in software processes were investigated by Abrantes and Travassos in [Abr11], like test-driven development, continuous integration, pair programming, planning game and many more. Their focus was “*what are the software practices that can be considered agile into the context of approaches to develop software?*”. Some of their results are given below (with respect to the test-driven development process).

- Refactoring as rewriting and optimizing existing code, especially the removal of code duplication
- Planning game, to always focus on the most important stories
- Small releases, to get fast feedback from customer and to create useful software
- Simple design, to reduce complexity and removal of extra – not currently needed – code
- Collective code ownership, anyone should be allowed to change anything at any time without asking a single person (code owner)

⁴⁴ kilo lines of code (1.000 KLOC = 1.000.000 lines of code)

⁴⁵ <http://www.fitnessse.org>

- On-site customer, should be available for answering arising questions and fast decision making⁴⁶
- Sustainable pace, with no need for working overtime
- Open-workspace, having developers sit together, with enough space for pair programming (e.g. size of desks, number of chairs, wireless keyboard and mouse)
- Coding standard, so everyone should be able to read everyone's code, without refactoring, renaming or reformatting
- Stand-up meetings, max. 15 minutes, used for daily task organisation and discussion of project status and eventual problems

Their conclusion [Abr11] on the performance of agile development methods is, that it is strongly depending on the environment, the team and the applied intensity.

[Erd05], [Wil09], [Abr11], [Gol10], [Tur10]

My experience in software development at Graz University of Technology results in the same factors as mentioned above by Abrantes, where the most success factors are environment and work context. With no clear working schedule for fixed meetings, programming hours and constantly being interrupted by messages from the support system, focus on programming cannot be kept.

Here is an example from the day-to-day work in software development:

All of the core development teams in our department still implement features by test-last waterfall methods. Sticking to quite long release-cycles of three months, the lack of continuous integration and an isolated Q&A department trying to test (this is done test-last) all new features by running integration and acceptance tests on two test databases lead to a development delay of about two weeks after a soft "code-freeze" by the pre-release-date. After the code-freeze, all new features will be installed on one of the two test database machines and tested by the feature description (documentation) of the programming teams. At this time, documentation normally is still missing due to bad project planning and debugging until the very last moment. By figuring out the new features, Q&A department is running tests for the next two weeks. While waiting for feedback, programmers start implementing new features and will have to move back to their old code, when a bug is found.

⁴⁶ This method in agile software development is one of the hardest to fulfill; normally there is not enough space and programmers will still get nervous when having customers around their desk day-by-day.

4.5. TDD of web-based applications and relational databases

The main differences between testing traditional and web-based applications was presented by [Luc06]. Web-applications have to be tested for “*reliability, usability, inter-operability and security*”. Tests should cover all functional as well as all non-functional requirements of the application. Strategies for non-functional testing of web-applications are listed in Table 1

Testing Activity	Description
Performance Testing	Verify system performance like response time and service-ability by simultaneous access of hundreds of users
Load Testing	Evaluation of system performance under a predefined load with lots of simultaneous users from low to high activity
Stress Testing	Evaluation of system components when used over the specified limit of resources if the system crashes or will be able to recover
Compatibility Testing	Verify system functionality with different web-browsers or operating systems
Usability Testing	Continuous testing of user interfaces and workflows to meet the target audience of the web application
Accessibility Testing	Verify system behaviour based on guidelines to be perceivable, operable, understandable and robust (e.g. text alternatives for non-text content, support use of keyboard shortcuts, readable and understandable text, maximization of compatibility) ⁴⁷
Security Testing	Verify system functionality for user access control, checking system vulnerabilities and used software components (e.g. PHP-versions ⁴⁸ , system patches and missing updates)

Table 1, [Luc06]

Web-application testing typically consists of the following three different testing processes: Unit-, integration- and system-testing. Unit-testing frameworks are available for many of the programming languages like Java, C#, PHP, Python and JavaScript just to mention some of

⁴⁷ <http://www.w3.org/WAI/WCAG20/glance/>

⁴⁸ <http://php.net>

them. Integration tests can either be automatized by GUI-testing frameworks (e.g. Selenium⁴⁹) or by the use of scripting languages like Python. At this level, some extra code and functionality in the system may be needed, e.g. to support the preparation of test-data within the testing process. Like user-registration or user interactions, based on previous entered or uploaded data. System testing as the last level of tests verifies the overall system behavior and performance. These tests can either be done using black box, white box or gray box (also known as hybrid) testing strategies, depending on the system knowledge and given requirements. Discovering failures in functionality or services of web applications is essential because nowadays, everybody wants a fast and meaningful response from a website, as instantly as clicked or activated the service on any device [Luc06].

Test-driven development techniques can also be applied to development of relational databases, called test-driven database development (TDDD) in [Amb07]. One crucial point of TDDD is the verification of the database supporting all business rules. In TDDD there are both aspects that have to be considered: verification of database functionality (views, stored procedures, saving data) as well as the data stored in the database. One of the five recurring carried out steps in classical TDD is refactoring, whereas functionality and behaviour is never removed or added. When refactoring a database (e.g. like adding or removing a column, adding new tables, relations or intersections) informational semantics and behavioural semantics shall not be removed [Amb07].

Database refactoring can also lead to massive refactoring of source-code, e.g. when a column is dropped or table structures are changed. Manipulation of data in more than one single function may come from a bad architectural design or inconsequent application of refactoring steps due to missing or incomplete tests. As a result, many “same-looking but in facets different behaving” functions will find their way in the source-code. This will lead to a big overhead in code maintenance. Encapsulation of functionality is essential when working with databases, especially in web applications, as the requirements tend to change quite frequently. [Luc06], [Amb07], [Mur08]

4.6. Debugging strategies

In [Mur08] different strategies of debugging were analysed and compared between novice (student) and professional developers. Debugging of applications is often done by tracing,

⁴⁹ <http://www.selenium.org>

commenting out some code, using print statements at different parts of the code and put inside condition branches or loops. While debugging strategies differ between students and professionals. Students tend to browse code having more problems when reading other's code and use output statements with debugging information. As professionals take advantage of debugging tools and also use output statements, but much less than students do. Good debugging strategies analysed found were: Tracing (using debug statements, print variable's values), testing, understanding of code and problem isolation. Bad strategies were: Tracing (by putting useless debug statements in the code like "hello"), not understanding the code, working around some problems and tinkering. Some teaching examples are given to help students improve debugging techniques like tracing the problem on paper when it is more complicated (e.g. more than two variables, loops, conditions), adding well placed and meaningful print statements or use some debugging tools for tracing the code [Mur08].

4.7. Common mistakes in TDD practices

Results from an online survey with developers [Ani10] showed that many times the required steps for practicing TDD are omitted when using this development-method. As the effects on software and design-quality of TDD where studied by [Cau12], [Wir09], [Jan08], [Ren08] and have been discussed in chapter 4.2, the investigation of Aniche et. al [Ani10] was on finding the most common mistakes made in TDD.

Following the five main steps of TDD is easy, but "*programmers need to be very disciplined*" on this. Omitting some of these steps might lead away from the clear goals that TDD manifests. The developers asked to participate in the online survey had the following qualifications and work-experience: *75% were experienced with TDD, 20% used TDD in academic, 50% in open source projects and about 90% in industry.*

- About 15% of programmers frequently forgot to watch the test fail. When not watching the test fail, one cannot write the – needed in TDD - small piece of code that will make the test pass. After writing the next test without seeing it fail, unwanted behaviour can become part of the code, which might lead to bugs that are later hard to trace.
- Almost 30% of programmers regularly forgot the refactoring step, mentioning that the newly written code seemed good enough for them and so it would not need to be

refactored.⁵⁰ Sometimes other parts of the code are refactored in this step (e.g. legacy code). Refactoring shall only be done when all tests pass, otherwise it will be hard to find bugs introduced within this refactoring step.

- Usage of bad test names was also kind of a problem reported to happen by about 25% of the programmers regularly. Understandable test names make the code easier to read, to understand and fulfil the demand for source code documentation.
- Not starting with the simplest test was reported by beginners to happen by about 25% of them. The more experienced with TDD the programmers are, the less they reported to not starting with the simplest test first.
- 20% of programmers regularly ran only the current failing tests instead of all the tests of the test suite. By not running all the tests after changes in the code, it will be hard to find the correct line in the code that breaks the test. If this is just done once at the end of day it may even be harder to isolate the change that makes the test fail.

Some other points of this survey are complex test scenarios. This can happen, when there is too much logic in some function or class and will lead to split up the functionality first instead of writing too complex test cases. Refactoring test code is also a must, and most important and already mentioned above, only *“the simplest thing that makes the test pass”* should be implemented to avoid *“unnecessary complex code and as a consequence, decreasing code quality”*. They suggest keeping test code clean, with easy to understand names for tests to support the documentation idea of TDD [Ani10].

When practicing TDD in the way as e.g. Kent Beck suggests, you are always one step away from a passing test, e.g. like R. Martin wrote in [Mar07]: *“... you’ll detect almost any bug that creeps into the system within minutes. You don’t need a debugger to find the bug. You don’t really need to do much debugging at all. You know where the bug is because you just added it!”*

4.8. Conclusions on current research topics

Many of the presented results in this chapter illustrated the need for a more detailed and practical way of teaching TDD in computer-science classes and more effort in training the TDD

⁵⁰ This is where TDD is often used in a wrong and sometimes misleading way when not applying the smallest possible change in the code, that will make the test pass. Adding too much logic in one step breaks the *„rhythm of TDD“* [Bec03].

programming style. Introduction of TDD is mostly done with very simple examples that are hard to use in real-world software development environments.

A good and simple way to improve the TDD programming style can be provided by a daily exercise like a small coding-kata⁵¹ or a coding-dojo⁵². These should be limited to half an hour per day to improve the programming skills when using TDD as part of a software development process. Many of the available coding-kata examples are based on typical American games like bowling or baseball, which are not all too well known by many European students in details. There is a need to find more localized examples for these kata or dojo programming exercises to effectively teach TDD and give students a better understanding of how to apply this technique fluently in ubiquitous (day-to-day) programming(-exercises).

Some results also showed that finding the right test-cases can be a big issue. Students tend to focus more on quantity than on the quality of tests. A very good and detailed overview on how to find all of the tests needed for a simple application was given by James Whittaker in [Whi12, pages 130-132] while interviewing test-engineers (TEs): “*Better is the candidate who asks clarifying questions*”. After the expected behaviour of the application is discussed, tests are much easier to write. The likelihood of misinterpreting functionality through not asking questions has been greatly reduced.

⁵¹ <http://www.butunclebob.com/ArticleS.UncleBob.ThePrimeFactorsKata>

⁵² <http://www.butunclebob.com/ArticleS.UncleBob.TheProgrammingDojo>

5. Agile Website Development

Since its conception in 2010, an extensive use of agile development methods has been the most crucial aspect in accomplishing the Catrobat project. Numerous students joined our project and contributed thousands of lines of code. Maintenance of this large code-base which contains over 25 subprojects and daily new features can be rather tasking. At the beginning, source-code revision control was done with mercurial. To achieve better integration of agile development tools, the subprojects started moving it's codebase to github.com in 2012 (e.g. Atlassian's Jira). Since adding many features at the same time in many different branches became a necessity in the development process we decided to use a modified Git branching model that will be explained in chapter 6.

Because of the Catrobat's project structure as a large open source community project, hosted at Graz University of Technology, the main focus was put on agile software development methods like pair programming, collective code ownership, test-driven development, code reviews, agile estimation and planning, user stories, daily standup-meetings, KISS (keep it small and simple), kanban, test automation, continuous integration and continuous delivery.

With a focus on test-driven-development, it could always be arranged to complete writing the necessary tests before the implementation of a new feature. Every feature had a valid test code to be validated against. Everyone could rely on a valid codebase when checking out the latest version from GitHub.

In the next sections I will provide a detailed overview of all agile methods used during development and improvement of the *Pocket Code* website.

5.1. Planning Games for Releases – agile estimation and planning

In the planning game, every member of the core team, as well as every member of any sub-team could develop their own user stories (see section 5.2) to be considered for the next release. To limit the scope of the release, we first defined some limitations for any new features. Because of the many different subprojects and teams, developing their applications in parallel, some general release planning had to happen first. After the ideas for the upcoming releases were decided upon for implementation, each team went on to their individual release planning game.

5.2. User-Stories

A user story “*is a chunk of functionality (some use the word feature)*” and shall fulfil the following criteria:

- *understandable to customer and developer*
- *testable*
- *valuable to the customer*
- *small enough to build within a few hours or a few days*

[Bec01]

User-stories come from our personas point of view. Like mentioned in [Gri11], we have defined three target users for our system: a young girl aged 10 (Silvia), a teenage boy aged 15 (Tobias) and a working mother of three kids aged 44 (Angelika). New user-stories were discussed in the planning game and were written on story cards. Those story cards could only contain one or two sentences and had to fulfil the above-mentioned criteria.

5.2.1. Story-cards

We have used white and yellow A5 sized paper sheets. White ones were used for features, yellow ones for bugs reported by users or developers. A story card contained the following information:

- User story (1 to 2 sentences)
- Name of customer (or creator) – in case if questions come up during development.

In the planning game, the story cards were presented and discussed. So every member of the team has the same understanding of the story. Then the “costs” for implementation and the priority of the story were discussed and added to the story card. After this, each story card was put on the Backlog-area of the Kanban-board, according to its priority.

5.3. Using Kanban

During the development process we found that putting some more relevant information on a story card pinned to our Kanban board was essential for the stories’ content. For user-interface issues we used an underlined red uppercase U, for design issues a blue dot was used. With this extra information, the acceptance process of user stories was changed as well. While normal stories could have been accepted by anyone in our development team, not involved in

programming that task, so called user-experience or design-stories, had to be accepted by the Usability and Design team as well.

5.4. Pair-Programming

The onsite pair-programming environment in the project room has been quite exceptional. There were four tables with one large monitor, a standard keyboard and a mouse. With two chairs in front of each table, there was enough space for two programmers to accomplish efficient pair programming. As every team used different development tools, each pair had to bring their own laptop to the project room. Only the chairs have not been very comfortable, as they were made of wood, without any possibility to adjust seat height and lean angle. Additional amenities like air-conditioning, a daily restocked refrigerator, a microwave oven, a water heater for cooking tea, a coffee machine to brew fresh cappuccinos and lots of cookies, chocolate and sweets made programming really comfortable and fun.



Figure 14: Pair programming setup in our project room

Programming pairs did not change all too often, resulting mostly in teams of students who took the same lectures during the term. In case there were questions, or a team got stuck on a problem, pairs have changed to solve the issue at hand upon accomplishing an acceptable solution, and then returned to their initial team constellations. In the beginning, we tried to pair

new developers with each other and let them develop short and simple stories of their own. Unfortunately this approach lead to poor results and our new developers soon got frustrated. So we changed the programming pairs: every new team-member had to do pair programming with one of our senior developers. This helped them to get into test-driven development much more effectively than before. Writing tests first was one of the hardest things to learn for many of us. Approaching a small new feature, which was easy to implement and could normally be accomplished within a couple of hours had to be restructured to be done in a TDD way. Writing tests first changed the way of problem solving tremendously. After each and every pair-programming session, we had a working program. Without this approach, it could have taken hours of trial-and-error programming and debugging. Sometimes problems were solved by simply explaining to others' what exactly the problem was all about.

5.4.1. Collective Code Ownership

Nobody at the Catrobat-projects owns any of the code. Every new developer is encouraged to modify existing and add new code by adding tests first. Collective code ownership helped us to keep our source-code clean and readable for everyone. Coding standards are a must and commits to our head-branch are only done after a code review. Reviews were done by a team-member who was not involved in development of the story to be accepted in the first place.

5.4.2. Code Reviews

Whenever a story is finished, the review process starts. Before running all the tests, new code has to be reviewed by the review partner. After review, all unit-tests were run on the local machine with all new features merged locally with the head-branch. Afterwards all selenium-tests had to pass as well, and finally – if needed – manual testing completed the first part of the code review. The next part is detailed code-review of all newly added or already existing features, as well as removed ones. A code will be passed as “reviewed”, when coding standards are fulfilled and software-development patterns have been used where possible.

If the implemented story does also add usability and/or interface changes, one of our usability and design-team-members had to attend the review meeting, too. This led to the following advantages and disadvantages:

Advantages:

- No graphical or interface changes were committed by the developers to the head-branch or went into production without being noticed by the design-team and the UI-team.
- No usability changes went into production without being approved by the UI-team.
- Nearly every developer within the team knew about implemented features before going into production.
- Developers had more social contact across different teams
- New developers could soon contribute to our project
- Everyone got to know all code-changes (this became ideal collective code ownership)

Disadvantages:

- Arrangement of review-meetings was quite lengthy, by finding a suitable date for all persons involved
- Commits to head branch were slowed, and so the number of releasable features decreased
- The vast amount of meetings slowed down our development speed

5.5. Testing

The effects of TDD in academic or educational tasks has been described by S. Edwards in [Edw04] as: *“TDD is attractive for use in an educational setting for many reasons. It is easier for students to understand and relate to than more traditional testing approaches. It promotes incremental development, promotes the concept of always having a “running version” of the program at hand, and promotes early detection of errors introduced by coding changes.”*

By using TDD, programs are developed in small iterations and achieving a running version of the code after every cycle. The first iteration producing only a few features. Errors are detected as soon as they make their way into the source-code during the next test-run. TDD even encourages junior developers to changing central-parts of the software or old legacy code (with full test-coverage) when necessary, without the fear of braking existing functionality. This is accomplished through *“continuous regression testing”*.

Testing is also a quite social activity. No developer is willing to write many pages of documentation, if all things need to be known are already written down as source-code and corresponding tests. Programmers would rather spend their time writing tests for new features

in pairs, instead of documenting functionality. So testing is a more effective way of source-code documentation than any other form of document can provide. A test can be called “ideal” if any programmer can build the required program-under-test with only the knowledge of the test and its use-cases.

Some very important aspects of writing tests presented on the Google testing blog⁵³ are:

- Test behaviour, not implementation⁵⁴
- Test behaviours, not methods⁵⁵
- What makes a good test?⁵⁶

When testing behaviours instead of methods, a test will fail if new behaviour is added to the code, but won't fail, if just the method itself is tested. Tests that are independent of implementation details are easier to maintain and easier to understand. Good tests have the following properties:

- *correct* – a test shall verify that the code is correct
- *clarity* – a test will be the only documentation of the source-code
- *complete* – a test provides all the information to understand it
- *concise* – a test shall contain no distracting information

[Gtb14].

5.5.1. Unit Tests with PHPUnit

One of the first things every new programmer in our web-development team had to do was setting up his local development environment. This was quite cumbersome in the beginning of our project since everyone had to make some tricky installation adjustments with help from senior-developers. In the beginning the installation of necessary tools took a few hours to complete, therefore we were looking for an easier way and provided an Oracle VirtualBox Ubuntu 12.04 image file, with all things setup right out of the box. Therefore developing tests first with unit tests could start right in the beginning after downloading and running the new development environment.

Writing unit-tests with PHPUnit is as easy as using any other unit-test framework. A classical unit test looks like this:

⁵³ <http://googletesting.blogspot.com>

⁵⁴ <http://googletesting.blogspot.co.at/2013/08/testing-on-toilet-test-behavior-not.html>

⁵⁵ <http://googletesting.blogspot.co.at/2014/04/testing-on-toilet-test-behaviors-not.html>

⁵⁶ <http://googletesting.blogspot.co.at/2014/03/testing-on-toilet-what-makes-good-test.html>

- a setup part:
 - loading all required libraries
 - set up objects needed for test execution
 - load test-data into the database
 - prepare database contents
- a test part
- a teardown part (optional where necessary)
 - restore original database
 - remove all created test-data
 - remove temporary objects
 - rollback transactions

```

catroid_api_loginTest.php
1  <?php
2  /* Catroid: An on-device graphical programming language for Android devices
18  */
19  require_once('testsBootstrap.php');
20  class loginTest extends PHPUnit_Framework_TestCase
21  {
22      protected $obj;
23
24      protected function setUp() {
25          require_once CORE_BASE_PATH.'modules/api/login.php';
26          $this->obj = new login();
27      }
28      /**
29       * @dataProvider validLogin
30       */
31      public function testCatroidLogin($postData) {
32          try {
33              $this->assertTrue($this->obj->doCatroidLogin($postData));
34          } catch(Exception $e) {
35              $this->fail('EXCEPTION RAISED: '.$e->getMessage());
36          }
37          $this->assertGreaterThan(0, intval($this->obj->session->userLogin_userId));
38          $this->assertEquals($postData['loginUsername'], $this->obj->session->userLogin_userNickname);
39
40          $this->assertTrue($this->obj->doCatroidLogout());
41          $this->assertEquals(0, intval($this->obj->session->userLogin_userId));
42          $this->assertEquals('', $this->obj->session->userLogin_userNickname);
43      }
44      /**
45       * @dataProvider invalidLogin
46       */
47      public function testInvalidCatroidLogin($postData) {
48          try {
49              $this->obj->doCatroidLogin($postData);
50          } catch(Exception $e) {
51              $this->assertEquals(0, intval($this->obj->session->userLogin_userId));
52              $this->assertEquals('', $this->obj->session->userLogin_userNickname);
53              return;
54          }
55          $this->fail('EXPECTED EXCEPTION NOT RAISED!');
56      }
57      /* *** DATA PROVIDERS *** */
58      public function validLogin() {
59          $dataArray = array(
60              array(array("loginUsername" => "catroweb", "loginPassword" => "cat.roid.web", "loginSubmit" => "login"))
61          );
62          return $dataArray;
63      }
64      public function invalidLogin() {
65          $dataArray = array(
66              array(array("loginUsername" => "invalidUser", "loginPassword" => "invalidPass", "loginSubmit" => "login")),
67              array(array("loginUsername" => "invalidUser", "loginPassword" => "cat.roid.web", "loginSubmit" => "login")),
68              array(array("loginUsername" => "catroweb", "loginPassword" => "invalidPass", "loginSubmit" => "login")),
69              array(array("loginUsername" => "", "loginPassword" => "", "loginSubmit" => "login"))
70          );
71          return $dataArray;
72      }
73  }
74  ?>

```

Figure 15: Catroid API LoginTest.php

The tests are run in a console and generate the following output for a successful run:

```
catroweb@webbox:~/Workspace/catroweb/tests/phpunit$ phpunit api/loginTest.php
PHPUnit 3.6.12 by Sebastian Bergmann.
.....
Time: 0 seconds, Memory: 9.25Mb
OK (5 tests, 14 assertions)
```

In case of a failing test, the following output is generated with detailed information about the error that occurred, is generated:

```
catroweb@webbox:~/Workspace/catroweb/tests/phpunit$ phpunit api/loginTest.php
PHPUnit 3.6.12 by Sebastian Bergmann.
F....
Time: 0 seconds, Memory: 9.25Mb
There was 1 failure:
1) loginTest::testCatroidLogin with data set #0 (array('catroweb',
'catroidweb', 'login'))
EXCEPTION RAISED: The password or username was incorrect.
/home/catroweb/Workspace/catroweb/tests/phpunit/api/loginTest.php:38
FAILURES!
Tests: 5, Assertions: 8, Failures: 1.
```

All PHPUnit-tests are executed during a run, even when the first test fails all other tests are still carried out. This could sometimes be a problem, when running very large tests or starting the whole PHPUnit-test suite. Within all our source-code, comments are only allowed where needed by the framework or wherever they are essential for understanding of the code or programming pattern.

5.5.2. Database-Testing

Testing databases can help finding bugs in database structures or relations and prevents the system from corrupting or deleting data when put into production. Sometimes a miss-spelled SQL-query can lead to the loss of massive amounts of data without specific tests. Database operations can be divided into two groups – selecting data and manipulating data.

- **Selecting data** can lead to errors if the expected results are part for some data manipulation. In general, bad or corrupted SELECT-statements normally will lead to no output on a website like an HTML “404 – page not found” error. In even worse cases, such a failure can show data or structural information to unauthorized users.
- **Manipulating data** can lead to errors, data-loss or data-corruption on execution of a faulty INSERT- or UPDATE-statement. This is one of the main reasons that database testing is essential for delivery of reliable applications and software.

One of the advantages of testing systems with transactional databases is their data-transaction-logic. After running a test the original data can be easily restored by running the built-in database *rollback()* function. Without the use of transactional logic, all data created and manipulated by the tests should be restored to their state before test execution or deletion from the database tables. Testing databases does also lead to programming for testability and the use of mocking and stubbing objects, depending on the code complexity and system performance. Further aspects of programming for testability will be discussed in section 5.5.3.

5.5.1. GUI-Testing with Selenium

Browser tests are a good way to reduce monkey testing in a team. Those kind of tests are easy to write: one can either use the Selenium built-in screen recorder to record actions for every particular use-case to be tested, or write own browser tests using Selenium's Java-API. As we wanted the tests to both be repeatable as much as possible and also be programmed in a test-first way, the Java-API was used.

We came across some problems while writing browser tests, when Ajax callback statements raised a timing issue and caused delays in page rendering. So, without making source-code changes, some of the tests became flaky from time to time. We tried to overcome these problems by setting up a Selenium Grid-server and no longer ran browser-tests on the developer's local machine. These changes decreased the number of flaky tests and helped our team save precious development time spent for debugging.

There will always still be the need for some manual integration testing, but the effort will be greatly reduced by the use of automated Selenium browser tests.

5.5.2. Test-Data-Generation

Controlling the data and measuring the results are only possible if we know exactly, when data is created, changed or deleted. By generating test-data-sets in our code-base, it simplified the implementation and execution of tests for every developer. Developers no longer had to recreate the test data needed for a specific test from scratch, just to verify their changes to the codebase. Test-data-sets were created from user stories and their corresponding use-cases.

5.5.3. Programming for Testability

When testing time-dependent objects or database operations, function blocks must be programmed to be testable. This means every variable of the program-under-test can be overloaded by the test-procedures. By overloading a variable like `sysdate()` (that returns current date and time), developers can change the expected behaviour of the program. Sometimes changes and dependencies can also be controlled by some configuration-management. In our experience, both control-mechanisms are required to write a good test. Various test-suites provide mock- and stub-objects helping to reduce a test's complexity and increase its performance.

5.6. Daily Standup-Meetings

As written by Johanna Rothmann in [Rot07, p.129]: *“Short iterations remove the need for weekly sit-down group serial status meetings (which you should never have anyway). With daily standup meetings, people can't hide their real status.”* – daily standup-meetings are a must for every XP team.

In the beginning, we found the daily standup meetings sometimes a little bit boring – especially, when another team, not directly connected to our work, was having one and we simply wanted to continue with our work. After a while we found that pausing and listening, or taking part at another team's standup meeting gave us a better overview of current development throughout the whole project. And this was the same experience for the other teams as well. Standup meetings helped our team to socialize even more, drink some coffee together and discuss current problems on writing code, adding functionality or writing tests.

5.7. Development-Environment

As already mentioned before in chapters 3.9 and 3.10, our development environment changed over the last two years. In the beginning, everyone had to set up his own local development tools. The set up process was even more complicated, when our Professor started to bring students to our project room for eight hours a week, during their 12 week lecture period. Space was limited and so pair programming changed from pairs of two, to up to 5 people in front of one keyboard and monitor.

Our development environment from 2010 to 2012 consisted of:

Eclipse with PHP extension

- PHPUnit
- XAMPP (Apache webserver, PHP, Perl and MySQL database)
- PostgreSQL database
- Mercurial version control
- Selenium browser automation
- Ant automation tools

The development environment could be installed on current Windows-, Linux- and Mac-operating-systems. Some of them had limitations like Windows (as there is no Unix shell available), others were quite easy to setup like Linux, and on the OS 10.x we used Oracle's VirtualBox to setup Linux (Ubuntu 10.04 and later 12.04).

Later we discovered, that the necessary setup for new developers to start programming could be accomplished much faster and far more efficiently. One of our senior developers (thanks to Christian Hofer at TU-Graz) setup a VirtualBox image with all development tools pre-installed on a Ubuntu 12.04 system, ready to download from our test-server at the institute.

5.8. Test-Automation and Deployment Process

Deploying changes of database structures and database setup for automated tests was done based on the ideas, the Scratch developers used for their website⁵⁷. Each deployment script has to check first if all changes have been applied to the database and, if no changes could be found, had do update the structures.

While automating database-deployment is essential for continuous database integration, this gives each developer the possibility to work with his own copy of the database [Amb07]. When the changes are done and accepted by another developer, the deployment scripts will be extended with the new changes in database structures. The setup-scripts are divided into two parts, one for initialisation of the database, which includes the creation of the main tables, roles, sequences and the pre-installation data, and the other one for the changes in database contents, when needed during website development and improvement. SQL-Files are numbered for an easy overview of the updates already installed.

⁵⁷ https://github.com/LLK/Scratch_1.4

We wrote numerous ant-scripts to help with the semi-automated start of test-suites on our test-servers, running selenium-tests on a selenium-grid server system and make deployment of changes easy to put into production, as well as backing-up the whole system or deploying the current development branch to our test-servers.

5.9. Continuous Integration

At the beginning of the project there was no continuous integration for website development. Every deployment on the test-servers happened manually (supported by ant-scripts) after a commit to the head branch. After deployment, developers had to connect to the webserver via SSH terminal session and start all tests. Numerous ant scripts were written to make deployment and starting all or single tests as easy and comfortable as possible. After passing all unit- and all selenium tests on our test server, the newly added feature was ready to go into production. Deployment on our production server happened after a release was finished. In case a severe bug was found and had been fixed, this bug fix was deployed immediately. We were planning to automate the build and test process by using our project's Jenkins server. As this thesis is being written, the implementation of continuous integration is still work in progress and not yet completed.

5.10. Usability Issues and Screen Design

The most often changed part of the website was the login and registration screen including the underlying process. Registration is now done automatically when uploading a PocketCode-project to the community website. Taking the user's registered Gmail-address from his Android smartphone or tablet. During the upload process only a username must be chosen, no other data input from the user is required. At the beginning of our project's community website, we had a much longer and more complicated registration process for users; later we realized, that registration should only be initiated when a user is uploading a project to our community website. So there was no more need to have a button called "register" on our website any more. Once a program is uploaded to pocketcode.org, there is currently no straightforward option for the user to remove or rename the program.

Screen design and page layout as well as page rendering have also changed many times since the launch of "Catroid" respectively after it has been renamed to "Catrobat" and "Pocket

Code”. The first website was hosted under the domain-name `catroid.org` – and had many different style sheets for mobile devices. We also had over 900 unit tests to check the right browser version. After the launch of the new website, the layout was changed to a floating grid, so no browser specific CSS style-sheets were no longer needed and the browser tests have been eliminated.

Our system’s backend website to manage newly uploaded projects, check for projects that have been reported by the community, manage registered users and block users on behalf of their IP-address or username had a very poor usability. For example: when looking for a particular project, the whole project overview page had to be rendered; there was no search function for a user or a projects’ name.

5.11. Version control

After some years of using mercurial as the version control system of the Pocket Code-Project with Google Code⁵⁸, the migration of our codebase to GitHub^{59,60} was completed by June 2012. GitHub has been chosen for several reasons, one of them was our intention to use Atlassian’s Jira⁶¹ for project and issue tracking as the Catrobat-Project was growing rapidly over the last months.

The more subprojects we started, the more problems we had when integrating the results of the subprojects (like AR-Drone and Lego Bricks). The main codebase of Catroid IDE (Pocket Code) had changed since the start of many of the subprojects and the subprojects’ teams neglected to constantly integrate the latest changes of the master-branch to their branches, the development branches diverged completely. There were some good ideas, code-snippets and algorithms available in some branches, but unfortunately they were useless at the moment because main parts of the Catroid-IDE had changed and integration became too complex. To overcome those problems we decided to use a customized Git branching model that will be presented in the next chapter.

⁵⁸ <https://code.google.com/p/catroid/>

⁵⁹ <https://github.com/>

⁶⁰ <https://github.com/organizations/Catrobat>

⁶¹ <https://www.atlassian.com/software/jira>

6. Git Branching Model

6.1. Git and GitHub

Git is a distributed version control system (DVCS): “*In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files: they fully mirror the repository*“, and following, “*every checkout is really a full backup of all the data*“ [Cac09, p.3].

Catrobat currently has 22 public repositories hosted on GitHub⁶², with most of them in an active development state. Some older development features like the control of a Quadrocopter AR-Drone from Parrot⁶³ or the Lego® Mindstorm⁶⁴ are no more in an active development state and therefore the code originally written, cannot be integrated into the newer versions of Pocket Code as many semantic changes have been made.

For future releases of our project we will introduce some rules for a Software-Development-Process that include [San13]:

- *Create a private branch off a public branch.*
- *Regularly commit your work to this private branch.*
- *Once your code is perfect, clean up its history.*
- *Merge the cleaned-up branch back into the public branch*

In our Project, the merge back into the public branch (master, releasable) can only be done via pull requests to assure a good code quality and walk through the necessary review process.

6.2. Some best practice Git branching models

Andrew Berry writes in his Blog-entry “Git Best Practices: Workflow Guidelines”

[Ber12]: “*Use small, logical commits, ... this keeps the history of code-development, ... write meaningful commit messages, ... that has the advantage of finding a culprit commit is easy with **git bisect**, ... use one commit for each bug fix. Use **git rebase** with caution but to keep history straight free of merge commits.*”

⁶² <https://github.com/catrobat>

⁶³ <http://ardrone2.parrot.com>

⁶⁴ <http://mindstorms.lego.com/en-us/default.aspx>

In “Understanding the Git Workflow” they suggest [San13]:

“Always use “-no-ff” to force a new commit, ... Revision control exists for two reasons – first to help sync changes with teammates and regularly backup your work, second, for configuration management, e.g. working on the next major version/release while applying bug fixes to the existing version in production. And configuration management can be used to figure out, when exactly something has changed.

Use Public and private branches: public include master and release branches, private is just for yourself – like scratch paper. So nobody in the team should base work on a private branch of someone else’s.

- *For short lived work: Use squash merge*
- *For larger work: use rebase to squash old commits, reorder them or split them up.”*

As suggested in [Sus08]: *“Merge early, merge often. If the work on your branch takes long enough, you might want to perform a merge several times. As other people push to origin/master, you can pull those changes and integrate them into your branch incrementally, rather than waiting until the end when you're done. “*

And in [Dym12]: *“Features should be as atomic as possible and small. Keep integrating – features should be integrated into an integration branch almost with every commit. This will give you immediate feedback. A feature should pass QA only if it has been integrated with all the other features that are completed. Integrate often from feature to integration branch (`git rerere` can be used for support) – and taking features out is more powerful than putting them in – so make a build and omit problem features.”*

When the use of shared libraries can cause problems with features, e.g. when they depend on different versions, a configuration management can help deploy the correct version of the used libraries and to keep track of the toggles needed for development and the build of different versions.

Another simple way of using Git is described in [Hen09]:

For feature development:

- Pull to update local master
- Check out a feature branch
- Do your work, commit often

- Rebase frequently to incorporate upstream changes
- Interactive rebase (squash) your commits
- Merge the changes with master
- Push changes to upstream

Run all the tests after a pull, merge or rebase, and before you commit

Rebase against the upstream frequently to prevent your branch from diverging significantly.

“Do the same with bug fixes: squash all commits in the bugfix branch together in one single commit and use a commit message like “BUGFIX:” [Hen09]

Regarding the style of commit messages the imperative [Pop08] should be used: *“Fix bug”* and not *“Fixed bug”* or *“Fixes bug.”*

6.3. A successful Git branching model

Based on [Dri10] there are two main branches in the central repository: “master” and “develop” (see Figure 16). The main branch always holds a production-ready state. The “develop” branch, which can also be seen as the “integration branch”, where nightly-builds come from, holds the current development state. Completed features are always merged back into master, and are by definition a new production release. For parallel development “feature branches” are used. These branches only exist as long as the feature is in development and are then merged back into “develop” branch. Feature branches exist only in developers’ repositories. When using pull-requests or collaborating with other developers, remote repositories can also be used to distribute community changes. In the need of a quick bug fix, these branches always branch off from “master” and must merge back into “master” and (!) “develop”.

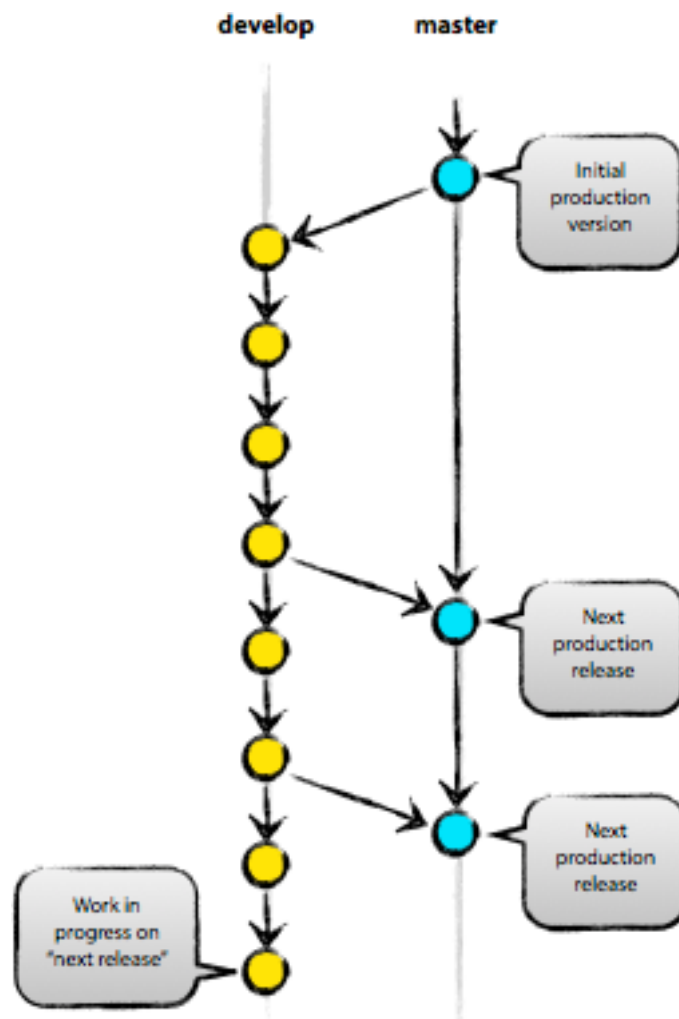


Figure 16: Git branches “develop” and “master” [Pe12]

The complete branching model is shown in Figure 17.

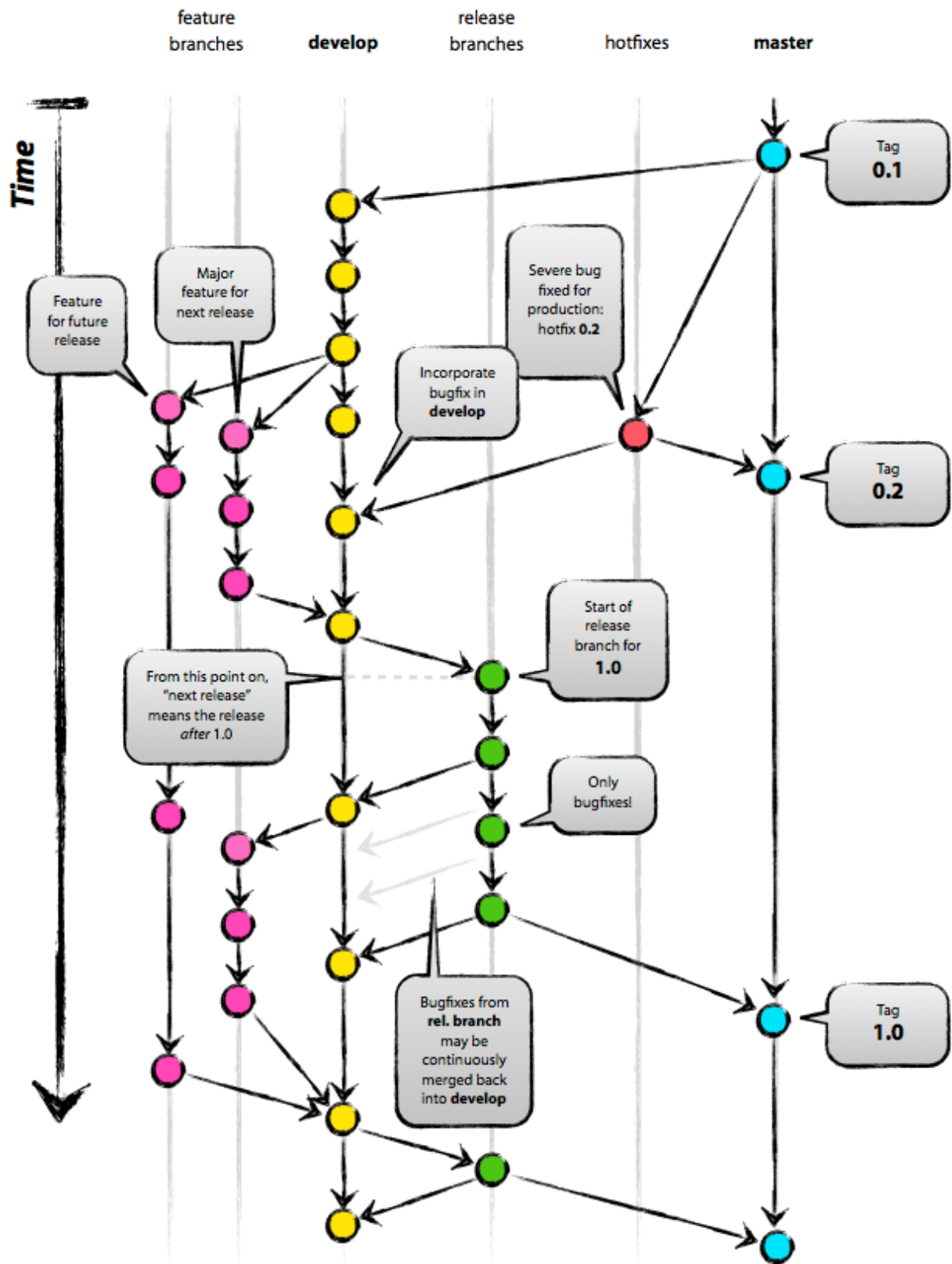


Figure 17: A successful Git branching model [Pe12]

6.4. Another Git branching model

Since the “Successful Git branching model” described in chapter 6.3 has some limitations, Aurélien Pelletier did define another Git branching model [Dri10], based on [Pe12] but with a different way a new feature branch is started.

With this model you can easily pick and choose what to merge. *“Because a feature branch is started from development, it is bound by its parents commits to other features not yet in production”* [Pe12]. Figure 18 shows the workflow to resolve these issues.

In contrast to the previous model [Pe12], there are three main branches in use which all have an infinite lifetime:

- Master (as in Git flow, master always reflects a production-ready state)
- Staging (holds features ready for release)
- Develop (for continuous integration)

*“Develop is there for continuous integration, this is where we constantly merge all the changes to detect bugs and conflicts as soon as possible. The source code in the develop branch never reach a stable point where it is ready to be released. Instead only some feature branches reach a stable point. Those stable feature branches are merge into the staging branch. Since feature branches were created from master and not from develop we can pick individually which one will be merged to staging. In fact this is the main point of this workflow: **We can easily choose which features will go into production next.**”* [Pe12]

For production release staging is merged into master.

As all the development is done in feature branches they can be merged into:

- *master, for a quick bug fix (of released versions)*
- *staging, for normal bug fix*
- *develop, constantly for continuous integration*

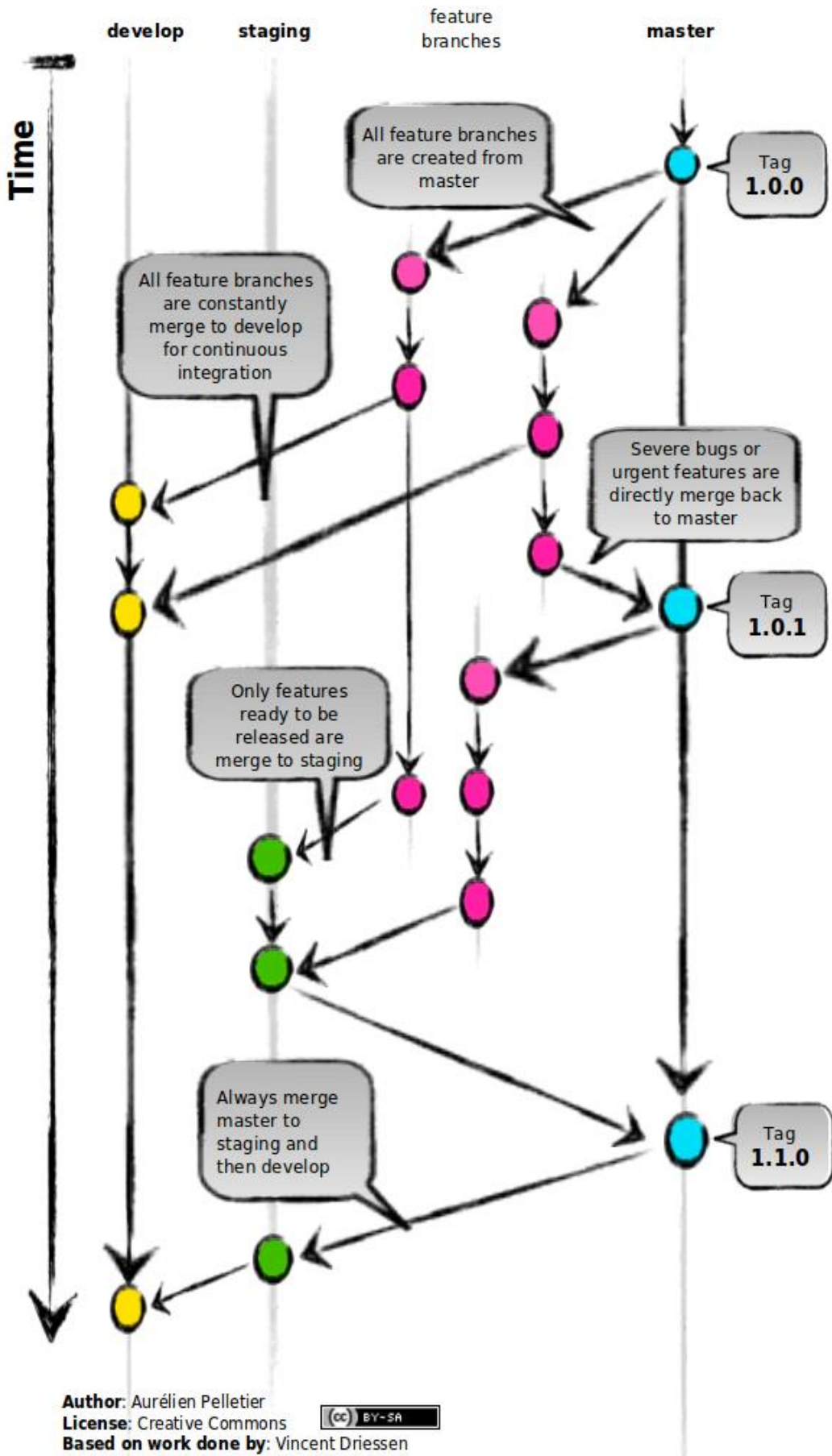


Figure 18: Another Git branching model [Dri10]

6.5. A new Git branching model

Another good branching model that can work within the Catrobat-Project is shown in [Hof11]. At first they had a straightforward branching consisting only of the following three stages:

Master ← Beta ← Develop

Recent features were done in “develop” branch, which was merged to beta once in a week. After shipping to the tester the features were merged into master. “So far so good, but this system left us with one problem: if we’re, say, 5 weeks after a new release, chances are the “develop” branch (and “beta”, as well) has a hodgepodge of new stuff. Bug fixes. New features. Partially done new features. Maybe even features we don’t want to talk about until the next major release.” [Hof11].

They changed the branching model to a little more complex one as shown in Figure 19.

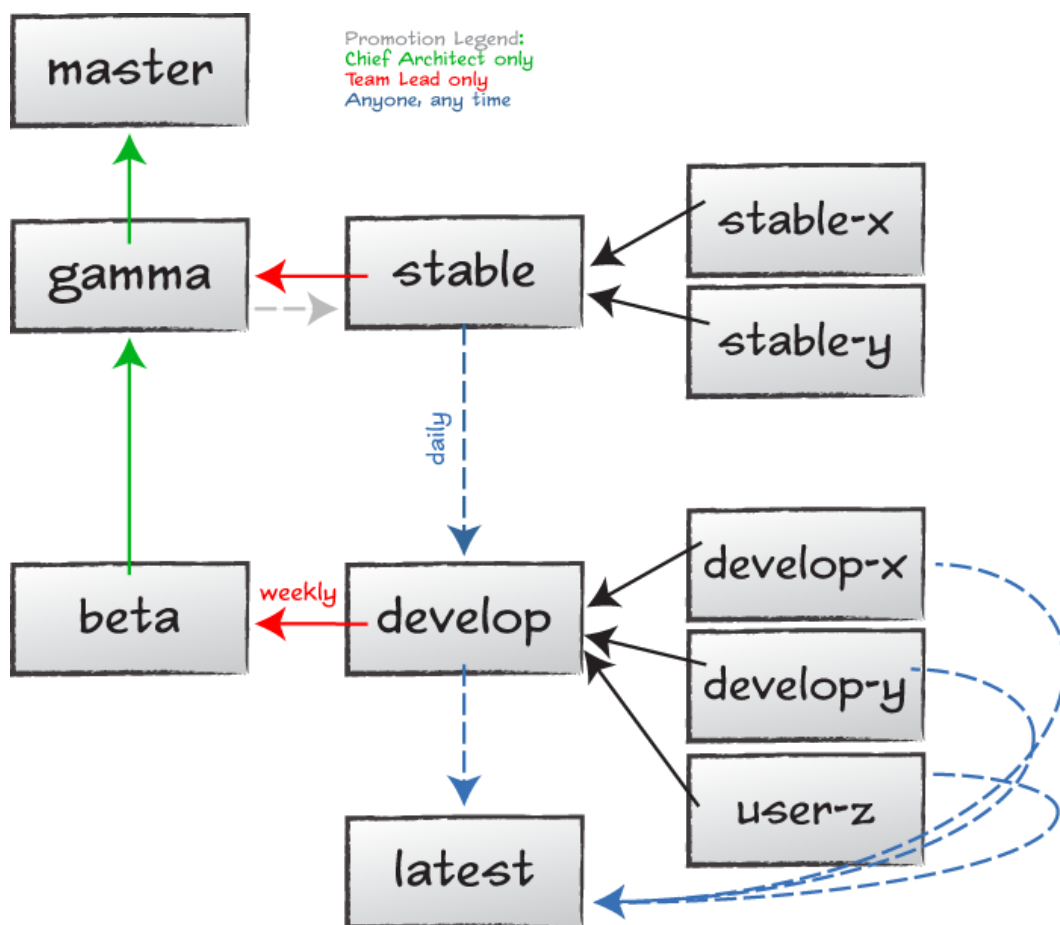


Figure 19: RemObjects Blogs, [Hof11]

The stable branches are used for changes that can ship at any time, including critical fixes or high priority bugs. All major development is done in “develop” branch and its sub-branches. Stable has to be merged into development frequently. One of the main differences to other branching models is, that here only a team-leader is allowed to merge from develop to beta and respectively from stable to gamma. Merging from beta to gamma and gamma to master may only be done by the chief-architect.

For a better understanding of the purpose of each branch and easier use in the Catrobat Project, it is suggested to rename the branches from:

- Latest → ContinuousIntegration
- Develop-x → Feature-X
- Develop-y → Feature-Y
- Beta → Integration
- Stable → Releasable (with Tags X.Y)
- Gamma → MarketRelease (e.g. the current available Version in the Google play-Store)

6.6. Results and Conclusion

As the Catrobat Project is a large FOSS project with many subprojects and a lot of features of which some may never go into production there is a need *”to easily choose which features will go into production next”*, [Dri10].

So the “new Git branching model” presented in chapter 6.3 [Pel12] will not suffice for our needs in the project. The best branching model from the current state of development and release planning is a combination of *“A new Git branching model”* [Dri10] and *“Our New Git Branching Model”* [Hof11]. The decisions, which features will be in the next release should only be done by team- or project-leaders as proposed in [Hof11].

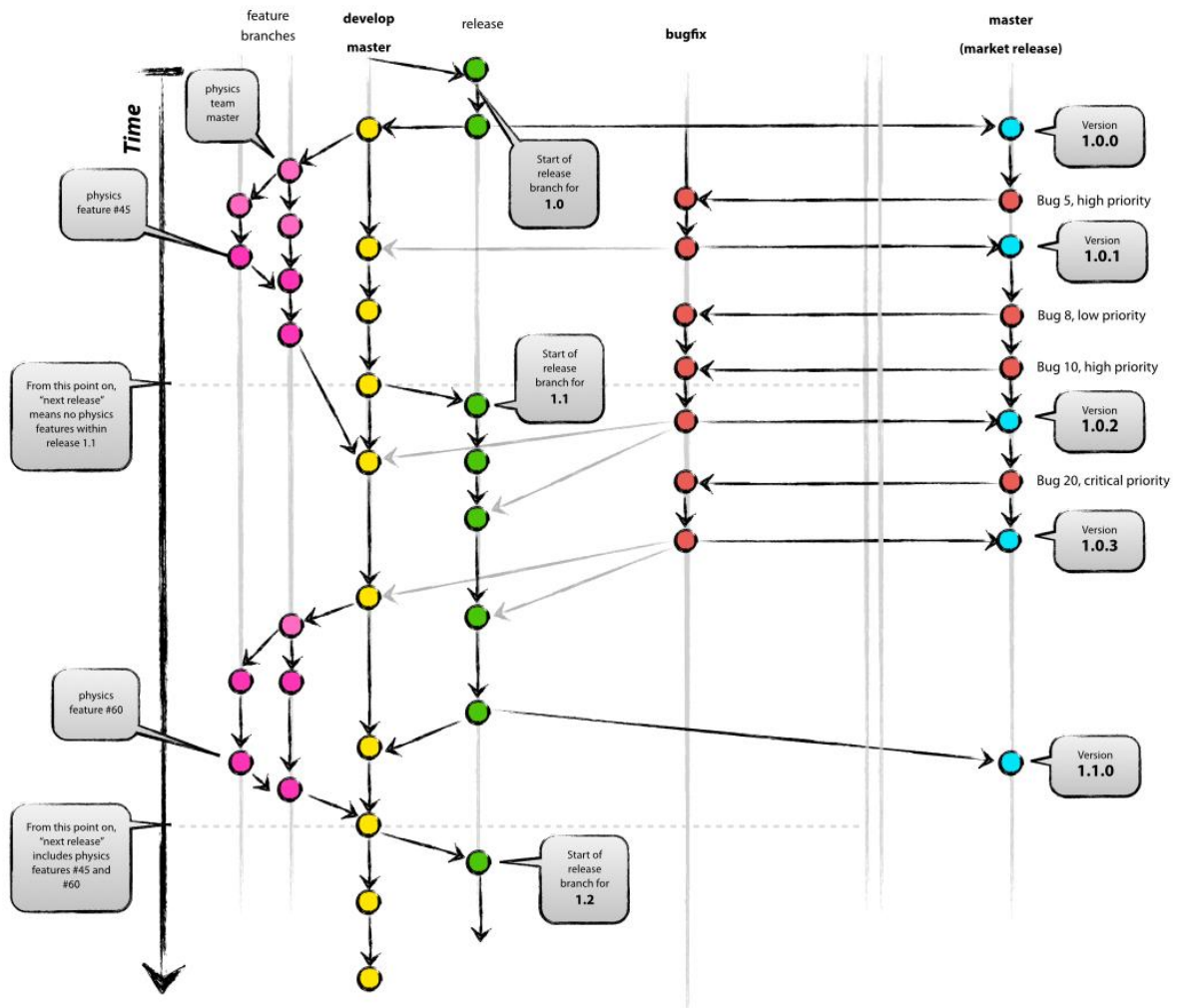


Figure 20: New Catrobat Git branching model

Our new branching model is based on two master branches:

There is a development master branch, available for all users and contributors to our project, as well as a second parallel master branch, available for only a small number of project owners. Everyone can checkout source-code from any of these two branches, as we only limit commits to *master (market release)* to a few developers.

6.6.1. Feature development

Features are only being developed in feature branches from *development master*. Each team creates its own team’s master development branch where again, all features are branched from the team’s master branch. During release planning meetings, all desired features for the next release are chosen and merged into the development release branch. After integration testing is complete and all features are finished, a release version will be created in *master (market*

release) branch. Development can be continued without interference of the current release branch.

This separation of feature development and feature release branches helps us keep features from going into production too soon. Sometimes even some features won't make it into production at all, or only by a far to come release version in the future. This decoupling helps us to keep our release branch up to date with bug-fixes without creating too much overhead by fixing them in all of our feature development branches.

6.6.2. Bug fixes in release branch

Bugs from current releases can be fixed immediately within the *bugfix* branch. This branch's sole purpose is to support fixing those bugs that made it into production. Depending on the bug's priority, fixes will be done immediately and merged back into production. Minor priority bugs will be fixed either by the next release version or together with a higher priority bug that was reported later in time. Every time the fixed bugs are merged back into production (market release branch). Depending on the timeline (see figure 20) they will also have to be merged back to our feature development and release branch as well.

This new modified Git branching model based on the three existing models presented by V, Driessen [Dri10], A. Pelletier [Pe12] and M. Hoffman [Hof11] can help us control all features that will not make it into production in a current release. As there are currently about 25 subprojects with more or less active development teams, this is essential for our project. Like before, some feature development took too long to be finished before a given release date, and without separation of development and release branch, these new and unfinished features were hard to remove from the code to be released.

7. Website Improvements and newly added Features

One of the main differences between the real agile way in theory and the execution in our project was the absence of a real customer with real user stories. In our planning games we decided by ourselves, what stories had to be done, made an estimation of effort together as a group and did our own prioritization in the team. Later in the development cycles we started to have a usability team helping us out with the definition of user stories – but, the lack of the missing “real” customer has always been quite a limitation of practising “real” extreme programming.

Working software could have been delivered almost any day, if we had really shipped it. As this thesis is being written, we are finally planning to have our first Google Play release by the end of 2013. We envisioned a 1.0 release version of our software already more than two years ago, when I first joined the Catrobat-Project in October 2010, thinking of shipping by the end of June 2011.

Requirements were changing rapidly, the team started to grow in a weekly rhythm, evolving at some time to over 30 subprojects with coordinators, weekly meetings, biweekly coordination meetings and so on. This overhead created by meetings and the necessary coordination for such a large group of involved individuals slowed down the teams’ development speed.

Participating for the first time in the Google Summer of Code (GSoC)⁶⁵ in 2011 we changed our release-planning and were thinking of a final release date by the end of 2011, for a successful integration of all newly implemented features during our first year with GSoC. But as it got even worse with frequently changing teams as well as struggling to get new developers adding value to our project, as we spent most of our time by introducing our way of coding to our new teammates and giving them a hand with installing all the required tools for development (this was sometimes really hard, because some of our students hadn’t been able to set up a running webserver, database and PHP development environment on their own). We started to have tutorial lessons in smaller groups, did pair-programming with new students in our team and taught them the way TDD works within our project.

⁶⁵ <http://www.google-melange.com/>

In the website-development team where I have been participating for the last two years, we were glad to have a virtual machine with a kind of automatic-installer and automatic-update scripts, to always have the right version of the code, the tests and the software needed available, no matter what kind of hardware and operating systems our students used. The previous development environment could only be run without problems on Ubuntu Linux and Windows XP or Windows 7. As many of our students started to use Apple Notebooks the idea of a VirtualBox OS image that could be used by everyone was born. With this easy to use Virtual-Box image people could get started programming within a few minutes.

The second required step for introducing them to the way we liked to test and program, we introduced a so called “buddy-system”. Every new member was introduced by a senior member of our team into the development process. For the first few weeks of participation, doing pair-programming sessions together were done. Without this kind of “care taking”, many new students got frustrated by our agile principles at the beginning.

The following features of the Catrobat-website have been implemented:

7.1. Uploading and browsing

Project upload can be done right within the Pocket Code-App. A Pocket Code project consists of a program file (XML), images, sounds and a screenshot. All data of the project is compressed before upload and put in a single zip-file (with .catrobat ending, so the file can be associated to Pocket Code app after download). After the project upload was finished, all project data is extracted by the webserver and stored in the file-system with public access to it. Additional data is extracted from the xml-file as well, for example:

- Catrobat project language version
- Remixing information (original author, original project)
- Project title and description
- Thumbnail/screenshot

This data is used for project visualization and is also stored in the database.

Tests for the project upload checked for the following information, provided by our generated testdata-sets and verified the returning status-codes and messages for the Pocket Code-app:

- Correct extraction of program-version from xml-file
- Correct extraction of thumbnails from project screenshot

- Correct extraction of all image- and sound-files
- Insert or database updates

Errors that have been found by these tests were:

- No creation of project directory because of insufficient rights
- Bad or corrupt image data within the project
- Bad or corrupt URL within the Pocket Code-app for project upload

Errors not covered by our tests were timeout-issues when uploading very large project files.

7.1.1. Automatic registration process on project upload

The first version of registration was quite complicated when being used on a smartphone, because many fields had to be filled out. We did also ask for far too many personal details, keeping all functionality close to the ones at the Scratch-website.

The next versions of registration were much simpler to use. There was no need for registration until uploading a newly created or remixed project to our community website. A user has to just choose his desired nickname and a password. The nicknames will we checked for bad-words before finishing the registration process. Currently these blacklists support only German and English and will not work with other languages (see future work).

All other relevant information like e-mail address and country were automatically taken from the smartphone/tablet. If a user starts using Pocket Code on another device (with the same e-mail address), he can immediately start uploading projects by just entering his username (nickname) and password. In previous versions the user login did happen automatically without the need of a password. Later we discovered this had some drawbacks in practice, as our target group (kids aged 10 to 14) usually share their smartphones/tablets and also let their friends create some programs with Pocket Code. Without the need of a password, any user could upload projects to the account connected with the smartphone or tablet. The consequences of uploading a project without the need for entering a password may be embarrassing for the owner of the device depending on the specific content of the upload. There is also a known limitation on user projects: once uploaded to the website, there is no button to remove the project by the user itself. A possible workaround is to “report the project as inappropriate”.

7.1.2. Automatic bad-words filter check on project upload

After uploading a project, a bad-words filter check is run on title and description entered by the user. There is no immediate feedback to the user, that his project contains bad-words. We did implement two different strategies:

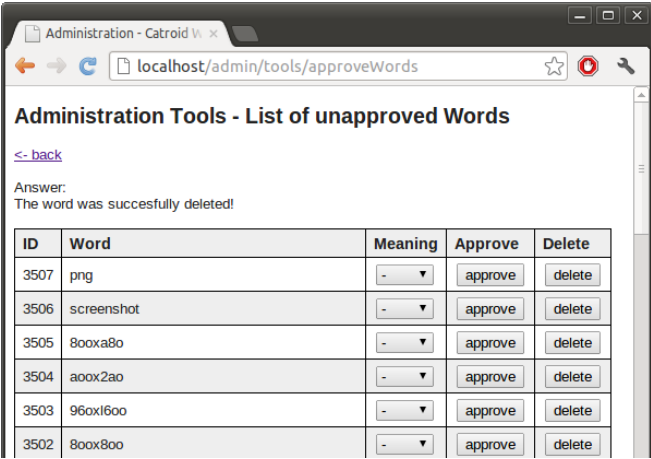
7.1.2.1. Black-word lists

Initially a table containing bad words was setup for every supported language first. This strategy was not very effective. When checking for bad words in a German project, there can also be lots of bad English words – and these wouldn't have been found by our checks. So we decided to have one single table of bad-words to check against. By adding more languages (e.g. Malayan) to the Pocket Code-project and website, the bad-words-filters needed to be maintained by the community.

This bad-words filter is still in use when checking for a bad username in registration process and serves as a first of two stage checks in uploading of projects.

7.1.2.2. White-words list

The white-words list contains all words from title, description and nicknames ever uploaded to our website. Words were added to the list - if they didn't already exist - by the bad-words-checker and neither marked as “good” nor “bad” in the beginning.



The screenshot shows a web browser window titled "Administration - Catroid" with the URL "localhost/admin/tools/approveWords". The page content includes a header "Administration Tools - List of unapproved Words", a back link "<- back", and a message "Answer: The word was succesfully deleted!". Below this is a table with columns for ID, Word, Meaning, Approve, and Delete. The table lists several unapproved words with their IDs and corresponding actions.

ID	Word	Meaning	Approve	Delete
3507	png	-	approve	delete
3506	screenshot	-	approve	delete
3505	800xa80	-	approve	delete
3504	a00x2ao	-	approve	delete
3503	960x1600	-	approve	delete
3502	800x800	-	approve	delete

Figure 21: Catrobat Administration tools (catrobat.org) - list of unapproved words

By default, all projects are visible after upload. When there is at least one unapproved word found in the title or description, the project will not be shown on our website and an automated e-mail is sent to Pocket Code-admins. Approving bad words – or even finding them – is quite cumbersome and will need attention with future releases, especially when the number of users

is growing. There are some other relevant features like blocking users on an IP-address or username basis to prevent uploading bad or humiliating projects to the website or posting comments to the discussion board. Handling these issues will be discussed in a later section of this chapter.

7.1.3. Lost username or password

A lost username or password can be easily recovered by entering one's e-mail address. If the e-mail address entered was registered on Pocket Code website, the user will get an e-mail with his username and a one-time token to set a new password. Updating password information was quite complicated in the beginning, because we also did have to synchronize passwords with the help-system (MediaWiki) and our discussion-board (PHPboard). Later in the project we removed the MediaWiki and the PHPboard from our scope using google-groups instead. This solution was not ideal, but we are currently thinking of a better integration of a discussion-board like Scratch did (see chapter 9 – future work).

Testing the lost password methods was done in two steps:

- An e-mail message was created by our mail-API and sent to a predefined Gmail address. After sending the mail message and some delay necessary to have it delivered, we did check the e-mail account for the new message our test just sent. This test ensured and verified that our mail-API was capable of sending mails.
- Our program generated all information on how to reset a password. The underlying test then checked the token stored in the database associated to the user and extracted the token from the e-mail message that will be sent. This test helped us verify that the sent token will be stored with the users' context in the database.

We did also add a Selenium browser test for password recovery, running through all steps necessary to reset the password (with a predefined token), so we did not have to implement checking e-mails with Selenium, which would have been quite difficult.

7.1.4. Editing a Project's Title and Description

When uploading a project, the user can provide a title and description. This information can be simply updated by uploading the project once again, with changes made to its description; changing the title results in another project created on the website. Currently it is not possible to change any of this information at the website, neither as authenticated user or anonymous guest.

Testing was done by verifying the information stored in database, compared to post-values provided by the upload-API. Errors that have been found by these tests were:

- a too long title-string
- bad mark-up or bad content in description

7.1.5. Browsing projects on website by category

On our first versions of the website, browsing projects was only possible by newest ones first. After uploading a few projects all previously projects seemed to have disappeared, because only a few (max. 10) projects were visible when navigating to the old version of the website catroid.org. More projects were loaded one by one, after clicking on the more button at the bottom of the page. This behaviour has to be changed before the go-live of the website can be announced. Otherwise our young users will get frustrated, when creating and uploading a project to the website and remaining there only for a very short time at the front page. Kids want to share their work and are proud of it, showing it to their friends and parents. A possible workaround for projects to last longer on the front-page is to show new projects group by country or region – this problem is addressed in chapter 9 – future work.

With the new version of the website, projects are grouped by the following categories:

- Featured (only shown when manually activated)
- Most downloaded
- Most viewed
- Newest

By clicking on a nickname, the user's profile and uploaded projects will be shown. Later a tagging and recommender system was implemented. More details will follow in the next sections.

Tests for viewing projects have been done in both, unit-tests and selenium tests. In the unit-tests, project view- and download-count were changed within the test and then the sorted data was verified by the database queries. Selenium tests were just done by checking if the newly uploaded projects were visible in the “newest projects” section.

7.1.6. Remixing projects

Every new project created with Pocket Code is marked as parent project for remixing. When a parent project is downloaded to the Pocket Code-app and uploaded again - no matter if there

were any changes made to the original project – it will be marked as a remix of the original project by our system. Information about a remixed program and the parent project’s ID is stored within the program’s xml-code.

Currently there cannot be given any credit to the creators of the original program. Even information about a remixed version is so far not visible on the website.

7.2. Project details

A lot more details for projects have been added for a successful launch of the Pocket Code-website. Some of them are: adding tags, adding comments, adding a list of similar projects and recommendations what other users did download or remix. Adding some social aspects like “loving a project” or giving it a “star” should also be available on our website. The following features for projects have already been developed and tested during our Google Summer of Code participation in 2012 and 2013⁶⁶:

7.2.1. Adding tags to projects

Adding tags to one’s project is currently only allowed for the user who did upload the project. Tags are visible as shown in Figure 23 with the new website design:



Figure 22: catrobat.org - project with tags added by the user

⁶⁶ <https://www.google-melange.com/gsoc/homepage/google/gsoc2013>

When logged in, every user can add tags to their own already existing projects or add tags when uploading a new project. Tags are stored in a relational way, so it will be easy to find all projects with a tag named e.g. “airplane”. Grouping all projects with similar tags is still in development – see future work for more details. Editing of tags has been optimized when using a smartphone or tablet.

7.2.2. Recommendation of similar projects

A recommendation system was implemented using Myrrix recommender system on a java basis. By the end of July 2013, Myrrix was no longer supported and became a part of Cloudera.⁶⁷ As the recommendation system did not work out as we expected, we choose to put the user-stories on our backlog again. By today, there is no need for reactivating the implementation as there are many important features still missing on the Pocket Code website. We also think that a recommendation system for a very large diversity of projects seems nearly impossible to implement.

7.3. HTML-5 player for projects

Based on the Google Web Toolkit (GWT) there was a HTML-5 implementation to play projects directly in the browser. The biggest limitation of the player was the lack of simulation of the phone’s sensors used in a Pocket Code program. Presentations, short movies or animations were the target programs for use with the HTML-5 player.

7.4. Internationalization

Internationalization of the Catroid-website was first done by different xml-files, one for each supported language. Tests for this implementation were quite trivial, as for each text-block inside a HTML-template, there had to be an xml-translation-file. As new languages were added and the website was redesigned and renamed to first catrobat.org (in 2011) and then to pocketcode.org (by September 2013), a more professional implementation of internationalization was needed. The best available open source project was “pootle – a community localization server”⁶⁸. Implementation was done during our first participation with

⁶⁷ <http://www.cloudera.com>

⁶⁸ <http://pootle.translatehouse.org/>

Google Summer of Code in 2011. Integration of pootle to our website was done without any automated tests.

7.5. Website design changes

Our website design changed from “catroid.org” – a derivation of cat & android. That is why our logo was a transformed android robot with a cat-head – to “catrobat.org”. Prior to the public launch of our visual programming language app and after many Google-Hangout-meetings with the MIT (Scratch) and the Google Education team, our programming app was renamed to Pocket Code, and as well, our new website was launched in September 2013 under the new name “pocketcode.org”.

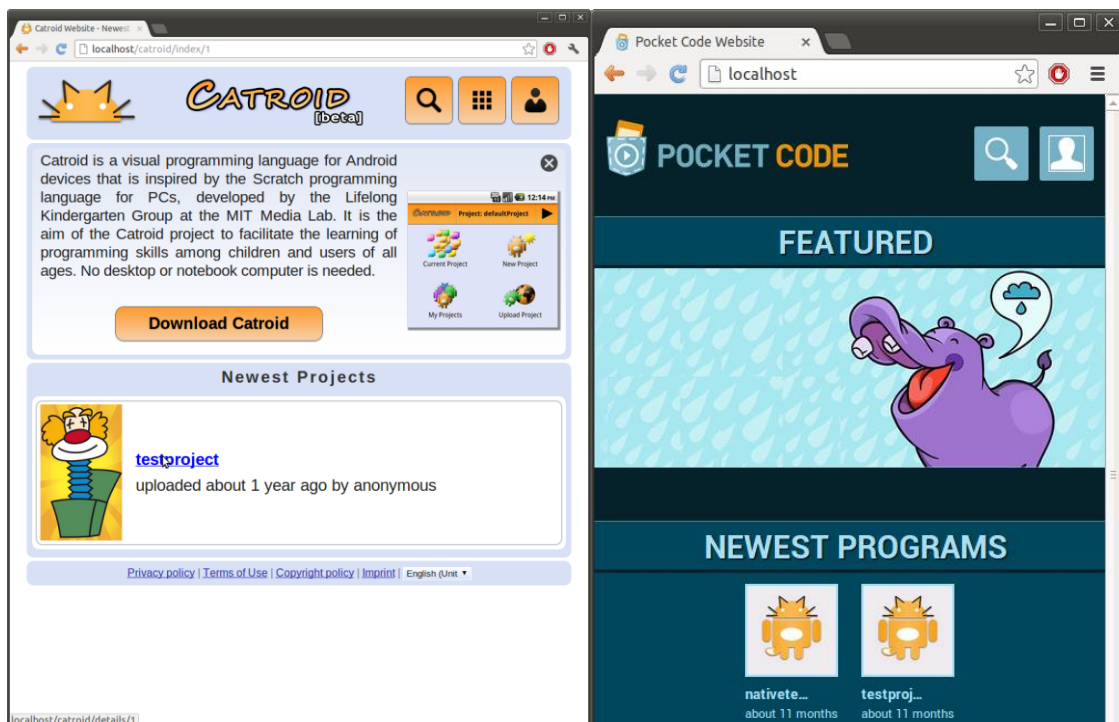


Figure 23: Old (left) and new (right) website design [Cat13]

Most of the tests for the old website could be used for the new one as well. Only some modifications had to be applied and some tests had to be removed. We did greatly benefit from the MVC-pattern used throughout website development. Even Selenium tests did run with only some small parts needing to be rewritten.

By bringing up a new design for the website, we also took the chance to remove all static content and setup a content-management-system for all relevant information like help-pages and

tutorials⁶⁹, introduction for new developers⁷⁰ and some general information about Pocket Code and the Catrobat umbrella organisation⁷¹. A community part of the website is still in development; we currently use Google Groups^{72 73} as a platform for our users and interested developers to discuss any relevant subjects connected to the project. Like Scratch (with Version 1.4), we first had a version of customized phpBB discussion board software with its own user management, but never put it into production. At the MIT, they did change their software to DjangoBB⁷⁴ with the release of Scratch 2.0. We had some discussion about integrating Pocket Code with Scratch as a mobile version of the MIT's visual programming language, sharing user-management, tutorial-materials, discussion forums and moderation, during the release of Google Play for Education⁷⁵ by November 2013.

7.6. Community tools

By listing a large number of projects in many different languages on the Pocket Code website, community tools are essential to give our young users and their parents a good feeling about security and safety. When bad content starts to spread on a website, the reputation is soon gone, and so will be most of the users. In our project, we want to foster young children to not just consume but start creating interactive content on their smartphones and tablets. To provide a safe and creative environment, we did implement some community tools for the Pocket Code website.

7.6.1. Report projects as inappropriate

As registration and contribution to our community is free with easy online registration, everybody can be part of the Pocket Code community. Bad content, e.g. humiliating pictures of someone else can be uploaded to our website and become visible to the whole world. We don't want bad things to happen – so there is a “report as inappropriate” button on every project's details page. Reporting something as inappropriate can be done by each member of the community. Since some foreign languages cannot be understood by our team of developers, the active participation of the community is a helpful way to accomplish a safe site in a wide

⁶⁹ <https://pocketcode.org/tutorial>

⁷⁰ <http://developer.catrobat.org/>

⁷¹ <http://www.catrobat.org/>

⁷² <https://groups.google.com/forum/#!forum/catrobat>

⁷³ <https://groups.google.com/forum/#!forum/pocketcode>

⁷⁴ <http://djangobb.org/>

⁷⁵ <http://developer.android.com/distribute/googleplay/edu/about.html>

variety of languages. Currently one has to be registered and logged in to the Pocket Code website to report something as inappropriate. We are still not sure if we should open this button to everyone without the need to register first. In our beta-version of our homepage (catroid.org), we provided this functionality to all users without registration, but we did ask for the user's e-mail address, when reporting a project as inappropriate.

In my opinion, this functionality should always be available for every visitor of our website. Regardless of any reported projects, our development- and support-teams will constantly have to monitor the website for new content, possibly not suitable for younger children. This can become a problem when a project's title and description cannot be understood because of its language. In our administration-tools, we have some tools to get a quick overview of new or updated projects, by their title, description and containing images (see figure 27, chapter 7.7.2). This functionality has been tested using PHPUnit and Selenium tests. We did write selenium tests after integration of all required functionality to verify its implementation and act as some sort of integration test. Unit tests were developed in a test-first-development style following these steps:

- Write a test `testFlagProject()` to mark a project as inappropriate that fails in the beginning (obviously)
- Write the function body `flagProject()::boolean`, that is empty at first
- Make the test run by adding `return true;` to `flagProject()`.
- Add a parameter `projectID` to `flagProject()`.
- The test fails again
- Make the test run by adding the `projectID` parameter to `testFlagProject()`
- Add the database operation to `flagProject` with `projectID`
- Check for the right status codes to return
- Write a test `testIsProjectInappropriate()` to check if a project is flagged
- Add a new function `isProjectInappropriate()::Boolean`, that is just empty at first
- Make the test run by adding `return true;` to `isProjectInappropriate()`.
- Add a parameter `projectID` to `isProjectInappropriate()`.
- The test fails again
- Make the test run by adding `projectID` to `testIsProjectInappropriate()`.
- Add the database operation to `isProjectInappropriate()` with `projectID`.
- ...

7.6.2. Discussion board

Development of our discussion board was cumbersome all of the time. We did not use single sign on by then, and so we had to register every user automatically to our phpBB-board, resulting in creating a user record with username, password and his e-mail in the phpBB-board database (running locally on our servers). Tests have been done using PHPUnit and Selenium as well, covering registration, login and logout process of the discussion board. Due to graphical limitations and the lack of a mobile version of phpBB-board, we decided to remove it from our project, using Google-Groups instead (until we'd find some better software for our project's purpose).

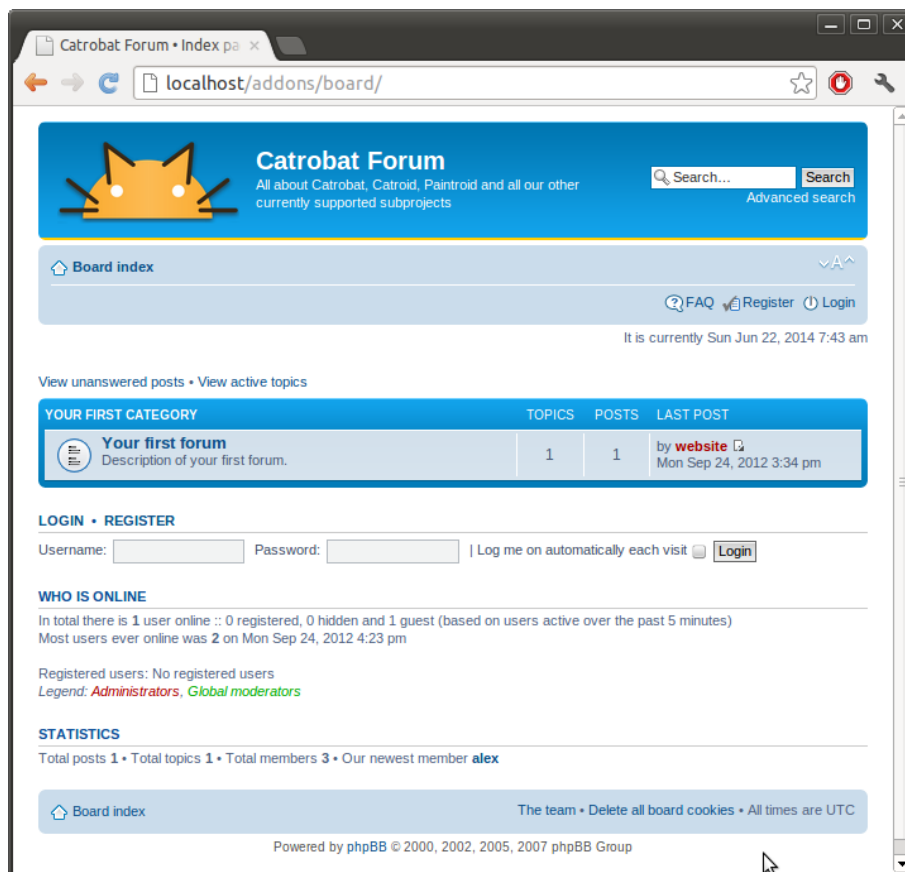


Figure 24: catrobat.org - phpBB discussion board integration

7.6.3. Help-pages and Tutorials

Additional materials have been created and provided on our website for parents, children and teachers. These tutorials include short introduction videos, step-by-step guides and programming hints with screenshots.

7.7. Backend functionality

Support for a large online community requires supervision and monitoring of content added by users, especially for our young target group. We want parents to feel safe when their children start using our software and community tools. To achieve a high quality level and keep our platform safe for all our users, some technical tools and organizational processes had to be implemented.

Our backend tools are simple HTML-pages with lots of functionality, all tested with PHPUnit and Selenium. Most of the functionality was implemented using test-first development and pair programming:

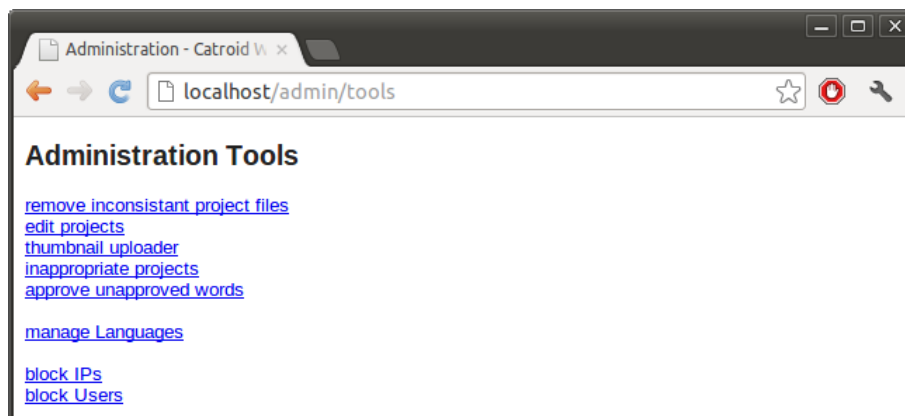


Figure 25: catrobat.org - Administration tools backend

7.7.1. Managing users

User management page gives an overview of all registered users by username, e-mail, country and gender. Access to the website (and uploading projects from Pocket Code-app) can be restricted for individual users.

7.7.1. Blocking users

Blocking users when uploading unsuitable projects or misusing the “report as inappropriate” button can be done with this tool. Additional blocking of IP-addresses is possible, if the user has registered more than one username.

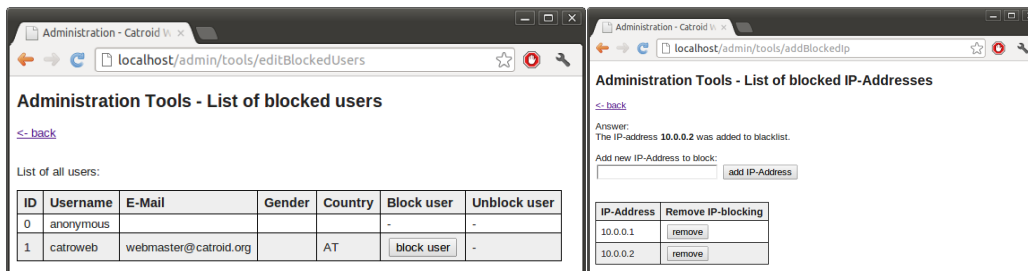


Figure 26: catrobat.org - administration tools for blocking users and IP-addresses

7.7.2. Managing projects

The project overview shows all projects sorted by newest one first, with information about the number of downloads, number of flags (as inappropriate) and the current status – visible or invisible. Admins can toggle a projects visibility after checking when reported as inappropriate and even delete projects that are not suitable to be listed on our website.

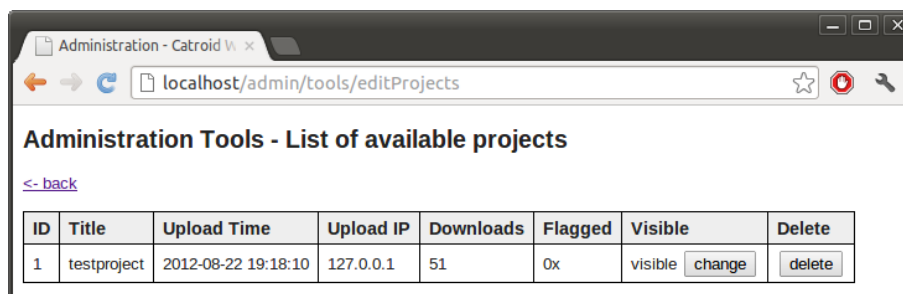


Figure 27: catrobat.org - uploaded projects

7.7.3. Check reported projects

For every project reported as inappropriate, an e-mail message is sent to our internal Google group. Admins can easily resolve a project’s status, if the report was unjustified. We asked the Scratch team, how they handle bad projects: they contact the project’s creator first, and ask him to remove or update his project accordingly. If there’s no feedback, the project will be made invisible by the administrators.

7.7.4. Bad-words filter and contents

Approval or rejection of bad-word in a project’s title and description can be done using this tool. By default, every unknown word gets “no” meaning. We will need to redo this tool for a better usability. As with all our programs, we have all functionality covered with tests, any refactoring can be done without the risk of breaking existing and working parts of our software.

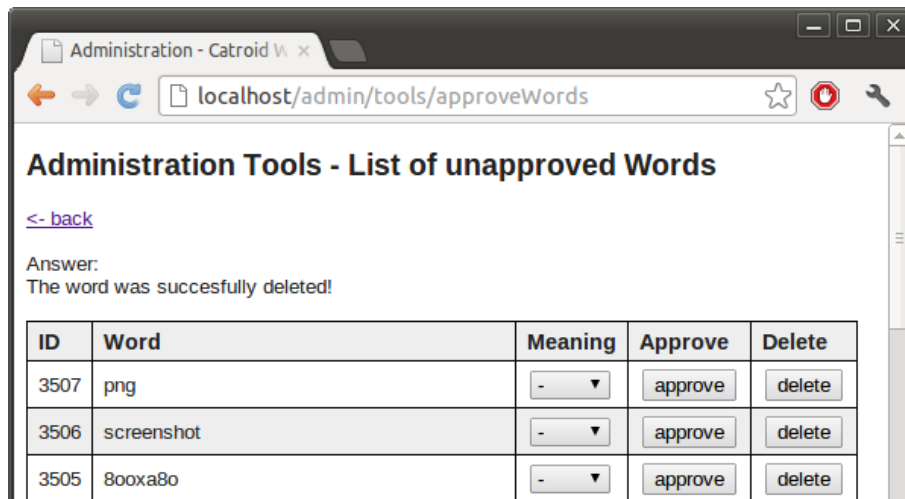


Figure 28: catrobat.org - list of unapproved words

7.8. Work in progress

Our development was driven always writing tests first, if possible. This led to a very small feature set, as we did never think too much ahead – we always tried to make the smallest possible thing work, adding new features at later times, when absolutely needed. Following this way of development kept us focused on the current problems, implementing tiny functions with a very high test-coverage.

As with every community, many automatic checks can help to keep a high quality and reduce misuse to a minimum. But there will always be the need to “review” a project’s content “by hand”, no matter how good automatic recognition tools will work.

All our tools will of course need a better usability, search functionality and some more features, to be able to manage large amounts of projects and users as during and after our public beta release period.

8. Results and Conclusions

Using all of the agile methods in our multi-year FOSS project where possible, we were facing some problems and limitations of these methods as listed in the next subsections.

8.1. Selenium Tests

Writing Selenium tests for all user interactions and input on the website was quite a lot of work. After changing the design and some user-interface parts of the homepage, many tests had to be rewritten, dropped or changed.

While selenium tests need quite a long time when started (currently about 15 minutes on an Intel core i5 2.5 GHz CPU with 8 GB RAM in a VirtualBox Ubuntu 12.04 environment) they could only be executed for integration tests.

Usability changes lead to some major design changes, and because of this, the test-first-design was not always possible (and even not useful) here. Switching to “test-last” was therefore sometimes unavoidable.

After Selenium has released its new version (Selenium WebDriver, or Selenium Version 2), we could run Selenium on our powerful test-server hardware in combination with Selenium Grid, supporting to run several tests in parallel mode. One of the best new features of Selenium 2 was the possibility of taking screenshots automatically for failing tests, while with Selenium 1, we often had to slow down the test-execution speed to watch the system fail; this mostly happened for incomplete or still running callback-requests from the website.

Running all of the projects’ Selenium-tests was done using ant. In February 2013 we changed for automation of test-scripts from ant to Python.

8.2. Kanban Board and Stories

Using a Kanban-Board with story cards helped us to get a quick overview of what features were currently in development. As many of our team-members have not regularly attended our project room, the need for an electronic version of the Kanban-Board became evident. After some research on available Kanban-software we decided to just use a shared google spreadsheet, with edit-rights for every member of the team. As by July 2013 we started to use

Atlassian's Jira⁷⁶ for project- and issue-tracking, and the monitoring of our Google Summer of Code 2013 projects⁷⁷.

8.3. Development environment

After two semester of providing tutorial sessions for new team-developers during courses they took in their computer science lectures and setting up the development environment on several different machines and hardware, our team decided to provide an automated script for installation of all needed components and required setup procedures to develop and run the Catrobat-website on their local machine using Ubuntu Linux⁷⁸. It was quite an effort to setup all needed software and tools, when not running Ubuntu Linux, especially in case of running Mac OS X or Windows 7. Because of those different development environments two of our senior team members⁷⁹ provided a ready-to-use VirtualBox⁸⁰-Image based on Ubuntu Linux as a "development IDE" and a perfect system for running the website on a local machine, including all the tests on all platforms like Windows 7, Apple OS X and even on Ubuntu itself.

8.4. Pair programming

By changing the arrangement of tables, monitors and keyboards in our project room, centered with free access to the different projects' Kanban-Boards, pair-programming was easier as before, when the tables were all in rows, like in traditional classrooms. Grouping around tables helped us discuss new features, sometimes with even more than one pair of programmers around one screen. Discussion of new features affecting more subprojects became also more evident after the changes.

Additional things like free coffee (with milk), free drinks (water, orange juice, apple juice), free cookies, a refrigerator, a microwave oven (with pizza function!) and access for team-members 24 hours a day, 7 days a week, made the project room a nice and comfortable working place for everyone.

⁷⁶ <https://www.atlassian.com/software/jira>

⁷⁷ http://www.google-melange.com/gsoc/org/google/gsoc2013/catroid_project

⁷⁸ <http://www.ubuntu.com>

⁷⁹ Christian Hofer, Roman Mauhart

⁸⁰ Oracle's VirtualBox is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2, <http://www.virtualbox.org>



Figure 29: Catrobat project room at Institute of Software Technology, TU-Graz

8.5. New team-members

To help new members of the team to get fast into testing and programming, we introduced a so-called “buddy-system”. Each senior-developer takes care of a new member of the team for the first few days or weeks, to help with the installation of the programming environment and testing framework, introduction to the test-server environment, introduction to test-first-design, pair programming and to complete the first smaller stories of our Kanban-board. In the beginning, we didn’t support new colleagues adequately, arranging no appointments in our project room and offering no assistance when it came to finding a programming-partner. That often led to frustration and error-prone commits to the project. For new members in the team it is essential to get a good lead and to have a senior developer on their side to help with the first steps of test-driven-development. As in our computer science classes, test-driven-development is educated, but not often used during programming assignments.

8.6. Test-Driven Development

As of today every programmer knows about test-driven development, test-first design and the importance of having tests to successfully change production code without introduction of new bugs or omitting built-in features. In our project we were facing the problem, that everyone knew *exactly* what test-driven development is all about and has read some books from Kent Beck, like “Extreme Programming Explained – Embrace Change”, or “Test-Driven Development By Example”. But the practical skills for writing tests first and start writing code next is not trained or even needed in the programming assignments at university courses. Students who have already working experience in IT-companies have been confronted with test-driven development or at least the need for writing tests. So it was easier for them writing meaningful tests. Finding what tests to write and keeping the small cycle of test-driven

development evident – write a test – see it fail – make it green – refactor the code – was one of the biggest challenges in our project. To overcome these challenges, some training for test-driven-development shall be introduced like a *Coding Dojo* [Sat08] or a *TDD Kata* [Pro13].

When designing and implementing new interface features, the test-driven development or test-driven design was leading us to writing many tests we then had to delete again. For future usability development and modifications of the website we will focus more on mockup-design first, starting implementation only when the main functions and features are approved by our usability-team. Good tools for supporting the creation of mockups are the commercial Balsamiq⁸¹ or the free available Invision-App⁸².

8.7. Standup-Meetings

We used to have standup-meeting for several months, but the effectiveness was decreasing, as there were too many different project members around in our project room to get a good and fast overview of the features and issues currently in development. In the beginning of the project, with only three main teams (Paintroid – later Pocket Paint, Catroid IDE – later Pocket Code and Catroid-Website – later pocketcode.org) the daily standup-meetings were used to coordinate features influencing other teams' functionality.

Basically, standup-meetings are a good way to share knowledge and discuss current issues of development. By our experience, those should be held only within one development team.

Overall, the diverse methods of agile software development helped us achieve the following goals: pair programming, test-driven development, large test-coverage, continuous integration, no source-code documentation, collective code ownership and to building a great team spirit.

⁸¹ <http://www.balsamiq.com>

⁸² <http://www.invisionapp.com>

9. Future work

9.1. Joint venture with scratch

With Google's EDU-play-store release at the beginning of November 2013, many meetings with the Scratch team from the MIT and Google have taken place. At first, the idea was to make the Catroid-App (at this time called: Pocket-Code) a mobile Version of Scratch, by integrating of some of our infrastructure like the website e.g., and renaming our app to Pocket Scratch whilst creating a new logo (as at this time, we already had a nice logo, developed by the FH-Joanneum⁸³ Usability and Design Team) and also changing some of our programming bricks, in both, their look and their functionality to better fit with MIT's Scratch 2.0.

A discussion about a brick called "transparency" within Scratch and "ghost-effect" within Pocket-Code, reviewed retrospectively, has been one of the main differences and possibly the main cause for the delay of our joint-venture for Scratch and Pocket-Code. The idea is still to bring our two projects closer together, but in quite a different way as first imagined: e.g. using one common platform for support (a forum), reusing teacher-materials (like tutorial cards) and tutorial-videos.

Our Pocket-code project was featured in the first weeks of the Google EDU-Play-Store release. To promote the project, a movie about how to use Pocket-code in a real-world educational environment was created by Google in G.I.B.S. International School⁸⁴ in Graz, Austria, supervised by Professor Wolfgang Slany. Since then, the school has been using the Nexus 7 Tablets provided by Google for educational purposes in their informatics-classes.

9.2. Registration

Before uploading a program created with Pocket-Code, some short and easy registration-process has to be passed. As our target age group will be kids ranging from the age of ten to fourteen, as few personal information as possible should be required (only nickname, e-mail, gender and country). Additionally there should be no need to enter a password. The e-mail address is also needed for registration, but this should be extracted automatically from the smartphone, in combination with its IMEI-number.

⁸³ <http://www.fh-joanneum.at/>

⁸⁴ <http://www.gibs.at>

Further on, the selection and use of a personal avatar should become possible. A personal avatar could be created within the Pocket-Paint App, also available from the Catrobat-Project.

9.3. Online tutorials and videos

For a successful use of our “Pocket Code” programming app, online material with tutorials and videos have to be produced explaining how the bricks can be used, how some animation or programming can be done or how to use external resources like images and music or sounds. Tutorials and sample projects should also be provided for school-teachers, so they find it easier to use our app to introduce and teach to their classes and have a better start with it. Some materials for parents are also needed, to give them some background-information of things that their children can learn when using Pocket Code.

9.4. Responsive web-design

Our first design of the website was quite simple (see figure 24, left) and only supported small screens on smartphones and tablets. There was no real desktop version of the website, except some change in the CSS-files showing larger images instead. Since the current devices like Google’s Nexus 7 or Samsung’s Galaxy Tabs support full HD resolution, the website will need a redesign regarding usability and the viewing of content. The website should also support many more different screen-sizes and react in a real responsive way, by showing more or less items, depending on the device used to view the pages.

9.5. Project management and feature-lists

The Catrobat project has grown from about five students in the beginning to – depending on the academic year – more than one hundred students in several project teams now. Of course, not all of them are productive and regularly work in our project room. Most teams are sized from 3 to 10 members with a so called “team coordinator”, responsible for feature implementation and communication with all other involved teams. Weekly coordination meetings to discuss feature integration, release planning and open issues will also have to be held.

For the overall communication and coordination, a full-time project management needs to be introduced to our project. Since Catrobat has been part of Google’s Summer of Code 2011,

2012 and 2013, and will probably be in 2014, there is quite a large amount of overhead that needs to be done. Integration of new features implemented by students during our Google Summer of Code participation needs a lot of attention and supervision.

Feature lists will be put into Atlassian's Jira (as user stories), for a better support of students, working from home or from some other parts of the world. This will help keep better track of implementation "costs" for new features and all occurring bug-issues. In the past, this could only be done by sorting out old story cards from previous releases from our Kanban-board archive.

9.6. Improvement of the community-parts of the website

The community part needs attention to make the project more popular. Currently the discussion forum⁸⁵ can be found at the help-pages of the Pocket-code website. By the time writing this thesis, there is no real discussion on projects and topics like there is at the Scratch community-website. There, children (and other users) can discuss how some projects were made, how special kinds of animations could be done, how counting high scores can be programmed and many more things kids are interested in, when creating or remixing projects.

9.6.1. Discussion board (forum)

A discussion forum like there currently is at the Scratch community website will be important to support users with questions on different kinds of topics. Fast response time – either by moderators or by the community itself – will be a major success factor in the long run. The forum should also be easy to use and have several predefined topics. Moderation of the forum's posts is very important and should be planned thoroughly. Posting to the forum should only be allowed for registered users or people providing an e-mail address, so the person is somehow "identified". A read only view for all postings should be possible for everyone.

9.6.2. Remixing of projects

Remixing projects by adding own ideas have been a success factor for Scratch [Mon11]. But it is also very important, to give credit to the creators of a project, this can be done automatically by the Pocket-Code app and website, when uploading a remixed version of a project. The remixing history

⁸⁵ Google Group at <https://groups.google.com/forum/m/?fromgroups#!forum/Pocket Code>

should be saved in our project database of the community website. So, it will be easy to extract the dependent projects from which a remix is derived. In the context of remixing is the next proposed feature – commenting on projects, to make the website more attractive to share one’s ideas.

9.6.3. Commenting on projects and automatic bad-words filter

Users like to voice their opinions of projects others have made. Sometimes they will also dislike things, and that is one point, where a community-website for kids and teenagers needs to keep an eye on. We already have a bad-words-filter on the website, used to check the username, title and description of a project, currently only in the two languages English and German. As our project has been translated by the community to many other languages, like Chinese (traditional and mandarin), Russian, Malayan and Romanian, filters also need some attention. For German and English we put the most common good and bad words in filters, to have something to start from. If someone uploads a project, the content is checked by our filters and marked accordingly. If unknown words are found the moderators get an e-mail message and will find new unapproved words on their dashboard to confirm or reject these contents. If content has to be rejected, then the user needs to be informed about this. If he or she continues to write unfriendly or bad things, they will have to be blocked for some time.

As available at the Scratch community website, adding comments should only be possible for registered users. As an additional feature it should be possible to report content of comments as inappropriate, similar to the way this is already available for a project’s title, description and contents.

Another possibility to automate recognition of bad words in other languages than German or English is to use e.g. Google’s translate API, to first, translate the words to English and afterwards run the bad-words-filter on the translated content. For alternative translation tools, further research on available open-source solutions needs to be done.

9.6.4. Reviewing projects and user control

Our community website will need a process for reviewing projects, especially when comments on projects will be implemented. As users will find it strange if their comments are not shown immediately, it is best, to just publish the contents and have a moderator overview things regularly [Mon09]. As the Catrobat-project is an open-source project at our University, with currently only two full-time employed academics, mostly handling coordination and management issues, moderators will be students working on programming projects or on their bachelor- or master-thesis. A dashboard for project review process and user control must be available for all of them, to

get a quick overview, if some things need attention as well as the information that someone is already working on resolving an issue. Moderators should also have a tool, to write internal comments on users, projects and users' comments. User control should also be possible from within these tools. A good way as learned from the Scratch-team is, to talk (write) to the user first and explain the rules as a first step. Only if the user doesn't stop their inappropriate behavior, some action should be taken, like blocking the user for e.g. one hour. If there is still no change in his behavior, the blocking can continue up to one day, one week or result in a permanent blocking of the user.

Blocking of IP-addresses should also be possible, because if blocked users will keep on with inappropriate actions by registering new IDs, instead of using their currently blocked user-account, all access from this single IP-address or an IP-range should be denied.

9.6.5. Organizational issues

As the main target groups of our project and our community website are children from ten to fourteen, we have to assure that all content provided either in projects or in comments, is suitable for them. To accomplish this we need to apply some organizational changes, those are:

- Having a sort of 24x7 support for the community website
- Reviewing projects and its descriptions frequently
- Removing inappropriate contents
- Checking project's comments frequently
- Replying to posts to our website's discussion-board

Compared to the MIT's Scratch project, where almost all of the team-members are employees of the institute, our project has only limited financial resources (used for the purchase of new hardware) and currently no possibility for long-term employment of students in our project.

9.7. Implement short development-cycles for continuous release of features

Like mentioned in chapter 6 about our new Git branching model for Catrobat, there is still some work to implement this new model in our development process.

9.8. Help-pages: getting started, exploring, guides

Some background-information on how to get started for kids and parents is needed. To give some easy entry points to programming, some hints, on what topics to start first and some

simple exercise for different target age-groups are important for the further success of the Catrobat-project. We will also need some material to supply to teachers, as shown on the Scratch website⁸⁶, to increase their interest in our project and to use it for their classes at school. All those materials should be easy to find on our website, easy to access and without the need for a login or registration to use them.

Some guides for programming bricks are also a must. There should be several parts of guides and explanations for all different kind of bricks like loops, variables, conditions and motions. A good example for those kind of guides suitable for kids can be found at Lego's EVO3 Mindstorms Series Website⁸⁷.

9.9. Test-driven-development exercises

Learning TDD in a university course or at school is one thing, using these techniques in daily programming work is another one. As sport professionals need to train their moves, their strength and skills every day, so does a (sooner to be professional) programmer.

Training TDD can be done in various ways. Some really excellent exercises mentioned in "Practices of an agile developer", "The art of agile development", "The pragmatic programmer" and "97 things every programmer should know" are to either have coding-dojos⁸⁸ from time to time or programming katas⁸⁹ every day for a short amount of time, not more than 15 to 30 minutes, by solving small exercises from scratch, using TDD-techniques [Hen10], [Hun09], [Ras10], [Hun11].

When students start working on our project, they often know how to write tests, but not exactly what to test. Writing good tests is essential for readability and maintainability. Many of the tests written in the beginning have been refactored and maintained over the time, and some of them became obsolete after changing some main functionality on our community-website. Since the structure of the test code was sometimes complicated and hard to read and because of this, even harder to understand, we came to the conclusion that tests should be written as simple as possible, with only one thing to test in each function. Learning to write good quality test code should be the one thing to be taught in our project, by having regular coding dojos on a team-basis in small groups. Website development needs many different aspects of testing, like unit-

⁸⁶ <http://scratched.media.mit.edu>

⁸⁷ <http://www.lego.com/en-gb/mindstorms/?domainredir=mindstorms.lego.com>

⁸⁸ <http://www.butunclebob.com/ArticleS.UncleBob.TheProgrammingDojo>

⁸⁹ <http://www.butunclebob.com/ArticleS.UncleBob.ThePrimeFactorsKata>

tests of core modules used in the content-management-system, selenium-tests of website usage by simulation of user stories and tasks, and last, database testing, that can be done within the unit-tests.

Training these parts of testing should be integrated into our daily project work, and should be done regularly, especially when new members will start to support our teams.

10. List of Figures

<i>Figure 1: Bricks used for graphical programming and how they can be put together [Cat13]</i>	13
<i>Figure 2: New Pocket Code (left), new Pocket Paint logo (middle) and old version (right) ..</i>	14
<i>Figure 3: Scratch – new project [Mit13]</i>	19
<i>Figure 4: Microsoft Kodu GameLab community website: project overview [Kod13]</i>	24
<i>Figure 5: Microsoft Kodu GameLab community website: project details page [Kod13]</i>	25
<i>Figure 6: Lego Mindstorm Programming environment [Leg13].....</i>	28
<i>Figure 7: Scratch 1.4 programming editor window [Res09].....</i>	29
<i>Figure 8: Scratch programming blocks [Res07]</i>	29
<i>Figure 9: Projects sometimes being delivered in today’s software development [Ssw13]</i>	35
<i>Figure 10: sticking to a plan [Ras10, p. 5]</i>	41
<i>Figure 11: Kanban-story board of Catrobat’s web-team</i>	43
<i>Figure 12: Virtual Kanban-story board of Catrobat’s web-team using Atlassian’s Jira</i>	43
<i>Figure 13: PHPunit test-results</i>	45
<i>Figure 14: Pair programming setup in our project room.....</i>	66
<i>Figure 15: Catroid API LoginTest.php</i>	70
<i>Figure 16: Git branches “develop” and “master” [Pe12]</i>	80
<i>Figure 17: A successful git branching model [Pe12]</i>	81
<i>Figure 18: Another Git branching model [Dri10].....</i>	83
<i>Figure 19: RemObjects Blogs, [Hof11]</i>	84
<i>Figure 20: New Catrobat Git branching model.....</i>	86
<i>Figure 21: Catrobat Administration tools (catrobat.org) - list of unapproved words</i>	91
<i>Figure 22: catrobat.org - project with tags added by the user</i>	94
<i>Figure 23: Old (left) and new (right) website design [Cat13].....</i>	96
<i>Figure 24: catrobat.org - phpBB discussion board integration</i>	99
<i>Figure 25: catrobat.org - Administration tools backend</i>	100
<i>Figure 26: catrobat.org - administration tools for blocking users and IP-addresses</i>	101
<i>Figure 27: catrobat.org - uploaded projects</i>	101
<i>Figure 28: catrobat.org - list of unapproved words</i>	102
<i>Figure 29: Catrobat project room at Institute of Software Technology, TU-Graz.....</i>	105

11. References

- [Mic13] Microsoft Fuse Labs, <http://fuse.microsoft.com/projects/kodu>, August 2013.
- [Kod13] Microsoft Kodu GameLab community website, <http://www.kodugamelab.com>, August 2013.
- [Lif13] Lifelong Kindergarten Group at the MIT Media Lab. Scratch Stats. <http://stats.scratch.mit.edu/community/>, August 2013.
- [Mit14] About Scratch, <http://scratch.mit.edu/about/>, January 2014
- [YoY13] YoYo Games and GameMaker: Studio, <http://www.yoyogames.com/>, August 2013
- [Nin13] Nintendo Flipnote Studio Website, <http://flipnotestudio.nintendo.com/notice/>, August 2013
- [Nin13a] Nintendo Flipnote Studio 3D Website, <http://flipnotestudio3d.nintendo.com/>, August 2013
- [Son13] Sony PlayStation, The Little Big Planet Website, <http://www.littlebigplanet.com/about>, August 2013
- [Sat08] Sato, D.T.; Corbucci, H.; Bravo, M.V., "Coding Dojo: An Environment for Learning and Sharing Agile Practices", *Agile 2008. AGILE '08. Conference*, pp.459,464
- [Pro13] Peter Provost's Geek Noise, <http://www.peterprovost.org/blog/2012/05/02/kata-the-only-way-to-learn-tdd>, August 2013
- [Kol08] Kollanus, S.; Isomöttönen, V., "Understanding TDD in academic environment: experiences from two experiments.", *In Proceedings of the 8th International Conference on Computing Education Research (Koli '08). ACM, New York, NY, USA, 25-31, 2008*
- [Kol11] Kollanus, S., "Critical issues on test-driven development.", *In Proceedings of the 12th international conference on Product-focused software process improvement (PROFES'11), Danilo Caivano, Markku Oivo, Maria Teresa*

Baldassarre, and Giuseppe Visaggio (Eds.). Springer-Verlag, Berlin, Heidelberg, 322-336, 2011.

- [Kol10] Kollanus, S., "Test-Driven Development - Still a Promising Approach?," *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the* , vol., no., pp.403,408, Sept. 29 2010-Oct. 2 2010
- [Bec01] Beck, K.; Fowler, M.; "Planning Extreme Programming", The XP Series, Addison-Wesley, 2001
- [Bec05] Beck, K.; Andres, C.; "Extreme Programming Explained – Embrace Change", Second Edition; Addison-Wesley, 2005
- [And10] Anderson, D. J.; "Kanban – Successful Evolutionary Change for Your Technology Business", Blue Hole Press, Sequim, 2010
- [Whi12] Whittaker, J.; Arbon, J.; Carollo J.; "How Google Tests Software", Addison Wesley, 2012
- [Pag09] Page, A; Johnston, K.; Rollison Bj; "How We Test Software at Microsoft", Microsoft Press, 2009
- [Tur10] Turhan, B.; Layman, L.; Diep, M.; Erdogmus, H.; Shull, F.; "How Effective Is Test-Driven Development?", from "Making Software, What really works, and why we believe it", O'Reilly, 2011
- [Wil09] Williams, L.; Kudrjavets, G.; Nagappan, N.; "On the Effectiveness of Unit Test Automation at Microsoft", 20th International Symposium on Software Reliability Engineering, IEEE, 2009
- [Mar07] Martin, R.C., "Professionalism and Test-Driven Development", *Software, IEEE* , vol.24, no.3, pp.32,36, May-June 2007
- [Gol10a] Goldman, M., "Test-driven roles for pair programming", *Software Engineering, 2010 ACM/IEEE 32nd International Conference on* , vol.2, no., pp.515,516, 2-8 May 2010

- [Gol10b] Goldman, M.; Miller,R.C.; “Test-driven roles for pair programming“, Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10); ACM, New York, NY, USA, p13-20., 2010
- [Abr11] Abrantes, J.F.; Travassos, G.H., "Common Agile Practices in Software Processes", Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, pp.355,358, 22-23 Sept. 2011
- [Ber12] Bergman, R., "Embracing Nihilism as a Software Development Philosophy and the Birth of the Big Book of Dead Code.", Agile Conference (AGILE), 2012 , vol., no., pp.86,91, 13-17 Aug. 2012
- [Pro12] Provost, P.; “Kata - the Only Way to Learn TDD”,
<http://www.peterprovost.org/blog/2012/05/02/kata-the-only-way-to-learn-tdd>, last visited: September 13, 2013.
- [Cod12] Codingdojo, “What is Coding-Dojo?”, <http://codingdojo.org/>, last visited: September 13, 2013.
- [Ren08] Rendell, A., "Effective and Pragmatic Test Driven Development", Agile, 2008. AGILE '08. Conference , vol., no., pp.298,303, 4-8 Aug. 2008
- [Cau12] Cauevic, A.; Punnekkat, S.; Sundmark, D., "Quality of Testing in Test Driven Development," Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the, pp.266,271, 3-6 Sept. 2012
- [Wir09] Rebecca J. Wirfs-Brock, "Design for Test", IEEE Software, vol. 26, no. 5, pp. 92-93, Sept.-Oct. 2009
- [Jan08] Janzen, D.S.; Saiedian, H., "Does Test-Driven Development Really Improve Software Design Quality?", Software, IEEE , vol.25, no.2, pp.77,84, March-April 2008
- [Bec03] Beck, K., “Test-Driven Development By Example”, Addison Wesley, 2003

- [Kos08] Koskela, L.; “Test Driven, Practical TDD and Acceptance TDD for Java Developers”, Manning Publications, 2008
- [Cri09] Crispin, L.; Gregory, J., “Agile Testing, A Practical Guide for Testers and Agile Teams”, Addison-Wesley, 2009
- [Agm01] Beck, K. et. al, “Manifesto for Agile Software Development”, <http://agilemanifesto.org>, last visited: September 23, 2013.
- [Amb13] Ambler, S. W., “Examining the Agile Manifesto”, Ambysoft Website, <http://www.ambysoft.com/essays/agileManifesto.html>, last visited: September 23, 2013.
- [Hun09] Hunt, A., “Practices of an Agile Developer”, The Pragmatic Bookshelf, 2009.
- [Hum11] Humble, J.; Farley, D., “Continuous Delivery, Reliable Software Releases through Build, Test, and Development Automation”, Addison-Wesley, 2011
- [Cha09] Chacon, S., “Pro Git”, e-book, 2009, <http://git-scm.com/book>
- [Sho08] Shore, J.; Warden, S., “The Art of Agile Development”, O’Reilly, 2008
- [Sel13] SeleniumHQ Browser Automation, <http://docs.seleniumhq.org>, last visited: September 30, 2013
- [Joh11] Johansen, Ch., “Test-Driven JavaScript Development”, Developer’s Library, Addison-Wesley, 2011
- [Php13] PHPUnit Website, <http://phpunit.de/manual/current/en/>, last visited: September 30, 2013
- [Zel12] Filipin, Ž., “Homebrewer’s Guide to Watir”, Leanpub books, 2012
PDF version: <http://leanpub.com/watirbook>
- [Bur12] Burtscher, D., “Master’s Thesis: Introduction of a Continuous Integration Process in an Open Source Project”, 2012

- [Sel13] Selenium HQ Browser Automation Website, <http://docs.seleniumhq.org/projects/>, last visited: October 2, 2013
- [Ras10] Rasmusson, J., “The Agile Samurai – How Agile Masters Deliver Great Software”, The Pragmatic Programmer, Pragmatic Bookshelf, 2010
- [Coh10] Cohn, M., “Agile Estimation And Planning”, Pearson Education, Prentice Hall, 2010
- [Cat13] Pocket Code on Google Play, <https://play.google.com/store/apps/details?id=org.catrobat.catroid>, last visited: October 7, 2013
- [Res02] Resnick, M. et al., “Scratch: A Sneak Preview”, 2002
- [Res09] Resnick, M. et al., “Scratch: Programming for All”, Communications of the ACM, November 2009, Vol. 52, No. 11
- [Mon11] Monroy-Hernández, A. et. al., “Computers can’t give credit: How automatic attribution falls short in an online remixing community”, CHI 2011, May 7-12, 2011, Vancouver, BC, Canada
- [Hil10] Hill, M.B. et al., “Responses to remixing on a social media sharing website”, Association for the Advancement of Artificial Intelligence, 2010
- [Gri11] Gritschacher, T., “Master’s Thesis: A Community Website for Interactive Mobile Content Created by Children and Teenagers”, 2011
- [Rot07] Rothman, J., "Manage It! Your Guide to Modern, Pragmatic Project Management, The Pragmatic Bookshelf, Raleigh, 2007
- [Edw04] Edwards, S.H., “Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action”, ACM, SIGCSE’04, March 3-7 2004
- [Gtb14] Google Testing Blog, <http://googletesting.blogspot.com>

- [Mon09] Monroy-Hernández, A., "Designing a website for creative learning.", Proceedings of the Web Science 09: Society On-Line, 18-20 March 2009, Athens, Greece.
- [Ssw13] SSW-Website, Rules to better Scrum using TFS
<http://rules.ssw.com.au/Management/RulesToBetterScrumUsingTFS/Pages>
last visited September 06, 2013
- [Ber12] Berry, Andrew, "*Git Best Practices: Workflow Guidelines*",
<http://www.lullabot.com/blog/article/git-best-practices-workflow-guidelines>
last visited September 06, 2013
- [San13] Sandofsky, Ben, "*Understanding the Git Workflow*",
<https://sandofsky.com/blog/git-workflow.html>
last visited: September 06, 2013
- [Dri10] Driessen, Vincent, "*A successful Git branching model*",
<http://nvie.com/posts/a-successful-git-branching-model/>
last visited: September 6th, 2013
- [Pel12] Pelletier, Aurélien, "*Another Git branching model*",
<http://blogpro.toutantic.net/2012/01/02/another-git-branching-model/>
last visited: September 6, 2013
- [Dym12] Dymitruk, Adam, "*Branch-per-Feature*",
<http://dymitruk.com/blog/2012/02/05/branch-per-feature/>
last visited: September 6, 2013
- [Cac09] Chacon, Scott., "Pro Git", 2009, retrieved from <http://git-scm.com/book>
last visited: September 6, 2013
- [Sus08] Susser, Josh, "Agile git and the story branch pattern",
<http://blog.hasmanythrough.com/2008/12/18/agile-git-and-the-story-branch-pattern>
last visited: September 6, 2013

- [Hen09] Henrichs, Rein, “*A Git Workflow for Agile Teams*”,
<http://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html>
last visited: September 6, 2013
- [Pop08] Pope, Tim, “*A Note About Git Commit Messages, April 2008*”,
<http://tbagery.com/2008/04/19/a-note-about-git-commit-messages.html>
last visited: September 6, 2013
- [Hof11] Hoffman, Marc, “*Our New Git Branching Model*”,
<http://blogs.remobjects.com/blogs/mh/2011/08/25/p2940>
last visited: September 6, 2013
- [Hen10] Henney, Kevlin, “97 Things Every Programmer Should Know”, O’Reilly Media
Inc, 2010
- [Hun11] Hunt, Andrew; Thomas, David, “The Pragmatic Programmer – from journeyman to
master”, Addison-Wesley, 2011