Karlsruhe Institute of Technology

# Collaborative and transparent Free Software development

Diploma thesis
by

**Lydia Pintscher**

Institute of Applied Informatics and Formal Description Methods (AIFB)
of the Faculty of Economics and Business Engineering

| | |
|---|---|
| Referent: | Prof. Dr. Rudi Studer |
| Betreuer: | Basil Ell |

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, ______________________________

for true friends and family
always close - no matter how far

# Zusammenfassung, deutsch

In dieser Diplomarbeit wird der Entwicklungsprozess von zwei Freien Software Projekten, Halo und Amarok, analysiert und verbessert. Halo ist eine Gruppe von Erweiterungen für Semantic MediaWiki (SMW) und Amarok ist ein Musikabspielprogramm. Sie repräsentieren ein breites Spektrum an Freien Software Projekten mit ähnlichen Herausforderungen. Diese Diplomarbeit konzentriert sich darauf die Kollaboration in den Projekten zu verbessern und die Transparenz in beiden Projekten zu erhöhen.

Im Rahmen dieser Diplomarbeit wurden mehrere Werkzeuge und Prozesse entwickelt. Diese beinhalten einen Weg eine Vision für ein Projekt zu erstellen, Werkzeuge zur Verbesserung der Kollaboration in einem Team, Methoden und Werkzeuge zur Verbesserung der Qualität der Software des Projekts, sowie Feature Tracking und Planerstellung. Diese Werkzeuge und Prozesse passen sowohl zu den Idealen als auch zur tagtäglichen Realität der Erstellung Freier Software und sind daher für eine große Auswahl an Freien Software Projekten geeignet.

# Abstract

In this thesis the development process of two Free Software projects, Halo and Amarok, is analysed and improved. Halo is a set of extensions to Semantic MediaWiki (SMW) and Amarok is a music player. They represent a broad spectrum of Free Software projects with similar challenges. This thesis focuses on improving the collaboration in these projects and increasing the transparency of both projects.

Through this thesis several tools and processes are developed. This includes a way to create a vision for a project, tools to improve collaboration in a team, methods and tools to improve the quality of the project's software and feature tracking and roadmap creation. These tools and processes fit both the ideals and day-to-day realities of Free Software creation, making them suitable for a wide range of Free Software projects.

# Disclaimer

I am involved in both projects that are the subject of this thesis. I am the Community Manager and Release Manager of Amarok as well as a member of the Amarok Committee. Additionally I am a member of KDE's Community Working Group and manage several mentoring programs (Google Summer of Code, Google Code-in and Season of KDE) for KDE, Amarok's umbrella community. I am also employed by ontoprise to work on community developer engagement for Halo. This gives me a unique view of the problems and needs of both teams as well as a very good position to implement the changes proposed in this thesis. However, it also means that sometimes I can not cite sources for some of my observations because I either received the information during communication with other team members or because I observed them over the years of working with those communities.

# Contents

# List of Figures

# List of Tables

# List of Listings

# List of URLs

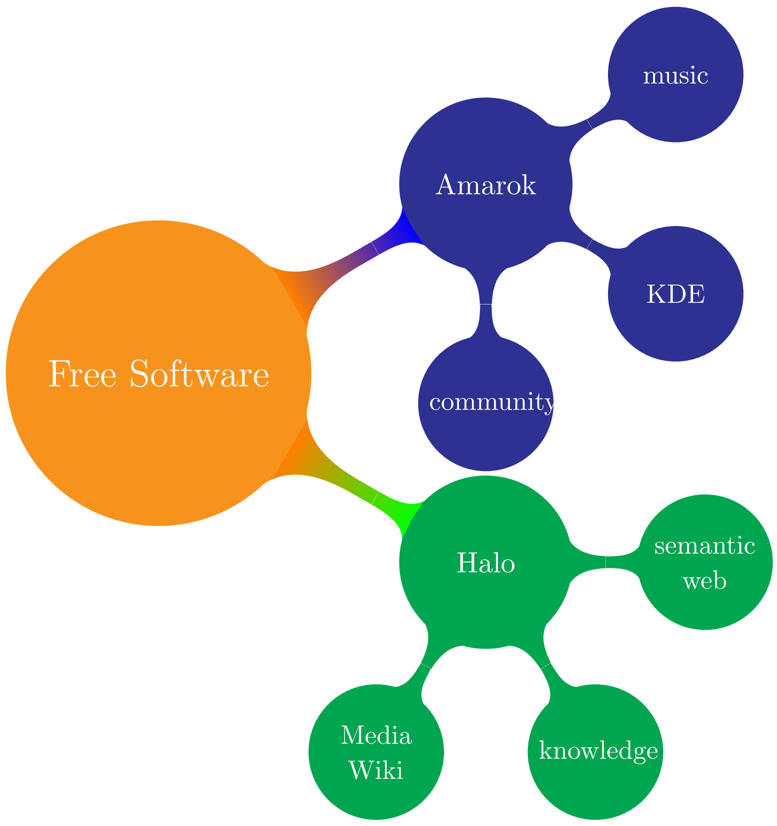
# List of Abbreviations

| | |
|---|---|
| API | application programming interface |
| CMMI | Capability Maturity Model Integration |
| CVS | Concurrent Versions System |
| FESCo | Fedora Engineering Steering Committee |
| FSF | Free Software Foundation |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IRC | Internet Relay Chat |
| MW | MediaWiki |
| PCS | Process Capability Score |
| PSS | Process Success Score |
| QA | quality assurance |
| RDF | Resource Description Framework |
| REST | Representational State Transfer |
| RSS | Really Simple Syndication |
| SC | Software Compilation |
| SCAMPI | Standard CMMI Appraisal Method for Process Improvement |
| SIPOC | Suppliers, Inputs, Process, Outputs, Customers |
| SMW | Semantic MediaWiki |
| SOAP | Simple Object Access Protocol |
| SPARQL | SPARQL Protocol and RDF Query Language |
| SVN | Apache Subversion |
| UAD | Unifying Action Declaration |
| UI | user interface |
| XML | Extensible Markup Language |

*"Everything that is really great and inspiring is created by the individual who can labor in freedom."*

Albert Einstein

# 1 Introduction



Free Software projects are a curious phenomenon. Many of them function without a clear leader and rely on people contributing to the project in their free time without any monetary compensation. At the same time, Free Software is an integral part of today's technology world.

Free Software projects vary in many ways, such as size, contributors and commercial interest. Despite these differences, many of them seem to face very similar problems. Two things are crucial for the survival of a Free Software project: collaboration and transparency. Due to their often distributed nature, these projects rely extensively on collaboration. Transparency is a requirement for good collaboration and project sustainability.

Amarok is an open source music player and one of the flagship applications developed by KDE, a large and mostly volunteer-driven Free Software project that, among other things, develops a desktop environment and hundreds of its applications. Halo, on the other hand, is a set of extensions for Semantic MediaWiki (SMW), developed in a mostly business-driven Free Software project. Halo and Amarok have similarly-sized developer teams (KDE as a whole is considerably larger). For both projects, it is important to strive for more transparency in the development process to make contributing easier for both team members and potential new contributors.

The goal of this thesis is to analyse and improve the development process of these two Free Software projects. The improvements will be enacted using tools the project members are already familiar with. These include but are not limited to MediaWiki (MW) and SMW. In the end, deeper knowledge of the development process behind middle-sized Free Software projects will be gained as well as tools and methods for improving this process. The focus of the work will be on increasing transparency and collaboration in both projects. Halo and Amarok have been chosen because they represent two very different kinds of Free Software projects in terms of commercial interest, target users, reasons for contributor involvement and leadership.

The remainder of this thesis is divided into seven parts. Chapter 2 provides an overview of the necessary fundamentals to understanding Free Software in general and the projects that are the focus of this thesis in particular. In Chapter 3 related work is explored. Chapter 4 contains an in-depth analysis of the Halo and Amarok projects and their development processes. The new development process is introduced in Chapter 5 and its implementation is explained in Chapter 6. The new process is then evaluated in Chapter 7. Chapter 8 provides conclusions and an outlook for future work.

*"The universe is full of magical things patiently waiting for our wits to grow sharper."*

Eden Phillpotts

# 2 Fundamentals

This chapter gives an overview of areas that are fundamental to understanding the following chapters of this thesis. It starts with an introduction to collaboration and transparency and a general overview of what Free Software is and continues with a characterisation of the projects that are the basis of this research.

## 2.1 Collaboration and transparency

One of Merriam-Webster's definitions for collaboration is: the act of "working jointly with others or together especially in an intellectual endeavor". In Free Software, this means working together on software. This collaboration can happen in the same physical location or virtually. It can happen in real-time or not. It can happen between members of the team and between members of the team and non-members (e.g., users).

In this thesis, transparency describes the easy access to and visibility of information. It is an essential ingredient of collaborative work. This transparency can exist or not exist in a team as well as between a team and outsiders. For this writing, transparency encompasses questions like "Who is working on task A?" or "What is person B working on?" and "What needs to be done to finish task C?".

## 2.2 Free Software and Open Source

Free Software describes a philosophy of developing and distributing software. The emphasis is on making the software freely available to both developers and users and encouraging them to modify it for their needs and share it with others. In its Free Software definition [Sta86] the Free Software Foundation (FSF)[1] defines four freedoms that a program must fulfil to be considered Free Software:

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and change it to make it do what you wish.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to distribute copies of your modified versions to others.

The Open Source definition [Per98] of the Open Source Initiative[2] as well as the Debian[3] Free Software Guidelines [Pro97] are the two other well regarded definitions of when a program can be considered to be free[4]. The difference lies mostly in whether one wants to emphasise the social or technical aspect of software being free.

Free Software has become a major player in the software industry in recent years. Some very successful programs include:

- *Apache*[5], a Hypertext Transfer Protocol (HTTP) server, runs on more than 57% of active webservers and more than 66% of the servers hosting the one million busiest websites [Cow10].
- *Android*[6], Google's mobile operating system, has climbed from a market share of below 5% of all smart phones to nearly 20% in one year [Com10].
- *Firefox*[7], a web browser, has a market share of around 30% [Wik10].
- *MediaWiki*[8], a wiki software, is the basis of thousands of websites, among them Wikipedia[9], one of the ten most visited websites [Ale10].

[1]http://fsf.org
[2]http://opensource.org
[3]http://debian.org
[4]From here on free will be considered as libre, not gratis.
[5]http://httpd.apache.org
[6]http://android.com
[7]http://firefox.com
[8]http://mediawiki.org
[9]http://wikipedia.org

Free Software projects are the entities in which one or more programs are developed. There are a lot of different ways a Free Software project can be set up. The spectrum goes from projects that are run by companies entirely, over mixed projects to projects entirely driven by a volunteer community (see Figure 2.1). The reasons for being involved vary depending on the set-up of the project. In company-driven projects the motivation tends to be more on the extrinsic side (e.g., money and social pressure) while in volunteer-driven projects the motivation tends to be more on the intrinsic side (e.g., fun, personal development and friendship). These differences in motivation often also lead to a different work and communication style. Almost all of the volunteer-driven projects tend to be spread out over different countries, time zones and cultures. This brings a number of challenges regarding team building and work coordination. Some of the measures taken to alleviate this can likely also help companies deal better in a world where working from home and global teams become more and more common.

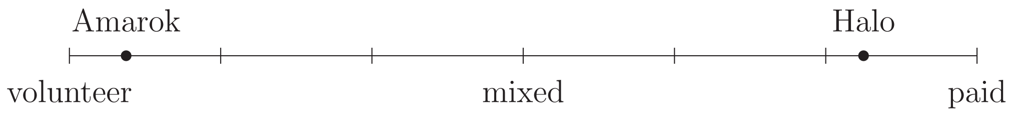


**Figure 2.1:** Spectrum of driving forces of a Free Software project (and approximate position of Amarok and Halo)

Another way Free Software projects can be characterised is by the perceived mean distance between the individual layers of its community (see Figure 2.2). The community consists of people who are closer to or farther away from the core team of the project. The bigger the perceived distance between each community layer is, the harder it is to create a feeling of belonging in the non-core team. This feeling of belonging, however, is what keeps contributors involved in a project [Bac09].

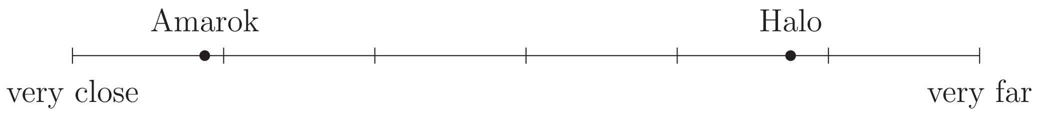


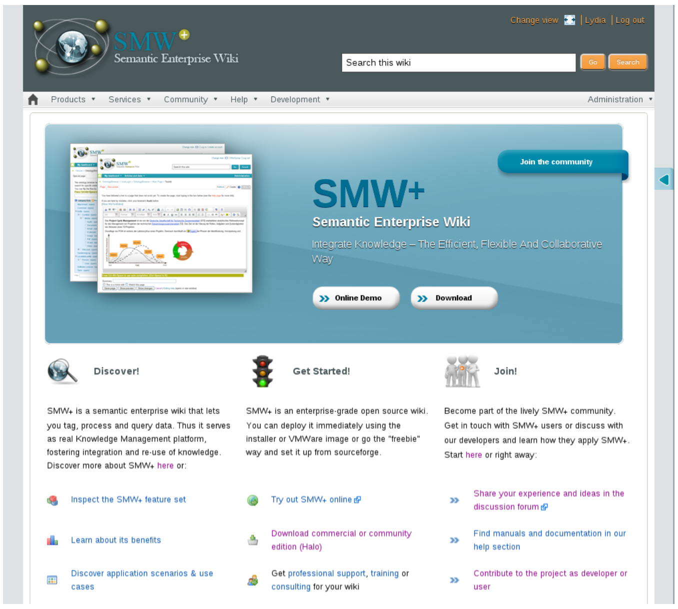
**Figure 2.2:** Perceived mean distance between community layers of a Free Software project (and approximate position of Amarok and Halo)

Releases are the heartbeat of a Free Software project. They are the source code snapshots that the authors deem suitable for public consumption – be it for testing or production use. Some projects develop their product in the open with everyone having access to the version control system history (KDE[10], GNOME[11]). Other projects only publish releases (Bash[12], XMind[13]). Mixed models in which the ver-

[10]http://kde.org
[11]http://gnome.org
[12]http://tiswww.case.edu/php/chet/bash/bashtop.html
[13]http://xmind.net

sion control history is published with a delay (Qt[14], Android) also exist. Releases are often labelled as major and minor releases based on whether they add new features or only bug fixes compared to the previous release.

## 2.3 Halo



SMW+ is a semantic enterprise wiki that lets you tag, process and query data. Thus it serves as real Knowledge Management platform, fostering integration and re-use of knowledge. Discover more about SMW+ here or:

- Inspect the SMW+ feature set
- Learn about its benefits
- Discover application scenarios & use cases

SMW+ is an enterprise-grade open source wiki. You can deploy it immediately using the installer or VMWare image or go the "freebie" way and set it up from sourceforge.

- Try out SMW+ online
- Download commercial or community edition (Halo)
- Get professional support, training or consulting for your wiki

Become part of the lively SMW+ community. Get in touch with SMW+ users or discuss with our developers and learn how they apply SMW+. Start here or right away:

- Share your experience and ideas in the discussion forum
- Find manuals and documentation in our help section
- Contribute to the project as developer or user

**Figure 2.3:** SMWForum running Halo 1.5.1

Halo[15] is the name for a suite of extensions to Semantic MediaWiki (SMW)[16]. They provide enhancements to the semantic annotations of SMW as well as other useful features that enhance MediaWiki (MW) and SMW with a focus on use in a business environment. They are mainly being developed by ontoprise[17]. The sponsor of the development activities is Vulcan Inc[18].

In addition to the free of charge Halo extensions ontoprise also offers a commercial version called SMW+. It comes with an installer and virtual machine image for easier installation and updates.

Halo includes the following extensions and tools:

- *Access Control List* is used to protect single pages, articles in categories and namespaces as well as semantic values. It can be used to restrict access to parts of the wiki to specific people or groups.
- *Application Programming* is an extension bundle that provides various functions useful for building applications in the wiki.

[14]http://qt.nokia.com
[15]http://smwforum.ontoprise.com
[16]http://semantic-mediawiki.org
[17]http://ontoprise.com
[18]http://vulcan.com

- *Collaboration* adds commenting on and rating of articles to the wiki.
- *Data Import* allows integration of external data either via import into wiki pages or via queries to Simple Object Access Protocol (SOAP) and Representational State Transfer (REST)ful web services.
- *Deployment Framework* provides easy administration of a wiki installation including installing and updating extensions.
- *Enhanced Retrieval* provides advanced searching capabilities including fuzzy search and spell checking.
- *Halo* supports knowledge authoring as well as browsing and querying of semantic data.
- *Linked Data* is used for importing data from the Web of Data and offering data export via a SPARQL Protocol and RDF Query Language (SPARQL) endpoint and Resource Description Framework (RDF).
- *MS Excel Bridge* allows the import of query results into MS Excel.
- *MS Project Connector* allows the import and export of data from MS Project.
- *Rich Media* enhances uploading and handling of media files and allows their semantic annotations.
- *Rule Knowledge* provides a graphical editor for logical rules.
- *Scriptmanager* provides shared libraries for the other extensions.
- *Semantic Gardening* is a bundle of tools for maintaining the knowledge base.
- *Semantic Notifications* allows monitoring of changes of the semantic data.
- *Semantic Treeview* provides a treeview of wiki elements like categories, pages or semantic data.
- *Triple Store Connector* connects the triple store and wiki.
- *Ultrapedia* is a support extension for the *Ultrapedia*[19] project.
- *User Manual* provides context-sensitive access to parts of the Halo documentation.
- *WikiTags* provides semantic data from the wiki when creating documents in MS Office.
- *WYSIWYG* provides a WhatYouSeeIsWhatYouGet Editor including semantic annotations and editing of templates and queries.

[19]http://wiking.vulcan.com/up

### 2.3.1 History

Halo was started as part of Project Halo[20]. Project Halo started in 2003 with the vision of creating the digital Aristotle, a system knowledgeable in a wide range of topics and able to teach them. It was initiated by Vulcan Inc. Since then it has seen a number of public releases making it more suitable for collaborative work that goes further than what MW alone can provide.

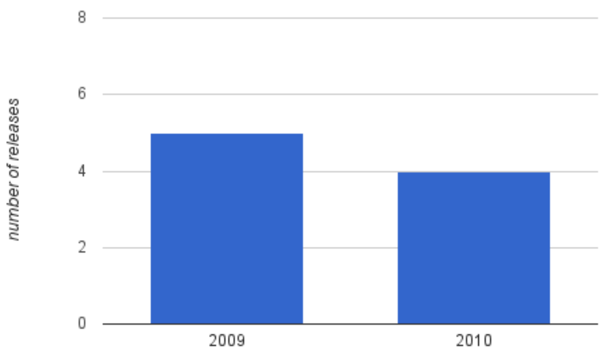


**Figure 2.4:** Number of final Halo releases in 2009 and 2010

Figure 2.4 shows the number of Halo releases per year and Figure A.1 shows a timeline of releases.

### 2.3.2 Goals

Halo aims to provide a collaborative environment where teams can share knowledge. It enriches the unstructured nature of a wiki with semantic annotations (together with SMW) and provides tools to use these annotations in order to make working with the knowledge stored in a wiki easier. It tries to make the semantic web usable for people who are not necessarily experienced with wikis and ontologies. With the ontologies which are provided by default in the commercial version, it is mainly targeting companies that need a platform for collaboration and project management.

### 2.3.3 Culture

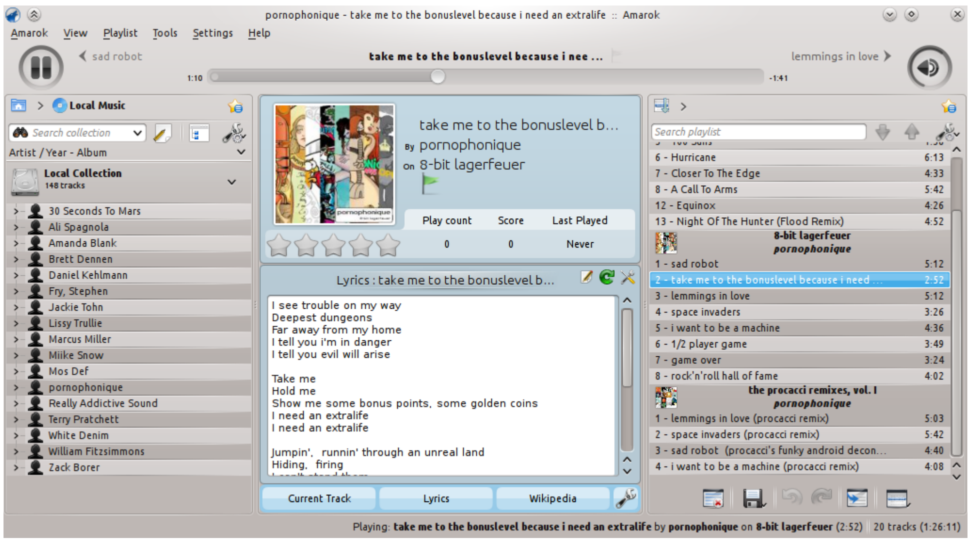
The project is to a large extent driven by the requirements defined by Vulcan Inc. There is a will to integrate other contributions from the community around it but the reality is that that is not happening a lot yet. The atmosphere is very much influenced by the fact that it is a company-driven project.

[20]http://projecthalo.com

### 2.3.4 Structure

The team is divided into a group of developers, a technical leader, a project manager and documentation writers. Requirements are mostly coming from Vulcan Inc. and then distributed in the team. Each developer has a set of extensions they are responsible for throughout the release cycle.

## 2.4 Amarok



30 Seconds To Mars
Ali Spagnola
Amanda Blank
Brett Dennen
Daniel Kehlmann
Fry, Stephen
Jackie Tohn
Lissy Trullie
Marcus Miller
Miike Snow
Mos Def
pornophonique
Really Addictive Sound
Terry Pratchett
White Denim
William Fitzsimmons
Zack Borer

| Play count | Score | Last Played |
|---|---|---|
| 0 | 0 | Never |

I see trouble on my way
Deepest dungeons
Far away from my home
I tell you i'm in danger
I tell you evil will arise

Take me
Hold me
Show me some bonus points, some golden coins
I need an extralife
I need an extralife

Jumpin', runnin' through an unreal land
Hiding, firing

| | |
|---|---|
| 6 - Hurricane | 6:13 |
| 7 - Closer To The Edge | 4:33 |
| 8 - A Call To Arms | 5:42 |
| 12 - Equinox | 4:26 |
| 13 - Night Of The Hunter (Flood Remix) | 4:52 |
| **8-bit lagerfeuer** ***pornophonique*** | |
| 1 - sad robot | 5:12 |
| 2 - take me to the bonuslevel because i need ... | 2:52 |
| 3 - lemmings in love | 5:12 |
| 4 - space invaders | 3:26 |
| 5 - i want to be a machine | 4:36 |
| 6 - 1/2 player game | 3:49 |
| 7 - game over | 3:24 |
| 8 - rock'n'roll hall of fame | 4:02 |
| **the procacci remixes, vol. I** ***pornophonique*** | |
| 1 - lemmings in love (procacci remix) | 5:03 |
| 2 - space invaders (procacci remix) | 5:42 |
| 3 - sad robot (procacci's funky android decon... | 4:40 |
| 4 - i want to be a machine (procacci remix) | 4:08 |

Playing: **take me to the bonuslevel because i need an extralife** by **pornophonique** on **8-bit lagerfeuer** (2:52) | 20 tracks (1:26:11)

**Figure 2.5:** Amarok 2.3.2

Amarok[21] is a Free Software music player developed by members of the KDE community. It is set apart from its competition mainly by the deep integration of internet services and how it helps the user manage and play a large music collection.

Amarok development is entirely volunteer-driven, although one developer has been paid part-time to work on Amarok by Magnatune[22], an artist and customer-friendly music label, and another developer hired by Wurlitzer[23], a well-known jukebox company, to develop a customized Amarok version for a jukebox. The project relies on donations from the user community to pay for infrastructure and event expenses.

Amarok is part of KDE Extragear[24], making it partly independent from the KDE Software Compilation (SC)[25]. It does not follow its release schedule for example. It does, however, use a lot of KDE's infrastructure, including bug tracker, version

[21]http://amarok.kde.org
[22]http://magnatune.com
[23]http://deutsche-wurlitzer.de
[24]A module for programs that are partly independent from the main KDE programs
[25]KDE's compilation of core programs that is currently being released approximately every 6 months

control system, translation infrastructure, mailing lists and forum. It does not share KDE's Internet Relay Chat (IRC) channels and website. Wikis[26] and legal foundation[27] are partly shared.

### 2.4.1 History

Mark Kretschmann says that he “started Amarok in 2002 because XMMS had usability problems”. Since then it has seen tremendous growth and adoption among Linux users. It has repeatedly won prizes in the Linux community. Among them are LinuxQuestions.org Members Choice Awards for 2005, 2006, 2007, 2008 and 2009 [Gar06, Gar07, Gar08, Gar09, Gar10] and Linux Journal Readers' Choice Awards in 2008, 2009 and 2010 [Gra08, Gra09b, Gra10].

With KDE switching to Qt 4[28] for its software, radical changes were made to the whole SC. This brought a lot of opportunities for improvements and innovation like the whole Plasma[29] stack as well as the social semantic desktop. Amarok went through a similar transition from Amarok 1 to Amarok 2 during that time. It included a rewrite of large parts of the underlying code-base as well as invasive changes to the user interface (UI). The changes were made to make the program more maintainable and to refocus on the core mission of the project. Both KDE and Amarok endured public backlash due to the adaptation required from users but are regaining popularity and are thriving after the needed clean-up and refocusing [Sei10].

Amarok was started when KDE was using Concurrent Versions System (CVS) as the version control system, then migrated to Apache Subversion (SVN) with the rest of KDE's software in 2005 and was one of the first KDE projects to move to git[30] in 2009. Figure A.2 and Figure A.3 show a timeline with important milestones in Amarok's history. Since the beginning of the project, six new versions on average have been released per year (Figure 2.6).

### 2.4.2 Goals

The goal for Amarok is to develop a program that can help its users to rediscover their music – leading to its tagline and motto “Rediscover Your Music”. These days users tend to have many thousands of music files scattered on different hard drives. On top of that local music files seem to become less and less important with the presence of always available web radios and web services like Last.fm[31],

[26]The Amarok project uses one of KDE's wikis, UserBase, for its end-user documentation.
[27]KDE e.V. and the Software Freedom Conservancy on behalf of Amarok share project expenses.
[28]A cross-platform development toolkit for C++
[29]http://userbase.kde.org/Plasma
[30]Distributed version control system originally developed for the Linux kernel
[31]http://last.fm

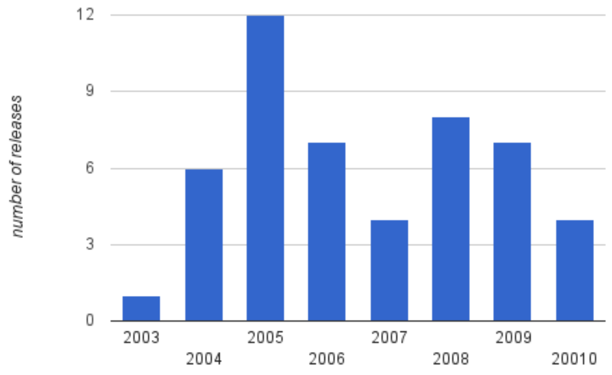


**Figure 2.6:** Number of final Amarok releases between 2003 and 2010

Pandora[32] and Co [Fra10] or the user might even set up their own music server with Ampache[33] or similar programs. It becomes increasingly important to offer help with finding the music one is looking for easily as well as rediscovering music that is getting lost in the large amount of available music. The other way that Amarok helps to rediscover music is through offering useful related information to a given song and making it easily accessible in a uniform way. This includes album art, lyrics as well as songs by the same or related artists. All of this is shown in applets in the UI removing the need to search for this information on different websites in a web browser for example.

### 2.4.3 Culture

The project is friendly and relies heavily on contributors working together to continuously make new releases.

The day-to-day operations and communication take place mainly on IRC and mailing lists. While no high-impact decisions are solely taken on IRC, because it is important that people not online at that point have a say in the decision, IRC is extremely important for the management of the project as well as creating a strong bond in the team. Some team members regularly meet at trade shows, development sprints and events such as KDE's annual conference Akademy[34]. The importance of these meetings for the team can not be emphasized highly enough. They help build friendships and motivation and turn drive-by contributors[35] into long-time community members. "I came for the code and stayed for the community." and "I found life-long friends here." are two commonly heard sentences.

[32]http://pandora.com
[33]http://ampache.org
[34]http://akademy.kde.org
[35]Contributors who only make a small number of contributions over a short time period to a project

### 2.4.4 Structure

Amarok has a flat team structure. There are only a few official titles: author, community manager, release manager and member of the Amarok Committee. It is a meritocracy like many Free Software projects, meaning titles and responsibilities are given based on demonstrated merit. Decisions are based on reaching consensus, and as in the rest of KDE, made pragmatically. Everyone can voice their opinion. Higher priority is given to those who contribute more than just opinion, i.e., "who does the work decides"[36]. It is comparatively easy to get involved and with enough dedication it is possible to get into the core team very quickly.

Contributions are recognised by adding the person's name to the about dialog of the program. It is split into several sections - Authors, Contributors and Donors. Contributors get awarded author status for long-time commitment to the project after recommendation by one of the existing authors and approval by the other authors. The second section contains names of people who have made non-minor contributions at some point in the lifetime of the project. Names are added based on recommendation by anyone in the community. The last section contains names of donors that get added for donating during Roktober, the annual fundraiser of the project.

The Amarok Committee is a body of five long-time contributors who oversee the financial operations of the project like approving travel expense reimbursements. It does not make decisions related to the development of the software just like KDE e.V.'s[37] board, which it is modelled after.

Day-to-day operations are supported by Rokymotion, the non-developer part of the team. Promotion, bug triage, end-user support and infrastructure maintenance are among their tasks but their main focus is on promotion.

## 2.5 MediaWiki

MW is a wiki software developed by the MW community and its supporting foundation. It is written in PHP and allows the user to easily collaborate on the web without needing to know Hypertext Markup Language (HTML). Instead, it uses a simple mark-up language. The low barrier to entry is one of the key factors to the success of Wikipedia, one of the most visited websites and biggest deployment of MW.

The success of the MW community is also enabled by making it easy to customize MW with extensions. This allows the growth of an eco-system around MW without potentially having to compromise the core of the project.

[36]One of KDE's tenets
[37]http://ev.kde.org

```
{|
|-
! Header 1
! Header 2
|-
| Cell 1
| Cell 2
|-
| Cell 3
| cell 4
|}
```

**Listing 2.1:** Example of wiki text mark-up for a simple 2x3 table

For the implementation in Chapter 6, two parts of the MW mark-up are important: tables and templates. An example of a simple table is shown in Listing 2.1. Templates are text snippets that are needed repeatedly across a wiki. Each of them has a page in the Template namespace. They are used by adding {{TemplateName}} in the wiki mark-up. Optionally they can be called with arguments using {{TemplateName|Argument1|Argument2}}. In the actual template these arguments can then be used by using {{{1}}}, {{{2}}} and so on.

## 2.6 Semantic MediaWiki

SMW is an extension to MW which allows semantic annotations of the wiki text as well as work with the semantic knowledge then contained in the wiki. This allows easy searching, aggregating, querying and more of the data in the wiki.

SMW was started at the University of Karlsruhe by Markus Krötzsch and Denny Vrandecic. They released the first version in 2005. Since then it has grown, seen over 20 releases and gained a significant contributor base, indicating a healthy project. During that time a number of extensions, including Halo, Semantic Maps[38] and Semantic Forms[39] have been build around SMW by volunteers and companies.

Most of the community activity happens on the user and developer mailing lists but there is also an active IRC channel.

For the implementation in Chapter 6, two things are important: queries and web services. The semantic data in the wiki can be queried and the result displayed in different ways. A simple example can be seen in Listing 2.2. Web services are used to display and use data from external sources in the wiki. External sources can be Really Simple Syndication (RSS) feeds, Extensible Markup Language (XML) files, websites offering a RESTful application programming interface (API) and more.

[38]http://mediawiki.org/wiki/Extension:Semantic_Maps
[39]http://mediawiki.org/wiki/Extension:Semantic_Forms

```
{{#ask: [[Category:Person]]
[[Affiliated with::ontoprise GmbH]]
| format=table
| headers=hide
| link=none
| order=ascending
|}}
```

**Listing 2.2:** Example for a query for everything in the category ”Person” that has the property ”Affiliated with” set to ”ontoprise GmbH”. The result will be returned as a table

## 2.7 KDE

KDE is a Free Software project that started out with the goal of creating a user friendly desktop environment for Linux. Matthias Ettrich started it in 1996 using C++ and Qt. Today it is one of the largest Free Software communities, creating a desktop environment and applications for it. Recently there have also been efforts to reach out to other form factors like netbooks and mobile devices. KDE’s software can be divided into KDE SC, which is the main desktop product plus some applications, and Extragear. Extragear is a place for programs that want to keep a certain degree of independence from the main module. It includes some of the community’s flagship applications like Amarok, digiKam[40] and K3b[41].

The project is almost entirely run by volunteers but there are companies supporting it and employing a large number of contributors. These include for example Nokia, Novell and Google. KDE is supported by the KDE e.V. in legal and administrative matters. The KDE e.V., however, explicitly does not influence development.

Communication and coordination largely take place on IRC and mailing lists, as well as at the annual conference Akademy and other smaller local events.

[40]http://digikam.org
[41]http://k3b.org

*"Alone we can do so little; together we can do so much."*

Helen Keller

# 3 Related Work

In this chapter, relevant related work is explored. It is input for the design of the new development process in Chapter 5.

The reviewed literature seemed to have a very strong focus on the developers in Free Software projects and often did not recognise the importance of non-coding contributions. Reasons for this might be the difficulty in obtaining data about non-developer contributors or the view that writing code is the most important work in such a project and should therefore be focused on. This thesis attempts a more comprehensive view of the contributor team.

## 3.1 Mapping of development processes

Capability Maturity Model Integration (CMMI) is a model for process improvement. It comes in three different versions, one for development, one for services and one for acquisition. In its current version 1.3 it lists 22 process areas in the development version [CMM10]. Some of them will be used to guide the analysis in Chapter 4: Casual Analysis and Resolution, Decision Analysis and Resolution, Integrated Project Management, Measurement and Analysis, Organisational Innovation and Deployment, Organizational Process Definition, Organizational Process Focus, Project Monitoring and Control, Project Planning, Process and Product Quality Assurance, Quantitative Project Management, Requirements Development, Requirements Management, Technical Solution. CMMI is accompanied by an appraisal process named Standard CMMI Appraisal Method for Process

Improvement (SCAMPI) [SCA06]. Organisations can be evaluated and assigned a maturity level from 2 to 5 based on the evaluation of their processes. It demands the use of interviews and document review as evidence for the evaluation. The analysis of the development processes in Chapter 4 will be following the suggested interview guidelines of SCAMPI. They include not interviewing two people in the same reporting chain together, preparing questions to guide the interview and communicating the interview schedule in advance.

Six Sigma is a business management strategy and associated tools. While it has received criticism for focusing too much on incremental improvements and not encouraging blue-sky ideas among other things [Hin07, Mor06], it does propose one tool that is useful for this thesis: Suppliers, Inputs, Process, Outputs, Customers (SIPOC). SIPOC provides a guideline for gathering data about a process (see Table 3.1). It will be used in Chapter 4 to give an overview of the most important processes. CMMI also provides a criterion for a defined process. It has to state purpose, inputs, entry criteria, activities, roles, measures, verification steps, outputs and exit criteria [CMM10]. For reasons of simplicity and scope we will use SIPOC.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| users, management | requirements | writing software | code | users |
| ... | ... | ... | ... | ... |

**Table 3.1:** Example table for SIPOC

## 3.2 Communication and coordination in distributed teams

Yamauchi et al. express a fundamentally important statement: “Open-source programmers are biased towards action rather than coordination” [YYSI00]. They tend to try out ideas without necessarily getting input from other contributors first. They reason, 'After all it might not work out,' or it is a controversial feature or change, in which case discussion is easier when there is code to look at and judge, among other motivations. The Linux kernel mailing list rules, for example, even explicitly state: “A line of code is worth a thousand words. If you think of a new feature, implement it first, then post to the list for comments.[1]” While this is extreme and a lot of other Free Software projects avoid this mentality at all costs it is a constant struggle to balance this bias towards action and the need for coordination between team members.

Wikis are used a lot in Free Software teams to coordinate work. Some of them are more successful than others. Based on a case study of three wikis by Wagner and Majchrazak [WM07], Choate identified three important factors for the successful

[1]http://kernel.org/pub/linux/docs/lkml

use of a wiki as a collaborative tool [Cho07]: alignment of goals, a culture of collaboration and community custodianship.

Espinosa and Carmel identified components that factor into the cost of time-separated teamwork [EC03]. The first factor is of course the cost required to execute a given task. In time-separated teams they add to that the *communication* overhead that is required, the *delay* that is happening because one person is waiting on another, *clarification*, meaning the additional communication and delay because of misunderstandings and *rework*, the additional production cost because of misunderstandings. They also say that some of these can be helped by better using the time when synchronous communication can happen, moving to more asynchronous communication and using tools that support it and educating team members about the implications of working with others over a time-distance. The latter can include making it easy to see what time it currently is at the other team member's location for example.

Espinosa et al. investigated team knowledge in geographically distributed teams in comparison with collocated teams [ESKH07]. They found that "task awareness[2] and presence awareness[3] help team coordination in software projects". They go on and say that while task awareness is important in collocated teams it is even more important in geographically distributed teams because it is harder to find out who has done or not done a particular task. Many projects mitigate a part of this through sophisticated collaboration tools. According to their findings, task awareness is more important for a collocated team and knowledge of the team[4] is more important for distributed teams. This might be because the collocated team already has the knowledge about the team and is more focused on the status of tasks. Task and presence awareness are part of what Hinds and Bailey call a "shared context" [HB03]. They cite a lack of shared context as one reason for conflicts in distributed teams. They propose a number of possible measures to prevent conflicts caused by distance between team members: i) regular face-to-face meetings to increase shared context, ii) purposefully conveying contextual information like schedules either manually or automatically iii) creating similar contexts at different locations[5], iv) adapting to the tools that are available over time and v) better adapting the tools that are available to the team's needs.

Holck and Jørgensen stress the importance of continuous integration in distributed development teams, meaning new features and bug fixes get into the source code repository constantly and tested quickly to reduce the time between the occurrence of a problem and someone noticing it [HJ07]. They also point out that both projects they reviewed (Mozilla and Apache) have guidelines which state that working on new features or bug fixes should be announced to avoid duplication of work and conflicting changes as well as work on changes that the community

[2]The knowledge of what is going on with an area of a task that is of concern for a team member

[3]The knowledge of the location and status of a team member

[4]e.g., Who is who? Who is an expert on a given topic? Who is responsible for something?

[5]This could include things like office policies, work environment or tools.

does not consider improvements for the program. This happens, for example, in Bugzilla tickets. To facilitate this further both projects have a notion of code-ownership to make it clear who is ultimately in charge of a given part of the code-base. Mozilla has an especially high focus on peer review, only allowing changes into the repository after at least one review and some changes only after super-review where the change's impact on the overall system is examined. A stabilization phase where new features are not allowed and development focuses on bug fixing only exists in both cases. In their conclusion Holck and Jørgensen suggest that continuous integration replaces many of the traditional engineering practices like plans and design documents to some degree.

## 3.3 Collaboratively working on a vision

There are various ideas of what constitutes the vision or mission of an organisation. Raynor, for example, defines a mission as "a concise statement of the customers and core competencies of the organization; in other words, the arena of competition for the organization and those characteristics of the profession that will allow it to perform successfully in that arena" and a vision as "a statement of the desired future state of the organization within the arena of competition defined in the mission" [Ray98]. He also created a framework that shows what influences and what is influenced by a vision and mission (Figure 3.1). It helps to structure and focus the discussions during the process of developing or refining a vision statement.

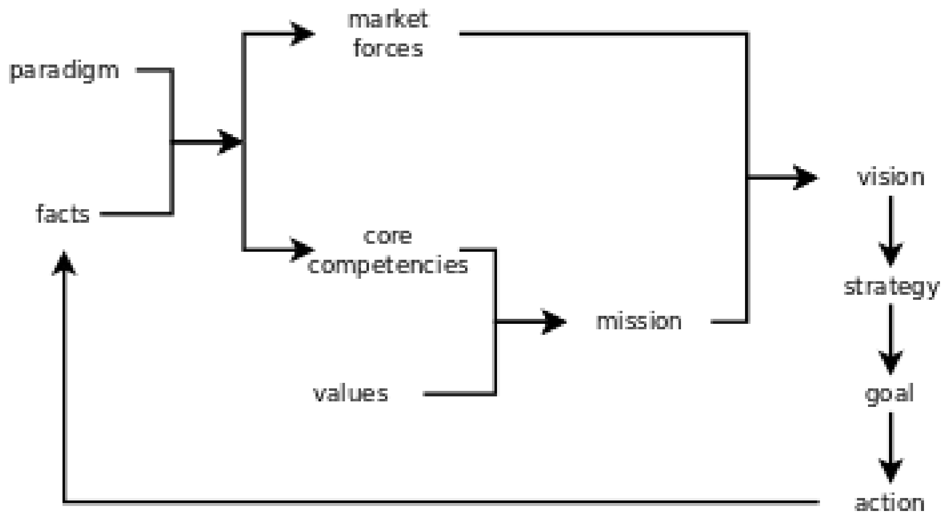


**Figure 3.1:** Raynor's mission framework [Ray98]

A vision as well as a mission statement are supposed to help focus on what is really important to a group of people. Without them it is easy for a group to lose sight of long-term goals. Short term actions might even damage long-term success if the team is not aware of where they are heading.

Levin sees the vision as an elaborate story of the future in contrast to conventional vision statements that are shorter and less elaborate [Lev00]. He claims that to really have an impact a short vision statement is not enough. It needs to be filled with a story people can relate to. To create this story he proposes four steps and lists a set of questions for each of them to facilitate them: i) becoming informed, meaning getting to know the organisation and its strengths and weaknesses as well as hopes and fears for the future, ii) visiting the future and recording the experience, meaning exploring what the favourable future of the organisation could look like, iii) creating the story, meaning the act of taking the information from the previous steps and writing them into a captivating story and iv) deploying the vision, meaning the dissemination of the vision among all stakeholders.

Lipton identified five reasons why a vision is important for an organisation or business [Lip96]:

1. A vision enhances a wide range of performance measures. (Studies showed that companies with a vision are performing better financially.)
2. A vision promotes change. (Lipton sees a vision as a vehicle for needed changes in an organisation.)
3. A vision provides the basis for a strategic plan.
4. A vision motivates individuals and facilitates the recruitment of talent.
5. A vision helps keep decision making in context. (Lipton stresses the necessity of an organisation knowing what it is doing and what it is not doing.)

Tarnow point out the power a vision statement can have when constructed as a Unifying Action Declaration (UAD) [Tar97]. When it is written as one it can lead to a concept called social categorization, that tells individuals who is included in or excluded from a group. A UAD according to him needs to suggests an action, identify this action only vaguely and include a social categorisation, which fits well with a mission statement.

Lipton lists three ingredients for a successful vision [Lip96]:

1. the mission or purpose of the organisation or business – the answer to the question why the organisation or business exists. What differentiates it from others in its field?
2. the strategy to achieve the mission
3. other elements of the culture of the organisation that seem necessary

Reasons why a vision fails, according to Lipton, are:

1. failure of management to live up to it and acting in disagreement with it

2. irrelevance for those who are affected by it
3. expecting it to solve all problems
4. too narrow scope or focus on the past instead of the future
5. the envisioned future is not grounded in reality
6. being too abstract or too concrete
7. lack of a creative process during its writing
8. poor management of participation during its writing
9. complacency – not realizing that it does take effort to make it reality

Based on this, all reviewed literature seems to agree that the key step during the process of creating a vision statement is to involve all stakeholders or a good representation of each group of them early in the process and have them participate in the creation of what is to guide them for the foreseeable future.

There are a few great examples of how Free Software projects create and communicate their mission or vision. **Mozilla**, the creators of Firefox, define their mission as "to promote openness, innovation and opportunity on the web"[6]. However, they communicate it slightly differently on the main page of their website[7]: "We Believe in an Open Web and we're dedicated to keeping it free, open and accessible to all." and then link to their mission statement and manifesto. The reason for this is likely that this second version is more captivating and encourages the reader to read more about Mozilla's work. **Fedora**'s mission statement[8] says: "The Fedora Project's mission is to lead the advancement of free and open source software and content as a collaborative community." The noteworthy part about this is that it does not mention Fedora's users. It focuses on driving innovation and this is exactly what Fedora does. This focus on innovation sometimes hurts casual users because of bugs and other problems using bleeding-edge software brings with it but at the same time it is very attractive for users who always want to try out the latest releases and innovations on the Linux desktop. In addition Fedora has four "Foundations"[9] (Freedom, Friends, Features, First), that are the core values of the project, and a recently created vision statement ("The Fedora Project creates a world where free culture is welcoming and widespread, collaboration is commonplace, and people control their content and devices."[10]). According to a Fedora board member the vision statement was driven by the board who gathered input from the community via blogs, mailing lists and social networks. **openSUSE** in contrast to Fedora's mission statement is working on a mission statement right now

[6]http://mozilla.org/about/mission.html
[7]http://mozilla.org
[8]http://fedoraproject.org/wiki/Overview
[9]http://fedoraproject.org/wiki/Foundations
[10]http://fedoraproject.org/wiki/Vision_statement

that will likely read: "The openSUSE project is a worldwide effort that promotes the use of Linux everywhere. The openSUSE community develops and maintains a packaging and distribution infrastructure which provides the foundation for the world's most flexible and powerful Linux distribution. Our community works together in an open, transparent and friendly manner as part of the global Free and Open Source Software community." In the first sentence it expresses a focus that is much more towards a very large userbase, something that is evidently not Fedora's main goal. The more elaborate strategy document for openSUSE makes another very important point at the start: "The following document is a statement describing the openSUSE users, community, products and goals. The document is for internal use and guides communication and decision making within the community. It does not aim to limit anyone within the community to work on what they want!" They do not want the mission to be used to tell someone that they can not work on something they want to do. It is a fine line between trying to make a team of volunteers work towards the same goal and driving them away by telling them they can not work on something they are passionate about. The mission statement was created together with a strategy document with the whole community after a previous attempt at creating such a document[11] in a smaller group and then publishing it for comments by the wider community had failed [Poo10]. An openSUSE contributor about the reason for the failure of the first attempt: "The first attempt went too deep, technical and corporate to get community buy-in – most people simply didn't see themselves in it." For writing the new document a collaborative platform called co-ment[12] was used. Everyone was able to give input and comment on the existing text. Comments were taken into consideration and the text changed accordingly where needed. In the end six iterations were done this way. Currently the text is under review by the board and will likely be voted on soon. The mission of the **Wikimedia Foundation** is "to empower and engage people around the world to collect and develop educational content under a free license or in the public domain, and to disseminate it effectively and globally."[13] In addition the Wikimedia Foundation also has a vision statement: "Imagine a world in which every single human being can freely share in the sum of all knowledge. That's our commitment."[14] A draft for both was written by the board of the foundation and then sent to the foundation mailing list for comments, edits and following that a vote for approval [Moe06]. There seems to have been some unhappiness with the process but overall it worked out. For both the mission and vision the foundation offers a separate wiki page for proposing changes that will be reviewed at least annually. This offers an easy way to propose changes in case the mission or vision statement no longer reflects what the community wants the foundation to be and do.

[11]http://en.opensuse.org/Portal:Strategy
[12]http://co-ment.com
[13]http://wikimediafoundation.org/wiki/Mission
[14]http://wikimediafoundation.org/wiki/Vision

Despite compelling reasons for having a vision or mission statement many companies or organisations do not. David identified a fear of the controversies that might arise during the development of it [Dav89]. The act of formulating it might reveal profound differences in the understanding of what the company or organisation is and where it should go in the future. These differences should be discussed as they might otherwise influence other decisions negatively. The other reason he mentions is that management is too focused on administrative and tactical tasks and does not pay enough attention to the clarification of the strategy behind them. Ireland and Hirc add a few more possible reasons for failure to create a mission statement [IH92]. The most relevant ones for this thesis are: i) the number of stakeholders that would have to be involved ii) the work it would require, iii) being comfortable with the status quo and iv) fear that the mission statement might provide important information to competitors.

## 3.4 Collaboratively creating a roadmap

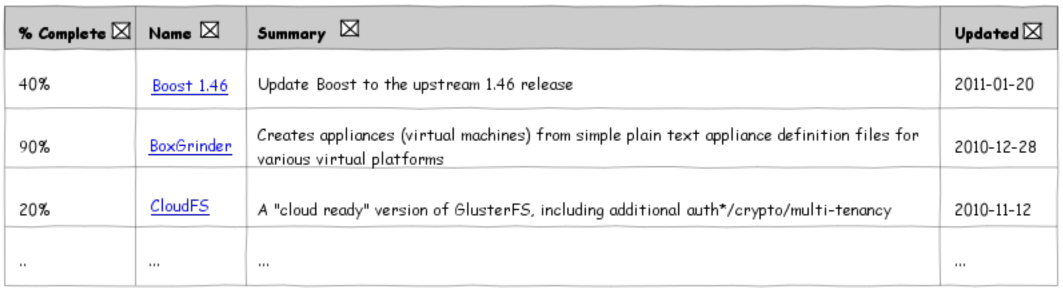
Many projects use wikis for their roadmaps. They seem to focus on features in them. A few of these roadmaps have been selected for closer examination here.

In its feature overview for Fedora 15, the Fedora community gives the name of the feature, a short summary, an indication of how much of a feature is done and when the status was last updated (Figure 3.2[15]). The table is maintained in Fedora's project wiki and features are added after approval by the Fedora Engineering Steering Committee (FESCo). The individual features link to a wiki page with more details (Figure 3.3[16]). Features that have not yet been approved for a release are also collected in the wiki and added to the FeaturePageIncomplete category for future consideration. Once the specification is complete it is marked as ready for review by the FeatureWrangler[17] by moving it to another category named FeatureReadyForWrangler. After his review for completeness he hands it over to FESCo for consideration for the next release by changing its category to FeatureReadyForFesco. FESCo votes on the features for inclusion in the next release and moves the approved ones to the category FeatureAcceptedFX [Fed].

The roadmap for UbuntuOne[18] is a wiki page with a list of high-level items. It gives only a short explanation for the features and does not link to any further information sources. On top of that it is outdated and lists features for a release that was done months ago. There is no indication of the progress on each of the planned features.

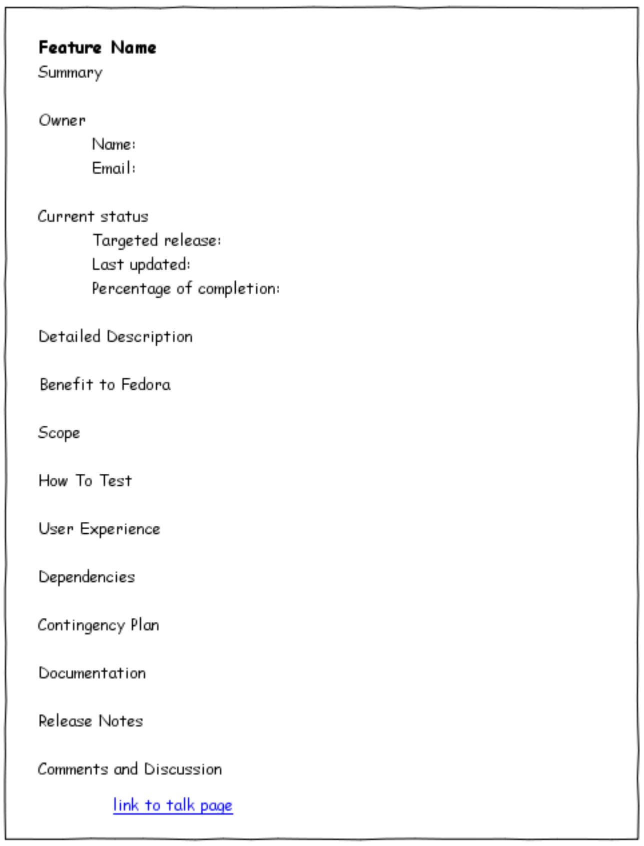
Apache Harmony has a high-level road map[19] that lists major milestones until 2008. Completed tasks have been struck through and prepended with DONE.

[15]http://fedoraproject.org/wiki/Releases/15/FeatureList
[16]http://fedoraproject.org/wiki/Features/EmptyTemplate
[17]Person appointed by FESCo to triage feature requests.
[18]https://wiki.ubuntu.com/UbuntuOne/Roadmap
[19]http://harmony.apache.org/roadmap.html



| % Complete ☒ | Name ☒ | Summary ☒ | Updated ☒ |
|---|---|---|---|
| 40% | Boost 1.46 | Update Boost to the upstream 1.46 release | 2011-01-20 |
| 90% | BoxGrinder | Creates appliances (virtual machines) from simple plain text appliance definition files for various virtual platforms | 2010-12-28 |
| 20% | CloudFS | A "cloud ready" version of GlusterFS, including additional auth*/crypto/multi-tenancy | 2010-11-12 |
| .. | ... | ... | ... |

**Figure 3.2:** Layout of the roadmap for Fedora 15



**Feature Name**
Summary

Owner
Name:
Email:

Current status
Targeted release:
Last updated:
Percentage of completion:

Detailed Description

Benefit to Fedora

Scope

How To Test

User Experience

Dependencies

Contingency Plan

Documentation

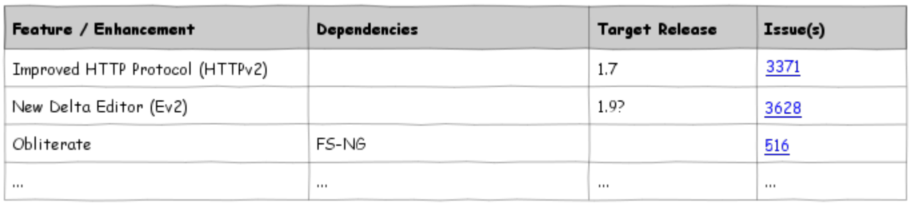
Release Notes

Comments and Discussion
link to talk page

**Figure 3.3:** Layout of the wiki template for new feature requests for Fedora

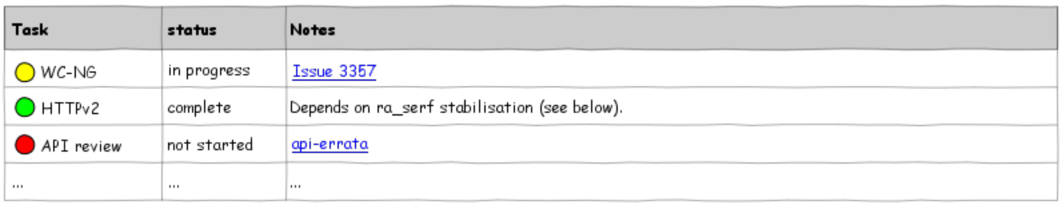
Apache Subversion (SVN) takes a different approach for their roadmap. They first show a table of the next planned releases, the most important deliverables with links to bug tracker entries and the quarter they are scheduled for. Then they offer a table of their most-wanted features including their dependencies, links to bug entries and the release they are targeted for. It can be seen in Figure 3.4[20]. Last they offer a more in-depth list of the features planned for the next major release. It contains a red, yellow, orange or green circle indicating the status (not started, in progress, explored and finished/deferred respectively), the name of the task, its status in words and notes like bug numbers. Figure 3.5 shows its layout.

[20]http://subversion.apache.org/roadmap.html



| Feature / Enhancement | Dependencies | Target Release | Issue(s) |
|---|---|---|---|
| Improved HTTP Protocol (HTTPv2) | | 1.7 | 3371 |
| New Delta Editor (Ev2) | | 1.9? | 3628 |
| Obliterate | FS-NG | | 516 |
| ... | ... | ... | ... |

**Figure 3.4:** Layout of table showing most wanted features for SVN



| Task | status | Notes |
|---|---|---|
| 🟡 WC-NG | in progress | Issue 3357 |
| 🟢 HTTPv2 | complete | Depends on ra_serf stabilisation (see below). |
| 🔴 API review | not started | api-errata |
| ... | ... | ... |

**Figure 3.5:** Layout of roadmap for SVN

Tracker's roadmap wiki page[21] first lists a number of issues and plans for a future release marked 0.x, likely indicating that they want to work on it for one of the not-too-far in the future releases but do not know which one of them yet. Most of the items do not have a link to further information. Below this list is the list for the release currently being worked on. Some items have links to Bugzilla entries or git branches. Completed items are marked with [DONE] or similar. Lists for past releases can be found below that.

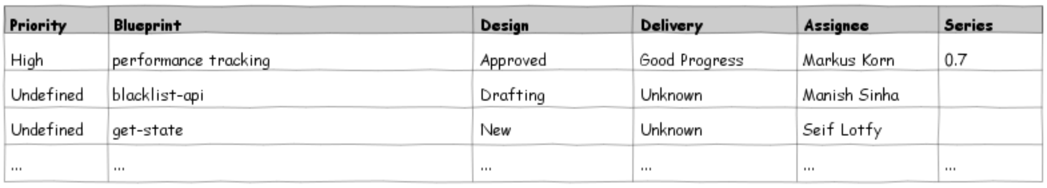
Plasma's roadmap wiki page[22] is more a brainstorming page for the next two releases. It divides ideas and todos sorted first by release and then the form factor the idea concerns (e.g., Plasma Desktop, Plasma Mobile). Each item is very short and for someone, who is not a member of the team, it is likely not clear what many of them mean.

Except for Fedora, in most of these cases it is not well communicated how a user or potential contributor can influence these roadmaps, how they can express that they think something should have priority for the next release or how they can help with making some of these proposals reality. Neary proposes to add and clearly mark items to a project's roadmap that are not going to be done by the core team to give direction to new contributors [Nea11].

Launchpad[23] is a development platform that was originally started for the Ubuntu project and now hosts nearly 22 000 projects. It provides a lot of the infrastructure a Free Software team needs including bug tracking, version control for source code, translations, an area for questions and blueprints, which is their name for feature specifications/tracking. Each person, team or project in Launchpad can have blueprints and has an overview of those related to them. Figure 3.6 shows the

[21]http://live.gnome.org/Tracker/Roadmap
[22]http://community.kde.org/Plasma/2011
[23]http://launchpad.net

layout of the blueprint overview table for the Zeitgeist project[24]. It lists priority, name, design (the status of the specification itself), delivery (the status/progress of the specified feature), assignee and (release-)series. In addition to this each blueprint has an individual page with additional information. Figure 3.7 shows the layout of the page for a blueprint for the Zeitgeist project[25]. It contains a description of the feature/task, information about the status and responsible people or teams, related code branches and bugs, which sprint it is a part of, who was asked to review the blueprint and a whiteboard area for free-text information. Canonical has long been criticised for not releasing the source code of the platform they are developing Free Software on and are asking other projects to use as well. By now Launchpad is Open Source but it is still hard to deploy your own instance because of licensing restrictions and lack of documentation.

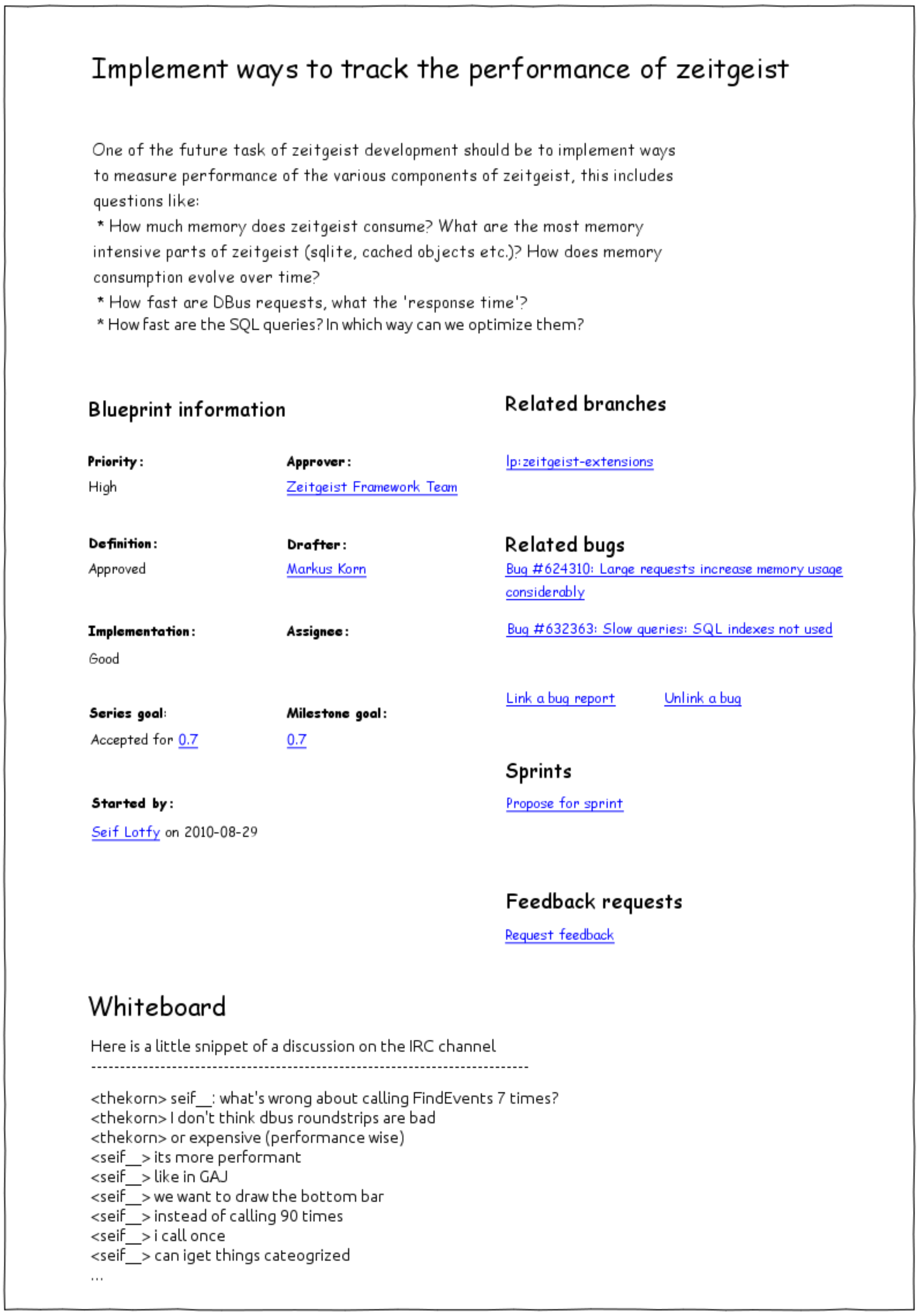


| Priority | Blueprint | Design | Delivery | Assignee | Series |
|---|---|---|---|---|---|
| High | performance tracking | Approved | Good Progress | Markus Korn | 0.7 |
| Undefined | blacklist-api | Drafting | Unknown | Manish Sinha | |
| Undefined | get-state | New | Unknown | Seif Lotfy | |
| ... | ... | ... | ... | ... | ... |

**Figure 3.6:** Layout of a blueprint overview table for a project in Launchpad

In contrast to this idea collecting and tracking, Fried et al. of 37signals advocate not caring about feature requests at all beyond reading them [FHL09]. Their rationale is that with enough users you do not need to keep a list of them. Users will keep reminding you again and again about what they want. This is likely true for the range of commercial proprietary web applications they are developing. This approach has several problems for Free Software development. The biggest one is probably that the whole process is not transparent beyond the small developer team. Users are not able to see what is planned and more importantly what is not planned. For a project that has to work with a shortage of human resources it also does not scale to deal with similar requests more often than absolutely necessary. 37signals, developing proprietary web applications, also does not have an interest in recruiting volunteers to support them. As a counter point, Krogh et al. researched the behaviour of people joining an existing Free Software project (Freenet) [vKSL03]. They found patterns that can predict if someone new to a mailing list will actually produce something of value for the project. One of their findings is that people tend to "lurk"[26] for weeks or even months in order to understand how the project works before becoming active. One of the criteria they identified as critical in the first email a newcomer sends to the mailing list is if that person is asking for a task to work on and the responses he receives. Only very few were given concrete task suggestions. Having a roadmap will make it easier to point them to possible tasks. The author's own experience from the

[24]http://blueprints.launchpad.net/zeitgeist
[25]http://blueprints.launchpad.net/zeitgeist/+spec/performance-tracking
[26]Reading a mailing list or IRC channel without actively participating



Implement ways to track the performance of zeitgeist

One of the future task of zeitgeist development should be to implement ways
to measure performance of the various components of zeitgeist, this includes
questions like:
* How much memory does zeitgeist consume? What are the most memory
intensive parts of zeitgeist (sqlite, cached objects etc.)? How does memory
consumption evolve over time?
* How fast are DBus requests, what the 'response time'?
* How fast are the SQL queries? In which way can we optimize them?

Blueprint information

| | |
|---|---|
| **Priority:** High | **Approver:** Zeitgeist Framework Team |
| **Definition:** Approved | **Drafter:** Markus Korn |
| **Implementation:** Good | **Assignee:** |
| **Series goal:** Accepted for 0.7 | **Milestone goal:** 0.7 |
| **Started by:** Seif Lotfy on 2010-08-29 | |

Related branches

lp:zeitgeist-extensions

Related bugs

Bug #624310: Large requests increase memory usage considerably

Bug #632363: Slow queries: SQL indexes not used

Link a bug report Unlink a bug

Sprints

Propose for sprint

Feedback requests

Request feedback

Whiteboard

Here is a little snippet of a discussion on the IRC channel
```
------------------------------------------------------------------

<thekorn> seif__: what's wrong about calling FindEvents 7 times?
<thekorn> I don't think dbus roundstrips are bad
<thekorn> or expensive (performance wise)
<seif__> its more performant
<seif__> like in GAJ
<seif__> we want to draw the bottom bar
<seif__> instead of calling 90 times
<seif__> i call once
<seif__> can iget things cateogrized
...
```

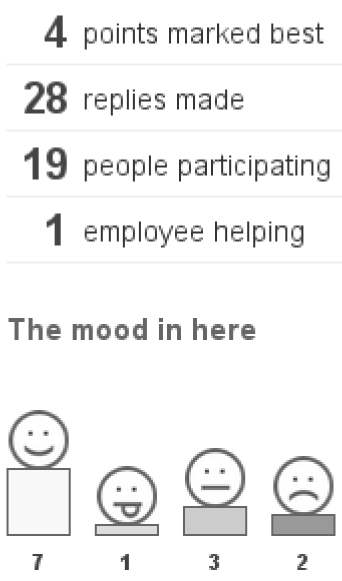
**Figure 3.7:** Layout of a blueprint details page for a project in Launchpad

KDE project [Pin10] also shows that keeping contributors engaged after their first task is an important part of turning them into long-term contributors. Krogh et al. also found a pattern they call feature gifts. It describes the case where a new contributor starts his work in a project with the announcement of a new feature. This is often appreciated by the team but sometimes not in line with what they had in mind for their product. A roadmap might facilitate discussions around such feature gifts before work on them is started and thereby prevent disappointment on the side of the new contributor because of wasted time and effort.

To let users influence the development process it might be useful to allow voting on features. Winkelmann et al. reviewed rating systems of some of the most popular websites [WHP+09]. They identified characteristics that help classify rating systems. They highlight the need for the input of ratings to be as easy as possible. The systems they reviewed largely required some sort of sign-in to improve the quality but they acknowledged that for some systems even that might be too high a barrier to entry. They also differentiate between different sorts of vote aggregation – for example showing just an average or cumulative score or showing a distribution of votes. In addition they distinguish between websites where ratings need to be approved and those where they are publicly visible immediately. They also highlight the fact that some websites offered incentives for ratings as the "rating provides value for the consumer rather than the evaluator". In the case of rating feature requests in Free Software this is however largely not the case. They point out the value of profile building through ratings as one possible incentive. This can be applied in a Free Software project.

Get Satisfaction[27] is a customer feedback platform that offers issue tracking, a place to ask questions and give positive feedback and a forum to propose ideas for companies wanting to engage with their customers. There are several things to note about Get Satisfaction. It clearly shows the degree of engagement of employees of a given companies with the community there. Some companies are not active at all (e.g., SAP, Debian) and it is just a platform for users to help other users, some companies only read (e.g., Facebook, Nokia) and other companies actively reply to issues and respond to requests (e.g., Diaspora, Intel). This sets the right expectation for anyone participating there and helps prevent disappointment caused by wrong expectations. In addition they show how many people participated in a given topic and how many replied and if an official representative of the company has participated (Figure 3.8). (Given the fluid nature of a Free Software community this official status can be hard to determine for anyone outside the core team.) Each answer can be marked as "best answer" and is then prominently shown at the top of the discussion to make it easier to see the right and helpful answers quickly. Each topic additionally has a mood indicator. When opening a topic or answering one Get Satisfaction offers four different smilies to indicate the poster's mood. These are then aggregated in a small bar chart representing the mood of the topic (Figure 3.8). Last but not least they make it very easy and obvious to indicate you are interested in a topic or affected by a problem as well. Updates to the topic are then sent by email to interested people. Overall it is a good system that seems to satisfy its users but it has two major problems for Free Software projects wanting to use it: i) it is not Free Software and ii) it focuses too much on companies and their representatives.

[27]http://getsatisfaction.com


4 points marked best
28 replies made
19 people participating
1 employee helping

The mood in here

**Figure 3.8:** Example of an involvement and mood indicator of Get Satisfaction

## 3.5 Quality Assurance

In [vKSL03] one of the Freenet developers is cited: “A public release always leads to increased testing by new users, which in turn leads to the discovery of new bugs in the software, and commit of debugged code.” This cycle is important for a project. Long sprints without feedback lead to undiscovered bugs due to the limited number of use cases of the core team. They might not use the product in the way some or even most of their target audience does and therefore not be aware of critical bugs or behaviour changes. Aberdour identifies a large sustainable community as one of the key factors of high quality in Open Source software [Abe07]. “The sheer size of the bug-reporting group will ensure that more people test the system (on more platforms) than any commercial organization could hope to achieve. This group plays a key role in reducing defect density.” Raymond has been cited many times “Given enough eyeballs, all bugs are shallow.”, meaning that with enough users all bugs will be discovered at some point [Ray01]. Mozilla, for example, has started Firefox Input[28] to identify problematic areas with the help of its large user base. Among other things it shows how important indicators of user satisfaction (start-up time, page load time, responsiveness, stability, features) change over time, popular system configurations and which websites are most problematic for use with Firefox for a large number of people. Identifying bugs is of course only the first step towards a high-quality product. The other important factors according to Aberdour are code modularity, project management and the testing process. The other important rule Raymond coined is “release early, release often” [Ray01]. The idea behind this being that only if the code gets to testers early and often they can give feedback, find bugs and contribute code. Peer review is a key part of quality assurance in Free Software and by its nature it is accessible to more reviewers than

[28]http://input.mozilla.com

any closed source program. As McConnell points out however, for this approach to quality assurance to work a large number of users that are both interested in and capable of debugging source code are needed [McC99]. Experience from a large number of projects suggests that these people do indeed exist. Zhao and Elbaum found that in large Open Source projects around 80% of the hard bugs are found by users and that these projects receive over 60% of their feedback within hours (the numbers are smaller for smaller projects) [ZE03]. McConnell also points out that this way of working is effective and fast but not necessarily efficient. Bollinger et al. point out however, that this way of development should rather be seen as the minimal set of necessary rules and guidelines for good software development that removes all unneeded management and tool overhead [BNST99]. They ask: "Can you justify adding a new control, method, or metric to the process when open-source methods already work fine without it?"

Halloran and Scherlis list important attributes a quality-related tool or process must have in order to be adopted in projects following Open Source practices. They identified: an incremental model for quality investment and payoff, incremental adoptability, a trusted server-side implementation that can accept untrusted client-side input and a tool interaction style that is adaptable by practising Open Source programmers [HS02].

Otte created a quality assurance (QA) framework for Open Source software [Ott10]. It helps identify areas a project needs to concentrate on to deliver a better product. It identifies 23 processes: Requirement Management; Requirement Review; Process Documentation; Product Documentation; Project Organisation; Project Coordination; Team Communication; Knowledge Capturing; Team Education; Infrastructure Management; Design Control; Development Control (Coding); Continuous Code Quality Control; Code Review/Inspection; Peer Review; Defect Management; Unit, Integration and Regression Testing; Release Management; Build and Release Check; Quality Management; Software Quality Assurance; Process Change Management; Defect Prevention. The project is rated with a score from 0 (non-existence) to 8 (existence) for each of them according to how much of the process is implemented. In addition each process is rated according to how functional, usable, reliable and efficient it is. Functionality is rated on a scale from -4 (not) to 4 (overly), usability and reliability on a scale from -1 (partially) to 1 (strongly) and efficiency from -2 (not) to 2 (overly). The sum of these points for all processes is the Process Capability Score (PCS). It can take values between 0 and 368. Each process is also assigned a value between 0 and 5 depending on how important it likely is for the whole project. Otte also suggests a Process Success Score (PSS) metric to determine if a project is a success or disappointment.

*"Our lives improve only when we take chances – and the first and most difficult risk we can take is to be honest with ourselves."*

Walter Anderson

# 4 Analysis of the Current Development Processes

In this chapter the current development processes of Amarok and Halo are analysed. The focus is on who the stakeholders are, which tools are being used and which problems the teams are facing currently.

The analysis has been performed based on twenty structured interviews with the different involved parties. For Amarok the interviews were conducted privately via Internet Relay Chat (IRC) and Jabber since the team is spread out across the globe. For Halo most of the interviews were conducted privately, face-to-face inside ontoprise. The interviews with contributors outside ontoprise were conducted via IRC. The interview questions have been kept very general and open to avoid leading questions and to minimize any influence on what the participants thought was important. They can be found in Chapter B. The interviews were conducted in private to allow the participants to express their opinions freely without having to fear any negative consequences. This ensures that the real problems are uncovered and get resolved.

The interviewees have been chosen based on their time in the project and their area of involvement.

During the interviews several problems and obstacles in the development processes were brought up. We will concentrate on the ones that are most important to each project and can be solved by technical means.

Figure 4.1, Figure 4.2 and Figure 4.3 give an overview of the concepts around the development processes and their relations. Figure 4.1 shows the basic model of a typical Free Software project where contributors move from the outer parts to the inner parts and back during their involvement in the project. As can be seen in Figure 4.3, there are different contributor roles. The developer writes code and influences the release plan and design documents together with the release or project manager. Through this they all influence the release of the actual source code. Bug triagers screen incoming new bugs reported by testers. The bug reports might be triggered by testing according to a test case; if not they might lead to the creation of a new test case. During the development period developers receive advice on usability from a usability advisor, who might create mock-ups to illustrate concepts. At the end of a release cycle the marketing team promotes the release through announcements and packagers package the code for easy installation. The program is translated by a team of translators and documented by documentation writers. Users are the consumers in this ecosystem, however, some of them are active contributors in the sense that they provide feedback to influence the development process, e.g., filing bug reports or feature requests.

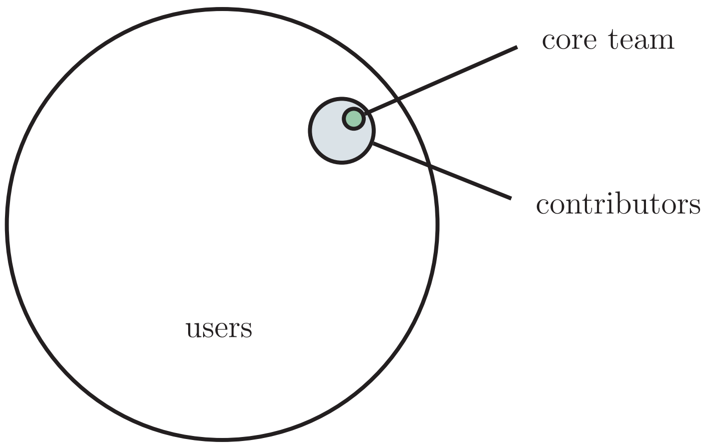


**Figure 4.1:** Onion model of a typical Free Software project

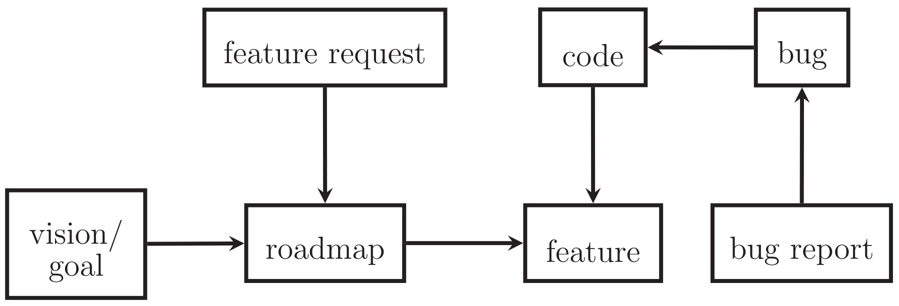


**Figure 4.2:** Important artefacts in the development processes (arrows indicate influence)

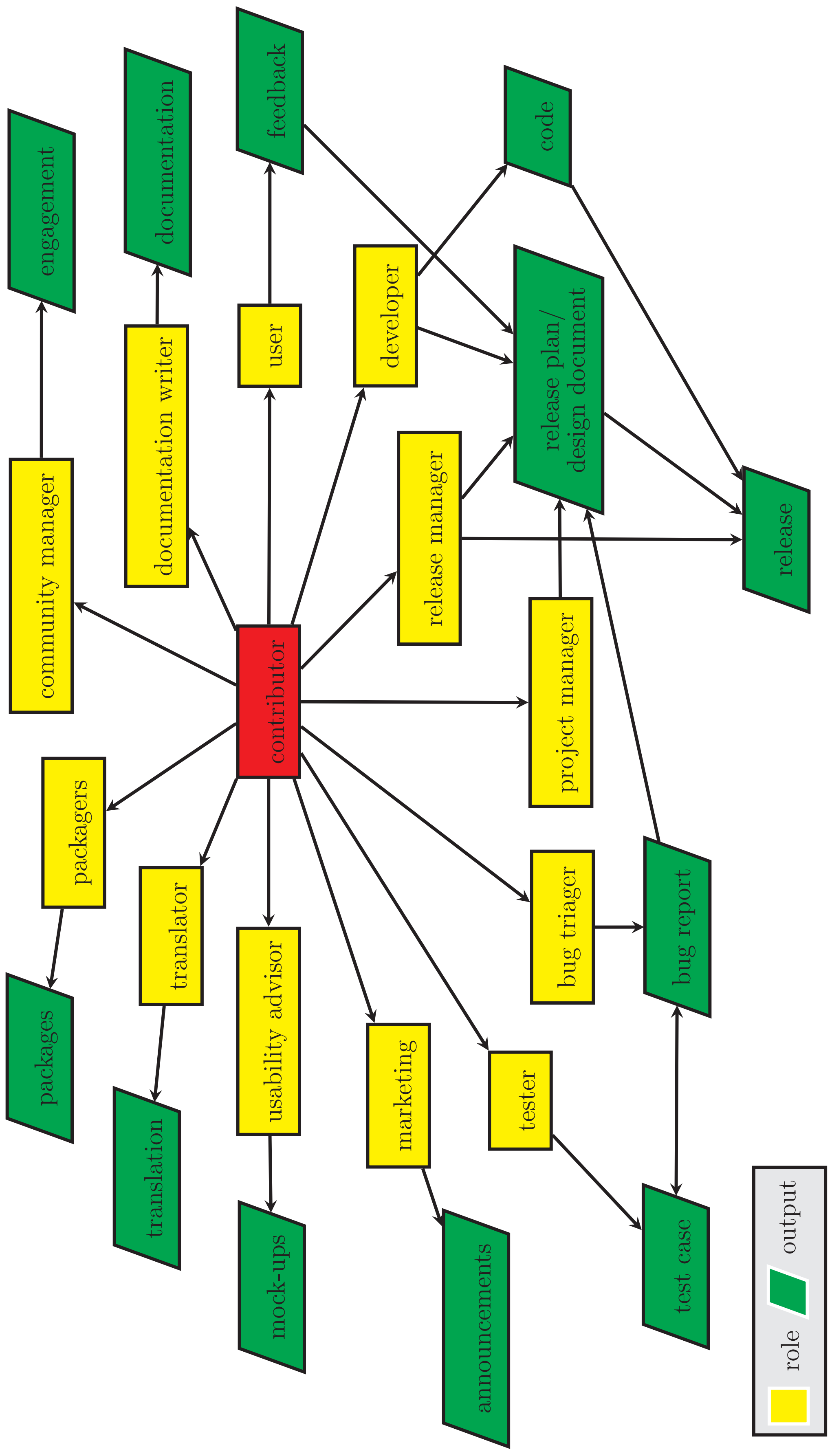


**Figure 4.3:** Contributor roles involved in the development processes and the output of their work

## 4.1 Halo

The interviews to map out the Halo development process were conducted with the current and former product manager, project manager, user documentation writer, usability expert, the quality assurance (QA) manager, two developers and two community contributors. All but the last two are employed by ontoprise. Figure 4.4 shows how long they have been in the team.

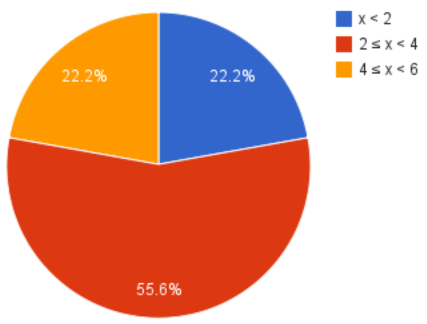


**Figure 4.4:** Distribution of the ten interview partners based on how many years they have been involved in Halo

### 4.1.1 Release Cycle

Halo's release cycle (Figure 4.5) is feature-based. This means a release is done when a planned set of features is implemented and tested. At the beginning of the cycle these features are chosen and designed in the design phase. This phase and its result are not publicly accessible. In the following implementation phase features are implemented, and in the quality assurance phase they are tested and polished. Parts of these phases are public, such as the Apache Subversion (SVN) repository containing the code. However, there are internal releases for these phases that are usually not published for wider testing, which means that there are no real pre-releases. The test cases used in the quality assurance phase are also not public. A timeline for the whole cycle exists, but finishing a given set of features is more important than meeting this schedule.

A feature freeze is not enforced and features can potentially be implemented until release.

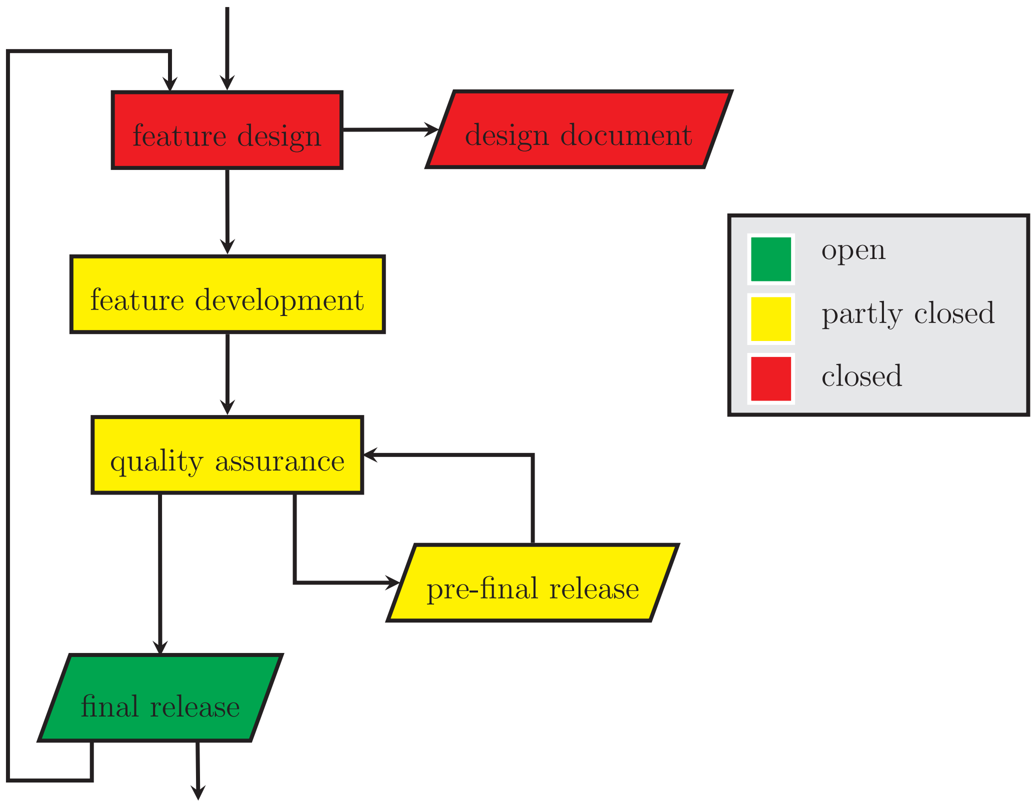
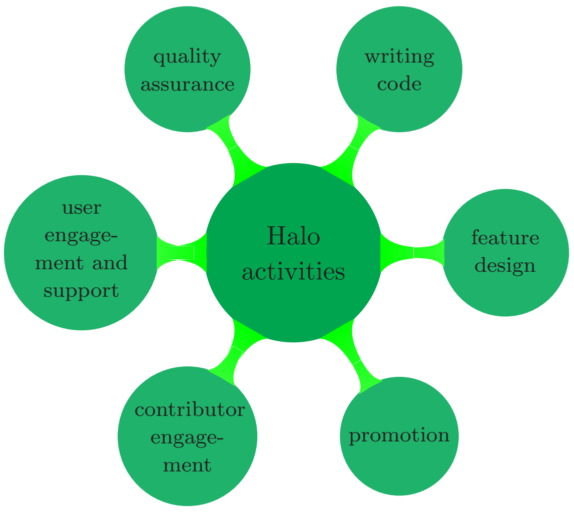


**Figure 4.5:** Halo's release cycle

### 4.1.2 Activities

The tasks and tools in Halo's development process can be grouped around six activities. They are described in the following sections. Usually each team member is only involved in a few of these activities.



#### 4.1.2.1 Feature design

At the beginning of the development cycle the whole team comes together to design features and plan the next release. This results in a design document that contains several milestones and is signed-off on by Vulcan. This is usually very detailed and is written in an internal wiki. It contains feature specifications, use cases and user interface (UI) mockups. Features are grouped into milestones there. These milestones constitute a roadmap for the next months.

In this phase everyone in the core team is involved with coming up with ideas for the next version(s). The actual feature specification is then written by the developers with help from the technical manager and project manager. The usability engineer gives advise on use-cases and provides mock-ups.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| users, core team, Vulcan | requirements, vision | brainstorming | ideas for features | core team |
| core team | ideas for features, feature requests | designing features | design document | core team, Vulcan |
| Vulcan | design document | signing-off on design document | signed contract | ontoprise |

**Table 4.1:** SIPOC for feature design in Halo

#### 4.1.2.2 Writing code

Code is developed mainly in SVN trunk by developers inside ontoprise. Each developer has a number of extensions he is responsible for and there is little overlap. Developers however regularly help each other in case they run into problems in their extensions and need advice. This happens on an internal developer mailing list if it is a general inquiry or in private email and Skype chats for specific problems concerning only two or three developers. In addition weekly status meetings are held to track progress and identify obstacles and problems.

As of December 2010 the suite of Halo extensions contains about 334,500 lines of code according to SLOCCount[1] and has an estimated development effort of around 1000 Person-Months. CLOC[2] reports a code:comment lines ratio of about 5:2 for the PHP and Javascript source files[3].

Delaying unfinished features to the next release is relatively hard since they are part of a contract. This means features are being developed until very late in a cycle. This can lead to poor quality because of a lack of proper QA during that time.

Not much code is being written outside of ontoprise by volunteer developers. The infrastructure for them to do so is however in place. The SVN repository is

[1]http://www.dwheeler.com/sloccount

[2]http://cloc.sourceforge.net

[3]Opinions vary widely about what an ideal code:comment ratio is. It is given here as one indicator that can be used to compare the two projects.

public for read access and it is possible to get an account with commit rights after submitting a few good patches[4].

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| developers | design document | writing code | features | users, Vulcan |

**Table 4.2:** SIPOC for writing code in Halo

#### 4.1.2.3 Quality assurance

Two things are done to ensure quality standards. First, a number of automated test cases is run every night on a Hudson instance[5], a continuous integration server. The results are emailed to the internal developer list once a day. In addition, before a release a test plan is executed. This means a number of test cases[6] are run manually to ensure important functionality is working as expected.

Bug reports are reported throughout the release cycle in a public bug tracker[7] running the Bugzilla software. Based on the component they are automatically assigned to the responsible developer or, if there is none, to an internal team mailing list that then assigns the bug to the appropriate developer. As Figure 4.6 and Figure 4.7 show, the vast majority of bug reports come from ontoprise employees and are also closed by them. In the weeks before a release (February, May, August, December) spikes in bug activity can be seen.

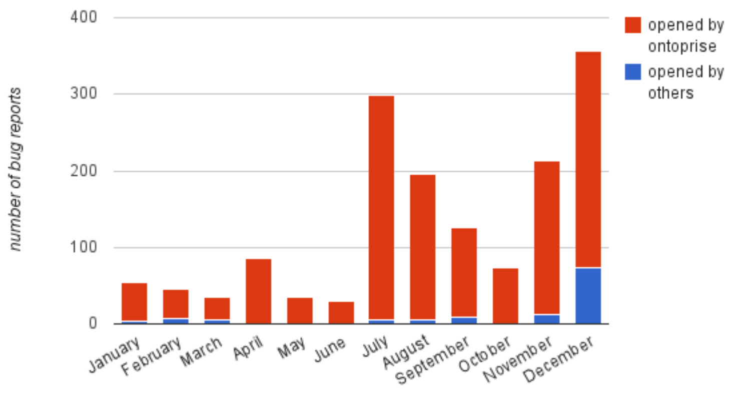


**Figure 4.6:** Number of opened bug reports in 2010 for Halo (ontoprise employees are identified by their @ontoprise.* email address)

[4]Code changes that fix a particular bug or implement a feature
[5]http://dailywikibuilds.ontoprise.com:8080
[6]Currently about 160 test cases are available.
[7]http://smwforum.ontoprise.com/smwbugs

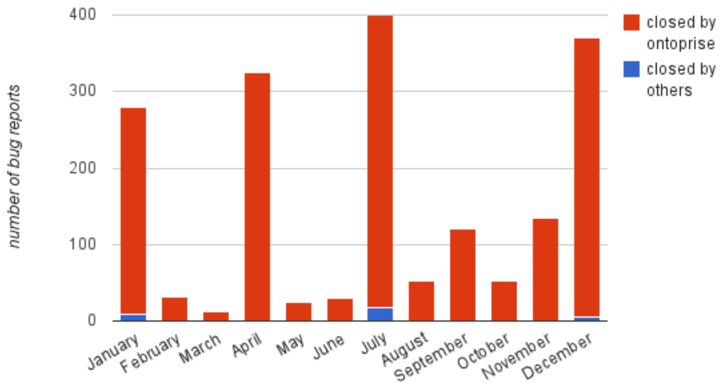


**Figure 4.7:** Number of closed bug reports in 2010 for Halo (ontoprise employees are identified by their @ontoprise.* email address)

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| Hudson | code from SVN | testing automatically | test result on website and in email | core team, testers |
| core team | code from SVN | testing manually | test result in test tracker | core team |
| users, core team | code | reporting bugs | bug reports | developers |
| developers | bug reports, test results | fixing bugs | bug fixes | users |

**Table 4.3:** SIPOC for quality assurance in Halo

#### 4.1.2.4 User engagement and support

User engagement and support happens largely in the SMWForum[8]. It contains the user documentation for all extensions, project news and a discussion board[9] where users can ask questions. The user documentation is created and maintained by a documentation writer and a few students helping her. Figure 4.8 shows the number of visits to SMWForum in 2010.

In the discussion board each sub-forum is assigned a moderator who has to ensure each request receives a reply within three working days. It is used actively and is the main place to find support. Close to 1600 posts have been written by about 250 users since September 2008. Most of the posts (about 45%) and topics (about 40%) are in the sub-forum "Installing SMW+, SMW or the halo Extension" which points to installation of the Halo suite being a major obstacle to its usage.

[8]http://smwforum.ontoprise.com
[9]http://smwforum.ontoprise.com/smwboard

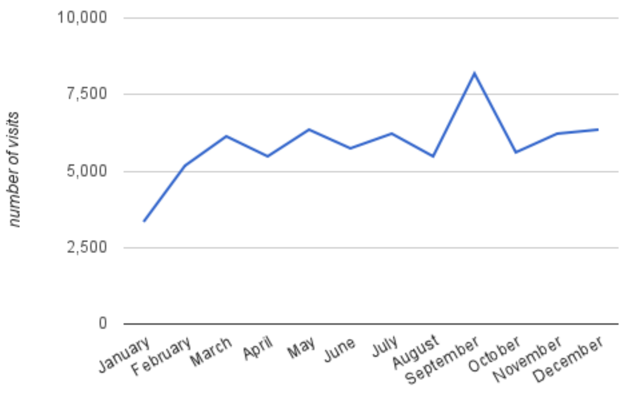


**Figure 4.8:** Number of visits to smwforum.ontoprise.com in 2010

In addition, support is also given on the Semantic MediaWiki (SMW) mailing list[10] and IRC channel[11]. The IRC channel however is not used by the Halo core team but instead monitored by two team members who direct people to other venues if necessary. Sharing these support channels has the advantages that it raises awareness in the wider community, the workload is spread among more people and that there are more people who can potentially answer a question. The disadvantage is that some SMW contributors/users feel uncomfortable with ontoprise's involvement there as has been expressed on the mailing list.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| community | support requests | giving support | answer to support request | user |
| core team | e.g., new release | announcing news | announcement | community |
| documentation team | design document, user questions | writing documentation | documentation | user |

**Table 4.4:** SIPOC for user engagement in Halo

#### 4.1.2.5 Contributor engagement

New contributors are important for the project as a source of feedback, requirements and testers as well as for dissemination of Halo.

For developers the SMWForum contains a Development section[12] which has information about how to get the source code, contribute patches, the release status and available junior jobs[13]. The information about the status of the current release

[10]semediawiki-user@lists.sourceforge.net
[11]#semantic-mediawiki on freenode
[12]http://smwforum.ontoprise.com/smwforum/index.php/Development/Main_Page
[13]Easy bugs for new contributors to get used to the code base

is updated by hand and therefore not always up-to-date. Potential contributors are also missing a list of open tasks to identify where they can help.

For testers a public testing contest was started where ontoprise gave away prizes for bug reports before a release. This was done only once so far and had problems due to the perception of ontoprise within the community. Outside of this contest bug reports are always possible though Bugzilla and the discussion board.

An interview series called Semantic Minds[14] was started to increase the visibility of people contributing to SMW and Halo.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| core team | design document, schedule | providing information about development process | wiki pages | contributors |
| core team | pre-release | public testing | bug reports, feedback | testers |
| users | software defect | reporting bugs | bug reports | developers |
| core team, community | contributions | increasing visibility of contributors | promotion | community |

**Table 4.5:** SIPOC for contributor engagement in Halo

#### 4.1.2.6 Promotion

Promotion is done via the SWMForum website and partially through the SMW mailing list. Activities are planned in the internal wiki and in internal meetings. Release announcements are written by the product manager and then reviewed by the project manager and someone involved in community engagement activities. Releases and other news are also posted on Twitter[15] (@SMWForum) and identi.ca[16] (!SMW group).

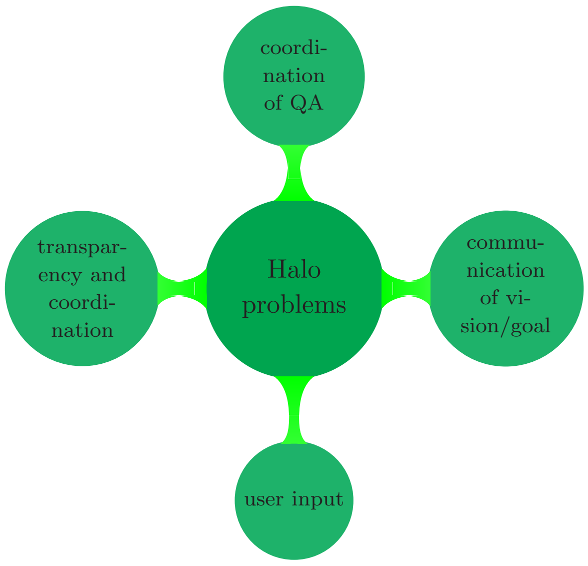
### 4.1.3 Problems

Halo is facing a number of problems that can be grouped into four main areas. They are explained in the next sections.

[14]http://smwforum.ontoprise.com/smwforum/index.php/SemanticMinds
[15]http://twitter.com
[16]http://identi.ca

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| core team | schedule, design document, ideas | planning and execution of promotion activities | promotional activities | community, users |
| core team | design document | writing and publishing release announcements | release announcement | users |

**Table 4.6:** SIPOC for promotion in Halo



### 4.1.3.1 Need for clearer communication of vision/goal

There seems to be a vision for Halo but it is not communicated enough both inside and outside of the team. People have a vague understanding what the vision is but are uncertain about it and can not express it in a few sentences. This leads to confusion about whether specific features are needed and requested by the target audience and if the suite is likely a good fit for a certain use-case. It makes it hard to argue in favour of or against the addition of features, which leads to tension. Two interviewees expressed their concern about the product becoming too complex for average users because of this lack of vision and defined target audience.

### 4.1.3.2 Need for better coordination of QA

Halo has a QA system but it does not seem to be effective enough due to the following reasons: i) short release cycle, ii) feature additions until shortly before release and iii) the clear focus on new features instead of stabilizing existing code.

The interviews made it clear that a bigger focus needs to be on stabilizing existing features for a while instead of adding many new ones. There is a feeling of development being very rushed. The adoption of Halo inside the community is hindered by these technical problems.

#### 4.1.3.3 Need for more transparency and coordination

Problems arise repeatedly inside the team due to extensive documentation and planning documents that do not reflect reality, e.g., if they are outdated. This leads to work in areas that are not urgent as well as work based on incorrect assumptions. Problems mainly occur where different teams need to work together (e.g., documentation and engineering) because they rely on outdated information. One of the main reasons for this seems to be that the internal wiki and its ontology are cumbersome to use for the employees. It was also brought up that the weekly developer meetings are very useful, but are not open and transparent for the rest of the team. One employee noted that they are sometimes missing the big picture, which makes it hard to make decisions, for example, about what is important to get done right now and what can be delayed.

Outside the team, missing transparency leads to potential contributors not turning into actual contributors. They are lacking an overview of what is being worked on, which phase of the development cycle the product is in and where help is needed. In addition, it leads to distrust inside the community since it is not visible who is involved, what their roles are and what is planned for the future. Since Halo depends on MediaWiki (MW) and SMW, among others, coordination with those projects is key. It was suggested to close the separate Halo SVN repository and move to MW's to make this easier.

One interviewee remarked: "A few people in the community have an image of Halo that largely consists of feature creep and a lot of patches." This image could be improved drastically with more transparency. As one example, the reasons for applying certain patches to MW and SMW would be clearer.

#### 4.1.3.4 Need for more user-input

Several interviewees raised their concerns about the small extent of user input. At the moment, development is driven largely by requirements from Vulcan. To create a product that can gain significant adoption in its market, increased focus on input from users is needed. To do this it needs to not only be easy (low technical and social barrier to entry, not too time-consuming) and clearly communicated that this is welcome, but for there to be follow-through and have this user input acted upon in the development of the product. Giving input is also one of the first and easiest contributions someone can make to a project on their way to becoming a valued part of the team. Past attempts to collect more user input likely often did not receive a lot of responses because they were too cumbersome (e.g., an online survey that took more than 30 minutes to complete).

#### 4.1.3.5 Other problems

The interviews and personal observations and conversations have hinted at other (mainly social) problems in the project that are out of the scope of this thesis. They should however be addressed to ensure that Halo can become widely adopted. Some of them will likely be improved as a side effect of the work in this thesis.

## 4.2 Amarok

The interviews to map out the Amarok development process were done with six developers, a usability advisor, a user documentation writer/community facilitator and a bug triager. All of them are volunteer contributors and represent the core team as well as occasional contributors. Figure 4.9 shows how long they have been in the team.

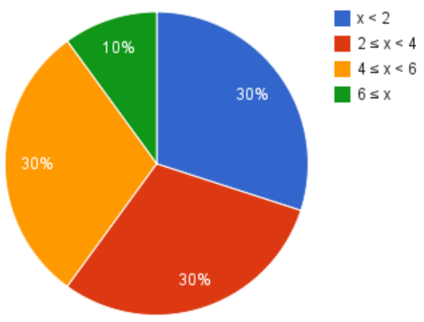


**Figure 4.9:** Distribution of the ten interview partners based on how many years they have been involved in Amarok

### 4.2.1 Release Cycle

Amarok's release cycle (Figure 4.10) is time-based and about eight to ten weeks long. This means that a release contains all of the features that are ready at a given point in time called feature freeze. Unfinished features are pushed to the next release. Being a volunteer-driven Free Software project this is not only very easy but is in fact necessary, since it is not possible to foresee the availability of the involved contributors.

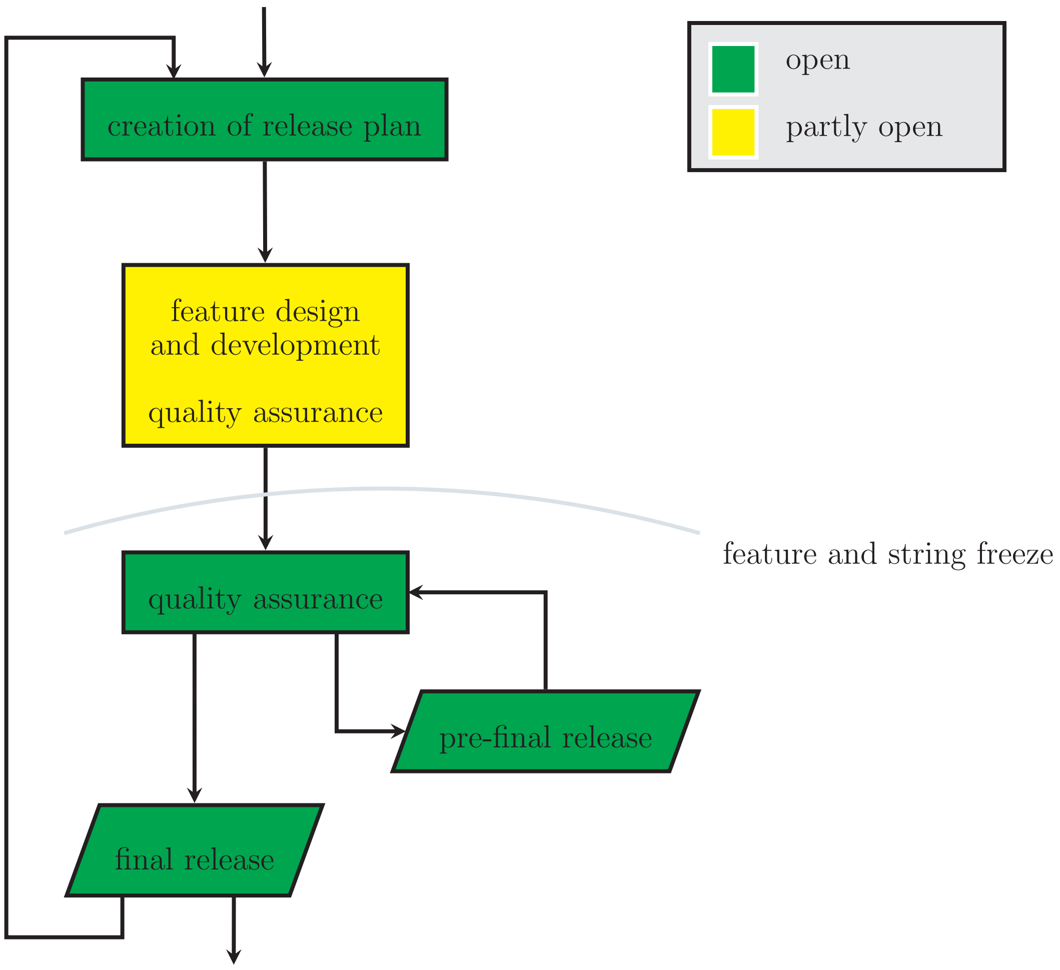


**Figure 4.10:** Amarok's release cycle

At the beginning of the cycle the team creates a plan that includes dates for string freeze[17], feature freeze[18] and releases.

There is no clear distinction between when features are designed and implemented. This largely happens at the same time and depends on the availability of team members and their other commitments. In general, they are free to design and implement features in git master[19] at any time during the design/implementation phase. If a feature is more involved than what can be accomplished in one cycle, it can be developed at any time in a branch and then get merged into git master when it is open for feature additions. A small part of this development is done in private git branches on the developer's computer before it is published.

The time between feature freeze and release is used for QA. Before the final release of a version there will usually be one or more public pre-releases that are tested by interested users and distributors. This gives a quick feedback cycle and allows for improvements before the final release.

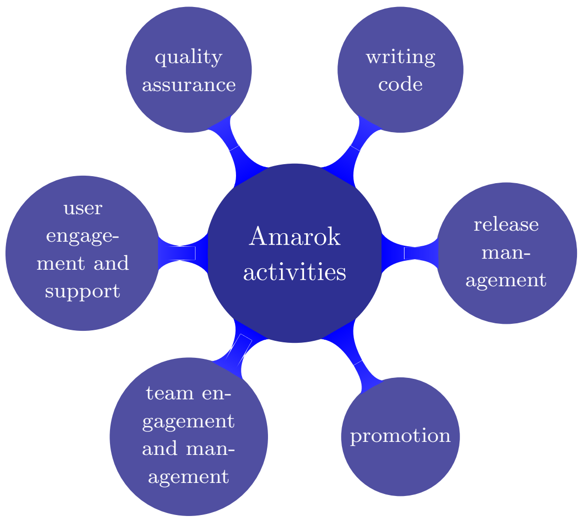
[17]The date from which no new translatable strings can be added to that release, to allow translation of the whole UI

[18]The date from which no new features can be added to that release, to allow enough time for testing and polishing. It is often on the same day as string freeze.

[19]The default development branch in the Amarok git repository

### 4.2.2 Activities

The tasks in the team can be grouped around six activities. They are described in the following sections. It is not uncommon for a team member to be involved in more than one of those activities.



#### 4.2.2.1 Writing Code

Developers are the ones actually writing the code that is the basis for everything else in the project. They make the ultimate decision of what goes into the product in the end. It is ultimately up to them to implement a feature or not – or to fix a bug or not. However, Amarok is a project that very much aims at multi-party consensus and finding the best solution for all parties to a given problem. Developers will listen to input from other contributors and users.

The current version control system in use is git. This has a few advantages over the previous version control system (SVN) but also brings challenges. Code is either committed to git master directly while it is being worked on, or it is developed in a branch (which is a lot easier with git than it was with SVN previously). These branches can either be private on a developer's computer or shared publicly, for example on KDE's main git server. This leads to a need for more coordination than was needed previously – something some developers still have to get used to.

As of December 2010, Amarok contains about 208,700 lines of code according to SLOCCount and has an estimated development effort of around 660 Person-Months. CLOC reports a code:comment lines ratio of about 7:1 for the C++ source files.

Most of the communication around writing code happens on IRC and the developer mailing list[20]. For longer discussions and problem-solving IRC is often preferred since it allows real-time communication. The developer mailing list is

[20]amarok-devel@kde.org

used for announcements and discussions that require input from a larger part of the core team. The list also receives notifications about review requests from ReviewBoard[21], where patches from new contributors can be reviewed as well as patches by established developers who want feedback. One-on-one communication is also performed via different instant messaging and collaboration services like Jabber and Skype.

Bugzilla[22] is used for bug handling. Most developers have searches based on their area of expertise and are subscribed to bugs in their component automatically. There is a tendency to avoid the web application and instead deal with bugs via email. This might be due to issues with Bugzilla itself or simply be due to the amount of bugs making the user interface difficult to deal with.

Sometimes a short design document or concept explanation is added to the project's wiki[23] to have a basis for a future discussion. However, this does not happen for most of the features being developed.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| developers | ideas, feature requests | implementing a feature | feature | users |
| developers | bug report | fixing a bug | bug fix | users |
| contributors | patch | posting a review request | review request | developers |
| developers | review request | reviewing a review request | comment, commit of a patch | contributors |
| contributors | ideas, feature requests | designing a new feature | design document | developers |

**Table 4.7:** SIPOC for writing code in Amarok

#### 4.2.2.2 Release Management

The release manager is responsible for proposing and executing the release schedule and making releases. The release schedule is created at the beginning of a cycle. The team is asked for input based on what each developer has planned. With this information the release manager makes a suggested release schedule that is published and accepted if nobody disagrees with it. For the actual release she needs to tag the release in git, create the tarball[24] and publish it for packagers and users at the appropriate times.

The work performed by other teams depends on the created release schedule. Translations of the software itself are done by KDE's translation teams. They

[21]http://git.reviewboard.kde.org
[22]http://bugs.kde.org
[23]http://amarok.kde.org/wiki
[24]Compressed archive of the source code of a release

depend on a string freeze in the release schedule to have a stable base to translate before release. As Amarok has started out on Linux and is still very Linux-centric, the project is dependent on distributions to package the software for easy installation. This is performed by packagers who receive the tarball of the release a few days before the official release to be able to have the packages ready for their users on release day.

Release management tasks are mostly coordinated on IRC. Input gathering for the release schedule creation as well as announcements are done on the developer mailing list and respective team lists for packagers and translators. The finished schedule is published in a public Google Calendar together with other Amarok-related events[25] so interested parties can easily subscribe to it in a calendar application.

Bugzilla is used for release management mainly to check the status of bugs that can potentially block a release. During the development cycle, such bugs are tagged with the keyword `release_blocker` and can then easily be searched for. This makes it possible to quickly judge if there are any bugs that must not be in a widely-used release, such as bugs that could cause data loss or make the program otherwise unusable.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| contributors | plans for features, other schedules | creating the release schedule | release schedule | community |
| release manager | bug reports, contributor feedback | evaluating the code for release | decision | contributors |
| developers | code | doing the release | tarball | packagers, users |
| release manager | release schedule | coordinating with other teams | communication, coordination | other teams including packagers and translators |

**Table 4.8:** SIPOC for release management in Amarok

#### 4.2.2.3 Quality assurance

Bugsquad is the group that looks at bug reports and triages them. This includes asking for further information when the reporter did not give enough details and making sure they are actually bugs in Amarok and not in another product or user errors. If the relevant developer is not already subscribed to the bug they notify him. They also make the release manager aware of release-blocking bugs.

[25]Team member birthdays, meetings, conferences and trade shows

IRC is used to get input and assessments of bugs from other team members and users. This solves questions like “is this already fixed in the development version?” and “can anyone reproduce this bug?” in case the triager is unable.

They are notified of new bugs most of the time via a special mailing list that is set up to receive all Bugzilla activity[26] for Amarok and an IRC bot that posts new bug reports to the user channel.

Figure 4.11 shows the number of new bug reports filed in 2010. It is hard to identify which of these are reported by core team members since not all of them have an @kde.org email address and use it for their Bugzilla account and some other owners of these addresses are not members of the Amarok core team. In addition, the boundary of the core team is flowing.

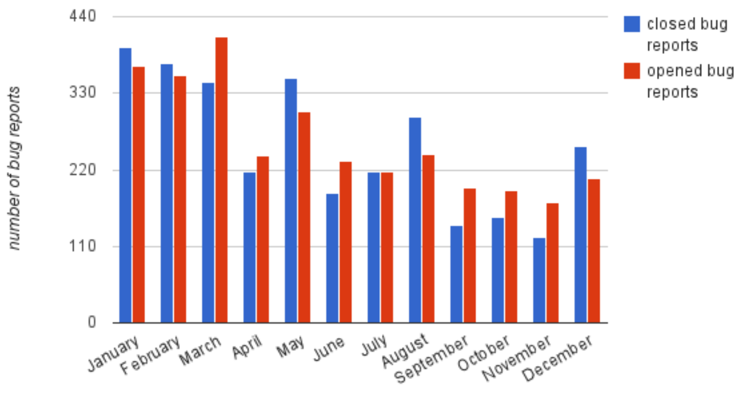


**Figure 4.11:** Number of opened and closed bug reports in 2010 for Amarok

On top of that the internal wiki contains an outdated checklist of important functionality that should be tested before a release. It is currently unused, however.

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| community | source code | testing | bug reports, feedback | developers |
| bug squad | bug reports | triaging bugs | comments on bug reports | community |
| users | bug reports | fixing bugs | bug fixes | users |

**Table 4.9:** SIPOC for quality assurance in Amarok

#### 4.2.2.4 User engagement and support

Users are important in the development process in various aspects: they provide the core team with requirements and feedback; they use and test the in-

[26]This includes new bug reports, comments on existing reports and changes to the status and subscriber list.

development code throughout the entire release cycle; and depending on how adventurous and knowledgeable each user is and how much he wants a given feature he might compile right from git master regularly, test pre-final releases or only use final releases. Therefore, the team tries to keep git master usable at all times and to give advanced notice of commits that might cause bigger problems. Users also provide support to other users in various venues, thereby taking a lot of support work off of the core team in some areas. In addition, they promote Amarok to their peers online and offline and thereby help the userbase grow. Ultimately, this is the pool new contributors are recruited from to keep the project running.

Support by users and team members is done mainly in three places: IRC[27], the user mailing list[28] and the forum[29]. Developers regularly provide support on the mailing list and on IRC but rarely the forum. Recently there is also a trend of users seeking help in social media like Twitter, identi.ca or Facebook[30]. These users usually end up being referred to other channels more suitable for support. Social media is also valuable as it gives an easy way to be close to and interact with the team. News can be spread quickly and a lot of potential volunteers can be reached for tasks.

User feedback comes in different forms. The main ones are bug reports in Bugzilla, KDE's forum, IRC and mailing list. KDE's forum additionally provides a place called Brainstorm[31] where users can post ideas and refine and vote on existing ones.

Documentation is done in two wikis, Amarok's own wiki for specialised documentation and KDE's UserBase[32] for general user documentation.

The main website and news outlet is a Drupal-based website[33] that also contains screenshots, feature lists, contributor blogs and links to support resources. Figure 4.12 shows the number of visits to the main website and wiki in 2010.

#### 4.2.2.5 Team engagement and management

Team engagement and management is the main task of the community manager and her team. This includes facilitating discussions and meetings when needed, providing access to resources and finding volunteers for urgent tasks. The communication for this is mainly done via IRC and mailing lists, but social media is also used where and when appropriate. Social media in this context is mostly used as a tool to strengthen the bonds between team members, which is crucial in a distributed team of volunteers. Facebook and other social media outlets make it simple

[27]#amarok on freenode
[28]amarok@kde.org
[29]http://forum.kde.org
[30]http://facebook.com
[31]http://forum.kde.org/brainstorm.php
[32]http://userbase.kde.org
[33]http://amarok.kde.org

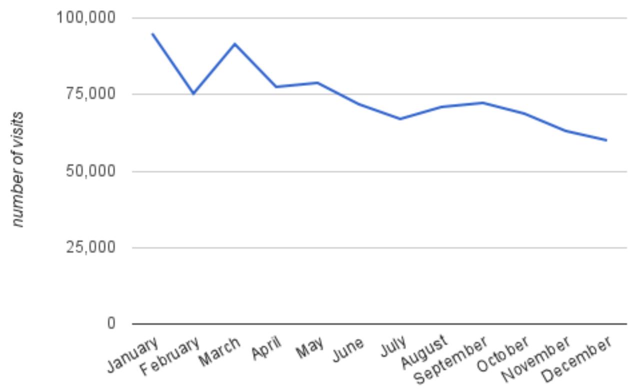


**Figure 4.12:** Number of visits to amarok.kde.org in 2010

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| community | support request | giving support | answer | users |
| community | new features, releases, events | promoting Amarok | promotion | users |
| users | feedback | receiving feedback | recommendations for changes | contributors |
| contributors | support requests, new features | providing documentation | documentation | users |
| contributors | requests for information | providing general information | information | users |

**Table 4.10:** SIPOC for user engagement and support in Amarok

to stay up-to-date with what is happening in the life of other team members, and thereby enables the team to support each other in times of need as well as share moments of joy.

#### 4.2.2.6 Promotion

The promotion team, called Rokymotion, is among other things responsible for everything having to do with outreach. This includes writing release announcements and blog posts about new developments, engaging in social media[34], giving talks at conferences and staffing booths at events. While all contributors perform

[34]Mainly @amarok on Twitter and identi.ca, the !Amarok group on identi.ca and the Amarok fan page on Facebook

| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| core team | request for meeting, urgent topic | facilitating meetings and discussions | meeting, discussion | community |
| core team | task | finding volunteers for (urgent) tasks | volunteer | community |
| community manager | team relationships | strengthening the bonds in the team | better teamwork | community |

**Table 4.11:** SIPOC for team engagement and management in Amarok

some degree of outreach, the Rokymotion members are responsible for the biggest tasks. They also help with the logistics of attending conferences.

Team communication happens on a special mailing list[35] and IRC channel[36]. A special wiki also existed but has been abandoned. Since then writing of announcements happens mainly on Google Docs[37] because it allows real-time collaboration which is often needed for announcement writing.

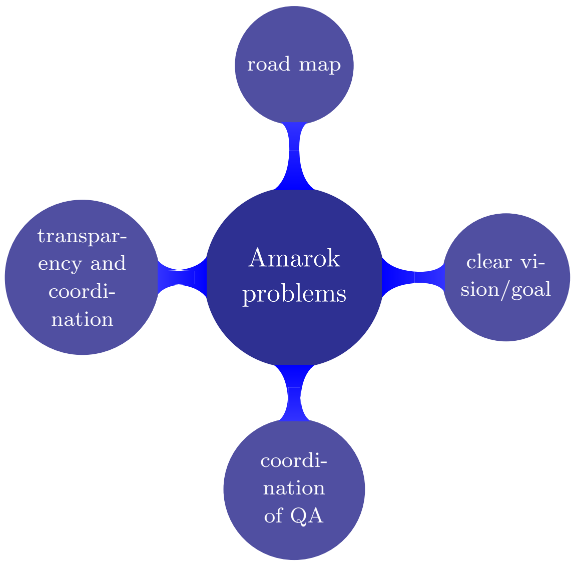
| Suppliers | Inputs | Process | Output | Customers |
|---|---|---|---|---|
| core team | release plan, changelog, new features | writing release announcements and blog posts | release announcements, blog posts | users |
| community | news, releases, new features | giving talks | talks, information | users |
| community | conference announcement | attending conferences | conference attendance, publicity | users |

**Table 4.12:** SIPOC for promotion in Amarok

### 4.2.3 Problems

Amarok is facing a number of problems that can be grouped into four areas. They are explained in the next sections.

[35]amarok-promo@kde.org
[36]#rokymotion on freenode
[37]http://docs.google.com



#### 4.2.3.1 Lack of a clear vision/goal

Amarok has always been about helping people rediscover their music. Amarok 2.0 was a reinvention, expansion and re-focussing on this goal. It is now reached and the team is lacking a clear target at which to aim. Such a target is needed for recruiting volunteers, positioning in a market of seemingly endless competitors as well as focusing of available resources.

Since Amarok 2 is a rewrite the team has decided to consciously drop some of the features of Amarok 1.4 because they were either very hard to maintain or were only used by a few users and made the experience worse for the others. Having a more clearly defined goal and target audience would make communication about these choices less difficult.

#### 4.2.3.2 Lack of a road map

Amarok is lacking a road map. However, it is important to have one for several reasons. Having a roadmap would make it easier to communicate to potential volunteers where help is needed and to users when their requested feature will likely be implemented. It would reduce duplication of work, which has become a larger problem since the migration from SVN to git, and would allow feedback from other team members as well as users to be heard early in the process. In addition it would ease the creation of a release plan at the beginning of a cycle and make it possible to plan further ahead than just one cycle.

During the interviews concerns were raised about a road map being restrictive and thereby potentially hurting the project. At the same, time all interviewees seem to see a need for more planning.

#### 4.2.3.3 Need for more transparency and coordination

At the moment, a lot of knowledge about Amarok's code and community and the plans around it are not written down but are instead communicated in conversations between contributors and contributors and users. Since Amarok is a volunteer project, it is always at risk of losing key contributors and their knowledge. This needs to be avoided. More transparency would also make it easier for the team to communicate the needs of the project and for new contributors to find their place in the project quickly.

Duplication of work and misunderstandings are also happening because of a lack of communication. In a distributed team having an overview of who is working on what is especially important.

#### 4.2.3.4 Need for better coordination of QA

Amarok's bug triagers, due to the size of the userbase, have to deal with a large number of bug reports. They have reached a volume where each developer can no longer look at all bug reports. Developers need help finding the bug reports that they need to concentrate on at any given time. Bug triagers need an easy way to communicate the pertinent bug reports to the appropriate developers and to the release manager. In addition, no test cases are available for volunteers to test before a release.

#### 4.2.3.5 Other problems

Just like Halo, Amarok has a number of other (mainly social) problems that are outside the scope of this thesis. The team is working on them and hopefully the work in this thesis will support that effort.

## 4.3 Comparison and conclusion

The two projects are evidently very different. Table 4.13 lists some of the features that distinguish both projects, all of which have large impacts. However, despite their differences, they use a lot of the same or similar tools (Table 4.14) and face very similar problems.

Figure 4.13 shows the points Halo and Amarok received for each of the processes tracked by the QAfOSS Model [Ott10]. The processes were rated based on the interviews and the author's experiences within the projects. Halo reached a Process Capability Score (PCS) of 172 and Amarok of 156. The maximum PCS is 368.

| | Halo | Amarok |
|---|---|---|
| age of project (time since first public release) | 2 years | 7 years |
| subject | semantic web | music |
| main programming language | PHP | C++ |
| motivation | mainly extrinsic | mainly intrinsic |
| volunteer vs paid | mostly paid | volunteer |
| boundaries of core team | clear | unclear |
| main interaction media | personal interaction, mailing lists, bug reports | IRC, mailing lists |
| community outside the core team | small | large |
| openness | relatively closed | relatively open |
| “release early release often” philosophy | no | yes |
| meetings | many | few |
| planning | extensive | little |
| direction set by | management and team | individual developer |
| release schedule based on | features | time |
| focused QA | yes | no |
| funding | Vulcan | users |
| legal entity | company | non-profit organisation |
| geographical location of core team members | Germany | worldwide |
| testing of development code by users | rarely | often |
| distribution done by | core team | distributions |

**Table 4.13:** Comparison of some of the distinguishing features of the projects

| | Halo | Amarok |
|---|---|---|
| version control system | SVN | git |
| wiki | MW, SMW and Halo | MW |
| bug tracker | Bugzilla | Bugzilla |
| build server | Hudson | Hudson |
| test case management | TestLink | simple wiki page |

**Table 4.14:** Comparison of some of the tools used by the projects

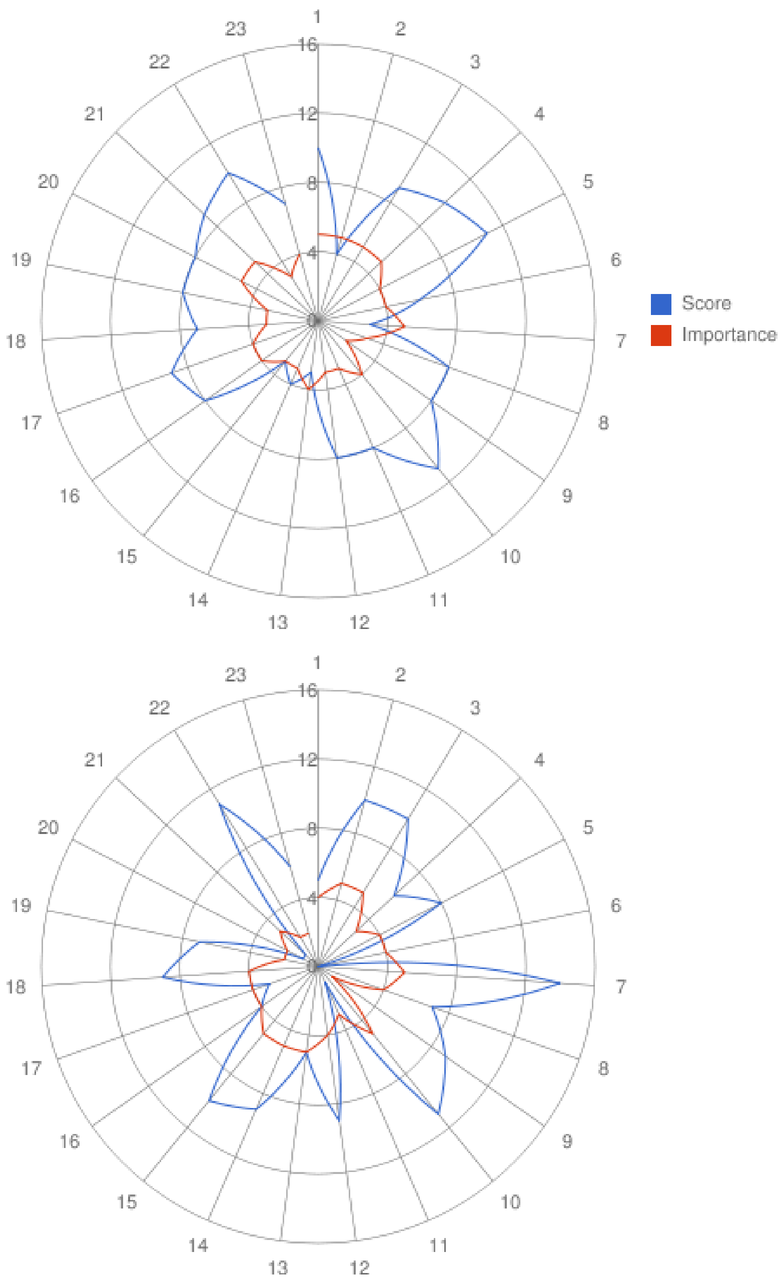


**Figure 4.13:** QAfOSS Model for Halo (top) and Amarok (bottom) based on [Ott10] (**1**: Requirement Management **2**: Requirement Review **3**: Process Documentation **4**: Product Documentation **5**: Project Organisation **6**: Project Coordination **7**: Team Communication **8**: Knowledge Capturing **9**: Team Education **10**: Infrastructure Management **11**: Design Control **12**: Development Control (Coding) **13**: Continuous Code Quality Control **14**: Code Review/Inspection **15**: Peer Review **16**: Defect Management **17**: Unit, Integration and Regression Testing **18**: Release Management **19**: Build and Release Check **20**: Quality Management **21**: Software Quality Assurance **22**: Process Change Management **23**: Defect Prevention)

*“Another world is not only possible, she is on her way. On a quiet day, I can hear her breathing.”*

Arundhati Roy

# 5 Design of an Improved Development Process

In this chapter the new proposed development process for Halo and Amarok will be introduced. The focus of the design is on improving the involvement of all stakeholders in the development process and the decisions and actions related to it by making them more transparent and collaborative. This involves a concentration on or change to a bottom-up approach instead of top-down processes.

## 5.1 Requirements, expectations and constraints

Both projects have a number of requirements, expectations and constraints for the new processes and tools. They are roughly ordered by importance.

**Building trust by transparency** First and foremost the new processes need to build trust by transparency. According to Grams, one of the reasons why an Open Source projects fails is a lack of trust. He says: “Collaboration works better when you trust the people with whom you are collaborating. Transparency is more believable when you trust those who are opening up to you. And it is much easier for the best ideas to win when there is a base level of trust in the community that everyone is competent and has the best interests of the project at heart.” [Gra10b] Blizzard also summarizes this nicely as ”Surprise is the opposite of engagement.” [Gra10a] This is especially important for Halo.

People gaming the system are likely not going to be a large problem at this point. Halo does not have a dedicated enough community for this and Amarok's is big enough to spot these cases and fix them.

**Allowing quick overview and contributions** The new processes need to allow quick contributions and give an easily accessible overview of the current development status – a dashboard-like overview. They need to facilitate moving to the centre of the community from the outskirts (Figure 4.1). They need to be low maintenance and easy to learn and use for newcomers. The barrier to entry they pose needs to be small.

**Avoiding "cookie licking"** The most important thing to be avoided is probably "cookie licking" [N+]. It describes a community anti-pattern in which someone claims a given task with the best of intentions and then does not have the time, motivation or skill set to complete it. Others who might have had the skills, time and motivation to do it in the meantime will refrain so as to not interfere with the work of the cookie-licker. The result is that the work does not get done or is delayed unnecessarily.

**Not wasting time** It is also important that the new processes do not require more time commitment from contributors than the current processes (without very good justification), as that would greatly hinder new process adoption in both projects.

**Using Free Software** For Amarok it is very important that any newly introduced tool is Free Software. This is a common requirement among Free Software projects [HS02]. This, and the will to lower the barrier to entry, leads to a number of de-facto standard tools which makes it easier to contribute to other projects once the contributor is familiar with them, e.g., Bugzilla, Apache Subversion (SVN), git, Mailman, MediaWiki (MW) and Internet Relay Chat (IRC).

**Setting expectations correctly** To the user it needs to be clear that his suggestions in these new processes are just suggestions. The teams will try to accommodate them but there are many potential reasons for not being able or willing to follow what the user community suggests. These include a lack of time, skills and manpower as well as technical problems and interference with other features or the vision of the project. The expectations need to be set correctly here.

**Changing mindsets** The most difficult thing to realize is likely the change of mindset that is needed in companies like ontoprise when working on Free Software projects. They fall into the Tom Sawyer trap too easily: "If you are looking to ideas like open source or social media as simple means to get what you want for your company, it's time to rethink your community strategy. [. . . ] Rather than asking how they can paint your fence for you, ask yourself how can we all paint everyone's fences together? You'll change everything if you take yourself out of the center of the community and instead become a humble contributor to a community where everyone, including you, will benefit from your efforts." [Gra09a] More transparency in all processes and the interaction with community members it brings will hopefully facilitate this mindset shift.

## 5.2 Communication and coordination in distributed teams

There are a number of tools and processes that can make communicating and coordinating in a distributed team easier. Those that will have a large impact on Halo and Amarok are investigated here.

### 5.2.1 Team and task awareness

Both projects struggle with team and task awareness. As seen in the analysis, it is often hard to answer the following questions: What is everyone working on? What are their current problems? Does he have time to help with an urgent problem? What needs to be done before the next release?

To create this task and team awareness a team and a release dashboard are proposed. Both dashboards aggregate data from different sources: repository, bug tracker, build server, feature request tracker, calendar, and more.

On the team dashboard everyone can quickly and easily see important information regarding the team and project. It should contain the following information:

- task count for each contributor
- bug count for each contributor
- feature request count for each contributor
- local time for each contributor
- build status, failing test cases
- countdown for the next releases
- calendar with other important dates
- links to important other sites (e.g., a list of release blocker bugs, checklists)
- accomplishments (X bugs and features already completed, Y still open for this release)

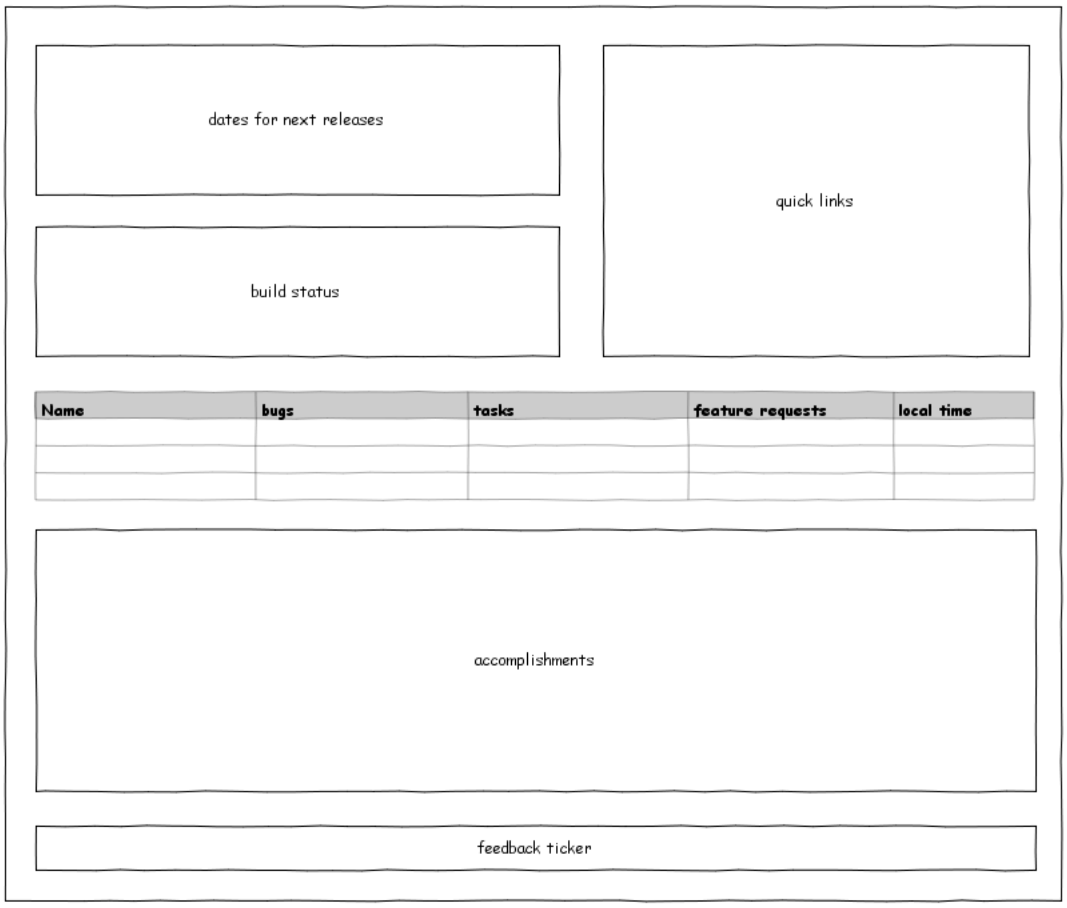


**Figure 5.1:** Mock-up of the proposed team dashboard

- ticker of user feedback (e.g., Twitter, identi.ca)

A mock-up can be seen in Figure 5.1.

The release dashboard gives an overview of the release currently being worked on. It shows the latest commits to the repository and the remaining bugs and feature request. A mock-up can be seen in Figure 5.2.

### 5.2.2 Release schedule

The release schedule should be public so that all involved parties can take these dates into consideration for their planning. There are two things that make this complicated, however. First, a lot of Free Software projects (including Halo) do not have a publicly communicated schedule. Creating one gives a rhythm, a heartbeat, to the project that everyone in the ecosystem can align with as needed. This heartbeat is also what keeps everyone engaged and involved. Importantly, it also communicates to the outside world that the project is alive. The other problem is the fear of a wrong schedule. It is no shame to adjust the schedule when one has new and relevant information or requirements that is best addressed by a schedule change. If a team can never meet the schedule they set themselves,

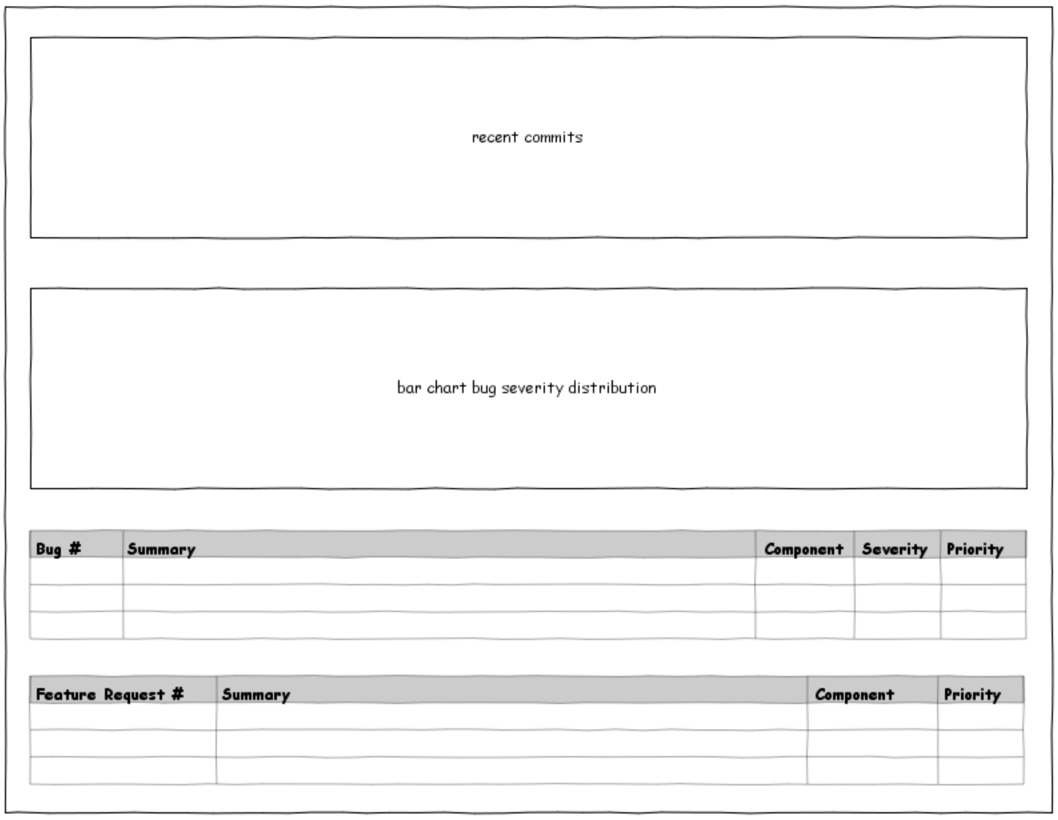


**Figure 5.2:** Mock-up of the proposed release dashboard

it is likely time to look for the fix in the underlying processes that are the reason for the constant delay or wrong schedule estimation.

It is therefore proposed to create a release calendar. It should be easy to maintain and a reasonable number of people (at least three) should have edit access to it to ensure it is kept up-to-date. Possible options here include a calendar in the wiki, a shared Google calendar, or a simple webpage on the project's website.

Amarok already has a Google calendar that is shared with the team and contains every release milestone. This will be used and made more prominently and publicly available. Halo so far only has a wiki page which lists the approximate date of the next release. A calendar in the wiki will be introduced here as this is the option that is easy and most likely to be kept up-to-date by that team.

### 5.2.3 Code ownership

In both projects code ownership[1] is a problem. It is a balancing act between making it exclusive and very explicit on the one hand (Halo mostly follows this model) and not having it at all on the other hand (Amarok mostly follows this model). The problem with exclusive and explicit code ownership is that it discourages or prevents others from contributing. In the end it is another "cookie licking"

[1]Someone is the maintainer of a part of the code base and feels responsible for it.

problem. The advantage of it is that there is always a person responsible for a given part of the code base that can be contacted with questions and problems. Logically, the disadvantage of having no explicit code ownership is the lack of the advantages that code ownership provides. The benefit of not having explicit code ownership is that it allows anyone to fill this vacuum. As a result, the goal is to make it explicit but not overly so, and not exclusive. The same applies to feature requests and bug reports.

The solution for this could be twofold. On the one hand it could be made clear that wherever a notion of ownership is publicly communicated, it is done so with a timer. A feature request could be assigned to a developer for three weeks, and if he does not work on it during this time he loses his exclusivity and fulfilling the request becomes a free-for-all. The other part would be an activity indicator, showing if the developer is currently actively participating in the project based on activity on mailing lists, the bug tracker, the source code repository and the IRC channel(s), among other things. The activity indicator will be discussed further in Section 5.5.

For Amarok a maintainer file will be introduced into the repository. It will have a short notice saying that the assignees of the particular area in Bugzilla is also the maintainer of that code and mention that this is not exclusive. For Halo the existing wiki page listing maintainers will be amended with a message making it clearer that while these people are the maintainers they are not the sole owners of that code.

### 5.2.4 Checklists

For new contributors it is often not easy to see what is required from them in their first commits.

To help them, and to maintain consistency among long-term contributors, a number of checklists are introduced. These checklists are not intrusive and the contributor can go through them on his own.

One such checklist is for the current commit. Another can be applied to feature requests or review requests. There are a few important questions these checklists should ask. For the commit checklist these could be: Does it fit the vision of the project or at least not work against it? Does it conform to the project's coding style? Is there a bug or feature request or test case associated with this? Does it introduce new dependencies? Does it add new strings? For a feature request or review request the questions could be: Does it fit the vision of the project or at least not work against it? Does it conform to the project's coding style? Who is going to maintain the code?

For Halo these checklists will be made available in the developer section of the public wiki. Amarok's will be placed in the HACKING directory in git which already holds other similar documents.

### 5.2.5 Building trust through engagement

Building trust is essential for collaborative work as laid out in Section 5.1, especially for Halo. One way to build this trust is by enabling face-to-face meetings of distributed teams. This only works up to a certain team size, as it might be too costly for a project with larger teams and does not necessarily engage people at the outskirts of the contributor circle (Figure 4.1). Other, additional ways have to be used.

A lot of tools can be used to engage with the community at large and show the more human side behind a Free Software project like Halo and Amarok. The most widespread tools are likely project planets[2] and social media tools (e.g., identi.ca, Twitter, Facebook), where team members can engage with other projects, users and themselves. This engagement builds social capital. In volunteer-driven projects there seems to be more willingness by developers to engage with users directly than in commercial projects. Part of this might be that it takes time and mental capacity that they do not have or that they do not deem it important enough. The other important factor is fear of consequences from management. In those cases, management needs to make a case for community engagement both by preferably engaging with the community in some form themselves and by making sure that employees see it as part of their job requirements. Social media guidelines can help take away some of the fear as well by making it explicit what the company does and does not deem acceptable. Being involved in social media channels has three very important benefits: i) the direct connection from the team to the user and the ability to interact with them; ii) the direct connection from the user to the team allowing users to express feedback that goes beyond bug reports, support tickets and feature requests; and iii) the direct connection between users without interference or the need for guidance from the core team. It is important to allow each of these connections in collaborative and transparent teams.

Amarok is already using all of the above-mentioned tools. Halo uses them only partially and wider use of them will be encouraged by integrating it into the team dashboard.

One of the biggest sources for non-transparency (and as a result, mistrust) is meetings. Too often they happen either face-to-face or via telephone. Both are notoriously bad for non-participants unless proper meeting minutes are taken and non-participants can provide their input beforehand and can still have influence on any decisions taken in these meetings afterwards.

To approach the first problem, text-based communication media like IRC, Skype chat and Jabber have proven useful. Participants are informed at the beginning of the meeting that a log will be made available either publicly or to a number of selected people. Decisions and conclusions of the meeting then become much more transparent to someone who did not take part. For important and non-urgent

[2]Blog aggregators that produce a website containing all relevant blog entries by contributors of the project

matters meetings should be avoided altogether. These discussions are often much more suitable to a non-realtime communication medium like a mailing list.

Amarok already holds very few meetings. Most day-to-day operations are handled on IRC and via the mailing lists. This should stay that way. Halo holds a lot of regular phone conferences and meeting minutes are only published in the internal wiki. To reach real transparency and collaboration with the wider community, these need to be transitioned. The first step would be to make meeting minutes public to as great a degree as possible. However, for various reasons (e.g., confidentiality concerns) the likelihood of that happening is small.

### 5.2.6 Connecting commits

It is often hard to keep track of the information that is scattered across different systems like bug trackers, mailing lists, repositories and patch review systems.

One way to connect the information in these systems are commit hooks. Modern version control systems support the execution of scripts, called commit hooks, upon each commit. This allows, for example, a bug or review request to be closed or a notification to be sent by email. They are especially useful if history needs to be reconstructed at some future point; if the commit is linked to a bug report, it is easy to find the bug report and reopen it in case the commit needs to be reverted because of regressions it introduced. If the committer wants to notify someone of a commit, emailing them the commit and what it changed (a diff) can be done. Table 5.1 lists a selection of the most useful keywords triggering such scripts in KDE. The keyword is simply mentioned in the commit message with the appropriate argument (e.g., “BUG 12345”). It is then parsed by the commit hook and the appropriate action is executed.

| Keyword | Action |
|---|---|
| BUG | close a bug and post the commit message and diff as a comment |
| CCBUG | post the commit message and diff as a comment to a bug |
| REVIEW | close a review request |
| FEATURE | close a feature request |
| CCMAIL | send an email with the commit message and diff |

**Table 5.1:** Useful actions that can be executed with a commit

## 5.3 Collaboratively working on a vision

The analysis in Chapter 4 has shown that both projects lack a clear and written-down vision.

### 5.3.1 Creating and communicating a vision

The creation of a vision for a team is a delicate undertaking. It can be especially troublesome in a volunteer team as it may bring up very different understandings of what the vision is or should be; team members may find that their existing contributions have been furthering different goals. At the same time it is needed to focus resources and have a clear understanding of what the project is and where it is going.

Three methods seem applicable towards create this vision. The first is an unstructured meeting of the whole team or a good representation of it. In this meeting the team would try to formulate a vision. The second option is that the Halo management team and Amarok committee sit down and formulate a vision for their respective team. The third option is an adaptation of the process proposed by Levin [Lev00] that was introduced in Chapter 3.

The first option has the risk of being too unguided and therefore not leading to a good result in an acceptable time frame. The second option risks alienating a large part of the team and thereby makes the resulting vision unlikely to succeed. The process proposed by Levin is both transparent and collaborative. It will therefore be used for the creation of the vision for Halo and Amarok. The 4-stage process (becoming informed, visiting the future and recording the experience, creating the story, deploying the vision) will be applied with modifications.

Each of the four stages in Levin's approach is associated with a number of questions. Since they are very focused on companies, they need to be adapted for Free Software projects. Their focus has been moved from a commercial company with employees to a Free Software project with contributors. Stages 2 and 3 have been merged to simplify the process. The adapted questions for stage 1 (the present) are:

- What is happening in the world external to our project that may impact our work?
- What trends are occurring that may affect the needs, expectations, and desires of key stakeholders (e.g., contributors, employees, upstream[3], downstream[4], etc.)
- What are other projects (within and outside our industry) doing or considering doing to prepare themselves for the future?
- What are the core values and beliefs for how contributions should be done that will not be compromised?

For stages 2 and 3 (the future):

[3]The teams providing the software a given program depends on or uses
[4]The teams using or depending on a given program

- What is our project's reputation? What is it known for?
- What do competitors and coopetition respect and envy the most?
- How and where are contributors/employees performing work and interacting with users?
- What is the user's experience?
- What major contributions have been made to the communities contributed to?
- What are contributors/employees saying to their closest friends and family about what it is like to contribute to our project?
- What new ideas, businesses or ventures have been pursued?
- What is going on in the marketplace?
- How are contributors/employees interacting with users? How are services being provided?
- What is the mood of the team?
- What are contributors/employees, users and other stakeholders experiencing and feeling?

And for stage 4 (the evaluation):

- What specific images or feelings emerged for you as you listened/read it?
- What key messages does it convey to you?
- What needs further clarification, explanation or elaboration?
- What, if anything, is missing from it that you believe is important to include?
- What do we need to do to translate this vision into action and make it a reality for our organization?

The questions for the first three stages are made available to the team in a shared space. Everyone is encouraged to add their answers and ideas. After a previously announced point in time, (some) team members come together to evaluate the results and decide which of them are useful/needed for the vision. Some key points should emerge from which a draft of a vision can be formulated. This draft should then be refined with the wider community via a mailing list discussion, collecting input on a wiki page, or some similar means. The questions of stage four can be used for this and the evaluation. Once a vision is agreed upon, it should be communicated on the project's website, in quarterly foundation reports or other appropriate locations.

For Amarok and Halo, the vision creation will be done using a simple wiki page that holds the questions and that is open to editing by the whole team.

### 5.3.2 Collecting updates for a vision

Projects change and because of that their vision is subject to change. In order to stay relevant, the vision needs to adapt to these changes. A way to provide input for changes therefore needs to be available. The proposed updates need to be checked regularly for inclusion in the vision.

Proposals for updates can be put forward on a mailing list, a wiki page or during regular meetings.

Proposals on a mailing list might be forgotten when the time comes to review the vision. If they are collected during regular meetings they are likely to be forgotten as well by the time it comes to review them. Therefore, for Halo and Amarok the input for updates of the vision will be collected on the wiki on the associated discussion page for the vision. Review will happen annually.

## 5.4 Collaboratively creating a roadmap

The analysis showed that both projects are lacking a public roadmap and a clear way to influence the direction of the project. A complete and well-maintained roadmap helps everyone involved in the project – from the beginning (receiving input and help), through the middle (coding) and all the way to the end (writing release notes). In a transparent and open project everyone should be able to contribute to the roadmap, be it in the form of ideas, opinion, advice or code. However, that does not mean that everyone can change the roadmap at will. A process is needed that allows easy contributions as well as clear guidelines detailing how each step can be influenced. For this thesis we will concentrate on feature roadmaps and will not include bugs.

### 5.4.1 Life cycle of a feature request

A feature request can be in different states. It has a life cycle from the point it is written down to the point where it is either implemented or discarded.

Table 5.2 lists the possible states a feature request can be in and Figure 5.3 shows the possible transitions between each of them.

These states will be used during the implementation of the feature request system for both projects.

### 5.4.2 Communicating expectations around feature requests

Communicating expectations regarding feature requests is non-trivial. It needs to be clear which of them are considered for inclusion in one of the next releases and

| State | Meaning |
|---|---|
| draft | feature request is being written and not yet ready for review |
| needs review | feature request is finished and can be reviewed |
| reviewed | feature request is reviewed and waiting for someone to implement it |
| in progress | feature request is being implemented |
| finished | feature request has been implemented |
| discarded | feature request has been reviewed but it was decided that it should not be implemented |

**Table 5.2:** Feature request states and their meaning

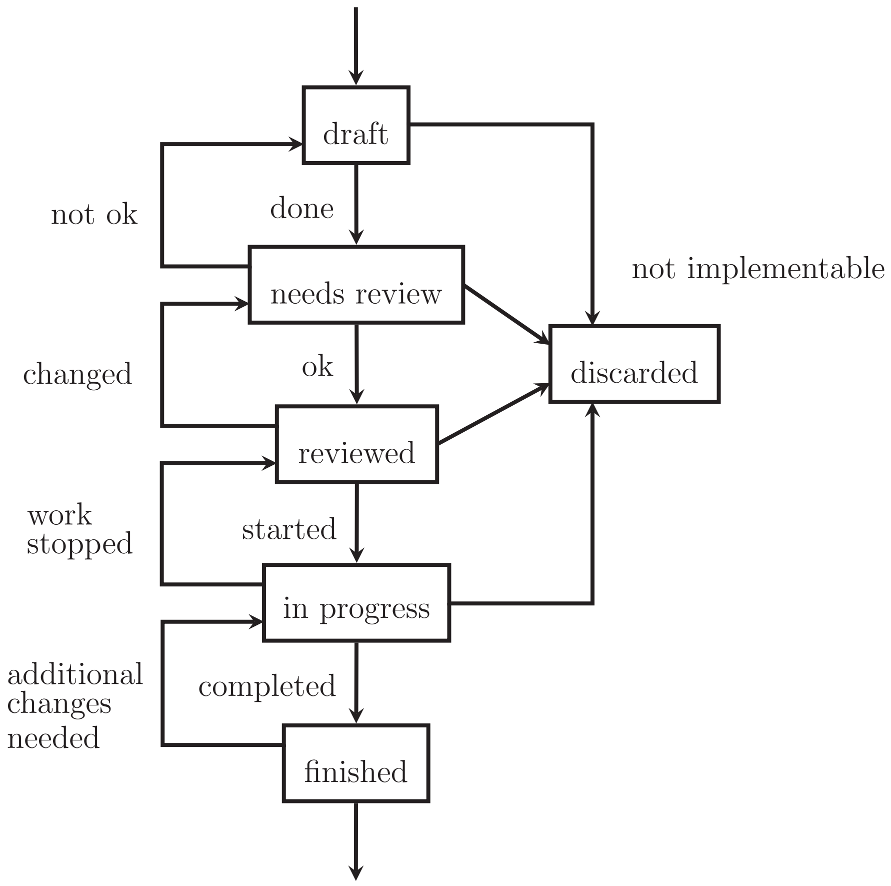


**Figure 5.3:** Feature request states and their transitions

which are not. Users should not be left waiting to see a feature implemented that the core team will never implement themselves or for which code contributions will never be accepted. At the same time, it can be hard to predict the availability of contributors and their ability to finish a given feature for the next release. This needs to be clearly communicated.

A classification of feature requests into categories P1 to P5 based on their priority is proposed. Table 5.3 shows a possible mapping of feature request priorities that clearly communicates expectations to users and potential contributors.

| Priority | Meaning |
|---|---|
| P1 | will be implemented by the core team |
| P2 | will potentially be implemented by the core team |
| P3 | will not be implemented by the core team but patches will be happily accepted |
| P4 | undecided or disputed |
| P5 | will not be implemented and patches will likely not be accepted into the main repository |

**Table 5.3:** Feature request priorities and their meaning

This mapping will be used during the implementation of the feature request system for both projects.

### 5.4.3 Scope and difficulty of feature requests

Feature requests can have different scopes and difficulties. It is hard for a new contributor to assess scope and difficulty of a given feature request in a code base that is new to him.

Table 5.4 proposes a classification that makes assessment easier. It allows quick judgement of the feature request, since a low number indicates a task with smaller scope and lower difficulty while a high score indicates a broad scope and higher difficulty.

| Difficulty / Scope | easy | medium | hard |
|---|---|---|---|
| small | 1 | 3 | 6 |
| medium | 2 | 5 | 8 |
| large | 4 | 7 | 9 |

**Table 5.4:** Feature request classification by scope and difficulty

This classification will be used during the implementation of the feature request system for both projects.

### 5.4.4 Claiming and assigning feature requests

It needs to be possible to claim and assign proposed feature requests.

Each feature request should have a creator, driver and assignee. They can but do not have to be the same person. The creator is the person that created the feature request, the driver is responsible for bringing it to completion (e.g., finding the right developer, collecting feedback) and the assignee is the person implementing the feature request. To avoid “cookie licking” it should be possible to automatically expire this to allow another contributor to step up. This will be done by using the number given in Table 5.4 as the number of weeks the feature request is marked as claimed since the last update.

It has to be clear how to go about getting involved with each of the roadmap's items. This could be done by: emailing the assignee or driver; contacting the team via their mailing list, forum or IRC channel; or even leaving a comment on the proposal page. The implementation depends on the team's preferred means of communication.

For Halo and Amarok, communication is supposed to happen on the talk page of the feature request. If there is no reply to that after a reasonable amount of time the driver and assignee of the feature request should be contacted by email.

### 5.4.5 Feature request page and overview

Based on the previous discussion, a feature page is needed. It should contain all of the relevant information for a feature request.

A mock-up for such a feature page can be seen in Figure 5.4. The status information includes the responsible people/teams, targeted release, how much of the feature is completed, scope/difficulty, priority and when the page was last updated. The extended information section lists more detailed information, such as the rationale for the request, use-cases, mock-ups, associated discussions (mailing list and forum), bug reports and repository branches. In the discussion section, people should be able to express their opinion/support and to provide input on the feature request.

For an easy overview a roadmap page is needed that lists all of the feature requests currently being worked on or under consideration.

A mock-up of the roadmap page can be found in Figure 5.5. It lists the release currently being worked on first, then the next release and then everything that is planned for one of the later releases. At the end, the roadmaps of past releases are displayed to visualize the progress the team has made and keep a record for future reference. Figure 5.6 shows the details of the roadmap page of the release currently being worked on.

## 5.5 Quality Assurance

The analysis of both projects has shown that they are having problems with quality assurance. We have to accept the fact that most software has bugs. The power of Free Software lies in the large audience that can help find and fix them. There are two important reasons why code that reaches the user still has bugs: i) it was not tested by a large enough audience to find them (mainly a problem for

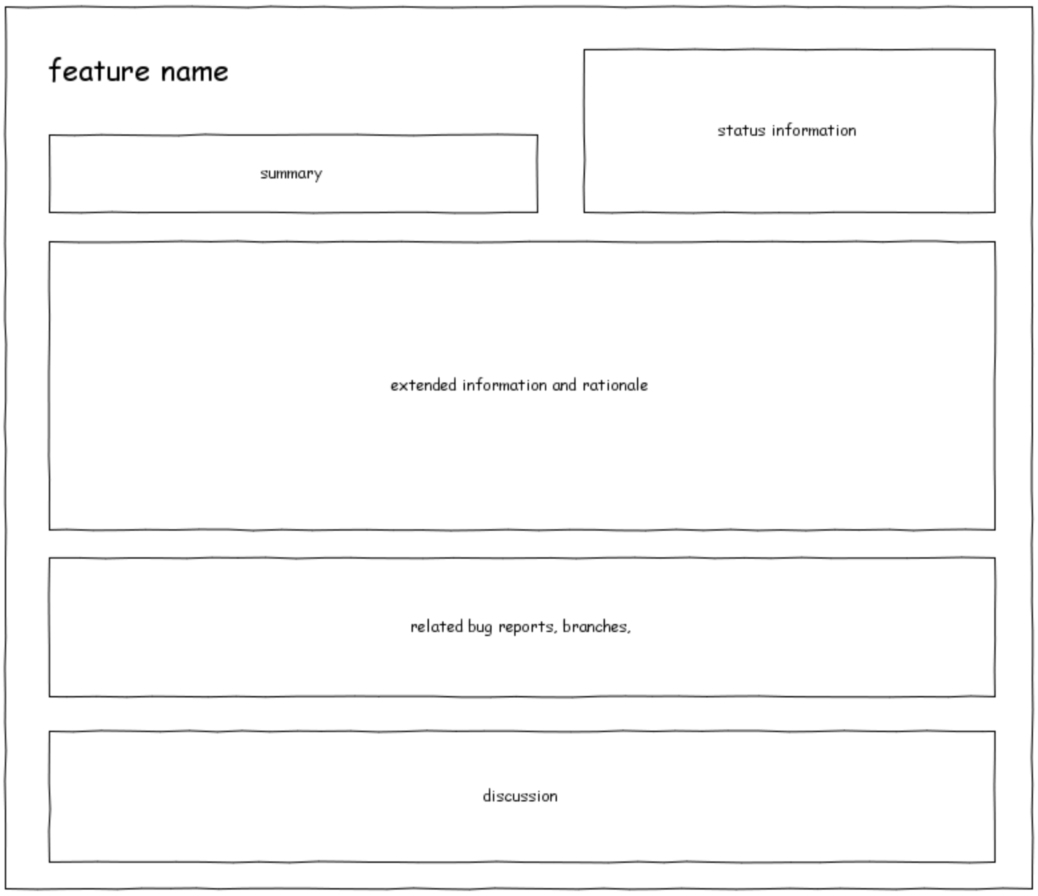

feature name
status information
summary
extended information and rationale
related bug reports, branches,
discussion

**Figure 5.4:** Mock-up of the proposed feature page

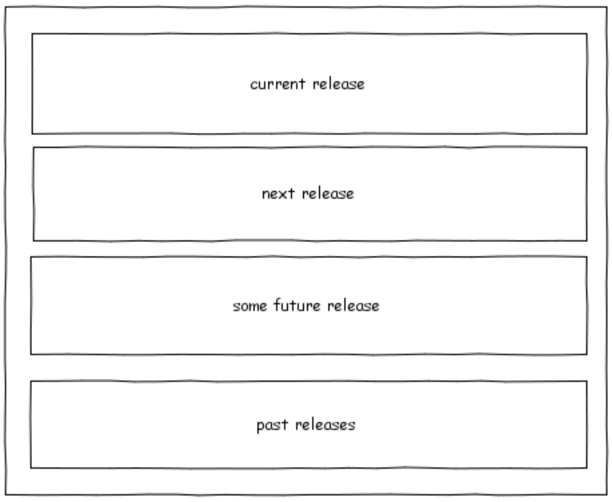
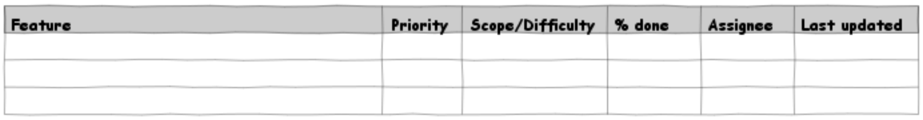


**Figure 5.5:** Mock-up of the proposed roadmap page

Halo); and ii) the bugs were found but no-one had the resources (e.g., time, skills, motivation) necessary to fix them, or the existence of the bugs was not brought to the attention of the appropriate person capable of fixing them (mainly a problem for Amarok).



| Feature | Priority | Scope/Difficulty | % done | Assignee | Last updated |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |

**Figure 5.6:** Mock-up of the current release section of the proposed roadmap page

### 5.5.1 Encouraging testing by a large group

To address Halo's problem, where code is not tested by enough people before a release, a number of measures can be taken.

A clear feature freeze and a schedule for test-releases for public testing needs to be created, communicated and followed. If the development branch starts to stabilize at certain predictable times, interested users will come and test it.

A list of milestones for a release schedule that have proven successful is outlined in Table 5.5. Amarok is already using this system and Halo is slowly transitioning to it with the introduction of public testing releases.

| Milestone | Meaning/intended audience |
|---|---|
| technology preview | showcase, likely very unstable |
| alpha | for very early testers, likely contains a lot of bugs |
| beta | for average testers, likely still contains a significant amount of bugs |
| release candidate | for late testers, should only contain very few bugs |
| final | production quality for users that need stability |

**Table 5.5:** Release milestones and their idealized meaning and intended audience

The users already willing to help with stabilizing the code-base need to be supported in their efforts.

It is a good idea to make test cases publicly accessible for testers and allow them to indicate which of the test cases they executed and what the results were.

### 5.5.2 Making problematic areas more visible

To address Amarok's problem, where areas of the code-base are being abandoned and this fact is not being noticed, the previously mentioned activity indicator (Figure 5.7) can be used.

The activity indicator is a colour-coded slider that shows how active a contributor is. This would largely simplify identification of unmaintained areas of the code-base and could help automate this process. Once it is clear that a contributor has

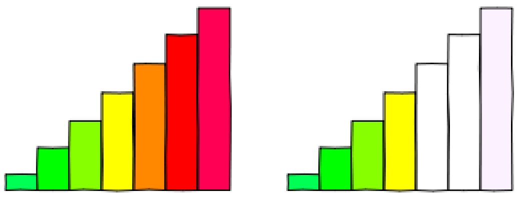


**Figure 5.7:** Mock-up of proposed activity indicator for a very active (left) and moderately active (right) person

not been active for two weeks, it is reasonable to assume that someone else needs to step in.

The value displayed by the activity indicator is based on a contributor's last IRC login, emails to mailing lists, forum posts, comments in bug reports, commits, wiki edits and blog entries. It also takes into account vacations or time-outs that the contributor himself indicates.

The activity indicator will not be implemented in this thesis because the large amount of diverse sources that would need to be taken into account are outside of its scope. It would however be a valuable addition to the tool set of Free Software projects and is therefore proposed for future work in this area.

The other thing that improves the situation is to regularly bring important issues to the attention of the team.

Regular updates on the quality status can be performed by periodic automatic or non-automatic emails, dashboards or shared bug queries. Useful information on a dashboard includes the build status of the current development branch and the number of bug reports considered critical. For regular push-updates on the quality status, it is important to find a balance between alerting the team fast enough (and with enough accompanying information) about critical issues and notifying them too often to the point that they begin ignoring these updates.

Both Amarok and Halo will use regular automatic notifications by email since this is the way it is most likely to be noticed. The notifications will be sent once every week by Bugzilla to the respective developer mailing list.

## 5.6 Building blocks

The proposed tools and processes provide the building blocks for a development process that is both transparent and collaborative. Figure 5.8 shows how they relate to each other.

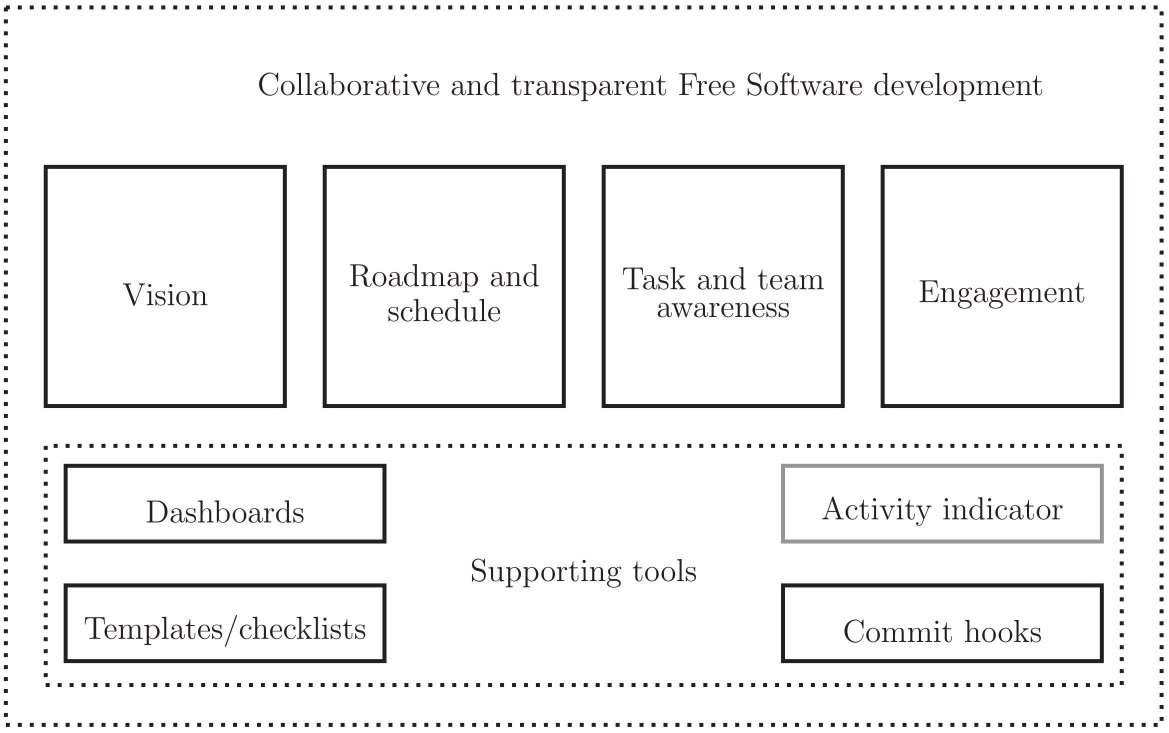


**Figure 5.8:** Building blocks (lines) and grouping (dotted lines) of collaborative and transparent Free Software development

*"They always say time changes things, but you actually have to change them yourself."*

Andy Warhol

# 6 Implementation of an Improved Development Process

This chapter explains the concrete implementation of the design proposed in Chapter 5 and their application in Halo and Amarok. Some parts are only described for one of the projects where they are identical for both of them. Other parts are implemented for both projects to show how the solution looks like in a pure MediaWiki (MW) environment and in a system using Semantic MediaWiki (SMW) and Halo.

## 6.1 Roll-out scenario

The building blocks outlined in the previous chapter are largely independent of one another. Each of them has therefore been introduced to the team separately to see how they are accepted, ease the transition and see where they need adjustments. Where applicable they have first been deployed for the Amarok and then Halo team.

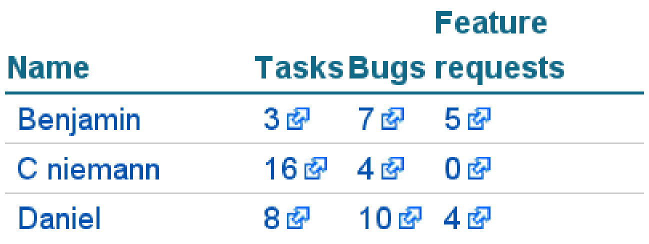
## 6.2 Communication and coordination in distributed teams

A number of tools and processes have been implemented, augmented or made more visible to improve the communication and collaboration in both teams.

### 6.2.1 Team dashboard

The team dashboard was implemented for Halo. It contains a table with the number of bugs, task and feature request for each team member, the current build status, the bug status of the current release and feedback from social media websites.

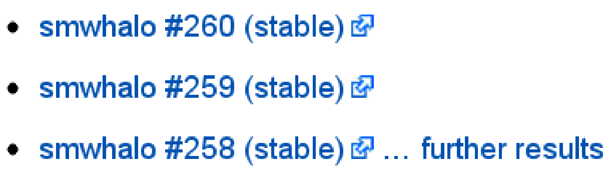
To create the table listing open bugs, tasks and feature requests the wiki is first queried for each ontoprise employee[1]. Then Bugzilla is queried via a web service for the open entries for each of the employees. A table is created with a row for each result of the ask query. The mark-up for this is shown in Listing C.1. The template of each of those rows can be seen in Listing C.2 (without links) and Listing C.3 (with links). {{{1}}} in the template is replaced by the result of the ask query. This can either be "User:accountname"[2] or "accountname". For the web service queries the namespace must be removed, which is done in "Remove_user_namespace" (Listing C.4) where "User:" is replaced with an empty string if the string it was given contains it. In "BugzillaQuery" the actual query to Bugzilla is executed. It is called with two arguments: the type of report (bug, task, feature request) and the account name of the person in question. {{{1}}} and {{{2}}} in Listing C.5 are then replaced by them respectively. The result of all this is shown in Figure 6.1.



| Name | Tasks | Bugs | Feature requests |
|---|---|---|---|
| Benjamin | 3 | 7 | 5 |
| C niemann | 16 | 4 | 0 |
| Daniel | 8 | 10 | 4 |

**Figure 6.1:** Team dashboard table for Halo

The build status is displayed using an Really Simple Syndication (RSS) feed as the source. It is published by Hudson, the continuous integration server used for Halo. Again a web service is used for that. Listing C.6 shows the web service call and Figure 6.2 the result.



- smwhalo #260 (stable)
- smwhalo #259 (stable)
- smwhalo #258 (stable) ... further results

**Figure 6.2:** Build status results from Hudson for Halo

The bug status chart (Figure 6.3) is created using three web service calls to Bugzilla which are then shown using a bar-chart result printer. It works as follows.

[1]Other criteria are of course also possible.
[2]"User" is the namespace of all user pages in the wiki.

First the number of open and closed bugs for the current development version as well as their sum is queried via a Bugzilla web service. Listing C.8 shows an example query. These results are stored in the wiki. In a second step these three numbers are queried using an ask query and displayed using a bar-chart result printer. Listing C.7 shows the query.

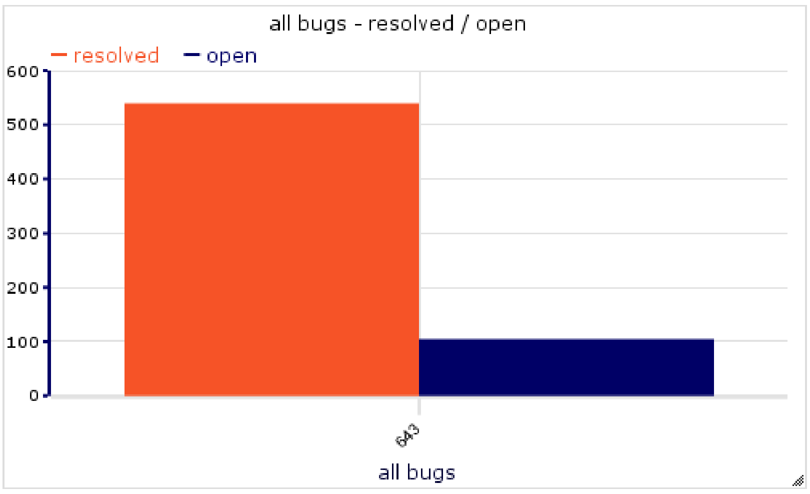


**Figure 6.3:** Bug chart on the team dashboard for Halo

The latest user feedback is gathered via an RSS feed from a Google real-time search. It searches for “Halo SMW+” since each search term individually returns few results relevant for the project, but rather returns results for the games Halo and Super Mario World. The mark-up is the same as in Listing C.6 but with a different URL.

### 6.2.2 Release dashboard

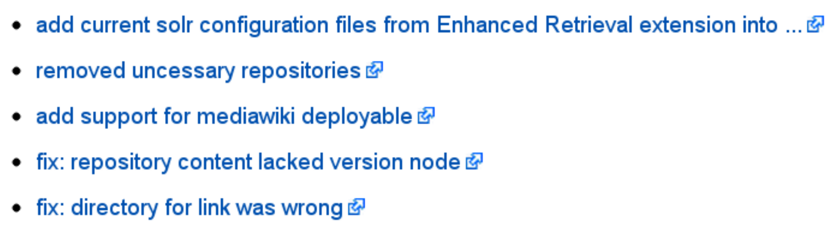
The release dashboard was implemented for Halo. It contains some general information[3], the latest commits, the bugs already closed and still open for the current release and features for the current release.

The latest commits to Halo’s Apache Subversion (SVN) repository are shown based on an RSS feed from websvn[4]. The mark-up is the same as in Listing C.6 but with a different URL. The result is shown in Figure 6.4.

The bugs are fetched via a web service call to Bugzilla and displayed in a table. The mark-up is shown in Listing C.9 (open bugs) and Listing C.10 (closed bugs). “CurrentVersionInBugzilla” is a template that only contains the release number of the next release in Bugzilla and is updated every release for convenience and maintainability.

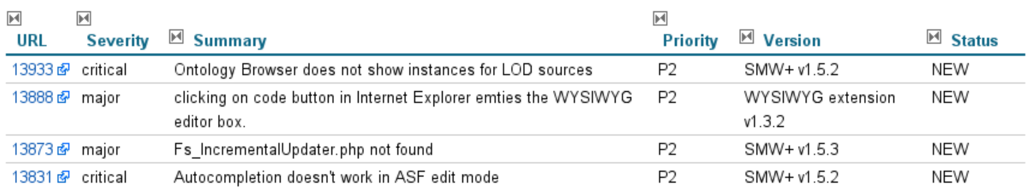
[3]Version number and planned release date of next release, development stage, links to important development resources

[4]Repository viewer to browse the content of an SVN repository via a website



- add current solr configuration files from Enhanced Retrieval extension into ...
- removed uncessary repositories
- add support for mediawiki deployable
- fix: repository content lacked version node
- fix: directory for link was wrong

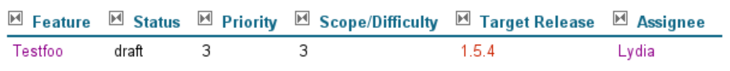
**Figure 6.4:** Latest commits to Halo's repository



| URL | Severity | Summary | Priority | Version | Status |
|---|---|---|---|---|---|
| 13933 | critical | Ontology Browser does not show instances for LOD sources | P2 | SMW+ v1.5.2 | NEW |
| 13888 | major | clicking on code button in Internet Explorer emties the WYSIWYG editor box. | P2 | WYSIWYG extension v1.3.2 | NEW |
| 13873 | major | Fs_IncrementalUpdater.php not found | P2 | SMW+ v1.5.3 | NEW |
| 13831 | critical | Autocompletion doesn't work in ASF edit mode | P2 | SMW+ v1.5.2 | NEW |

**Figure 6.5:** Table of bugs for the current Halo release

Features that are tracked in Bugzilla (mainly feature requests from users) are queried via a web service call similar to the one for the bugs table. The features tracked in the wiki are queried using an ask query and shown with a table result printer. The mark-up can be seen in Listing C.11 and the result in Figure 6.6.



| Feature | Status | Priority | Scope/Difficulty | Target Release | Assignee |
|---|---|---|---|---|---|
| Testfoo | draft | 3 | 3 | 1.5.4 | Lydia |

**Figure 6.6:** Table of features tracked in the wiki for Halo

### 6.2.3 Release calendar

Amarok's existing release calendar was placed more prominently on the main website. Halo's calendar is still being worked on for the next releases as the release dates are not known yet. Once they are known they will be shown using the Semantic Results Formats extension for SMW. It is able to show the result of a query in the wiki in different output formats, among them a calendar.

### 6.2.4 Code ownership

A MAINTAINERS file has been added to Amarok's git repository containing the following text:

> "Amarok does not have clear maintainers of any part of the codebase. You are welcome to work on any part of it provided you coordinate with others working in that part.
>
> If you need to contact someone about a particular part it is best to contact the developer mailing list at amarok-devel@kde.org or the assignee of the corresponding component on http://bugs.kde.org directly."

### 6.2.5 Checklists

Commit checklists were developed for both projects to help new contributors avoid common mistakes in their first commits. Halo's new checklist has been added to the developer section of its wiki. Amarok's new checklist has been added to a text file in the HACKING directory in its git repository along with other similar files. The checklists can be found in Chapter D.

### 6.2.6 Commit hooks

The existing commit hooks from KDE for closing bugs and emailing commits are going to be used for Halo as well. However they still need to be set up by the responsible system administrator. They are available in KDE's SVN repository.

## 6.3 Collaboratively working on a vision

For the vision creation in Amarok a wiki page was created with the questions of stage 1 to 3 in Section 5.3. It was seeded with initial thoughts for the first three questions to encourage participation. Then an email was sent to the two team lists asking contributors to participate in the vision creation process by adding their thoughts on each of those questions within one week. After that, a structured meeting on Internet Relay Chat (IRC) was scheduled to go through each of the questions again and discuss the ideas and add important points that were missing so far. This meeting was strictly limited to a length of two hours which helped tremendously to keep it on track. After the meeting the wiki page was updated with the results and three people created an initial draft of the vision. It was sent to the mailing list to receive feedback. After minor edits it was accepted as the vision for Amarok:

> "The Amarok team strives to develop a free and open music player that is innovative and powerful, yet easy to use. Amarok helps rediscover music by giving access to a vast amount of different music sources and related information. In a world where music and computing are everywhere, Amarok aims to provide the best music listening experience anywhere, anytime. The Amarok team promotes free culture. Amarok makes people happy."

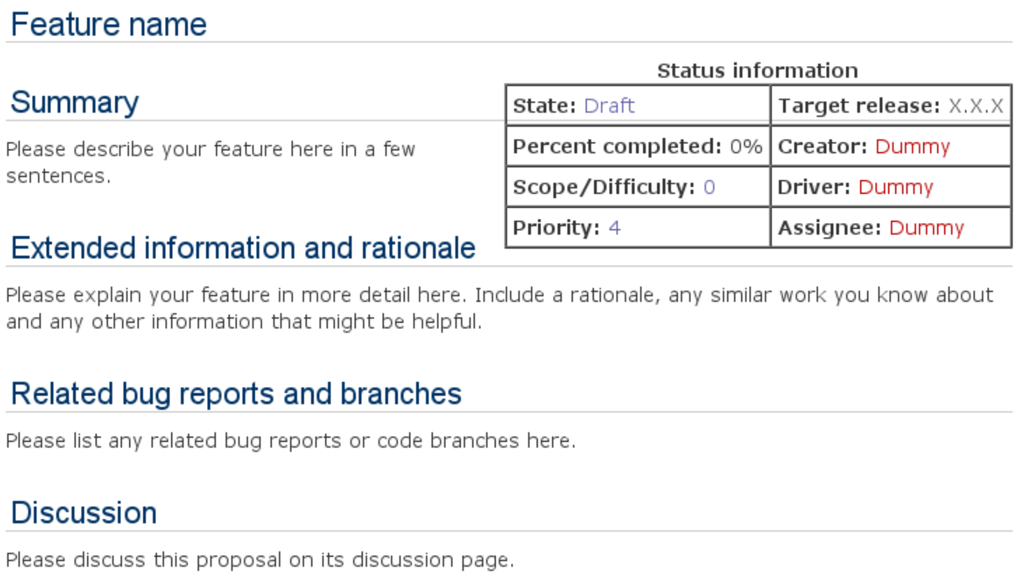
It has been posted to a wiki page and a link to it was added to the main website of the project. A note pointing to the discussion page of the vision page has been added to encourage update proposals for the vision.

The same process was started for Halo but failed to produce a tangible result so far because people did not participate in the first step, i.e., answering the questions in the wiki. Possible reasons for this are a lack of time or the fear that their ideas might not align with those of the management.

## 6.4 Collaboratively creating a roadmap

Two feature tracking systems have been implemented – a simple version for Amarok and a more elaborate version for Halo.

Amarok's version is a simple wiki text template that can be used in a new page for each new feature. Figure 6.7 shows an example feature page using it. The wiki mark-up for it can be found in Listing C.12. Listing C.13 shows an example of how it will be used in a wiki page. These features are then aggregated on a wiki page sorted into four categories (next release, some future release, unsorted, old releases).



Feature name

Summary

Please describe your feature here in a few sentences.

Status information

| | |
|---|---|
| **State:** Draft | **Target release:** X.X.X |
| **Percent completed:** 0% | **Creator:** Dummy |
| **Scope/Difficulty:** 0 | **Driver:** Dummy |
| **Priority:** 4 | **Assignee:** Dummy |

Extended information and rationale

Please explain your feature in more detail here. Include a rationale, any similar work you know about and any other information that might be helpful.

Related bug reports and branches

Please list any related bug reports or code branches here.

Discussion

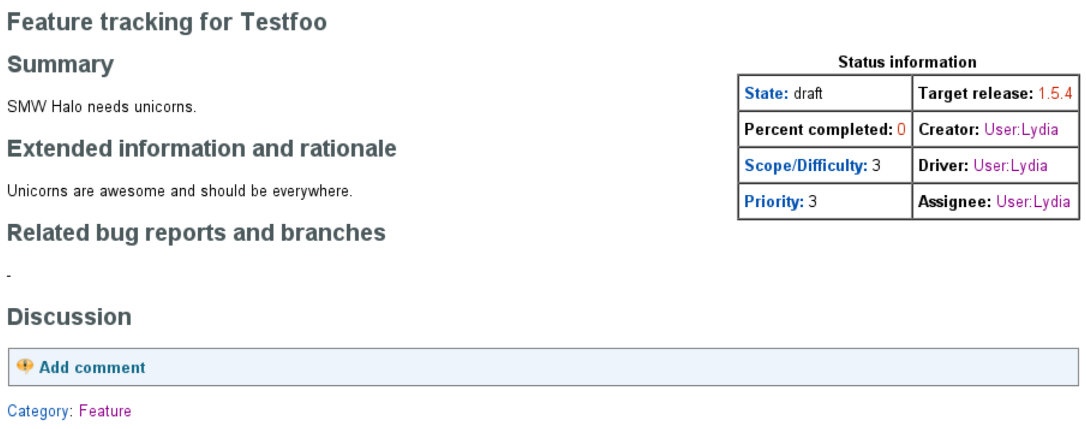
Please discuss this proposal on its discussion page.

**Figure 6.7:** Example of a wiki page for one feature for Amarok

An example of Halo's feature tracking system can be seen in Figure 6.8. It is built using the MW extension Semantic Forms. Two things are necessary to build the system, a form and a template. The form is used to create and edit the feature pages. It is shown in Figure 6.9. The mark-up of the form is shown in Listing C.15. The information entered via the form is then saved on the wiki page as a template call with the entered data as parameters. The mark-up of the template can be seen in Listing C.14.

## 6.5 Quality Assurance

Steps have been taken to improve the quality of both programs.


Feature tracking for Testfoo

Summary

SMW Halo needs unicorns.

Extended information and rationale

Unicorns are awesome and should be everywhere.

Related bug reports and branches

-

Discussion

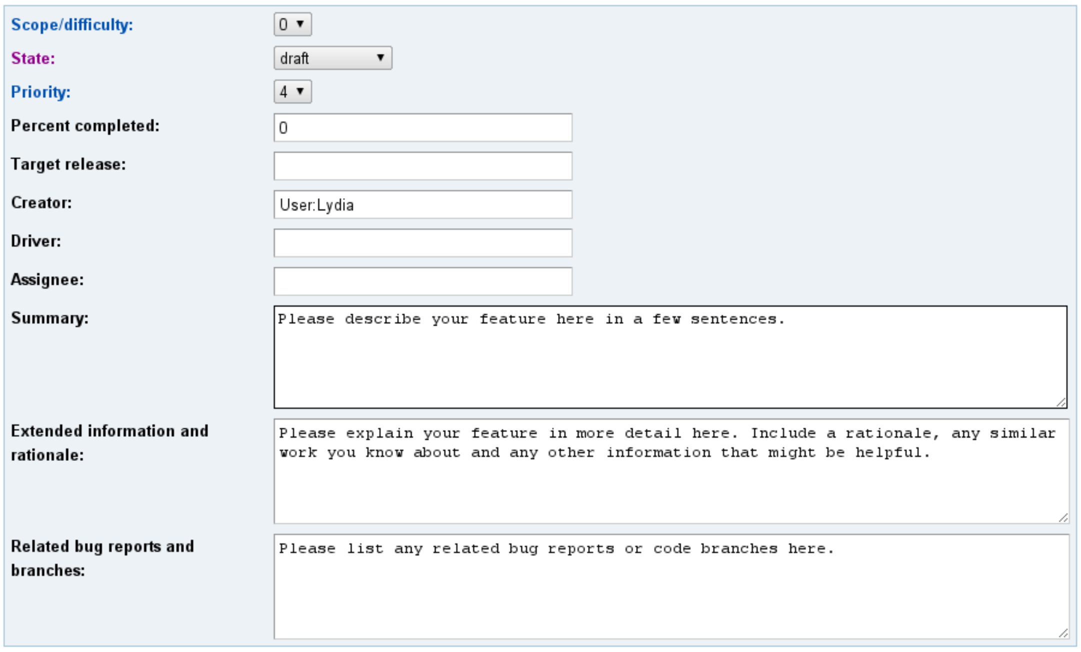
Add comment

Category: Feature

Status information

| | |
|---|---|
| **State:** draft | **Target release:** 1.5.4 |
| **Percent completed:** 0 | **Creator:** User:Lydia |
| **Scope/Difficulty:** 3 | **Driver:** User:Lydia |
| **Priority:** 3 | **Assignee:** User:Lydia |

**Figure 6.8:** Example of a wiki page for one feature for Halo



| | |
|---|---|
| **Scope/difficulty:** | 0 |
| **State:** | draft |
| **Priority:** | 4 |
| **Percent completed:** | 0 |
| **Target release:** | |
| **Creator:** | User:Lydia |
| **Driver:** | |
| **Assignee:** | |
| **Summary:** | Please describe your feature here in a few sentences. |
| **Extended information and rationale:** | Please explain your feature in more detail here. Include a rationale, any similar work you know about and any other information that might be helpful. |
| **Related bug reports and branches:** | Please list any related bug reports or code branches here. |

**Figure 6.9:** Form to create and modify feature tracking pages for Halo

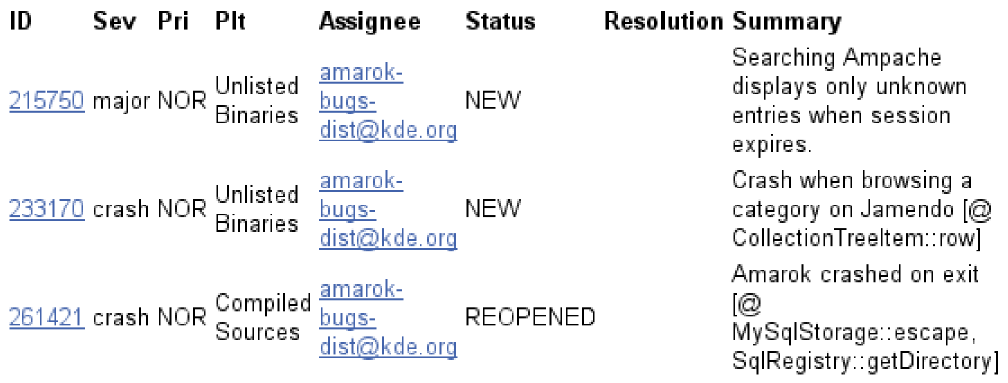
### 6.5.1 Encouraging testing by a large group

For Amarok a test checklist was developed by a student mentored by the author as part of Google's Code-in contest. It guides the tester through all menus and functionality in Amarok. He is then encouraged to file bug reports for any problems he finds. The starting point when developing this checklist was a list of all menu and context menu entries of the program.

For Halo a second and improved public testing contest was started that built on the lessons that were learned during the first one. This includes fixing problems caused by bugs in the used tools like the deployment framework, clarifying instructions for participants and making patches to MW and SMW optional to lower the barrier to entry.

### 6.5.2 Making problematic areas more visible

To make problematic areas more visible to the whole team in Amarok Bugzilla whines were set up. They notify the team weekly about bugs marked as release blockers via an email sent to the developer list. The email content is shown in Figure 6.10.



| ID | Sev | Pri | Plt | Assignee | Status | Resolution | Summary |
|---|---|---|---|---|---|---|---|
| 215750 | major | NOR | Unlisted Binaries | amarok-bugs-dist@kde.org | NEW | | Searching Ampache displays only unknown entries when session expires. |
| 233170 | crash | NOR | Unlisted Binaries | amarok-bugs-dist@kde.org | NEW | | Crash when browsing a category on Jamendo [@ CollectionTreeItem::row] |
| 261421 | crash | NOR | Compiled Sources | amarok-bugs-dist@kde.org | REOPENED | | Amarok crashed on exit [@ MySqlStorage::escape, SqlRegistry::getDirectory] |

**Figure 6.10:** Weekly reminder about bugs that block an Amarok release

*"Your mother was right. It's better to share."*

The Red Hat Story

# 7 Evaluation

In this chapter the tools and processes implemented in Chapter 6 are evaluated. The evaluation was done to answer two main questions:

1. Are the new tools and processes improving collaboration in a Free Software project?
2. Are the new tools and processes increasing transparency in a Free Software project?

The evaluation focuses on subjective rather than objective indicators because they are the more appropriate indicators for the long-term adaptability of the new tools and processes by both teams.

## 7.1 Survey

A survey was sent to 10 people in each project. For Amarok the survey was sent to everyone who took part in the initial interviews. For Halo the survey was sent to 9 of the 10 people who took part in the initial interviews[1] and one additional developer. At the end of the survey period 8 replies were recorded for Halo and 7 for Amarok. Most questions in the survey had to be answered on an ordinal scale from 1 (not at all) to 5 (a lot/absolutely) and the rest in free-text

[1]One of them left the Halo team in the meantime and could not be reached for the evaluation.

fields. The answers were collected via an anonymous online survey to minimize the influence of the personal relationship between the author and the interviewees on the answers. The survey questions can be found in Chapter E.

## 7.1.1 Communication and coordination in distributed teams

The commit checklists are considered helpful for new contributors (Figure 7.1). However, the answers from the Halo team are not entirely positive. The average score[2] from the Halo team is 3.75[3], from the Amarok team 4.14 and combined 3.93. A reason for the lower score from the Halo team might be the lack of regular interaction with new external contributors and the problems they face. In addition the comments indicate that some of the points in the checklist should have explanations.

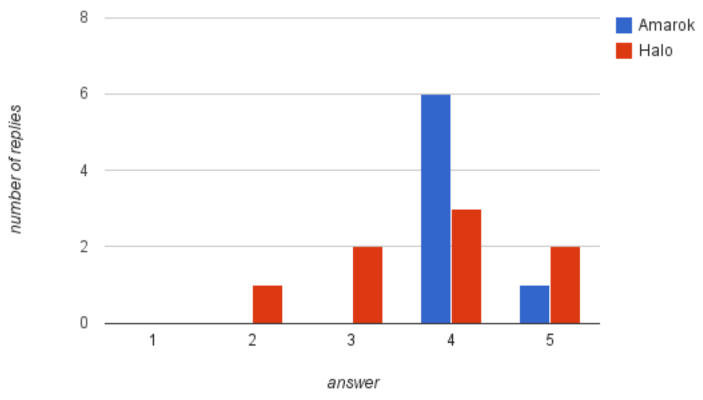


**Figure 7.1:** Result for evaluation question “Will the commit checklist help new contributors?” (Halo, Amarok)

The team dashboard received a positive average score of 3.63 from the Halo team for increasing transparency (Figure 7.2) but only an average score of 2.75 for improving collaboration between contributors or contributors and users (Figure 7.3). This was expected since the team dashboard’s main purpose is to increase transparency. However, it is also intended to improve collaboration by creating team awareness.

The release dashboard is considered suitable for increasing transparency by the Halo team. It receives an average score of 3.5 (Figure 7.4). They seem to doubt its ability to improve collaboration between contributors and contributors and users (Figure 7.5). It received only an average score of 2.88 for both. The reason for this might be that the release dashboard does not improve collaboration directly

[2]The average score is the sum of all scores for a question divided by the number of replies for that question.

[3]Scores are rounded when necessary.

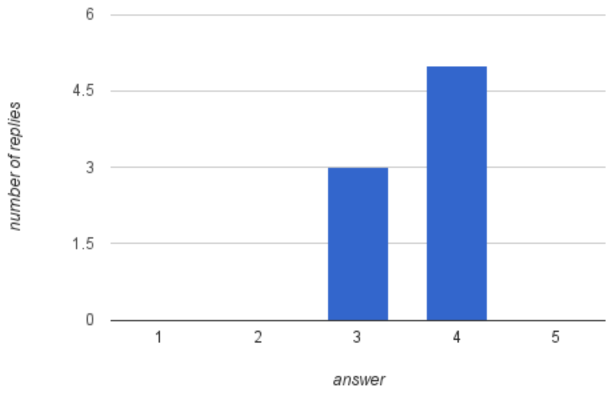


**Figure 7.2:** Result for evaluation question "Will the team dashboard increase transparency?" (Halo)

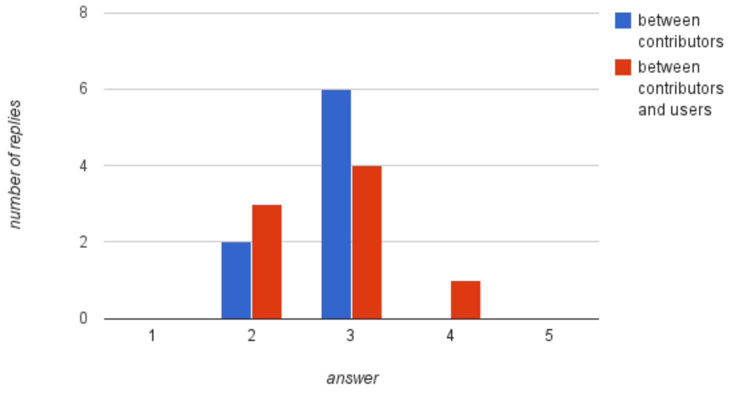


**Figure 7.3:** Result for evaluation questions "Will the team dashboard improve collaboration between contributors/contributors and users?" (Halo)

but rather indirectly by increasing transparency and is therefore not considered as something that improves collaboration by itself.

### 7.1.2 Collaboratively working on a vision

The vision creation for Amarok was a success. During the vision creation process about 70 edits have been made by four people in the wiki. Eight people actively took part in the Internet Relay Chat (IRC) meeting while a few others only followed it. The team members seem to be very pleased with the resulting vision statement and believe it will help the project make better decisions in its next development steps (Figure 7.6, average score of 3.86). The process has already lead to a few additional major decisions on the future direction of the project. The replies in Figure 7.7 indicate that the new vision is going to help increase transparency for users (average score of 3.57) and improve collaboration for con-

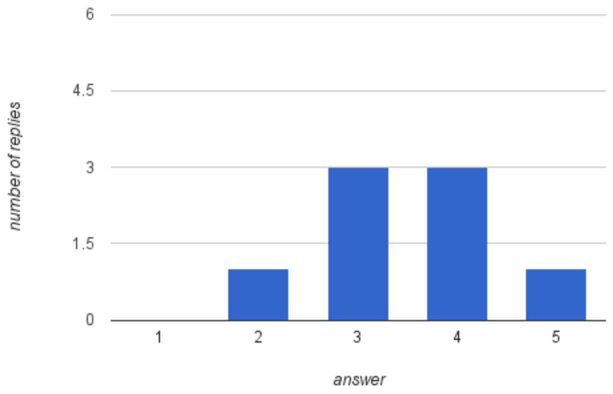


**Figure 7.4:** Result for evaluation question “Will the release dashboard increase transparency?” (Halo)

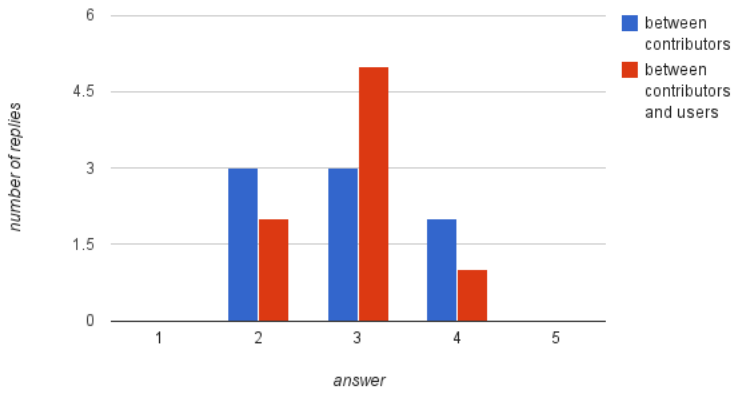


**Figure 7.5:** Result for evaluation questions “Will the release dashboard improve collaboration between contributors/contributors and users?” (Halo)

tributors (average score of 3.43). The replies testify this vision creation process can be used by other Free Software projects as well (Figure 7.8, average score of 4.43) and is considered transparent and collaborative (Figure 7.9, average score of 4 and 4.43 respectively).

The Halo team was asked for reasons why the vision creation failed there so far. The responses indicate a combination of reasons. These include a lack of time, fear of being judged for the given responses in the initial stage, not being used to having a voice in decisions like this and being discouraged by the open-ended questions. This indicates that more support from management is needed both in terms of time commitment as well as encouraging participation.

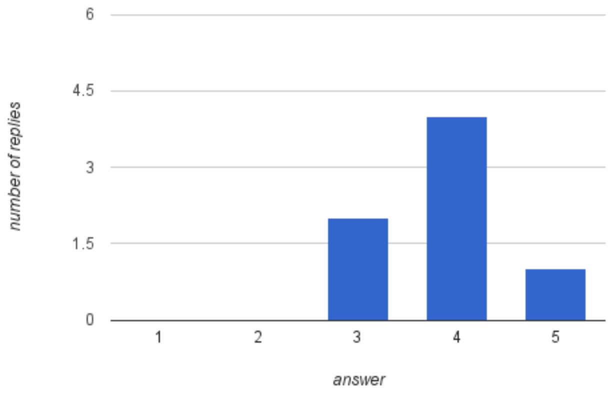


**Figure 7.6:** Result for evaluation question “Will the new vision help the project make better decisions in its next development steps?” (Amarok)

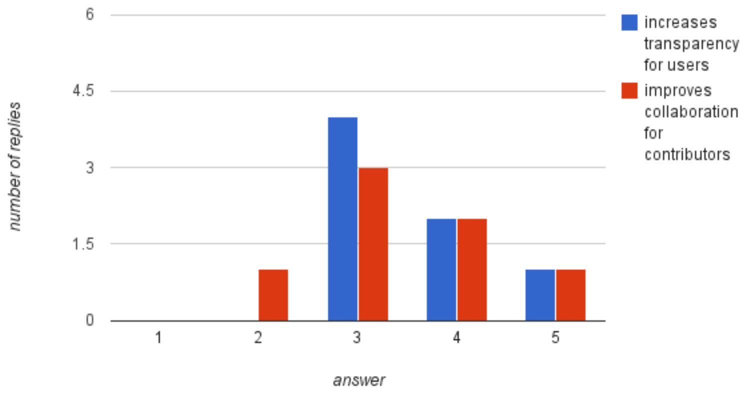


**Figure 7.7:** Result for evaluation question “Will the new vision help increase transparency in the project for users?” and “Will the new vision help improve collaboration in the project for contributors?” (Amarok)

### 7.1.3 Collaboratively creating a roadmap

The new feature tracking form/template and roadmap received a very good average score of 4.07 for being easy to use (Figure 7.10). However, Halo’s system based on Semantic Forms was rated higher (4.25) than Amarok’s template based system (3.86).

Halo’s new feature tracking system received an average score of 3.5 for improving collaboration, both between contributors as well as between contributors and users. Amarok’s received an average score of 3.57 for improving collaboration between contributors and 3.86 for improving collaboration between contributors and users (Figure 7.11). The teams give it an average score of 3.125 (Halo) and 4 (Amarok) for being able to increase transparency (Figure 7.12). Concerns were raised by the Halo team that feature tracking already happens in Bugzilla and the internal wiki.

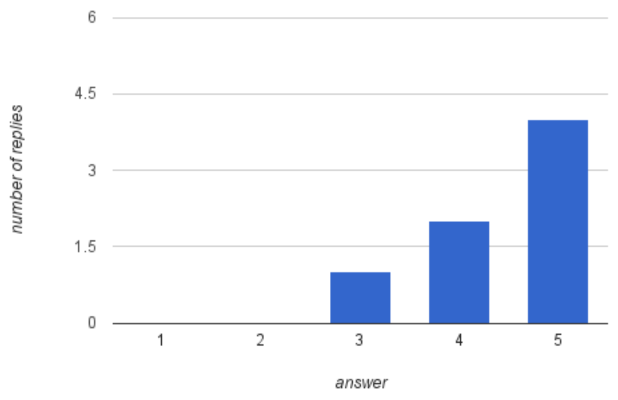


**Figure 7.8:** Result for evaluation question "Do you think the process of creating the new vision can be used by other Free Software projects as well?" (Amarok)

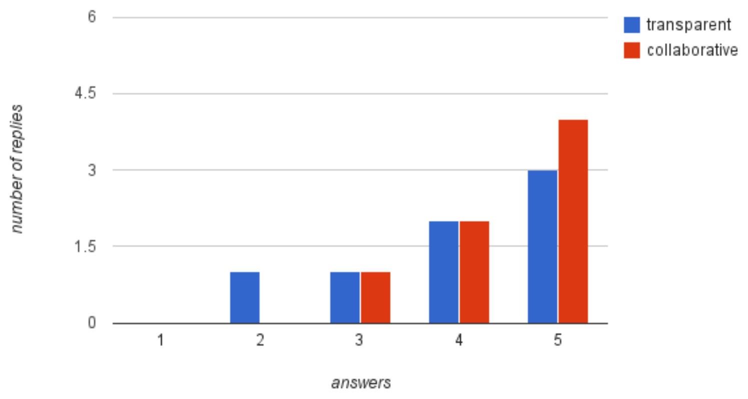


**Figure 7.9:** Result for evaluation questions "Do you think the process of creating the new vision was transparent/collaborative?" (Amarok)

The feature tracking in the internal wiki can hopefully be replaced with this more transparent and public version in the future. Bugzilla should then only be used for feature requests from users who do not intend to be part of the implementation process of their requested feature.

The Halo teams seems divided when it comes to the question if other Free Software projects might be interested in using this feature tracking system as well (Figure 7.13). The reason for this might be the lack of exposure to other Free Software projects and their needs. Their average score is 3.25. The Amarok teams seems to think that the system is indeed of interest for other Free Software projects and gives it an average score of 4.14.

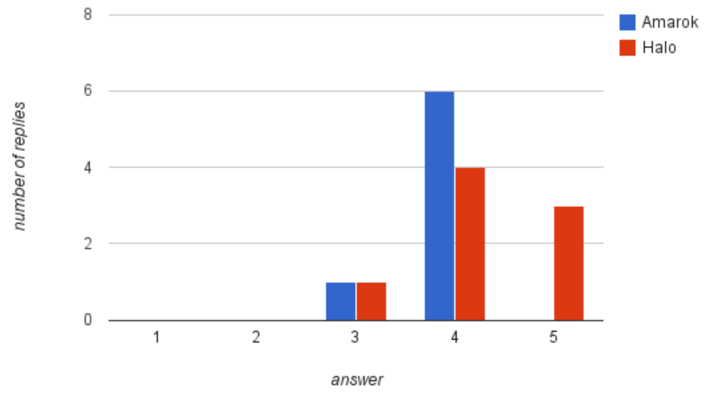


**Figure 7.10:** Result for evaluation question “Do you think the new feature tracking form/template and roadmap are easy to use?” (Halo, Amarok)

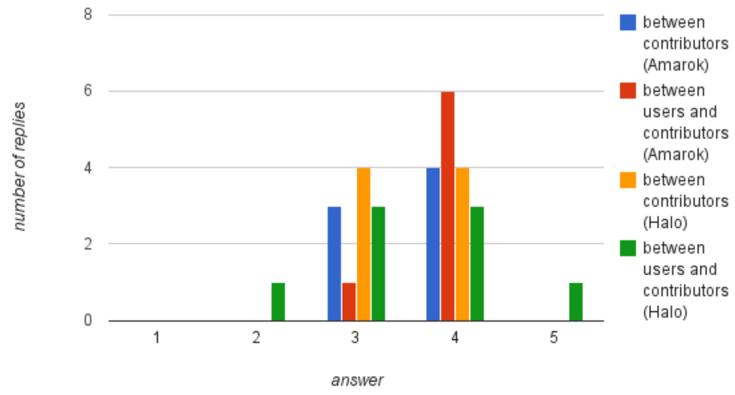


**Figure 7.11:** Result for evaluation questions “Will the new feature tracking form/template improve collaboration between contributors/contributors and users?” (Halo, Amarok)

### 7.1.4 Quality Assurance

All measures to improve the quality situations have been considered very useful (Figure 7.14). Weekly emails about release blockers have been considered most useful before public testing and the testing checklist. They received average scores of 4.29, 4.25 and 4.14 respectively. Public testing is considered very helpful because it provides input from a range of people with very different use-cases, work-flows and expectations.

## 7.2 Changes in the openness of the development processes

Significant parts of the release cycle of both projects have been made more transparent and collaborative, in short more open, during the writing of this thesis.

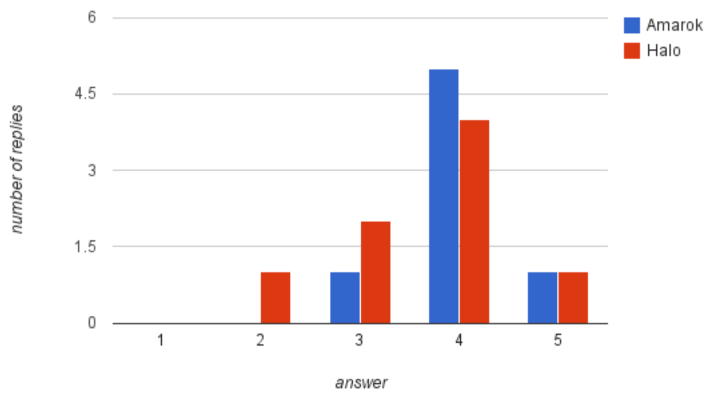


**Figure 7.12:** Result for evaluation question "Will the new feature tracking form/template and roadmap increase transparency?" (Halo, Amarok)

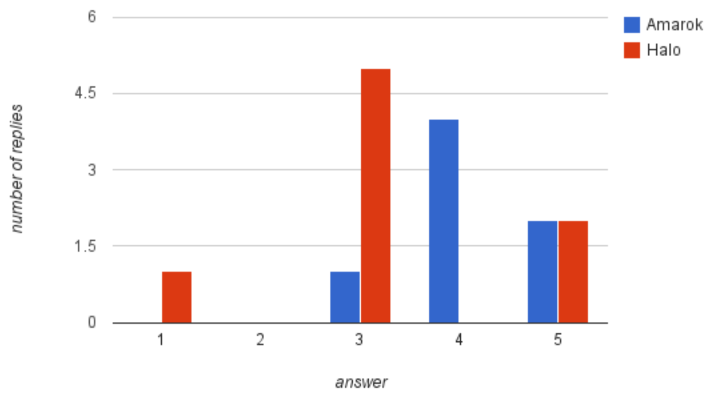


**Figure 7.13:** Result for evaluation question "Do you think other Free Software projects might be interested in this too?" (Halo, Amarok)

Figure 7.15 and Figure 7.16 are the release cycle charts from Chapter 4 adapted to show which stages became more open for each project.

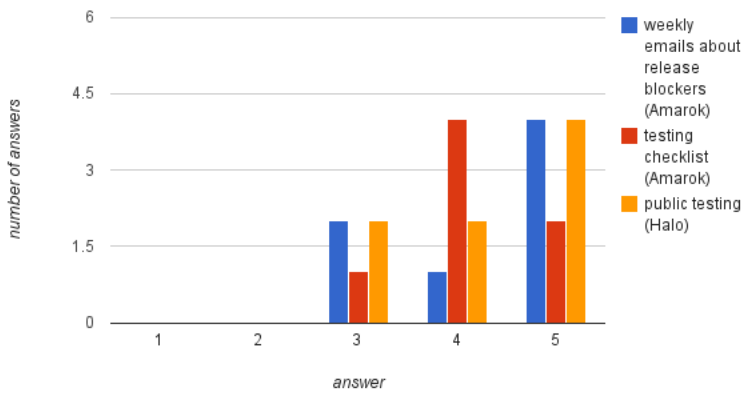


**Figure 7.14:** Result for evaluation questions "Do you consider the weekly emails from Bugzilla to the developer mailing list about release blockers/the testing checklist/the public testing useful for improving the QA situation?" (Halo, Amarok)

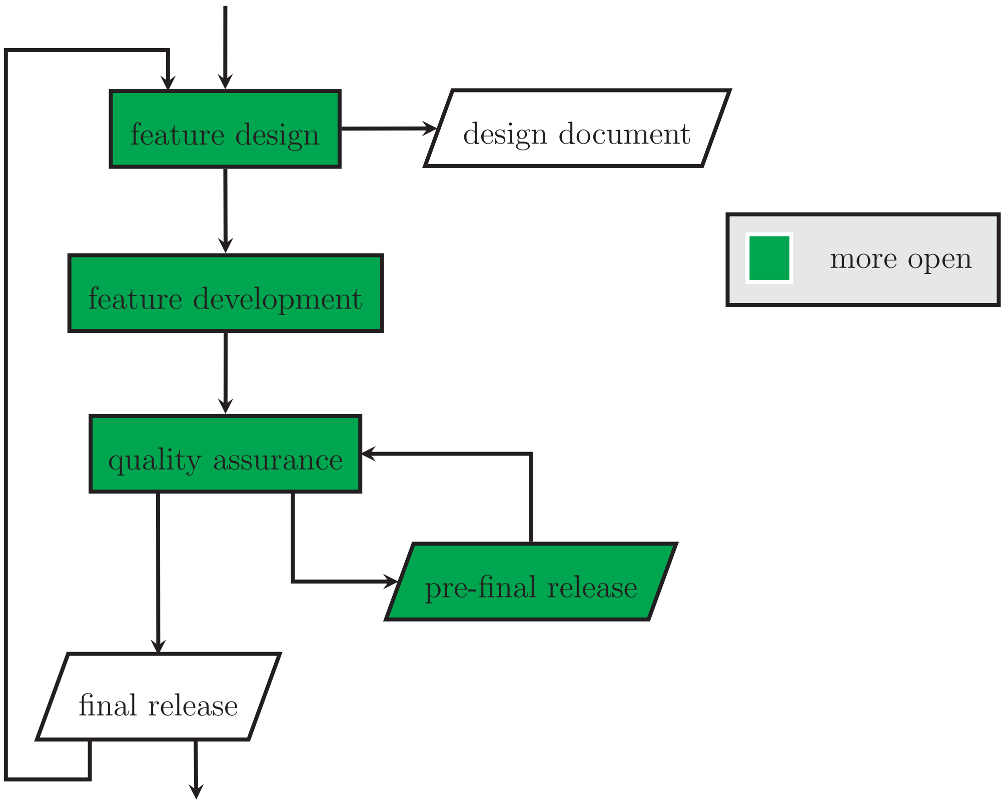


**Figure 7.15:** Changes in the openness of Halo's release cycle

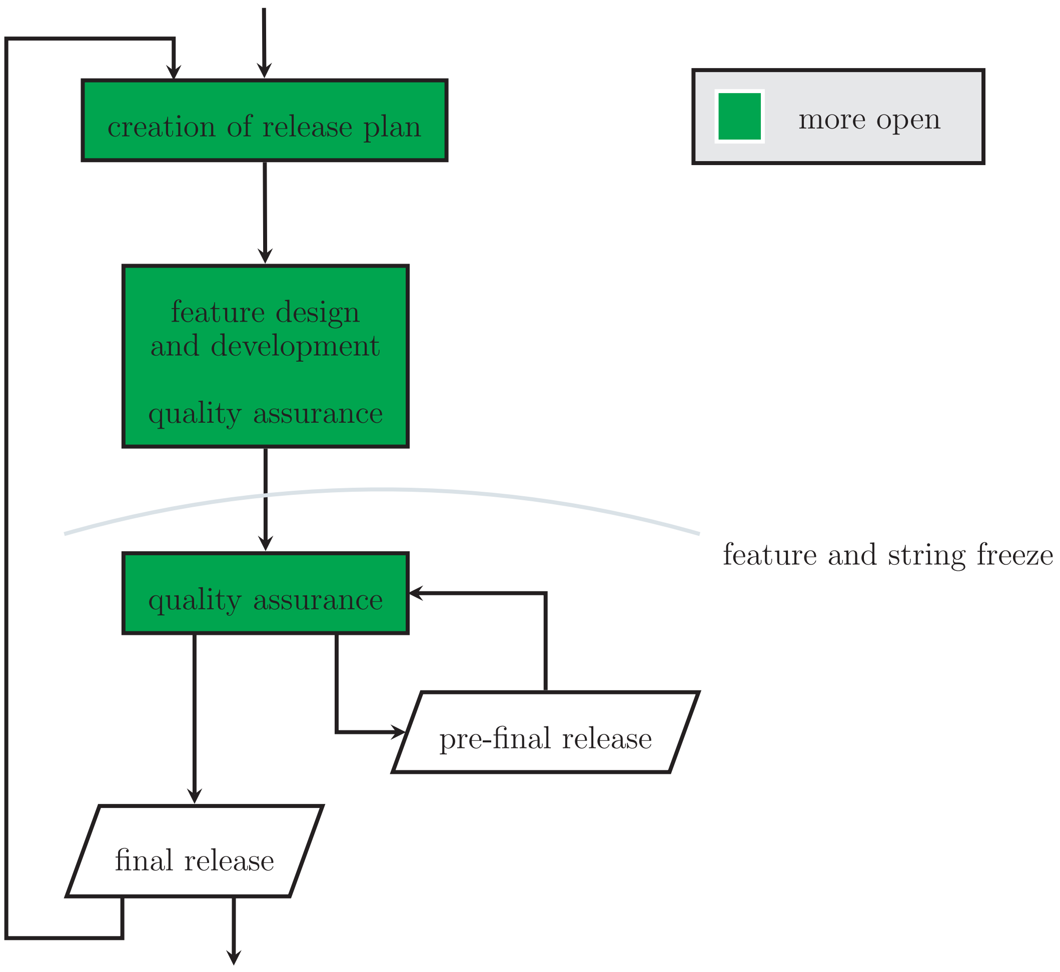


**Figure 7.16:** Changes in the openness of Amarok's release cycle

*"All is riddle, and the key to a riddle is another riddle."*

Ralph Waldo Emerson

# 8 Conclusions and Outlook

The work in this thesis has helped two Free Software projects make large parts of their development process more transparent and collaborative. In addition, it has helped the projects focus on pertinent issues that they are facing; after the initial interviews with participants, feedback was given to the author indicating that the participants appreciated someone asking important questions, even if those questions are not always easy or pleasant. It forced the participants to think about the future: where they want the projects to end up and how to get there. Hopefully, this work will be a significant stepping stone on that path.

Several tools and processes provided in this thesis can be adapted to other Free Software projects as well, and used to make their development more transparent. This includes ways to create a vision for a project, tools to improve collaboration in a team, methods and tools to improve the quality of the project's software and improvements to the simple feature tracking and roadmap creation that are bedrocks of Free Software development. Each of them is in line with how a Free Software project functions.

In the future, the proposed tools and processes should be tested with other teams and will be even more tightly integrated with projects' existing infrastructures. In addition, the proposed activity indicator should be implemented.

# Acknowledgements

I would like to thank the people that accompanied me along the way while I wrote this thesis. There are many of them, but I would like to give special thanks to:

- Basil Ell for always asking the right questions and saving me from my own enthusiasm when necessary
- the Halo team for making this thesis possible
- the Amarok team for running with my ideas and working on the next steps already

*"Only work if you have the feeling, it could start a revolution."*
Joseph Heinrich Beuys

# A

# Release timelines

The following timelines show the public final releases of Halo and Amarok. Technology previews, alpha and beta releases as well as release candidates are not included.

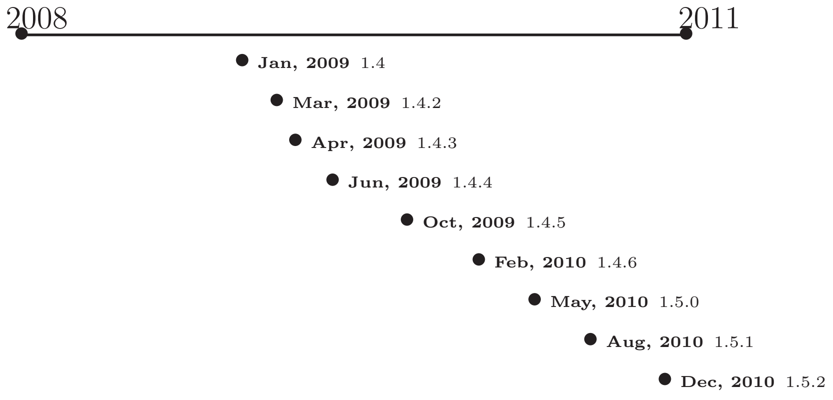


**Figure A.1:** Halo timeline (release dates according to [Ont10])

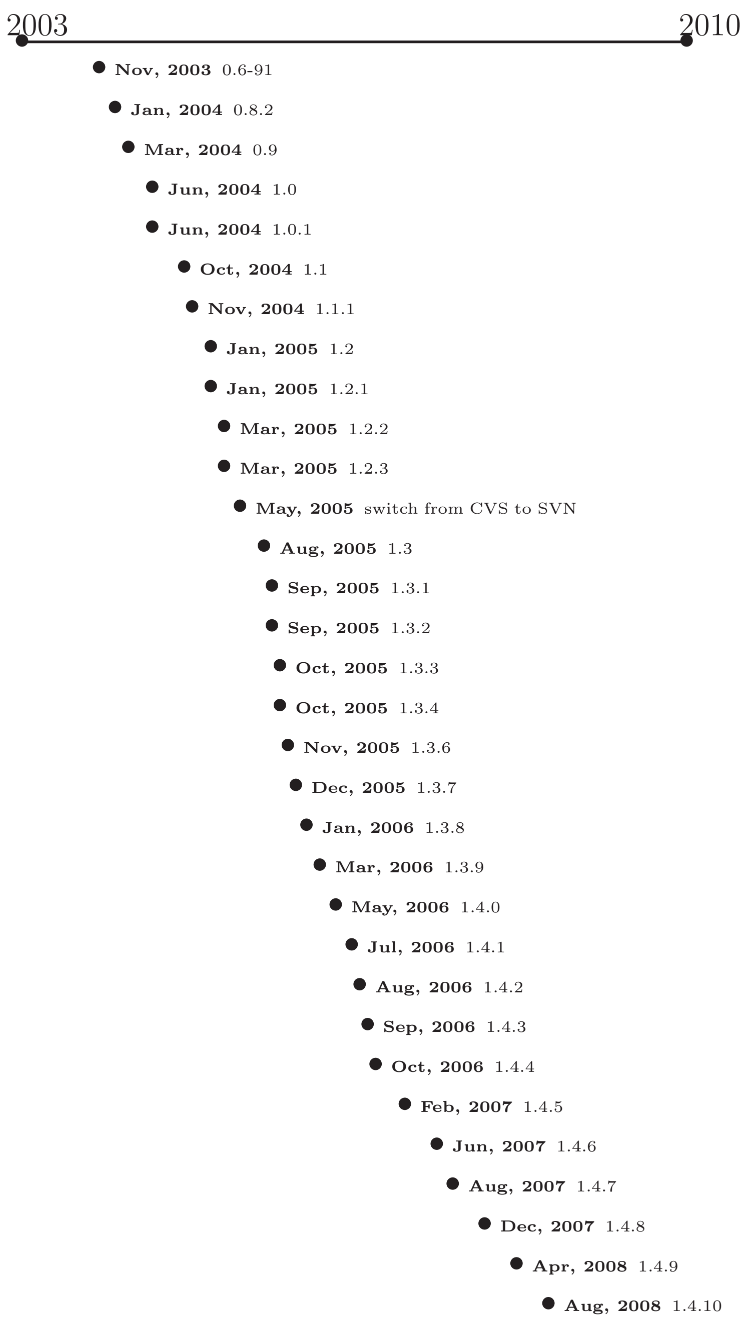


**Figure A.2:** Amarok 1 timeline (tagging dates according to [git10], source code management system switch according to [Rid05])

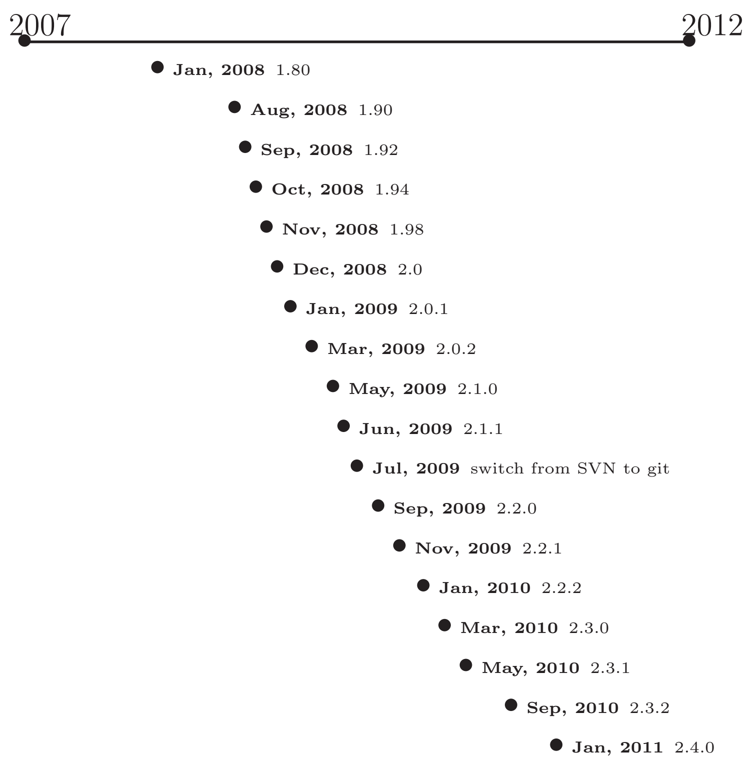


**Figure A.3:** Amarok 2 timeline (tagging dates according to [git10], source code management system switch according to [Pin09])

*"It seems like once people grow up, they have no idea what's cool."*
Calvin (Calvin and Hobbes)

# B

# Interview questions

The following questions were asked during the interviews to capture the initial state of the projects.

- For how long are you involved in the project?
- Why are you working on Amarok/Halo?
- Which parts of the development process are you involved in?
- Which tools do you use for your day to day work and how do you use them?
- What is good/bad about the tools you use?
- How do you decide which task to work on next?
- Who do you interact with usually and why?
- How do you interact with them?
- What are the areas that create most problems and confusion inside the core-team?
- What works well inside the core-team?
- What are the areas that create most problems and confusion outside the core-team?
- What works well outside the core-team?

- Where does the development process need to be improved?
- Which parts of the development process work well?
- Is there anything else you would like to add?

“*Talk is cheap. Show me the code.*”

Linus Torvalds

# C

# Mark-up

The following mark-up snippets are part of the implementation. They are provided to illustrate how to replicate a similar system for another project.

```
{|
|-
! Name
! Tasks
! Bugs
! Feature requests
{{#ask: [[Category:Person]]
[[Affiliated with::ontoprise GmbH]]
| ?Email address
| format=template
| template=Team dashboard row
| headers=hide
| link=none
| order=ascending
| merge=true
|}}
|}
```

**Listing C.1:** Mark-up for the team dashboard table showing task, bug and feature request count for each team member

```
|-
| [[{{{1}}}|{{Remove_user_namespace|{{{1}}}}}]]
| {{BugzillaQuery|Task|{{Remove_user_namespace|{{{1}}}}}}}
| {{BugzillaQuery|Bug|{{Remove_user_namespace|{{{1}}}}}}}
| {{BugzillaQuery|Feature Request|{{Remove_user_namespace
  |{{{1}}}}}}}
```

**Listing C.2:** Template for the individual rows of the team dashboard table showing task, bug and feature request count for each team member

```
|-
| [[{{{1}}}|{{Remove_user_namespace|{{{1}}}}}]]
| [http://smwforum.ontoprise.com/smwbugs/buglist.cgi?
  emailassigned_to1=1&query_format=advanced&bug_status=NEW&
  bug_status=ASSIGNED&bug_status=REOPENED&cf_issuetype=Task&
  email1={{{2}}}&emailtype1=exact {{BugzillaQueryIndividual|Task
  |{{Remove_user_namespace|{{{1}}}}}}}]
| [http://smwforum.ontoprise.com/smwbugs/buglist.cgi?
  emailassigned_to1=1&query_format=advanced&bug_status=NEW&
  bug_status=ASSIGNED&bug_status=REOPENED&cf_issuetype=Bug&
  email1={{{2}}}&emailtype1=exact {{BugzillaQueryIndividual|Bug
  |{{Remove_user_namespace|{{{1}}}}}}}]
| [http://smwforum.ontoprise.com/smwbugs/buglist.cgi?
  emailassigned_to1=1&query_format=advanced&bug_status=NEW&
  bug_status=ASSIGNED&bug_status=REOPENED&cf_issuetype=Feature
  %20Request&email1={{{2}}}&emailtype1=exact {{
  BugzillaQueryIndividual|Feature Request|{{
  Remove_user_namespace|{{{1}}}}}}}]
```

**Listing C.3:** Template for the individual rows of the team dashboard table showing task, bug and feature request count for each team member with links

```
{{#replace:{{{1}}}|User:|}}
```

**Listing C.4:** Parser function to remove the User namespace

```
{{#ws:BugzillaConnector
| componentName = *
| productName = *
| IssueType={{{1}}}
| Assignee={{{2}}}
| Status=NEW
| ?result.URL
| _format=count
}}
```

**Listing C.5:** Web service query for the team dashboard table

```
{{#ws:RSSFeed
| url = dailywikibuilds.ontoprise.com:8080/job/smwhalo/rssAll
| ?result.title
| ?result.link
| _striptags
| _format=template
| _template=RSSFeedLinkList
| _limit=3
}}
```

**Listing C.6:** Web service query for the build status from Hudson

```
{{#ask:[[Development/Release_Status/Facts_for_reports]] [[
   CountAllBugsForCurrentRelease::+]][[
   CountResolvedBugsForCurrentRelease::+]] [[
   CountOpenBugsForCurrentRelease::+]]
| ?CountAllBugsForCurrentRelease=all bugs
| ?CountResolvedBugsForCurrentRelease=resolved
| ?CountOpenBugsForCurrentRelease=open
| format=ofc-bar
}}
```

**Listing C.7:** Query for the bug status chart of Halo

```
[[CountOpenBugsForCurrentRelease::{{#ws:BugzillaConnector|
   componentName = *
| productName = SemanticWiki
| IssueType= Bug
| Severity =  blocker, critical, major, minor, normal, trivial,
   enhancement
| Status = UNCONFIRMED, REOPENED, NEW, ASSIGNED
| Milestone = {{CurrentVersionInBugzilla}}
| ?result.Id
| _limit = 2000
| _format=count
}}]]
```

**Listing C.8:** Web service query for number of open bugs for the current Halo development version

```
{{#ws:BugzillaConnector
| componentName = *
| productName = SemanticWiki
| Severity = blocker, critical, major
| Status = UNCONFIRMED, REOPENED, NEW, ASSIGNED
| Milestone = {{CurrentVersionInBugzilla}}
| ?result.URL
| ?result.Severity
| ?result.Summary
| ?result.Priority
| ?result.Version
| ?result.Status
| _format=table
}}
```

**Listing C.9:** Web service query for open bugs for the current Halo release

```
{{#ws:BugzillaConnector
| componentName = *
| productName = SemanticWiki
| Severity = blocker, critical, major, minor, normal, trivial
| Status = RESOLVED
| Milestone = {{CurrentVersionInBugzilla}}
| ?result.URL
| ?result.Severity
| ?result.Summary
| ?result.Priority
| ?result.Version
| _format=table
}}
```

**Listing C.10:** Web service query for closed bugs for the current Halo release

```
{{#ask: [[HasFeatureStatus::in progress||draft||reviewed||needs
   review]]
| ?HasFeatureStatus = Status
| ?HasFeaturePriority = Priority
| ?HasFeatureScopeDifficulty = Scope/Difficulty
| ?HasTargetRelease = Target Release
| ?HasFeatureAssignee = Assignee
| format=table
| headers=show
| mainlabel=Feature
| link=all
| order=ascending
| source=tsc
| queryname=FeatureTracking
| merge=false
|}}
```

**Listing C.11:** Query for features tracked in the wiki for Halo

```
= {{PAGENAME}} =

{| style="float: right;" border="1" cellpadding="4" cellspacing
   ="0"
|+ '''Status information'''
|-
|'''State:''' [[Feature_States|{{{State}}}]]
|'''Target release:''' {{{TargetRelease}}}
|-
|'''Percent completed:''' {{{PercentCompleted}}}
|'''Creator:''' [[User:{{{Creator}}}|{{{Creator}}}]]
|-
|'''Scope/Difficulty:''' [[Feature_ScopeDifficulty|{{{
   ScopeDifficulty}}}]]
|'''Driver:''' [[User:{{{Driver}}}|{{{Driver}}}]]
|-
|'''Priority:''' [[Feature_Priorities|{{{Priority}}}]]
|'''Assignee:''' [[User:{{{Assignee}}}|{{{Assignee}}}]]
|}

== Summary ==
{{{Summary}}}

== Extended information and rationale ==
{{{ExtendedInfoAndRationale}}}

== Related bug reports and branches ==
{{{RelatedBugReportsAndBranches}}}

== Discussion ==
Please discuss this proposal on its discussion page.
```

**Listing C.12:** Wiki mark-up of Amarok's feature tracking template

```
{{Feature
|State=Draft
|PercentCompleted=0
|ScopeDifficulty=3
|Priority=2
|TargetRelease=2.4.6
|Creator=Nightrose
|Driver=Nightrose
|Assignee=Nightrose
|Summary=Please describe your feature here in a few sentences.
|ExtendedInfoAndRationale=Please explain your feature in more
   detail here. Include a rationale, any similar work you know
   about and any other information that might be helpful.
|RelatedBugReportsAndBranches=Please list any related bug reports
   or code branches here.
}}
```

**Listing C.13:** Usage example of Amarok's feature tracking template

```
= Feature tracking for {{PAGENAME}} =
{| style="float: right;" border="1" cellpadding="4" cellspacing
   ="0"
|+ '''Status information'''
|-
|'''[[Feature_Tracking_States|State:]]''' [[HasFeatureStatus::{{{
   State|}}}]]
|'''Target release:''' [[HasTargetRelease::{{{TargetRelease|}}}]]
|-
|'''Percent completed:''' [[HasPercentCompleted::{{{
   PercentCompleted|}}}]]
|'''Creator:''' [[HasFeatureCreator::{{{Creator|}}}]]
|-
|'''[[Feature_Tracking_ScopeDifficulty|Scope/Difficulty:]]''' [[
   HasFeaturePriority::{{{Priority|}}}]]
|'''Driver:''' [[HasFeatureDriver::{{{Driver|}}}]]
|-
|'''[[Feature_Tracking_Priorities|Priority:]]''' [[
   HasFeaturePriority::{{{Priority|}}}]]
|'''Assignee:''' [[HasFeatureAssignee::{{{Assignee|}}}]]
|}

== Summary ==
{{{Summary|}}}

== Extended information and rationale ==
{{{ExtInformationRationale|}}}

== Related bug reports and branches ==
{{{RelatedBugReportsBranches|}}}

== Discussion ==
{{ShowComments|show=True}}

[[Category:Feature]]
```

**Listing C.14:** Wiki mark-up of Halo's feature tracking template

```
{{{for template|Feature}}}
{| class="formtable"
! [[Feature Tracking ScopeDifficulty|Scope/difficulty:]]
| {{{field|ScopeDifficulty|mandatory|default=0}}}
|-
! [[Feature_Tracking_States|State:]]
| {{{field|State|mandatory|default=draft}}}
|-
! [[Feature Tracking Priorities|Priority:]]
| {{{field|Priority|mandatory|default=4}}}
|-
! Percent completed:
| {{{field|PercentCompleted|mandatory|default=0}}}
|-
! Target release:
| {{{field|TargetRelease}}}
|-
! Creator:
| {{{field|Creator|mandatory|default=current user}}}
|-
! Driver:
| {{{field|Driver}}}
|-
! Assignee:
| {{{field|Assignee}}}
|-
! Summary:
| {{{field|Summary|input type=textarea|mandatory|default=Please
   describe your feature here in a few sentences.}}}
|-
! Extended information and rationale:
| {{{field|ExtInformationRationale|input type=textarea|default=
   Please explain your feature in more detail here. Include a
   rationale, any similar work you know about and any other
   information that might be helpful.}}}
|-
! Related bug reports and branches:
| {{{field|RelatedBugReportsBranches|input type=textarea|default=
   Please list any related bug reports or code branches here.}}}
|}
{{{end template}}}
```

**Listing C.15:** Wiki mark-up of Halo's feature tracking form

*"Nothing great in the world has been accomplished without passion."*

Georg Wilhelm Friedrich Hegel

# D Checklists

Commit checklists were developed for both projects. They are supposed to help new contributors avoid common mistakes in their first commits to a public repository.

## D.1 Halo

For every commit:

- Does it conform to Halo's coding style?
- Is there a feature request or bug report associated with it?
- Does it introduce new dependencies?
- Is the code documented where needed?
- Has it been added to the changelog?
- Does the diff contain only the changes you made and files you added/removed? Does it contain all of them?

Additionally for commits introducing new features:

- Does it fit Halo's vision or at least not work against it?

- Who will maintain the code?
- Is Halo in feature freeze?
- Do existing regression tests still pass?
- Are new tests needed?
- Has it been reviewed by a usability person?
- Does user documentation exist?

## D.2 Amarok

For every commit:

- Does it conform to Amarok's coding style?
- Is there a feature request, bug report or review request associated with it? (Please close them with the commit message.)
- Does it introduce new dependencies?
- Does it add new strings? Is Amarok in string freeze?
- Has it been added to the changelog?
- Does the diff contain only the changes you made and files you added/removed? Does it contain all of them?

Additionally for commits introducing new features:

- Does it fit Amarok's vision or at least not work against it?
- Who will maintain the code?
- Is Amarok in feature freeze?
- Do existing regression tests still pass?
- Are new tests needed?
- Has it been reviewed by a usability person?
- Does user documentation exist?

*"The future is already here – it's just not very evenly distributed."*
William Gibson

# E
# Evaluation questions

The following questions were used as the basis for the evaluation in Chapter 7. Each of them was to be answered on a scale from 1 (not at all) to 5 (a lot/absolutely) with the exception of the comment questions in each section and Halo's first question. These were free-text fields instead.

## E.1 Halo

1. **the new vision**
   (a) What do you think are the reasons the vision creation for Halo failed so far? (No-one added answers to the questions in the wiki so far.)
2. **communication and coordination**
   (a) Will the team dashboard increase transparency? (You can find it at http://smwforum.ontoprise.com/smwforum/index.php/Development/Team_Dashboard.)
   (b) Will the team dashboard improve collaboration between contributors?
   (c) Will the team dashboard improve collaboration between contributors and users?
   (d) Will the release dashboard increase transparency? (You can find it at http://smwforum.ontoprise.com/smwforum/index.php/Development/Release_Status.)

   (e) Will the release dashboard improve collaboration between contributors?
   (f) Will the release dashboard improve collaboration between contributors and users?
   (g) Will the commit checklist help new contributors? (You can find it at http://smwforum.ontoprise.com/smwforum/index.php/Development/Commit_Checklist.)
   (h) comments

3. **roadmap and feature tracking** (You can find the new feature tracking form at http://smwforum.ontoprise.com/smwforum/index.php/Form:Feature and an example at http://smwforum.ontoprise.com/smwforum/index.php/Testfoo. These are primarily intended for contributors who want to communicate what they are working on.)

   (a) Do you think the new feature tracking form is easy to use?
   (b) Will the new feature tracking improves collaboration between contributors?
   (c) Will the new feature tracking improves collaboration between contributors and users?
   (d) Will the new feature tracking increase transparency?
   (e) Do you think other Free Software projects might be interested in using this too?
   (f) comments

4. **quality assurance**

   (a) Do you consider the public testing useful for improving the QA situation?
   (b) comments

## E.2 Amarok

1. **the new vision** ("The Amarok team strives to develop a free and open music player that is innovative and powerful, yet easy to use. Amarok helps rediscover music by giving access to a vast amount of different music sources and related information. In a world where music and computing are everywhere, Amarok aims to provide the best music listening experience anywhere, anytime. The Amarok team promotes free culture. Amarok makes people happy.")

   (a) Will the new vision help the project make better decisions in its next development steps?

   (b) Will the new vision help increase transparency in the project for users?
   (c) Will the new vision help improve collaboration in the project for contributors?
   (d) Do you think the process of creating the new vision can be used by other Free Software projects as well? (This means answering the questions on the wiki, having a meeting to discuss them and drafting a vision statement after that in a smaller group. You can see the questions again at http://amarok.kde.org/wiki/VisionCreation.)
   (e) Do you think the process of creating the new vision was transparent?
   (f) Do you think the process of creating the new vision was collaborative?
   (g) comments

2. **communication and coordination**

   (a) Will the new commit checklist help new contributors? (You can find it at https://projects.kde.org/projects/extragear/multimedia/amarok/repository/revisions/master/entry/HACKING/commitchecklist.txt.)
   (b) comments

3. **roadmap and feature tracking** (You can find the new feature tracking template at http://amarok.kde.org/wiki/Template:Feature, an example in action at http://amarok.kde.org/wiki/Proposals/Example and the new roadmap page at http://amarok.kde.org/wiki/Proposals.)

   (a) Do you think the new feature tracking template and roadmap are easy to use?
   (b) Will the new feature tracking template and roadmap improve collaboration between contributors?
   (c) Will the new feature tracking template and roadmap improve collaboration between users and contributors?
   (d) Will the new feature tracking template and roadmap increase transparency?
   (e) Do you think other Free Software projects might be interested in using this too?
   (f) comments

4. **quality assurance**

   (a) Do you consider the weekly emails from Bugzilla to the developer mailing list about release blockers useful for improving the QA situation?
   (b) Do you consider the testing checklist useful for improving the QA situation? (You can find it at http://amarok.kde.org/wiki/Development/Testing.)
   (c) comments

# Bibliography

[Abe07] Mark Aberdour. Achieving quality in open source software. *IEEE Software*, 24:58–64, January 2007.

[Ale10] Alexa. Alexa top 500 global sites. online, November 2010.

[Bac09] Jono Bacon. *The Art of Community - Building the New Age of Participation*. O'Reilly, 2009.

[BNST99] Terry Bollinger, Russell Nelson, Karsten M. Self, and Stephen J. Turnbull. Open-source methodology: Peering through the clutter. *IEEE Software*, 16(4):8–11, 1999.

[Cho07] Mark Choate. Factors that improve wiki success. online, December 2007.

[CMM10] CMMI Product Team. Cmmi for development, version 1.3. Technical report, Carnegie Mellon - Software Engineering Institute, November 2010.

[Com10] The Nielson Company. Mobile snapshot: Smartphones now 28% of u.s. cellphone market. online, November 2010.

[Cow10] Jennifer Cownie. November 2010 web server survey. online, November 2010.

[Dav89] Fred R. David. How companies define their mission. *Long Range Planning*, 22(1):90 – 97, 1989.

[EC03] J. Alberto Espinosa and Erran Carmel. The impact of time separation on coordination in global software teams: a conceptual foundation. *Software Process: Improvement and Practice*, 8:249 – 266, 2003.

[ESKH07] J. Espinosa, Sandra Slaughter, Robert Kraut, and James Herbsleb. Team knowledge and coordination in geographically distributed software development. *J. Manage. Inf. Syst.*, 24:135–169, July 2007.

[Fed] Fedora Project. Features/policy/process - fedoraproject. online.

[FHL09] J. Fried, H.D. Hansson, and M. Linderman. *Getting Real: the Smarter, Faster, Easier Way to Build a Successful Web Application*. 37signals, 2009.

[Fra10] Leonardo Franchi. Blurring the boundaries of music. In *Akademy 2010*. KDE, July 2010.

[Gar06] Jeremy Garcia. 2005 linuxquestions.org members choice award winners announced. online, March 2006.

[Gar07] Jeremy Garcia. 2006 linuxquestions.org members choice award winners announced. online, February 2007.

[Gar08] Jeremy Garcia. 2007 linuxquestions.org members choice award winners. online, February 2008.

[Gar09] Jeremy Garcia. 2008 linuxquestions.org members choice award winners. online, February 2009.

[Gar10] Jeremy Garcia. 2009 linuxquestions.org members choice award winners. online, February 2010.

[git10] gitweb. Kde gitweb - amarok.git/tags. online, October 2010.

[Gra10] James Gray. Readers' choice awards 2010. online, October 1010.

[Gra08] James Gray. Readers' choice awards 2008. online, May 2008.

[Gra09a] Chris Grams. Tom sawyer, whitewashing fences, and building communities online. online, September 2009.

[Gra09b] James Gray. Readers' choice awards 2009. online, June 2009.

[Gra10a] Chris Grams. Community-building tip: surprise is the opposite of engagement. online, April 2010.

[Gra10b] Chris Grams. Trust: the catalyst of the open source way. online, June 2010.

[HB03] Pamela J. Hinds and Diane E. Bailey. Out of sight, out of sync: Understanding conflict in distributed teams. *Organization Science*, 14:615–632, November 2003.

[Hin07] Brian Hindo. At 3m, a struggle between efficiency and creativity - how ceo george buckley is managing the yin and yang of discipline and imagination. online, June 2007.

[HJ07] Jesper Holck and Niels Jørgensen. Continuous integration and quality assurance: a case study of two open source projects. *Australasian Journal of Information Systems*, 11(1), 2007.

[HS02] T. J. Halloran and William L. Scherlis. High quality and open source software practices. In *Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering*, pages 26 – 28, 2002.

[IH92] R. Duane Ireland and Michael A. Hirc. Mission statements: Importance, challenge, and recommendations for development. *Business Horizons*, 35(3):34 – 42, 1992.

[Lev00] Ira M. Levin. Vision revisited. *The Journal of Applied Behavioral Science*, 36(1):91–107, 2000.

[Lip96] Mark Lipton. Demystifying the development of an organisational vision. *Sloan Management Review*, 37(4), Summer 1996.

[McC99] Steve McConnell. Open-source methodology: Ready for prime time? *IEEE Software*, 16(4):6–8, 1999.

[Moe06] Erik Moeller. Rfc: Mission & vision statements of the wikimedia foundation. email, November 2006.

[Mor06] Betsy Morris. Fortune: The new rules. online, July 2006.

[N$^+$] Dave Neary et al. Cookies licking - community management wiki. online.

[Nea11] Dave Neary. Drawing up a roadmap. online, February 2011.

[Ont10] Ontoprise. News - smw+ semantic enterprise wiki. online, November 2010.

[Ott10] Tobias Otte. *An Investigation into quality assurance of the Open Source Software Development model*. PhD thesis, University of Wolverhampton, June 2010.

[Per98] Bruce Perens. The open source definition. online, February 1998.

[Pin09] Lydia Pintscher. we're testing the water for everyone. online, July 2009.

[Pin10] Lydia Pintscher. Mentoring in free software projects - a review of 6 years of gsoc and sok in kde and what we learned so far. In *Akademy 2010*. KDE, July 2010.

[Poo10] Jos Poortvliet. Strategy sucks. online, September 2010.

[Pro97] Debian Project. Debian social contract - the debian free software guidelines (dfsg). online, July 1997.

[Ray98] Michael E. Raynor. That vision thing: Do we need it? *Long Range Planning*, 31(3):368 – 376, 1998.

[Ray01] E.S. Raymond. *The cathedral and the bazaar: musings on Linux and Open Source by an accidental revolutionary.* O'Reilly Series. O'Reilly, 2001.

[Rid05] Jonathan Riddell. Kde's switch to subversion complete. online, May 2005.

[SCA06] SCAMPI A Upgrade Team. Standard cmmi appraisal method for process improvement (scampism) a, version 1.2: Method definition document. online, August 2006.

[Sei10] Aaron Seigo. +10 on linux journal reader's choice awards. online, November 2010.

[Sta86] Richard Stallman. The free software definition. online, February 1986.

[Tar97] Eugen Tarnow. A recipe for mission and vision statements. *Journal of Marketing Practice: Applied Marketing Science*, 3(3):184–189, 1997.

[vKSL03] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, July 2003.

[WHP$^+$09] Axel Winkelmann, Sebastian Herwig, Jens Pöppelbuß, Daniel Tiebe, and Jörg Becker. Discussion of functional design options for online rating systems: A state-of-the-art analysis. In *17th European Conference on Information Systems (ECIS 2009)*, 2009. Verona, Italy.

[Wik10] Wikimedia. Wikimedia traffic analysis report - browsers. online, October 2010.

[WM07] Christian Wagner and Ann Majchrzak. Enabling customer-centricity using wikis and the wiki way. *Journal of Management Information Systems*, 23(3):17–43, January 2007.

[YYSI00] Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. Collaboration with lean media: how open-source software succeeds. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 329–338, New York, NY, USA, 2000. ACM.

[ZE03] Luyin Zhao and Sebastian Elbaum. Quality assurance under the open source development model. *J. Syst. Softw.*, 66:65–75, April 2003.