research-report

BEP-store

Published
with GitBook

Table of Contents

Introduction

With the rise of open-source software over the past years, open source communities have grown as well. In large open source projects, the community plays a significant role in providing feature requests, prioritization of future features and feedback on existing functionalities. This increased influence is reflected in the tools that are used in software development. Open issue trackers, pull requests and public communication channels like IRC, Slack, Gitter and Twitter are just a few of many options for software developers to engage with the community around their software.

However, in the current situation, the software plays the central role in the process. Issues are listed under repositories and pull requests are about code. This results in alienation of the part of the community that doesn't have the technical expertise to deal with code, usually the majority. This seems suboptimal, since the software development process deals with a lot of concepts beyond code as well. This document identifies the difficulties in centering the software development process around the community, with the goal of utilizing as much of the expertise available in the community. It formulates a clear problem description and splits this problem into different research questions.

Problem description

Existing tools offered by platforms (such as GitHub) focus on collaboration and distributed software development. These tools however are created around the needs of developers, rather than facilitating the software development process in general. This excludes a large part of the community from participating actively in the process.

On the other hand, the tools developed at FeedbackFruits are focused on the improvement of education. The tools are targeted at students and teachers, many of which are non-programmers and as such are excluded from the traditional development process. In addition, feature request are often too vague and plentiful for developers of the core platform to accommodate all of them. To include these individuals, who are often eager to improve the educational system, FeedbackFruits aims to create a platform where anyone with useful expertise can be included in the software development process. The goal of this platform is to create software that can connect to existing FeedbackFruits ecosystem, but can be developed without having to know its inner workings. The general problem description, with respect to FeedbackFruits, then becomes as follows:

How can the existing platform be extended in such a way that everyone in the FeedbackFruits community can contribute to the software development process?

Research questions

To better understand how to solve the problem, the following main research questions and several sub-questions have been formulated. These questions are rather general, but serve to split the problem into smaller pieces, which can then be broken down even further. The smaller sub-questions will give this report its structure, but won't be anwsered explicitly in the conclusion.
The research questions from the Project Plan have been rephrased during the research, wich means that the research questions below are rephrased versions of the original questions.

How should community driven software development be structured?

- What can be learned from existing (non-software) communities?
- What current efforts exists to involve the community in the software development process?
- How are requirements established and used in software development?
- What limitations does the existing FeedbackFruits ecosystem impose on the software?

Sub-questions

What can be learned from existing (non-software) communities?

- What communities are there and how do they work?
- What are the requirements for a successful community-centered ecosystem?
    - What are the strengths of these communities?
    - What are the weaknesses of these communities?
- What are factors that can be utilized for software development?

What current efforts exists to involve the community in the software development process?

- What elements are essential in community involvement?
- What techniques are currently used for community involvement?
- How can the development process be regulated with regards to time and maintainability?
- Is it possible to keep everyone engaged during the entire development process?

How are requirements established and used in software development?

- What is the necessity of requirements?
- What are requirements based on?
- Are there standards for establishing requirements?
- Can requirements change during the software development process?

What limitations does the existing FeedbackFruits ecosystem impose on the software?

Report structure

To anwser the four main research questions, the report will have four sections. The first research question zooms in on what can be learned from (non-software) communities. Before theories can be defined about what can be learned from all sorts of communities, a definition must be established of what a community is. Because communities exist in different sizes, with different structures and on different platforms, a few of these communities will be analysed in section 1.

Some ways to involve the community in building software may be composed based on the previous section. However, this research will probably not be the only attempt to involve a software community in building software. The second research question is therefore focussed on existing platforms where software communities are involved in one way or another in the software development process. Section 2 will focus on the structure of the existing software communities. The techniques used by those communities and the way that the development process engages the community will also be examined.

Software development in its current form has some methodologies to establish requirements for a (big) software development project. The third research question will take a better look at the dynamics of requirements. These requirements are the basis of how the software will be built and are therefore an essential part of the software development process. In section 3, a few of the methods wich are used to establish requirements will be described. The dynamics of the requirements, like management and implementation, will also be given a closer look.

The first three research question focus on determining the critical attributes of the problem in a general sense. However, because these attributes have to be applicable on the current FeedbackFruits-platform, that platform may pose some limitations. With research question four, the structure of the FeedbackFruits-platform will be examined, as well as the extensiblity of the platform. The result of this research question can be found in section 4.

Communities

A community is, according to the Oxford Dictionary, *a group of people living in the same place or having a particular characteristic in common*[1]. In traditional research, the study focuses on localy bound communities like neighbourhoods and villages, because they have clear boundaries of who is involved in the community[2][1].
With the introduction of modern media, like games, social media and messenger-applications, communities are not bound anymore by location and the characteristic becomes the central point of the community. A concequence of this is that boundaries of the community are less clear. Therefore, one of the challenges of doing research on these virtual communities is determining who is and who is not part of a community[12].
Virtual communities have four necessary characteristics: interactivity, communicators, a publicly shared mediated (virtual) communication place and a minimum level of sustained membership[13]. This means that in a virtual community, there must be a longlasting, self-sustaining communication *(interactivity)* about subjects (related to the relating characteristics) by two or more persons *(communicators)* over one ore more platforms like a website or a chat-group *(communication place)* where there is a certain amount of active members *(sustained membership)*.

Before we can compare different communities and their structure, we first need to define a few (types of) communities. We will look at two types of real-life communities and two types of virtual communities, such that we can discuss their similarities and differences.

> [1]. http://www.oxforddictionaries.com/definition/english/community ↩

Ecosystems

An ecosystem can be seen as a natural form of a community. It is bound to a location where a diversity of animals and species interact with eachother[1]. Although there is no real communication, the combination of individual self-interest (the survival of their offspring or themself) creates a network of dependencies and interactions that one can interpret as a community.

A key factor of an ecosystem is stability. In a stable ecosystem[2], each species acts in such a way that they directly enable or inderectly do not interfere with the survival of another species. . An ecosystem will remain stable as long as the system is able to deal with outside disturbances.

> 1. http://www.oxforddictionaries.com/definition/english/ecosystem ↩

> 2. https://stats.oecd.org/glossary/detail.asp?ID=2530 ↩

Intentional Communities

Intentional communities are man-made communities, where its members attempt to realise a common alternative way of life outside of mainstream society[1]. Although there is no one list of criteria to characterise them, the most cited criteria are:

1. No bonds by familial relationships only.
2. A minimum of three to five adult members.
3. Members join voluntarily
4. Geographical and psychological separation from mainstream society.
5. A common ideology that is adhered to by all members.
6. Sharing of (a part of) one's property.
7. The interest of the group prevails over individual interests.

Not all intentional communities fulfill all of these criteria but they give a good idea of what is and what is not a intentional community. Intentional communities can be quite diverse. To get a better understanding of the different ways an intentional community can work, they can be split into four groups: ecological, communal, religious and practical communities[1].

Ecological Communities
Ecological communities are communities that withdraw to remote locations in an effort to live according to their own ideals. The community members actively separate themselves from society and aim to be self-sufficient. The separation is what identifies this type of intentional communities[2][1].
Ecological communities can manifest itself on different scales. It can vary from a group living together in a house (cohousing) to a whole village (ecovillages)[3], but all ecological communities have a support-structure and cooperate to achieve the ideals of the community[4].

Communal Communities
Communal communities center their ideals around the contact between communities members[1]. This means that the platform (chat-groups, website, etc.) or place (offices, pub, etc.) plays a central key in this type of community. The place or subject of the platform focuses on the particular characteristic that the members have in common. Examples of such communities are a cooking-forum for people who like cooking or a chat-group for parents whose kids are in the same class.

Religious Communities
Religious communities have their ideals centered around a belief or religion. If you look at these communities based on religion, then there is no boundary to their location. However,

there are religious communities that also behave like a communal community, like local church-communities. The ideals of the religion are the ideals of the community.

Practical Communities

Practical communities are special because they don't center around ideals at all. Practical communities arise out of pure convenience, when people are better of sharing information or resources (on a regular basis). This can be a community of farmers sharing farming equipment, or students sharing a house.

MMO-RPG Communities

Massive Multiplayer Online Role-Playing Games are, like the name describes, online games where the player explores a virtual world as one of many types of characters and can interact with an enormous amount of other players. Popular MMO-RPG's often have two types of communites: communities outside of the game (fans and forums) and communities inside of the game (parties and guilds). For this research, we will be looking at the communities inside of the game, because the dynamics of these communities often require active participation.

Most MMO-RPG's have two in-game systems that can be seen as communities: parties and guilds. A party is a small group of players that train or do quests (in-game missions) together for a short period of time (possibly one game session). The benefit of forming a party are experience-sharing (to some degree) and strength in numbers. Strength in numbers means that players can kill high-level in-game creatures they could not kill on their own. If a party is formed for the purpose of doing a quest, the composition of the party (the distribution of roles or functions) may be adapted towards the goal of the quest to ensure optimal performance[5].

A guild is a group of players forming a community. The purpose of a guild is creating a social connection or to make it easier to form a trust-worthy or well-composed party over a longer period of time (longer than one game session)[6][7][8]. Within a guild, members can be classified based on connectivity, type of player, role within the guild and involvement[9].

Software Communities

Software communities are groups of people involved in the development or improvement of a piece of software, but can also be focussed on a programing language (model). This makes software communities a broad term. For this research, the focus will be put on Software Development Communities, communities centered around software development/improvement.

A software development community can be divided into two groups: developers and users. Developers create software, wich is used by the users. The users are able to give feedback about the software, wich the developers can use in the next version of that piece of software. Users sometimes discuss the (lack of a certain) functionality in a software program on dedicated forums. This creates a feedback loop for the programmers. A smaller feedback loop is the bug-reporter. Organisations, like Apache and Mozilla, often have a part of their software dedicated to enabling the users to submit bugs[10]. This enables the developers to fix problems with the software in a much faster way than searching for feedback.
One special community is the Open Source Software community. Open Source means that the source-code of the software program is publicly available on the internet and that everyone can download and use the software for free[11]. Because the code is publicly available, it enables users who understand the code to look in the code for the source of bugs and failures and give feedback related to a piece of the code. It also enables other programmers to expand the program based on it's source code.

Requirements for a successful community-centered ecosystem

This section introduces the term Software Ecosystem and its relation to a community, as well as a model for quantifying its characteristics. It also lists several challenges in creating and maintaining a successful community.

Software Ecosystems

A Software Ecosystem (SECO) is defined by van den Berk, Jansen, & Luinenburg, 2010 [36] to be: 'A set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them.' A SECO consists of a central hub, a platform and niche players. In context with the subject of this research report, the platform is the FeedbackFruits platform, the niche players are its users, mostly teachers and students, and the hub is the FeedbackFruits company. In this context, the problem description becomes: how can the hub stimulate niche creation in the SECO?

Niche creation can be stimulated by determining the desired future state of the SECO [36]. To do this, both the current state and the desired improvements need to be known. In order for the hub to gain insight about the current state of the ecosystem, the SECO Software Assesment Model can be used. It defines a broad spectrum of characteristics of the SECO that can be made quantifiable.

To gain insights about the desired improvements, some kind of knowledge management will be needed in order to aggregate the information from all of the niche players. However, due to the use of personalized learning methodologies in the FeedbackFruits platform and the broad spectrum of subjects covered by niche players, a lot of this information is tacit knowledge. McDermott states that sharing tacit knowledge requires interaction and proceeds to provide a list of critical factors for building a community that can share such knowledge [37].

Communities of practice

Communities of practice enable person-to-person interaction while still engaging a whole group in advancing their field. As such, they are ideal vehicles for sharing tacit knowledge. The following are 10 critical success factors in building communities of practice [37], separated into four challenges:

Management Challenge

- Focus on topics important to the business and community members.
- Find a well-respected community member to coordinate the community.
- Make sure people have time and encouragement to participate.
- Build on the core values of the organization.

Community Challenge

- Get key thought leaders involved.
- Build personal relationships among community members.
- Develop an active passionate core group.
- Create forums for thinking together as well as systems for sharing information.

Technical Challenge

- Make it easy to contribute and access the community's knowledge and practices.

Personal Challenge

- Create real dialogue about cutting edge issues.

These challenges are centered around very human problems, since sharing tacit knowledge is thinking together and discovering what insights of the past are relevant for the future. To make a community of practice really valuable, inclusive and vibrant, they need a human touch of nurture.

What techniques are currently used to involve a community with software development?

There are many projects that are built with the help of an online community. The largest projects include the development of the Linux kernel[1], the Apache project[2] and the Mozilla browser[3]. To include the online communities during the development of such online software it is necessary to keep everyone up to date and informed about the current state of the project. In short, the means needed can be divided into five categories: communication, version control, issue tracking, testing and package management. The techniques that are currently being used are specified below for each category. In "Practices in Commercial Projects Using GitHub" by Kalliamvakou et al. the reasons to use GitHub, a popular online Git hosting service, were investigated. Their results yield interesting insights into techniques that can be utilized to enhance the Feedbackfruits platform [20].

Communication

Communication is perhaps the most important factor when it comes to collaborative software development. Much communication is traditionally done via e-mail, instant messaging, phone, web meetings and groupware [21]. However as of late new, more enhanced tools like Slack[4] have emerged. With such new tools, it is possible to integrate updates from other development tools such as GitHub and make the visible to the whole team. A conclusion drawn by Kalliamvakou is that the use of GitHub and its tools reduce the need for communication and coordination needs. However challenges remain in collaborating with non-technical persons because commercial projects still resort to external tools outside GitHub [20].

Version control

Many software projects makes use of version control. According to the annual Eclipse Community service this percentage totals to almost eighty percent (Git 43.9%, Subversion 30.7%, Other 5.8%) in 2014 [22]. From the statistics in the figure below it can be concluded that the Git has become the most popular version control method. The main reasons for the growing popularity of Git are its feature set and the usage in large projects such as the aforementioned Linux project [23]. Two of the most important features of Git are branching and pull-requests stimulate working together trough independent work [20].

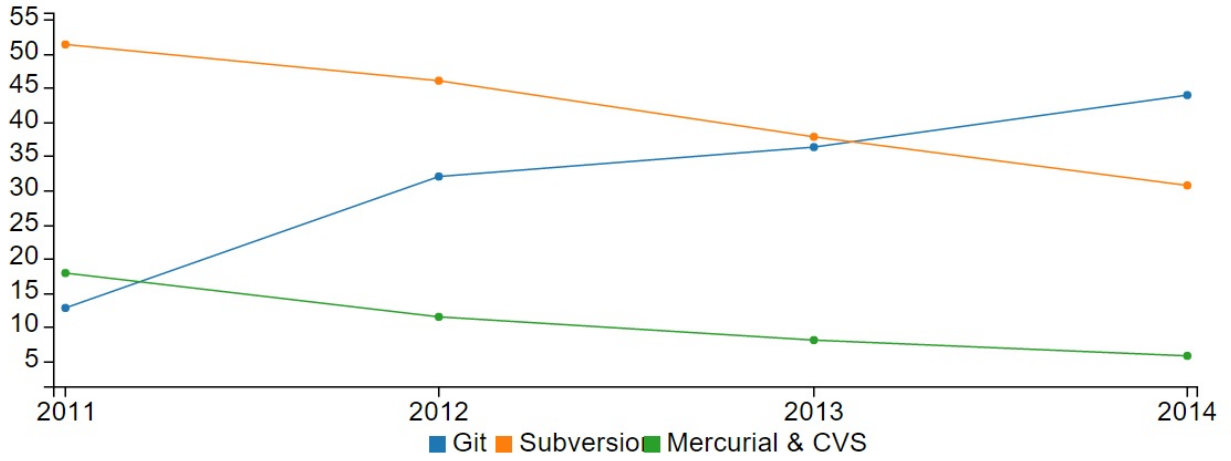

Image 2.1.1 - Popularity of different version control tools

Issue tracking

The quote "Software does not allow itself to be finished. Software is never finished" by Paul LeBlanc [24] shows the need for good issue tracking. There is an enormous list of issue-tracking solutions available [25], each with their own strengths and weaknesses. Interviews conducted by Kalliamvakou show that for example the combination of the GitHub issue-tracker enhances self-organization when assiging tasks and resolving conflicts [20]. However the GitHub solution is not optimal because the issue tracker lacks some critcial features such as severity, components and bug report templates [26].

Testing

The citation from Everett (2007) signifies the need for software testing. An estimated amount of $59.5B has been lost due to poor quality software in the period 2000-2007. With proper testing this amount could have been drastically lower [27]. With the software industry being booming, the need for testing will also increase [28]. While testing will not immediately cause users to be involved in the software development proces, it may attract users because they like working on high quality software.

Package management

Just as testing, package management might not be a primary driver for developers to start working on a project. In the article "Achieving quality in open-source software" by Aberdour (2007), the author identifies several key factors for participation and motivation in open-source software projects. These include code modularity and opportunities to learn new tools [29]. When a project uses package management such new tools are quickly identified. Furthermore the presence of package management mostly indicates modular software.

1. https://github.com/torvalds/linux ↩
2. http://subversion.apache.org/ ↩
3. https://hg.mozilla.org/ ↩
4. https://slack.com/ ↩

Engagement

One of the possible issues in Open Source Software Development is engagement in the project. Commercial development has several mechanisms to ensure the stability of the project, like interface specifications, plans and habits regarding how things are done[10]. Communication can be a substitute for most of these mechanisms, although the "communication-only" approach is not scalable. The stability of the project engages the participants in the project and ensures a proper end product. Extrinsic motivation, like money, reputation and peer pressure, can also boost the engagement in the project [29]. Open Source Software Development (OSSD) does not always have this stucture to build on. Most contributors in OSSD act based on intrinsic motivation[30][31], like gaining experience, being part of the community or doing it for fun. This type of motivation does not guarantee engagement, but does improve the mental condition while participating and the quality of the performance, due to the lack of outside pressure[32][33].

Still, OSSD has some ways to engage the contributors/developers in the process of software-development. One way, like *Netbeans.org* does, is the distribution of management positions that are limited to coordination roles[34]. Their task is to determine and announce what needs to be done and wait for volunteers to accept responsibility. This method gives structure without controlling the development process.
Another way is enabling quality control in a project. Thanks to peer review and quick feedback loops, contributors can get a lot of responses on the code they produced. This constant communication between contributors and community drives the contributors to develop higher quality code[35]. Thus, the community can be actively engaged in the quality of the code, by either giving or reacting to feedback.

Requirements Engineering

The field of requirements engineering is a part of systems and software engineering, and deals with structuring requirements. Requirements are useful in measuring to what degree software solves the problems it was built to solve. Requirements engineering can be viewed in terms of two groups of activities, corresponding with the iterative and incremental nature of the process: requirements development and requirement management [14]. To consider these activities, first a definition of a software requirement has to be established. Finally, a model for structuring the process of requirements engineering is considered.

The definition of a software requirement

In the "Guide to the Software Engineering Body of Knowledge" (SWEBOK) the IEEE[1] defines a software requirement as: 'a property that must be exhibited by something in order to solve some problem in the real world' [15]. Requirements can be imposed on the product as well as the process. An essential property of software requirements is that they are verifiable, either as an individual feature (functional requirement) or at the system level (non-functional requirement). In addition to this behavioral properties, requirements may have attributes such as a priority rating and a progression status.

Requirements development

Discovering, analyzing, documenting and validating requirements are all part of the process of requirements development [14]. This process is essential to defining a software architecture that works the way the end user expects it to work. This part of the requirements engineering process takes place before a software implementation is developed. The goal is to end up with a specification of the requirements the software has to adhere to.

One of the main challenges in requirements development in large systems is coordination [16], specifically coordinating dependencies. Dependencies may arise between requirements, resources and software that consumes these resources. To manage this coordination mechanisms are needed, such as standardization, problem decomposition or (third-party) expert counsel. Early dependency detection and analysis is key to determining which coordination mechanism to apply.

Requirements management

Throughout the software development process a requirement may change. This change may affect other parts of the system and in turn affect other requirements as well. Requirements management consists of activities related to managing these changes and tracing them to the corresponding parts of the system [14]. The goals of the this part of the process are [17]:

- supporting acquisition, specification, grouping and attribution of the raw captured requirements
- supporting their derivation to more detailed levels, keeping and adjusting attributes
- enabling test cases and test environments to be defined and linked
- enabling relationships between requirements, design, realization and test to be tracked and traced
- enable distributed development within the organization

Requirements management facilitates an iterative requirements engineering process [14]. In "Requirements for Requirements Management Tools", Hoffmann, Kühn, Weber, & Bittner [17] provide a requirements specification for any tool that attempts requirements management for large scale projects.

The Twin peaks model

In order to ensure a software implementation adheres to its requirements and keeps doing so, an iterative workflow is required. An example of an iterative model is the Twin Peaks model. Requirements and an architectural specification are developed concurrently, while continually separating problem structure and specification from those of the solution [18]. Combining this approach with problem frames [19] allows for a structural decomposition of problems into smaller problems that can be solved with a modular solution. The final workflow looks like this [18]:

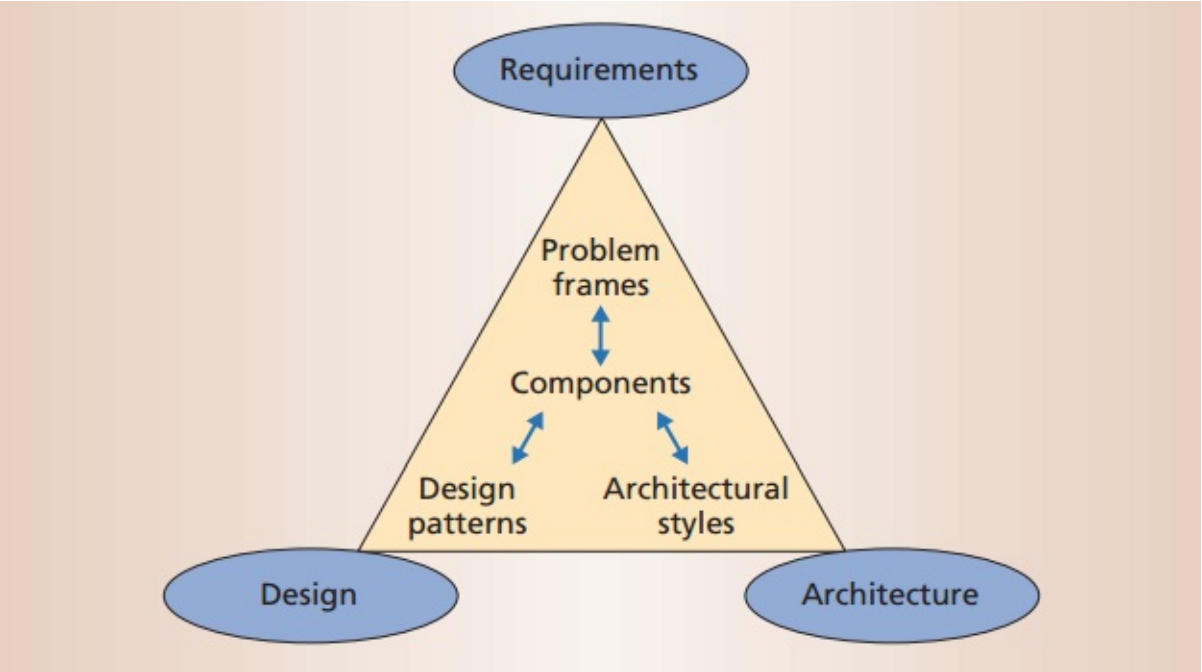

Image 3.1 - Workflow

> 1. http://www.ieee.org/ ↩

Integration with FeedbackFruits

To integrate with the existing FeedbackFruits platform, some requirements have to be met by any solution to the problem. This section lists these requirements, and offers some suggestions for meeting them.

Current platform structure

FeedbackFruits is an online education platform, built on modern web technologies such as Ruby on Rails [1], Ember.js [2], MongoDB [3] and Node.js [4]. The platform is structured with micro-services in the form of different servers that serve a single front-end, and are also capable of talking to each other.

The platform is heavily structured around activities, such as videos, documents and assignments. These activities have an API and can be extended to incorporate custom functionality. It comes with a test suite, that ensures any activity type complies with de API. A Ruby on Rails back-end aggregates the different engines that offer types of activities and serves the activities as part of the main FeedbackFruits API.

The activities get requested in an Ember.js front-end and displayed in a material design [5] user interface. Each type of activity can optionally be rendered using custom logic for that type, contained within an Ember addon [6].

Extensibility

At the moment of writing this report, there are two modular ways to extend the FeedbackFruits platform. The first and simpler way is to add a new type of activity. This can be done by creating a Rails engine [7] and implementing the activity according to the existing API. This will allow the engine to run within the FeedbackFruits platform. The activity can then be displayed in the front-end with an Ember addon for the new activity type. Alternatively, either an Ember addon or a Rails engine could be added by itself to offer respectively an existing activity type in a new way or a new activity type in an existing way.

The second way to extend the platform is to circumvent the back-end and link the Ember addon to an external back-end in the form of a micro-service. An advantage of this approach is that the service is only restricted by the communication with the Ember addon. effectively removing all restrictions in terms of implementation. One disadvantage is that another source of latency is introduced in the form of a new server.

1. http://rubyonrails.org/ ↩
2. http://emberjs.com/ ↩
3. https://www.mongodb.org/ ↩
4. https://nodejs.org/en/ ↩
5. https://www.google.com/design/spec/material-design/introduction.html ↩
7. http://guides.rubyonrails.org/engines.html ↩
6. https://www.emberaddons.com/ ↩

Conclusions

No existing solutions for the problem were found. The following subsections present the conclusions for the individual research questions. These conclusions can be used as a guideline towards implementing a solution for the problem.

What can be learned from existing (non-software) communities?

Looking at the different communities, it can be concluded that most of the communities have one thing in common: they work because they have a goal. In the case of the intentional communities: all but the practical communities have their ideals as a goal to fulfill. The practical communities make life easier rather than fulfilling an explicit goal. The goal of MMO-RGP communities is to make some part of the game-play possible or easier. This is because some quests can only be done in a party that forming one is a goal to achieve. Late in the game, when enemies can become exponentialy stronger, the need of trust-worthy party members becomes greater. This can be the reason and the goal to form a guild. A clear and good goal can make a guild stable and well-defined.
Software communities often have the goal of improving the quality of the software product. While the users report bugs or elements of the program that they are unsatisfied about, the developers can improve the program using the feedback from the users. This will result in greater users-satisfaction, wich is profitable for both user and developer.

Every community that undertakes something must have a clear and relatable goal. This goal is the foundation for the stability of the community and ensures that a community is worth joining.

What can be learned from existing efforts to involve a community with software development?

In short, there are a few key factors that can be identified in succesful software development communities:

- a small core team that actively communicates about the current and desired state of the software;
- use of GitHub (git, pull-requests, issue-tracker) or a similar system;
- modular code with standardized documentation, easy set up and tutorials;
- use of both peer reviews as well as formal testing;
- further stimulation of intrinsic and extrinsic motivation.

A small core team is necessary to provide a stable basis on which the rest of the community can rely. This team should contain members respected by the community and provide clear information about the state and future of the project. In relation to FeedbackFruits, this team can, consist of one person from FeedbackFruits and two or three people from the community. The members from the community can change per project and at least project issuer should take a place in the team.

GitHub is a proven tool that is familiar to almost every software developer. Studies have shown that is a very effective way of communicating within the community. However, when the code is difficult to understand, not properly tested or documented, or when it is difficult to set-up, the barrier for a new user to contribute is much higher. The use of modern tools and new techniques enables people to learn and intrinsically motivates people to contribute.

With the use of peer reviews not only the quality of the code is enhanced but also the motivation of the contributor. This motivation can be boosted even further by publicly acknowledging the contributors work in combination with some sort of a reward program.

How are requirements established and used in software development?

Section 3 describes the process of requirements engineering. Implementing structured requirements development and management early on can help deal with the challenges of building a large software systems. One of the main challenges is coordinating dependencies between requirements. This challenge can be dealt with using coordination mechanisms.

Additionally, the requirements engineering process should be kept iterative. The Twin Peaks model facilitates this and can be extended to work in conjunction with software architecture and design processes. This makes it possible to develop software modularly while still being able to verify the software meets a set of predefined requirements.

What limitations does the existing FeedbackFruits ecosystem impose on the software?

Section 4 describes how to integrate with the current FeedbackFruits platform. The first limitation imposed is the usage of material design [1]. Other limitations depend upon the method of extending the platform.

This first method is to provide a new type of activity. As such its scope is limited to activities. Furthermore, the implementation is restricted to be a Rails engine [2] and/or an Ember addon [3] and its API must extend the activity API. These restrictions stem mostly from the current implementation of the FeedbackFruits platform back-end.

The second method circumvents the FeedbackFruits back-end. It limits implementations only on the front-end side to be an Ember addon and does not impose the limition of the activity API. However, this also takes away the test suite that comes with the activity API. Because of this, it is more difficult to assess if the extension will be able to integrate with the current FeedbackFruits platform properly.

> 1. https://www.google.com/design/spec/material-design/introduction.html ↩

> 2. http://guides.rubyonrails.org/engines.html ↩

> 3. https://www.emberaddons.com/ ↩

Recommendations

No existing solutions were found that solve the specific problem posed in the problem description. However both Wikipedia and GitHub contain various aspects that can be used in a possible solution. Wikipedia is one of the few community based platforms that really is a sustained success. One of the reasons is that Wikipedia consists of three main groups: donators, contributors and moderators [1]. A shared factor between Wikipedia and GitHub is that subjects can be decomposed in multiple smaller parts. This modularity is something that repeatedly occurs in the previous parts of the research report and thus is likely to be essential for a successful community based platform. For software development GitHub is one of the best and most used tools. The challenge is to extend its philosophies to non-software development.

Based on these success stories and conclusions, a description is given of what an ideal solution looks like. This solution is presented in the form of a platform.

The ideal platform

The platform converts resources, mainly in the form of time and knowledge, into an extension to the current FeedbackFruits platform. The goal of this extension should be to fulfill a particular need. A goal can be identified as the answer to finishing the phrase: "With FeedbackFruits I would like to ..". Each goal can be divided in challenges, sub-goals that need to be met in order to meet the goal. If these challenges are to large to be solved by one or two pioneers, then the challenges must be divided into sub-challenges in such a way that they can be solved by one or two pioneers. This implies that the whole project can be solved by lots of different people by tackling smaller problems at the same time.

Use cases

Bob is a teacher, he would like to be able to ask question via the FeedbackFruits-platform. However, this functionality does not exist yet. Luckily for Bob he can request this functionality using the Bepstore. When he visits the bepstore, he sees a button 'create new goal' and clicks it. Bob is then able to fill in the title and description of his request. Afterwards he gets the possibility to share is newly made goal on twitter, facebook and feedbackfruits.

Esther is a computer science student but is a bit bored and is browsing facebook. She runs into a post by teacher Bob whose course she followed last year. She sees he would like to ask questions on the FeedbackFruits-platform and thinks this is a good idea. She has got

some idea about this feature and joins the core team of goal. Then she starts to break down the initial goal into smaller subgoals: multiple choice and open questions. Esther submits these proposals to the platform for review and already starts setting up the development toolset.

Peter is a designer and already actively contributes in the bepstore. In the 'new goals' section he sees that a design for multiple choice and open questions is desired. Peter joins the project and indicates he would like to be responsible for this. After an hour he has finished the initial designs and uploads them to the platform. The platform automatically shares his designs on 'myawesomedesigns.com'. A very experienced designer sees Peters design and comments on it.

After a while Bob checks on his goal and sees that a core team has formed and the initial idea has been broken down into smaller goals. Additionally, designs have been drafted. Bob has some ideas about these designs and comments on them. Esther gets a notification that some designs are ready for implementation and starts to develop them. When she is done, the core team gets a test report. Bob is happy and gives an update to everyone involved.

Resources

The platform facilitates a structured process to reach a goal. On a high level, this process consists of two parts: the gathering and spending of resources. The most important resources are time and knowledge. Less important resources can include money, servers and licensed materials. Resources can be gathered by creating a hype around the goal. Once enough resources have been acquired a team of pioneers can start implementing an extenions to achieve the goal. This will result in resources being spent.

Pioneers

Pioneers can be identified according to the resources they contribute. Pioneers can fall into one of three categories:

- Core team (time and knowledge)
- Contributors (knowledge)
- Backers (other resources)

The core team are a select group of pioneers that take the responsibility for a project. Dividing the goal into challenges and sub-challenges is the job of the core members, as such they will provide the structure for the software development process. Contributors can pick up one or multiple of these sub-challenges and implement the goal of those sub-

challenges. Reviewing code or designs also is a contribution to a sub-challenge. Backers are pioneers that try to improve the project without implementing or reviewing solutions for a (sub-)challlenge. This can be done with money, teaching material e.g.

The process

There are four main factors that are essential for the development of an extension: hype, requirements, code and updates. These factors correspond to different types communication between different categories of pioneers. For the development process to be clear, an iterative approach must be used. Each iterative step can be linked to a essential factor of the development. First hype must be created in order to gather necessary resources. When all the necessary resources are gathered, the requirements of the project can be created based on the goal. Code can be created based on the final requirements. The creation of code must also be done in iterations to ensure progres and code quality, where each iteration ends with an update to the contributors and the community.

Hype

Hype is created by promoting the project, either on social media or other means. Anyone, pioneer or not, can create hype for a project. Creating more hype not only stimulates backers to support the project but also motivates contributors. This effectively generates resources that can be consumed in the other parts of the process.

Requirements

Requirements are needed to structure how resources will be consumed in order to produce code. Good requirements are fundamental to a successful process, so the specification falls on the core team. The requirements need to be formulated clearly and dependencies between requirements need to be identified early on, resulting in communication dictating this part of the process.

Code

Code is produced by the core team and the contributors as a means of achieving the goal. The requirements should be reflected in the code produced by both the core team and the contributors. For this reason, the means of communicating about the code should facilitate referencing the relevant requirements as well.

Process update

One of the most important factors in maintaining a successful community is the ability to keep everyone engaged with the common goal. To achieve this, everyone in the community should be kept up-to-date on the current state of the project, as well as future plans. In addition to keeping the community engaged, this part of the process also bridges the current iteration with the next iteration. It ends the current iteration with a summary of the results, and provides an opportunity for hype to be created around the next iteration.

Integrations

Many of the suggested features already exist as separate tools. If such tools have an API, they could be neatly integrated into the proposed platform which prevents reinventing the wheel. For example GitHub has an extensive API with, among others, support for the creation of teams, pull-request and issues. Other integrations that can be explored are:

- crowdsourcing platforms (eg. Kickstarter, tilt.com, WePay, monster.com, pinterest and gratipay);
- integration testing (eg. Jenkins and Travis-CI);
- chat tools (eg. gitter, slack).

MVP/EVP

An initial list of requirements can be found below. This list is not exhaustive but should rather be seen as an itererative list. Positive and negative signs (++/+/-/--) correspond with the importance of the feature. The importances can be must have, should have, could have, won't have, which is according the MoSCoW model.

ToDo

As a user of the platform I would like to:

General improvements

- [+] have a general information page (v0.6)
  - [-] get tips on how to use the platform (v0.6)
- [+] goal tags (v0.5)
- [+] search/browse for goals (v0.5)
- [+] see my contributions (v0.6)

User engagement

- [++] communicate with other pioneers (v0.5)
  - [+] use different tools per category (v0.5)
  - [+] get updates (push, email, etc.) (v0.5)
- [+] create an account more easily (v0.6)
- [+] hype my goal (v0.5)
- [+] share my goal on social media (v0.5)
- [-] edit a user (v0.4)

Requirements engineering

- [++] prioritize sub-challenges (v0.4)
- [++] get additional information about the status of my goal, such as:
  - [++] quality (v0.6)
  - [+] functionality (v0.6)

Quality control

- [++] give feedback on someone else's work (v0.5)
- [++] get feedback on my work (v0.5)
  - [++] via peer review (v0.5)
  - [+] via automated testing (v0.5)

New

- [+] automatic repository generation (v0.6)
  - [+] boilerplate (v0.6)
  - [+] teams (v0.6)
  - [+] status checks (v0.6)
  - [+] milestones (v0.6)
  - [+] issues (v0.6)

Out of Scope

- [-] follow a tutorial on how to use the platform
  - [--] per user type
- [-] be able to join a project as a backer
- [-] view someone's reputation
- [-] get reputation
  - [--] have a ranking system
- [-] have some sort of gamification
- [-] bounty

- [-] available resources
- [-] hype factor
- [--] join a training party
- [--] be matched with an apprentice
- [--] be matched with a senior

Done

- [++] see a landing page
- [++] create an account
- [++] be able to join a project
  - [+] as a contributor
  - [++] as a core team member
- [++] create a goal
  - [+] set estimated necessary resources (actual results from problem decomposition)
- [+] edit a goal
- [++] view a list of goals
  - [-] view a list of trending goals
  - [+] view a list of newest goals
- [++] see a goal description
- [++] see a list of sub-challenges for a goal
- [++] use GitHub integrated
- [++] be able to decompose a goal into challenges in a structured way as specified in Requirements for Requirements Management Tools (Hoffmann, Matthias et al., 2004)
- [++] get information about the status of my goal, such as:
  - [++] progress
  - [++] milestones
  - [+] timeline

> [1]. https://en.wikipedia.org/wiki/Wikipedia:User_access_levels ↩

References

[1]	Meijering, Louise and Huigen, Paulus and Van Hoven, Bettina, Intentional communities in rural spaces, *Wiley Online Library*, 2007.
[2]	Meijering, Louise and van Hoven, Bettina and Huigen, Paulus, Constructing ruralities: The case of the Hobbitstee, Netherlands, *Elsevier*, 2007.
[3]	Sanguinetti, Angela, The design of intentional communities: A recycled perspective on sustainable neighborhoods, *Walden Fellowship, Inc.*, 2012.
[4]	Svensson, Karen, What is an Ecovillage, 2002.
[5]	Chen, Mark G, Communication, coordination, and camaraderie in World of Warcraft, *SAGE Publications*, 2009.
[6]	Rossi, Luca, MMORPG Guilds as Online Communities-Power, Space and Time: From Fun to Engagement in Virtual Worlds, 2008.
[7]	Kelly, Jeremy Neal, Play time: An overview of the MMORPG genre, 2004.
[8]	Verhagen, Harko and Johansson, Magnus, Demystifying Guilds: MMORPG-playing and norms, 2009.
[9]	Ang, Chee Siang and Zaphiris, Panayiotis, Social roles of players in MMORPG guilds: A social network analytic perspective, *Taylor \& Francis*, 2010.
[10]	Mockus, Audris and Fielding, Roy T and Herbsleb, James D, Two case studies of open source software development: Apache and Mozilla, *ACM*, 2002.
[11]	Osterloh, Margit and Rota, Sandra, Open source software development—Just another case of collective invention?, *Elsevier*, 2007.
[12]	Sundaram, Hari *et al.*, Understanding community dynamics in online social networks: a multidisciplinary review, *IEEE*, 2012.
[13]	Jones, Quentin, Virtual-Communities, Virtual Settlements \& Cyber-Archaeology: A Theoretical Outline, *Wiley Online Library*, 1997.
[14]	P *et al.*, An effective requirement engineering process model for software development and requirements management, *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, 2010.
[15]	Bourque, Pierre and Fairley, Richard E., {Guide to the Software Engineering Body of Knowledge - SWEBOK v3.0}, *IEEE CS*, 2014.
[16]	Crowston, Kevin and Kammerer, Ericka Eve, Coordination and collective mind in software requirements development, *IBM*, 1998.
[17]	Hoffmann, Matthias *et al.*, Requirements for requirements management tools, *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, 2004.
	Nuseibeh, Bashar, Weaving together requirements and architectures, *IEEE*,

[18]	2001.
[19]	Hall, Jon G *et al.*, Relating software requirements and architectures using problem frames, *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, 2002.
[20]	Kalliamvakou, Eirini *et al.*, Open Source-style Collaborative Development Practices in Commercial Projects Using GitHub, *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, *IEEE Press*, 2015.
[21]	Thissen, M Rita *et al.*, Communication tools for distributed software development teams, *Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research: The global information technology workforce*, 2007.
[22]	Ian Skerrett, Eclipse community survey 2014 v2, *Eclipse*, 2014.
[23]	Stephen, The case for Git, *NetInstructions*, 2015.
[24]	LeBlanc, Paul J, Writing Teachers Writing Software: Creating Our Place in the Electronic Age. Advances in Computers and Composition on Studies Series., *ERIC*, 1993.
[25]	Multiple authors, Comparison of issue-tracking systems, 2016.
[26]	Zhou, Bo and Neamtiu, Iulian and Gupta, Rajiv, A Cross-platform Analysis of Bugs and Bug-fixing in Open Source Projects: Desktop vs. Android vs. IOS, *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, *ACM*, 2015.
[27]	Everett, Gerald D and McLeod Jr, Raymond, Software testing: testing across the entire software development life cycle, *John Wiley \& Sons*, 2007.
[28]	Asif Mohammadullah, Software Engineering – One of the most Booming Streams of Engineering, *Ignite Engineers*, 2014.
[29]	Aberdour, Mark, Achieving quality in open-source software, *IEEE*, 2007.
[30]	Bitzer, J{\"u}rgen and Schrettl, Wolfram and Schr{\"o}der, Philipp JH, Intrinsic motivation in open source software development, *Elsevier*, 2007.
[31]	Roberts, Jeffrey A *et al.*, Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects, *INFORMS*, 2006.
[32]	Ryan, Richard M and Deci, Edward L, Intrinsic and extrinsic motivations: Classic definitions and new directions, *Elsevier*, 2000.
[33]	Edward L. Deci, Richard M. Ryan, The "What" and "Why" of Goal Pursuits: Human Needs and the Self-Determination of Behavior, *Taylor & Francis, Ltd.*, 2000.
[34]	Jensen, Chris and Scacchi, Walt, Collaboration, leadership, control, and conflict negotiation and the netbeans. org open source software development community, *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, 2005.
[35]	Von Krogh, Georg *et al.*, Carrots and rainbows: Motivation and social practice in

[35]	open source software development, 2012.
[36]	van den Berk, Ivo and Jansen, Slinger and Luinenburg, L{\'u}tzen, Software ecosystems: a software ecosystem strategy assessment model, *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 2010.
[37]	McDermott, Richard, Knowing in community, 2000.

Glossary

Communities of practice

Groups of people who share information, insight, experience, and tools about an area of common interest.

2. Software ecosystems 2. Software ecosystems

Niche creation

The increase of niche players in a software ecosystem.

2. Software ecosystems 2. Software ecosystems

Niche players

A company holding a specialized and profitable part of a commercial market.

2. Software ecosystems 2. Software ecosystems

Tacit knowledge

Ideas and insights that are not documented and hard to articulate.

2. Software ecosystems 2. Software ecosystems