University of Passau

Department of Informatics and Mathematics

Bachelor Thesis

Activity of Core and Peripheral Developers: A Case Study on Mozilla Firefox

Author:

Ferdinand Frank

frankfer@fim.uni-passau.de

Matr. No.: 72705

September 7, 2017

Advisors:

Prof. Dr. Sven Apel

Chair of Software Engineering I

Claus Hunsen

Chair of Software Engineering I

Thomas Bock

Chair of Software Engineering I

Frank, Ferdinand:
Activity of Core and Peripheral Developers: A Case Study on Mozilla Firefox
Bachelor Thesis, University of Passau, 2017.

Abstract

On typical open-source projects, developers work on a voluntary basis and can decide on their own how much they want to contribute to the project's source code. Hence, the contribution activity as well as the collaboration patterns differ a lot between the developers of such projects. As identifying the most active and influential developers can be crucial for the purposes of coordination and support between the developers, several approaches have been established to classify the developers as core and peripheral in open-source projects. In this work, we investigate the developer classifications produced by four different established operationalizations in terms of consistency, the resulting role stability of the developers, and the number of classified developers on 15 consecutive versions of the Web browser Mozilla Firefox. Since Firefox is a large open-source project, which is especially well known for its rapid release model, we further investigate the resulting contribution activity patterns of both developer groups within such a short development range of a new version. We found that our applied operationalizations lead to similar results according to the number of classified developers in each class, since the most Firefox developers are classified as peripheral in each produced classification. Further, only on a classification based on the number of contributed commits to the Firefox version-control system, we could provide evidence of a characteristically high role stability of core developers, whereby a characteristically low role stability of peripheral developers could be shown best on a classification based on the number of collaborations with other developers. Regarding the contribution patterns of core and peripheral developers, we could not find any significant differences between both developer classes, primarily because our applied operationalizations for the classification lead to different activity distributions among these two classes. However, both classes are embossed by a pretty instable contribution pattern in terms of their number of contributed commits. Because of these findings, we conclude that our selected operationalizations lead to a consistency between "fair" and "slight" according to the produced developer classifications on the Firefox project, but that the agreements are clearly not as high as on similar large open-source projects investigated on related studies.

Contents

List of Figures

List of Tables

List of Acronyms

OSS	open-source software	1
VCS	version-control system	1
LOC	lines-of-code	5

1 Introduction

Since the introduction of open-source software (OSS) in the 1990s, the use of the open-source model within software projects has gained a lot of popularity [FA05]. However, as an OSS project is characterized by giving all of its users the possibility and the right to collaborate to the source code on a voluntary basis, there is no assigned definition and separation of different developer roles as it is on conventional software projects with a closed-source model. OSS developers, therefore, generally decide on their own how much they want to contribute to specific parts of a project's source code and with whom they want to collaborate without getting assigned by a supervisor. Hence, there are some developers contributing the main development work in a long-term involvement on the project, which can be seen as the core developers of the project, and also developers having a short-term involvement with irregular development contributions and which are mostly just operating in the role of a bug fixer or bug reporter, which can be defined as peripheral developers [YK03, JAHM17, NYN$^+$02].

Although it is easy to separate the developers into the classes core and peripheral in theory, it is rather difficult to realize this classification on the contribution activity data of real OSS projects. But, this can be crucial to provide an insight into the composition of the development team and the collaborative relationships of developers in large OSS projects. This insight can further identify influential developers according to their knowledge, collaborative patterns, and skills for coordination and support purposes within OSS large projects [CH13]. Therefore, an appropriate operationalization is necessary which classifies the developers of an OSS project as core and peripheral based on their corresponding contribution activity or further developer characteristics. A popular approach, therefor, is the application of count-based operationalizations on the project's commit data of the version-control system (VCS) to count the number of contributions, e.g., lines of code or commits, by each developer and to thereon apply thresholding to classify the developers as core and peripheral according to their amount of contributions. However, core developers are also distinguished by broadly communicating with many other developers for the reasons of support and coordination, because of their characteristically detailed knowledge of the project's source code. As count-based operationalizations do not reflect any interactions between the developers, an additionally common approach is to build networks consisting of developers as nodes and edges between developers contributing technically related changes or making subsequent contributions to a developers mailing list to represent those relationships between the developers. Appropriate network-based operationalizations are then used to classify the developers in their corresponding roles according to their position in the network and their communication ties to other developers [JAHM17].

The classification of developers as core and peripheral using count-based as well as network-based operationalizations have already been investigated on several OSS projects, e.g., Linux, QEMU, and Django, whereby it has been shown that count-based and network-based operationalizations produce developer classifications which are largely consistent and valid [JAHM17].

In this study, we use those established operationalizations to classify the developers as core and peripheral and analyze the agreement of our classifications with the characteristic contribution activity and involvement patterns of both developer groups on a further large OSS project, which is well-known for its fast release model: Mozilla Firefox

Mozilla Firefox, which is one of the most popular Web browsers in the world, was released under an open-source license since version 1.0 [Mozf]. With their huge amount of source code using over ten different programming languages, the Mozilla Firefox project offers a lot of opportunities for potential developers to collaborate on their source code, whereby already over 2,500 volunteer developers contributed to the Mozilla Firefox code base [Moz17a, Mozl, Mozd]. As in other open-source projects, the development activity strongly varies between the Firefox developers especially since an adaption to a faster release model after the release of Firefox version 4.0 in March, 2011, which lead to a release of a new Firefox version every six to eight weeks [KADZ15]. As developers only have such a short time period for the development of new features as well as bug fixes in this adapted release model, our goal is to further investigate the distribution of the core and peripheral developer activity within such short version development ranges of Firefox.

Therefore, we firstly analyze previous works about the classification of developers as core and peripheral and examine the used operationalizations to conduct the classification of the developers in our study. After that, we take a closer look on the adapted faster release model of Mozilla Firefox, their version control management, and their feature development process, so we are able to understand the release cycle of a new Firefox version and can thereon analyze the developer activity within the development process of new Firefox versions. We then describe, how we extract the necessary commit data from the official Firefox VCS, whereby we focus on the Firefox versions 37 to 51. This data is used to classify the developers as core and peripheral according to their technical contribution activity and collaboration patterns using established count-based and network-based classification metrics, which we discuss in the study design. Subsequently, we introduce our research questions and hypotheses, on which we focus on the analysis of the role stability of each developer class over multiple version development ranges to examine the typical long-term involvement of core developers and the short-term involvement of peripheral developers. Another focus is to analyze the peripheral developer activity right before a release of Firefox, as we assume that especially peripheral developers will mostly commit bug fixes or small enhancements right at the end of a version development range, whereby their activity should increase at that point. At the main part of this Bachelor's thesis, we then present the results of our hypotheses and critically evaluate them by discussing some threats to the validity.

In our study, we found that the most developers are classified as peripheral in each of our selected FIREFOX version for each applied operationalization. Further, we only observed a high role stability of core developers when classifying the developers based on their number of contributed commits to the FIREFOX VCS. Since all our applied operationalizations lead to inconsistent results regarding the contribution patterns of core and peripheral developers, we could not provide evidence of a potential higher contribution activity of peripheral developers at the end of a version development range. Due to these inconsistencies, we could further not observe such a high agreement of the classifications produced by our selected operationalizations as it has been found on other large open-source projects in the context of previous empirical studies.

2 Background and Related Work

Before we present a detailed look on the release model and the development process of Mozilla Firefox as well as the details regarding the design of our study, we firstly take a closer look on previous studies about the classification of developers in OSS projects and analyze the results of existing related work on Mozilla Firefox.

Multiple empirical studies already investigated the composition of development teams by analyzing technical contributions of developers to the source code of different open-source projects like Apache httpd [MFH02] or Linux [LC03, NYN$^+$02]. These studies found that development teams in open-source projects often have a hierarchical structure, in which only a small percentage of the developers is responsible for the majority of the development work. Small enhancements, bug reporting, and bug fixes on the other hand are quite decentralized and are mainly conducted by developers with generally small contributions to the project, what leads to the common finding that developers can be well classified in different roles according to their technical contribution activity [YK03, CWLH06, MFH02, LC03, JAHM17].

Some studies even went further and additionally analyzed the social perspective of OSS by studying *communication patterns* on mailing lists as well by investigating *collaboration patterns* within the source code using VCS data. These investigations resulted in the finding that developers of OSS can also be divided into different groups according to these social patterns [MFH02, JAHM17].

Even though the widespread onion model, introduced by Nakakoji et al. [NYN$^+$02] within a case study of four different open-source projects, classifies developers within OSS in eight different roles extending from passive users over bug reporters to project leaders, the prevailing simplification of this model distinguishes the developers into the coarsely groups of core and peripheral developers [CWLH06, JAHM17].

Under consideration of technical aspects, Joblin et al. define core developers as those, who ”play a central role in the communication and leadership structure and have substantial communication ties to other core developers” [JAHM17]. Further, core developers distinguish themselves by a stable long-term involvement within the project. Regarding the social perspective, core developers ”typically exhibit strong ownership over particular files that they manage, they often have detailed knowledge of the system architecture, and they have demonstrated themselves to be extremely competent” [JAHM17]. Peripheral developers on the other hand ”are primarily involved in identifying code-quality issues and in proposing fixes, while also participating moderately in development-related discussions” [JAHM17] and usually have an ”irregular or short-term involvement” [JAHM17].

Next to the finding that developers can be classified as core and peripheral according to their technical contribution activity as well as to communication and collaboration patterns, several metrics have already been elaborated to accomplish this classification of OSS developers. The most common ones are based on the data of VCS, mailing lists, bug trackers, and even interviews with corresponding developers [JAHM17, TRC10, MFH02]. Within a case study of ten well-known open-source projects to analyze the classification of developers into core and peripheral, Joblin et al. [JAHM17] divide the applicable metrics for the classification into count-based and network-based operationalizations.

2.1 Count-based Classification

Count-based operationalizations are based upon the count of distributions to the project's source code on the VCS. Common approaches therefor are to count the number of commits a developer has authored, which is defined as the *commit count*, or the number of changed lines a developer has contributed within these commits, formally defined as the *lines-of-code (LOC) count*. Joblin et al. assume that core developers generally have more commits and much more changed lines than peripheral developers, due to their responsibility for the main development work, whereby these count-based operationalizations should lead to a realistic classification of the developers. Further, as core developers often have a deep knowledge of the project and a distinctive technical expertise, it is assumed that these developers make recommendations for new features or problem solutions within the developer mailing list more often than peripheral developers. Therefore, core developers should have more mail contributions to the developer mailing list, whereas the count of these contributions, defined as the *mail count*, is also a good approach for the classification of the developers as core and peripheral [JAHM17].

To define the transition between core and peripheral developers based on the classifications resulting from count-based operationalizations, a specific threshold is crucial. A popular approach established by previous works on OSS by Terceiro et al. [TRC10] and Mockus et al. [MFH02] is to classify developers as core that are together responsible for 80% of the work load based on the examined metric, as it is also applied on the investigation of developer classifications by Joblin et al. [JAHM17]. Therefor, as exemplary shown on Figure 2.1, the developers are ordered descending according to either their commit count or their LOC count and are sequentially classified as core until the sum of the work load of the classified core developers make up 80% of the total work load of all developers. Consequently, all remaining developers are classified as peripheral.

2.2 Network-based Classification

The above mentioned threshold can also be applied on a further approach of operationalizations, which are based upon networks built by considering social and/or technical aspects of the investigating OSS project [JAHM17]. Here, developer-mailing-list data is used to express the social aspects of the project by connecting developers within the network making contributions to the same communication thread in the developer mailing list (mail network). Additionally, the influences

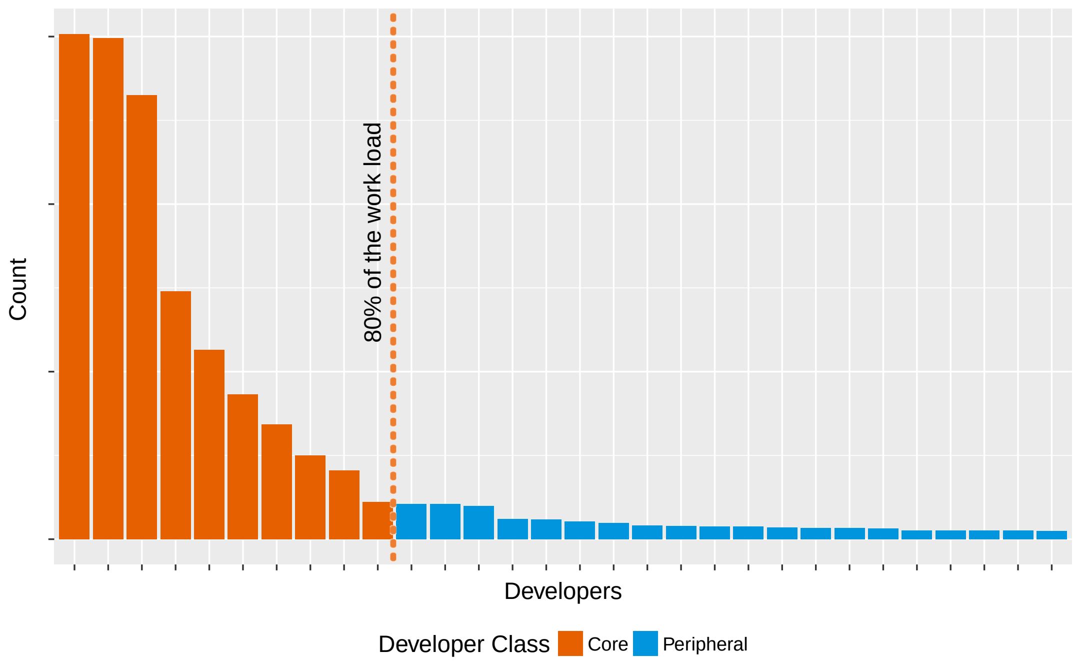


Figure 2.1: Exemplary classification of developers based on a count-based operationalization using an 80% threshold

of the technical aspects within the network are realized by analyzing function-level information on the commit data of the project's VCS. The network, therefore, connects developers which edited related lines of code or the same functions of the source code to represent the technical relationship between the developers (VCS network).

To classify the developers as core and peripheral based on the developer networks, Joblin et al. [JAHM17] examined five different operationalizations. The first one, defined as the *degree centrality*, counts the number of in- and outgoing edges of each developer within the network to extract their local importance. This operationalization is based on the assumption that core developers modify more distributed lines of code than peripheral developers to support others with their technical expertise and therefore have more technical relationships to many different developers. A similar approach which represents the global importance is the *eigenvector centrality*. Next to the degree of a developer, this metric also considers the importance of the connected developers. Here, it is assumed that peripheral developers have less connections to developers in central positions than core developers because of their lower contribution activity on main features. Another operationalization, defined as *hierarchy*, analyzes cohesive groups within the developer network and classifies developers as core which can not clearly be assigned to one single cohesive developer group, but are connected to multiple groups, as it is assumed that these developers are partially responsible for the coordination of these cohesive developer communities. A similar approach is introduced as the *core-peripheral block model*. In this model, it is assumed that core developers are highlighted as densely connected nodes within the network, due to their high communication and coordination responsibility to other core and peripheral developers. Peripheral developers, on the other hand, are primarily working isolated on the OSS project and are mainly connected to few core developers for technical support requests instead of being connected to other peripheral developers. The fifth network-based operationalization aims at studying the transition between different developer roles over time, defined as the *role stability*. As core developers distinguish themselves by a consistent long-term involvement in the project, their role stability should be clearly higher than for peripheral developers, which are characterized by an inconsistent short-term involvement.

2.3 Statistical Consistency of the Classification Approaches

As the count-based and the network-based operationalizations represent two different approaches for the classification of developers, Joblin et al. [JAHM17] investigated the validity of these operationalizations within his study of ten different open-source projects. The results have provided evidence of the statistically consistency of the developer classifications based on count-based operationalizations using mailing-list data and on count-based operationalizations using VCS data. Moreover, they could confirm the previously described outstanding characteristics of core developers within developer networks, which are the placement in globally central positions with a high positional stability and hierarchical embedding. Further, the different classifications of developers as core and peripheral on the ten different open-source projects using all of the previously described operationalizations produced sufficient consistent results, whereas the count-based and the network-based operationalizations can be seen as equally valid. However, an additionally conducted online survey, in which the developers of the investigated projects were asked to report the developer roles of their project according to their own perception, resulted in a stronger agreement with the applied network-based operationalizations.

2.4 Related Work on Mozilla Firefox

Next to the analysis of good approaches to classify developers in OSS projects, also several studies have already been conducted on the Mozilla Firefox project which are related to the topic of this Bachelor's thesis.

In 2002, Mockus et al. [MFH02] analyzed the development process, the composition of the development team according to core and peripheral developers, as well as their coding activity, especially regarding bug fixes, on the Firefox project. These analyses were conducted on the change history of selected modules of the Firefox source code maintained on CVS as well as problem report data on their bug tracking system Bugzilla. They found, amongst others, that on each investigated module much more developers only contributed bug fixes or reported bugs than contributed a high number of commits with many changed LOC. Since core developers of a Firefox module according to this study are the most active developers who together contributed, at least, 83% of the total commits and 88% of the total changed lines in the corresponding module, only up to 35 developer could be classified as core on each module. On the other hand, up to 129 developers contributed bug fixes, and a yet larger group of up to 3,000 people reported bugs on each module. Further, only one of those core developers was active in all investigated modules and two-thirds in only one of those modules. However, it should be mentioned that this study of the Mozilla Web browser has been conducted before the official release of version 1.0.

Furthermore, Khomh et al. [KADZ15] investigated the impact of the change of the Firefox release model from a traditional one with irregular scheduled releases of new versions to a rapid release model with frequent releases, which we will further discuss in the following chapter. Next to the findings that crashes of the Firefox browser occur earlier in the Firefox versions resulted from the rapid release model and that the development of those versions are distinguished by faster bug fixes, the study further analyzed the correlation between the release model and the coding activity on the case of Firefox. In this context, it was found that the majority of the Firefox developers make changes on multiple different files rather than exhibiting responsibility over a few files and that they contribute more bug fixes for existing features than developing new features, since the change to the rapid release model.

3 Mozilla Firefox

In this chapter, we present the current release model, the version control management, as well as the feature development process of our case study project Mozilla Firefox in order to correctly specify and conduct our analyses.

Mozilla Firefox is one of the most popular Web browsers and is developed by the Mozilla Corporation under an open-source license with a world-wide market share of approximately 7.97% in 2016 [Sta].

In the beginning known under the name Phoenix, the Web browser was initially released under open-source license on September 23, 2002. The official release of the first main version named as Firefox 1.0 occurred two years later, on November 9, 2004 [Mozb][Mozc]. Since then, a traditional release model was established with irregular scheduled releases of new versions and three different release channels to test and stabilize the features before they got released [Moz11]. As some competitors like Google Chrome already had a faster release model with frequent releases of new versions back then, the Mozilla Corporation decided to also change to a faster release model with releases of new Firefox versions every six weeks after the release of Firefox version 4.0 in March, 2011 [Sha11].

In the following sections, we discuss the design of this adapted so-called rapid release model as well as its resulting development process, to be able to analyze the activity of the developers within this release model.

3.1 Release Model

As mentioned above, Firefox is currently following a rapid release model since version 4.0, originally based on a new Firefox release every six weeks. To respond better to emerging user and market needs, the Mozilla Corporation even decided to make the release cycle more variably by moving to a six-to-eight week release cycle for Firefox in 2016 [Moz16]. But this does not mean that the development of a new version only takes six to eight weeks. Before the introduction of the rapid release model, every Firefox version flowed through three release channels called *Nightly*, *Beta*, and *Release* to test and stabilize new features. With the switch to the rapid release model, a fourth channel with the name *Aurora* was introduced [Moz11]. So, every six to eight weeks, a developing Firefox version moves from one channel to the next one leading to a complete development time of 18-24 weeks for one Firefox version until it is published on the Release channel, as shown in Figure 3.1.

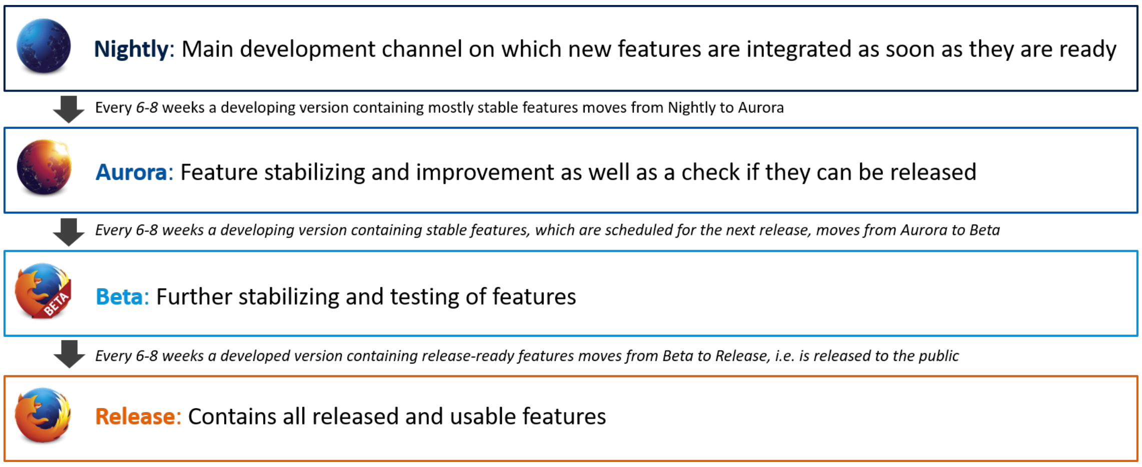

Nightly: Main development channel on which new features are integrated as soon as they are ready
Every 6-8 weeks a developing version containing mostly stable features moves from Nightly to Aurora
Aurora: Feature stabilizing and improvement as well as a check if they can be released
Every 6-8 weeks a developing version containing stable features, which are scheduled for the next release, moves from Aurora to Beta
Beta: Further stabilizing and testing of features
Every 6-8 weeks a developed version containing release-ready features moves from Beta to Release, i.e. is released to the public
Release: Contains all released and usable features

Figure 3.1: Release channels of the FIREFOX rapid release model

The Nightly channel as the first release channel is responsible for the main development, on which a new FIREFOX version starts to get developed. New features and bug fixes will be integrated into this version on the Nightly channel as soon as they are ready developed, successfully tested, and got reviewed by a responsible reviewer. After six to eight weeks the version from the Nightly channel including all stable features will be transferred to the Aurora channel. On this channel, the features will be further stabilized, improved, and checked if they are able to get released by the MOZILLA management. The Beta channel then inherits these features with its version from the Aurora channel, where further testing of the features occur with a broader audience of users and developers. After this complete development time of 18-24 weeks, the developed FIREFOX version including its release-ready features from the Beta channel will then be pushed to the Release channel, i.e., a new FIREFOX version is released to the public. [KADZ15, Mozj].

To be able to release a new FIREFOX version to the public every six to eight weeks, three FIREFOX versions are always developed in parallel. When the features for one version are imported from the Aurora channel to the Beta channel, the features from the next version are simultaneously imported from the Nightly channel to the Aurora channel and so on. This leads to the official release cycle, exemplary demonstrated for FIREFOX versions 48–52 in Figure 3.2.

FIREFOX version 48 was developed in the Nightly channel for six weeks, before the version including its stable features got pushed to the Aurora channel. With this push, the development of FIREFOX version 49 has started on the Nightly channel. After seven weeks of feature stabilizing of version 48 in the Aurora channel, it was pushed to the Beta channel for another six weeks of testing by Beta users. At the same time version 49 got pushed from the Nightly channel to the Aurora channel and the new FIREFOX version 50 has been introduced on the Nightly channel. After further six weeks, version 48 then got finally released as an official FIREFOX version on the Release channel, while version 49 moved to the Beta channel and version 50 to the Aurora channel [Moz17c].

In the following sections and chapters of this work, we use the term ”version development range” to designate the six-to-eight week time interval for the development of a FIREFOX version in any of the four release channels.

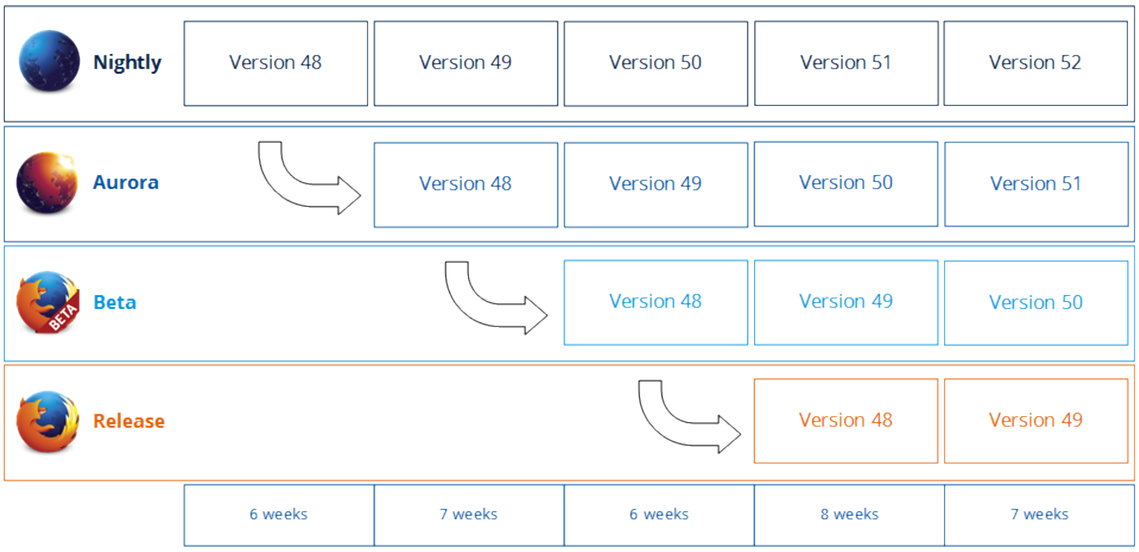


Figure 3.2: Firefox release cycle for versions 48–52

3.2 Version-Control Management

Each release channel of Firefox has its own version-control repository managed with Mercurial[1]. Important milestones of each version development range of a release channel, e.g., the start point and the end point, are tracked by VCS tags on the corresponding release-channel repository. Additionally, a release-channel repository inherits all tags of the previous release-channel repositories. The repositories of the Nightly, Aurora, and Beta channels further maintain an extra tag representing the start of a version development range in the following release channel. The tag management of each release channel repository is exemplarily shown for Firefox version 46 in Figure 3.3.

Nightly, as the first release channel, maintains no specific tags corresponding to its own channel on its repository named *mozilla-central*. But, as mentioned before, Nightly manages tags representing the start of a new version development range on Aurora which are the only tags on the Nightly repository. The repository *mozilla-aurora*, which represents the release channel Aurora, maintains tags representing the start and the end point of version development ranges on the Aurora channel as well as a tag representing the start of a version range on the Beta channel. The Beta channel repository *mozilla-beta* inherits all tags of the Aurora channel and further maintains tags representing the start point, the end point, and multiple build as well as release milestones of version development ranges on the Beta channel. As on the Nightly and on the Aurora channel, the Beta channel repository also maintains a tag for the start point of the version development ranges on the following channel, which is the Release channel in the case of the Beta channel and, therefore, marks the release of a new Firefox version. Finally, the Release channel repository *mozilla-release* inherits all tags from the Beta repository and additionally tracks the end of a release cycle on the Release channel as well as further build and release milestones [Mozh, Mozj].

[1]https://hg.mozilla.org/

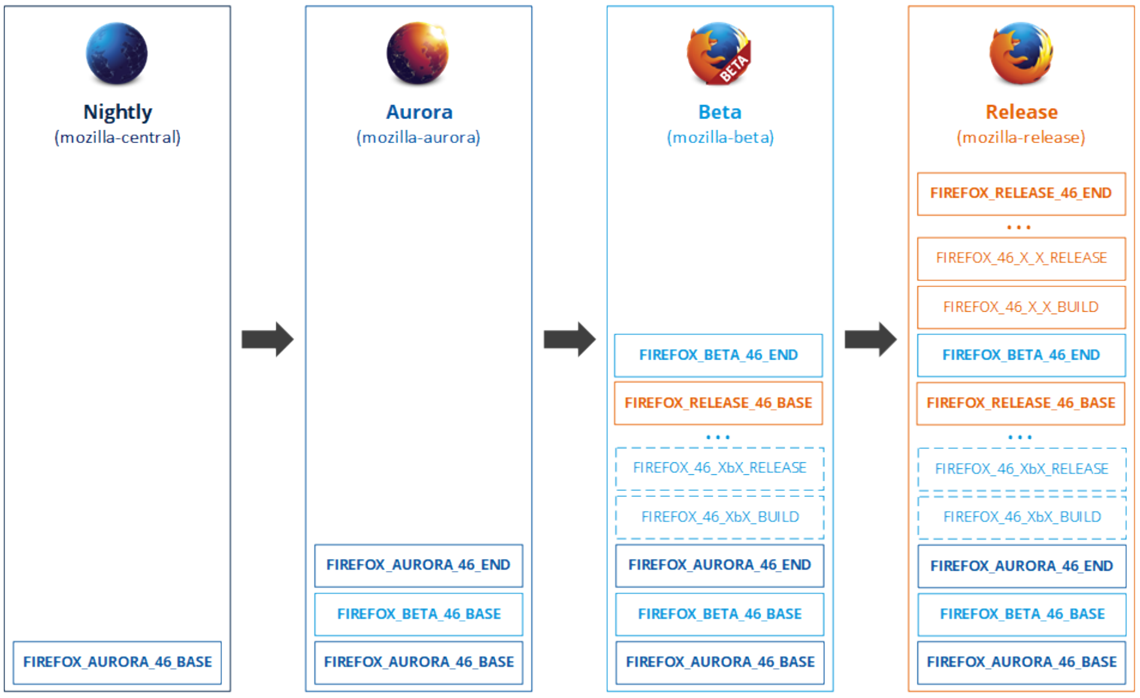


Figure 3.3: VCS tags on all release-channels for FIREFOX version 46

Each release-channel repository on MERCURIAL is also mirrored on an independent branch on the VCS GIT, which can be viewed on GITHUB[2]. The only issue with this mirror is that no version tags are managed on GIT since January, 2016, whereas no assignment of the commits to their corresponding versions can be done automatically by using tag names [Moza].

3.3 Feature Development Process

Since MOZILLA FIREFOX is a very large project developed using more than ten different programming languages with a predefined guideline for developers to contribute to the FIREFOX source code [Moz17a], it is important to understand the development process of a new FIREFOX feature or bug fix, to be able to later understand and judge the results of the developer activities.

Every new contribution to the source code, which is called a "patch" on the FIREFOX project, starts with an entry in the FIREFOX bug database called BUGZILLA, which needs to be referred to in the later commit message. If a contributed patch does not refer to one of these filed bugs in BUGZILLA, the code will not be accepted. Therefore, the first step for every potential FIREFOX developer is to choose or to create a new bug in the bug database and register oneself as the assignee of the bug, to notify other developers about the responsibility for that bug. This first step of a contribution to the source code is defined as *Preparation* in the FIREFOX contribution workflow, as shown in Figure 3.4.

Since a responsible developer exists for every module on the FIREFOX source code, the next step in the FIREFOX contribution workflow is to verify with the corresponding module owner that the proposed changes are acceptable. The module owner in

[2]https://github.com/mozilla/gecko-dev

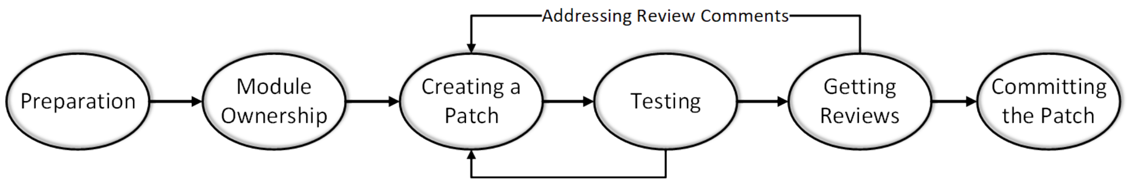


Figure 3.4: Workflow to submit a patch to the Firefox source code, adapted from [Moz17b]

that context is a developer who has an advanced expertise with the code in the corresponding module as wells as with its placement in the Mozilla Firefox codebase and is largely involved in developing parts of the module himself [Moz17b, Mozk].

So, only after the presence of a corresponding bug in the bug database Bugzilla and the verification of the proposed changes with the module owner, a potential developer can start to develop the new patch [Moz17a, Moz17b].

Further, the contribution guideline requires every patch to be tested with automatic or manual tests, called *Litmus tests*, after it has been developed. If these are successful, the patch can be submitted and a review by the module owner can be requested on the official review board. By this time, the patch will not occur on Nightly or on any other of the four release channels, yet. Only after the patch has been successfully reviewed and the contributor has fixed any complained problems with the patch, it can be pushed to the official Mercurial repository of the Nightly channel. This push can only be executed by community members with commit access. Contributors which do not have these permission need to request a push by adding the "checkin-needed" keyword to the bug in the Bugzilla database, whereby it will be pushed by a member with commit access, as soon as the commit is recognized. The developed feature or bug fix will then be released with the currently developed version in the Nightly channel (see Figure 3.2) [Moz17a, Moz17b].

Urgent security fixes are treated differently to normal bug fixes. As soon as a security vulnerability gets reported, an individual group of developers called *Mozilla security bug group* coordinated by a responsible security module owner will try to fix the vulnerability as soon as possible. In dependence of the severity of the vulnerability, an unscheduled release including urgent security fixes may be performed, which flows faster through the four release channels than normal bug fixes, to protect their users from potential consequences [Moze, Mozh, KADZ15].

4 Study Design

In this chapter, we give details of our Firefox case study to analyze the classification of the developers as well as their development activity and role stability over multiple Firefox version development ranges. Therefore, we first specify our addressed research questions, followed by a description of the corresponding data collection including the procedure of processing and validating the data with respect to our research questions. Based on the extracted data, we then define our hypotheses and the corresponding investigation in relation to our specified research questions.

4.1 Research Questions

In this study, three different research questions are addressed, whereby the first two are focused on the case of Firefox itself with its classification of developers as core and peripheral, the role stability of the developers, as well as their activity distribution over multiple version development ranges. The third one, on the other hand, investigates the agreement of the applied classification metrics to verify the validity of these operationalizations for the classification of the developers as core and peripheral.

Since core developers play a central role in OSS projects and should be the minority in large-scale OSS development teams according to their definition presented in Chapter 2, we are interested in the actual composition of the Firefox development team regarding the two defined developer classes. Additionally, core developers are defined to have a stable long-term involvement, whereas peripheral developers usually have an irregular or short-term involvement. To address and verify these characteristics of our developer classes, we further investigate the role stability of core and peripheral developers on the case of Firefox in the context of this research question:

> ***RQ1:*** *How many* Firefox *developers can be classified as core or peripheral for each considered development range and how stable is the classification of developers into core and peripheral over time?*

Further, we focus on the influence of the developer roles on the contribution process. As already described in Chapter 2, peripheral developers distinguish themselves by mainly contributing bug fixes or small enhancements to the source code of OSS projects instead of developing new main features as done by core developers

[JAHM17, CWLH06]. Since Khomh et al. [KADZ15] found that the contribution of bug fixes has immensely increased in relation to the contribution of new main features with the adaption of FIREFOX to the rapid release model (see Section 2.4), the activity of peripheral developers should, therefore, also have been increased significantly. However, it is disputable, if the development work of those peripheral developers is primarily executed after the contribution of new features by core developers, i.e., at the end of each release range and after the integration of the newly developed features, since bugs can only be fixed if a potential source of bugs exist, or if there is no discernible contribution pattern of core and peripheral developers. This line of arguments leads to our central research question:

> ***RQ2:*** *How do the contribution patterns differ among core and peripheral developers?*

Last, we aim at validating the consistent developer classifications by network-based and count-based operationalizations as it has been analyzed by Joblin et al. [JAHM17]. The study indicates that the classifications of developers as core and peripheral by applying count-based and network-based operationalizations is largely consistent for different open-source projects. Therefore, we examine these findings by analyzing the agreement of our applied count-based and network-based operationalizations on MOZILLA FIREFOX:

> ***RQ3:*** *To what extent do the applied count-based operationalizations agree with the network-based operationalizations?*

4.2 Data Collection

After we defined the research questions of this study, we now describe the details of extracting the necessary data from the FIREFOX VCS as well as processing this data in order to correctly investigate our research questions.

As described in Chapter 2, OSS developers can be classified as core and peripheral according to their technical contribution activity to the projects VCS repository and to their communication and collaboration patterns. To analyze these technical and social aspects, we focus our data collection on the commit data of the official FIREFOX VCS repositories. Since MOZILLA FIREFOX has four different release channels a new version has to go through before it gets released (as shown in Figure 3.2), we need to choose the most representative one to extract the commit data for our research.

Because of Aurora as the second release channel is the first one which is mainly responsible for feature stabilizing and improvement as described in Section 3.1, we assume that this channel contains almost all developer contributions from the first release channel Nightly which are partly not included in the version's corresponding Beta or Release channel due to stability issues, as well as additional bug fixes and feature enhancements firstly made in the Aurora channel. As more peripheral developers are working with Aurora than Nightly because of their responsibility for feature stabilizing and small enhancements as described in Chapter 2, the release channel

Aurora should be more representative for the rapid release model of Firefox as well as for our specified analyses than the other release channels. Following from these assumptions, we extract the necessary commit data from the VCS repository of the release channel Aurora as the base for our analyses.

4.2.1 Tooling and Data Extraction

For this extraction from the VCS, we use a framework for analyzing technical and social aspects of software development called Codeface, firstly used in [JMA+15] [Sie]. Even if the best way to retrieve the commit data from the Firefox Aurora channel would be the extraction from the main Mozilla Firefox VCS Mercurial, we are limited to extract the data from the mirror of the Mercurial repositories on Git, as Codeface is currently restricted to the version-control systems SVN and Git[3]. As the main problem of this Git mirror is the missing tag and version assignment of the commits, we need to do a manual mapping of the managed version tags on Mercurial to the mirrored commits on Git to be able to correctly assign the extracted commits to their corresponding Firefox version on the Aurora channel. This has been done by retrieving the last commit for each tag, which represents the start of a new version development range on the Aurora channel repository on Mercurial, e.g., the tag "FIREFOX_AURORA_46_BASE" for version 46 (see Figure 3.3), and finding the corresponding commit on the Git mirror by comparing the dates, messages, and authors. To always map the commits to the Firefox versions in which the changes have been developed, we use the *author dates* of the commits on Git representing the date of the first push to the VCS instead of the *commit date* which would change on every push to the next release channel and, therefore, would lead to an invalid mapping. The full mapping is shown on Table A.1.

Furthermore, we decided to focus on the commit data of the Firefox versions 37–51 on the Aurora channel for our investigations, as Firefox version 51 has been the newest release at the state of the research for this thesis. These 15 versions are robust against diverging development ranges, as each possible length of a Firefox version development range between five and eight weeks is covered by these versions, whereby we assume that this should be enough to get a sufficient insight of the development activity of the Firefox developers as needed for our analyses.

Through the extraction via the framework Codeface, we, therefore, retrieve the whole commit data between each of those 15 Firefox versions as text files, whose content we can further process using scripts of the programming language R[4].

4.2.2 Data Processing

After we extracted the commit data of the Aurora channel for our 15 investigating Firefox versions, we need to apply appropriate operationalizations on the processed commit data to classify the Firefox developers as core and peripheral, before we can conduct our analyses about the activity of these developer classes. Therefore, we consider count-based as well as network-based operationalizations for our Firefox

[3]https://github.com/siemens/codeface
[4]https://www.r-project.org/

case study, analogously to the case study of ten different open source projects by Joblin et al. [JAHM17].

As the metrics for the count-based approach, we consider two different developer activity measures of technical contributions to the VCS, like they have also been used by Joblin et al., as described in Chapter 2. The first one is the number of commits a developer has authored within a specific time period, which is defined as the *commit count*. The second one is the sum of added and deleted lines of code a developer has authored within a specific time period, which we define as the *lines-of-code (LOC) count* [JAHM17].

For our analyses, we choose an appropriate time interval, e.g., one development range of a Firefox version in the Aurora channel, and calculate either the total number of commits or the total number of LOC for each developer within this time interval. In dependence of the calculated contribution count, we then classify developers as core, whose count is above a defined threshold. Since Mockus et al. [MFH02] already found within a previous case study about Mozilla Firefox that only a few developers are responsible for approximately 80% of the changes on the source code as well as the established application of this 80% threshold on further previous case studies (see Chapter 2), we also consider this threshold for our case study on Mozilla Firefox. Hence, we classify core developers as those which are together responsible for 80% of the development work based on the chosen metric, i.e., either the commits or the LOC count.

The network-based approach for the classification of developers, on the other hand, is a bit more complex. As the first step, we build a co-change-based developer network using function-level information of the VCS to retrieve information about developers contributing technically-related changes. The result is a network in which the developers are represented as nodes connected through edges representing technical relationship by a modification of the same function within the source code.

On the basis of this co-change-based developer network, we select two network-based metrics for the classification of developers as core and peripheral, as they are also described by Joblin et al. [JAHM17]. The first one is defined as the *degree centrality* representing the number of edges a developer has to other developers within this network. The second metric extends the first one by additionally considering the edges to developers that are themselves in globally central positions next to the general number of edges to other developers to define the expected importance of a developer, which we define as the *eigenvector centrality*.

Therefore, each developer has a specific centrality value due to his position and edges within the build network on both metrics. To classify the developers as core with the most edges to other developers (degree centrality) or with the highest importance within the network (eigenvector centrality), we take the same threshold of 80% as on the count-based operationalizations and choose the corresponding developers whose centrality values together sum up to these 80% of the total centrality value of all developers. All other developers are accordingly seen as peripheral developers due to too few edges to other developers or due to their low importance within the network.

4.2.3 Data Validation

Before we are able to make assumptions about the developer activity in the Firefox release model, we need to do a final data validation, whether the Firefox release model and its feature development process, as described in Section 3.1 based on the information of official Firefox Web pages [Moz16, Mozj], applies to the real Firefox project. This is necessary to make sure that we are able to extract the correct VCS data and realistically execute our analyses on the Aurora channel. Therefore, we need to check if the main implementation for a Firefox version indeed occurs, when this version is maintained in the Nightly channel as described in Section 3.1, so we can assume that these implemented features will also exist in the following channel Aurora, on which we are extracting the data for our analyses.

To carry out the check, the commits of six features and bug fixes of the Firefox versions 51, 50, and 49 were tracked on the official Mercurial repositories of the four Firefox channels Nightly, Aurora, Beta, and Release. The results of the check are listed in Table A.2. The investigation has shown that every tracked commit, which has been released with a specific version, has been pushed to the Nightly channel when its version was currently maintained in the Nightly or in the Aurora channel and has always been pushed to the next channel when its version has also been pushed to the next channel. This excludes the push to the Release channel, on which the releasable features get pushed a few days before the corresponding Firefox version, due to necessary technical work according to a statement on the official Firefox blog [Leg11].

Due to this validation, we can verify the previously described development process of Firefox and can assume that we will have all main development commit data, when operating our analyses on the Aurora channel. Therefore, our further analysis of the developer activity will refer to the correct and realistic development data and time intervals of the Mozilla Firefox project.

4.3 Hypotheses

To address our described research questions, we investigate on five hypotheses. The first three address *RQ1* by analyzing the Firefox developer classification itself, as well as the resulting stability of the core developers. The fourth hypothesis aims at validating a potential increase of the peripheral developer activity right before a release of Firefox as a characteristic of the contribution pattern of peripheral developers in the context of *RQ2*. Finally, the consistency of the applied count-based and network-based operationalizations as questioned by *RQ3* shall be analyzed with the last hypothesis.

As described in Chapter 2, previous studies already investigated different OSS projects and have found that only a few developers are responsible for the majority of the development work. Mockus et al. [MFH02] as well as Terceiro et al. [TRC10] found within individual case studies of different OSS that only a few developers, who were considered as core, contributed more than 80% of the development work. So, core developers are seen as the minority in comparison to peripheral developers within OSS development teams, whereby we assume that the most of the developers will be classified as peripheral in each version development range of Firefox.

To investigate this hypothesis, we classify the Firefox developers as core and peripheral on each development range of our investigating Firefox versions 37 to 51. As described in Section 4.2.2, we can use the count-based operationalizations commit count and LOC count, as well as the network-based operationalizations degree centrality and eigenvector centrality for our classification of the developers. To get comprehensive results in respect to our hypothesis, we conduct the analysis based on each classification produced by these four operationalizations using the commit data of one complete version development range independently. For each classification, we then count and compare the number of classified active developers in each class on each individual version development range. In respect to the hypothesis, we then expect the number of classified peripheral developers to be higher in each selected version development range than the number of classified core developers.

Since core developers typically work on main features whose development often takes more than one version range and are, therefore, classified as core in only the version range in which the features get integrated, we further investigate an additional approach for the comparison of the classified developers in each class which incorporates those characteristics of core developers. In this approach, when analyzing the number of developers in each class on a version development range, we additionally consider the developers as core which were in a core state in, at least, one of the previous three version development ranges. Accordingly, those developers are not additionally considered as peripheral in the corresponding version range. We refer this approach as *sliding window* when analyzing the results. Nevertheless, we still expect the number of classified peripheral developers higher in each selected version development range than the number of classified core developers on this approach:

> ***H1:*** *The most of the* Firefox *developers will be classified as peripheral in each version development range.*

Typically, core developers distinguish themselves by having a long-term involvement in an OSS project which leads to a recognizable higher role stability than for peripheral developers, which only have an inconsistent, short-term involvement [JAHM17, CWLH06]. Therefore, we assume that this is also the case for the Mozilla Firefox project, whereby the majority of the developers who are classified as core on one development range of a Firefox version should also be classified as core on the following version development range, what would represent the expected high role stability.

To address this hypothesis, we firstly investigate the turnover values of the core developers of each version development range for each classification produced by our four operationalizations. The turnover values in this context represent the proportion of active developers in a version development range, which were not active in the previous three version development ranges. In respect to the hypothesis, we regard a developer as active in a version development range if he/she has contributed, at least, one commit to the VCS within the considered version development range. The lower the resulting turnover values of the core developers are, the higher we consider the role stability of those developers, as it indicates that the most active core developers in one version development range have been active in the previous ones as well.

As the turnover values represent the role stability of the core developers regarding their activity in multiple previous version development ranges, we further analyze the probability of developers to be classified as core in a version development range, if they have been classified as core in, at least, the latest version development range, which we define as "stable" developers. Therefore, we classify all FIREFOX developers as core and peripheral using again each of our four classification metrics for each version development range. We then calculate the sum of the developers which are classified as core in one version development range and can also be classified as core in the following version development range. Thereupon, we calculate the proportion of these "stable" developers on the sum of all core developers in each version development range, which we assume to include the most of those classified core developers in order to accept our hypothesis:

> ***H2:*** *The majority of* FIREFOX *developers classified as core in one version development range will also be classified as core in the following version development range.*

While hypothesis *H2* only investigates a simplified definition of role stability based on consecutive development ranges, we additionally analyze the role stability based on all selected FIREFOX versions. This avoids to neglect potential core developers who are only highly active in, e.g., every second version development range or who are not active at all in multiple development ranges due to vacations or other circumstances. Additionally, with this approach, we also consider developers who are working on features whose development takes more than one version development range and are, therefore, classified as core in only the version range in which the features get integrated, as on our sliding window approach of the analysis regarding hypothesis *H1*.

Therefore, we count the number of version development ranges, in which a developer can be classified as core according to each of our four operationalizations, regardless if those ranges are contiguous or not. As our analysis of the MOZILLA FIREFOX project is based on the data of 15 consecutive versions, we assume typical core developers to be active in, at least, eight of these 15 investigated FIREFOX version development ranges to validate the characteristic high role stability of core developers without necessarily being active as core on two successive version development ranges:

> ***H3:*** *A* FIREFOX *developer classified as core in one version development range will also be classified as core in the most of all other selected version development ranges.*

Due to the rapid release cycle of FIREFOX, which leads to a short development time in each release channel of six to eight weeks, we assume that most of the time of a release cycle will be used for the development and the contribution of new main features which is mainly the responsibility of the FIREFOX core developers according to our definition in Chapter 2. Therefore, approximately the last third to the last quarter of the range, i.e., the last two weeks, should be embossed by an increased contribution activity of bug fixes and refinement works of those newly added features to make them stable enough for the push to the next release channel or even for the release. Since peripheral developers distinguish themselves by primarily contributing such bug fixes and small enhancements (see Chapter 2), we expect their contribution activity to significantly increase in those last two weeks of a version development range.

For the investigation of this hypothesis, we analyze the developer activity of peripheral developers on each development range of our selected FIREFOX versions 37 to 51. To get comprehensive results in respect to our hypothesis, we again conduct the analysis based on each of our four operationalizations independently using the complete commit data of each selected version development range, as also specified on the previous hypotheses.

For the measurement of the developer activity, we can rely our analysis on either the *commit count*, i.e., the number of commits conducted by the developers of each class, or the *LOC count*, i.e., the sum of added and deleted lines of code by the developers of each class, since both measurements are a direct contribution to the source code of a project and thereby represent some kind of developer activity [JAHM17]. But since a rename of a code file in a VCS is considered as a full deletion of all the contained lines of code within the code file and a subsequently addition of all the deleted lines, the LOC count as an activity measure would lead to distorted results. Therefore, we focus our analyses of the developer activity on the number of commits per developer class. Hence, we calculate the sum of the commit count for all developers within a developer class in each individual week of a version development range and then specifically analyze the ratio of the commit count for the peripheral developers on the last two weeks in relation to their commit count on the previous development weeks. To also neglect developers with a very high or very low commit count, i.e., developers with extreme values, we further analyze the median number of commits for each developer class in respect to our hypothesis. As we further want to incorporate the developers who work on main features whose development often takes more than one version range, we additionally analyze the sliding window approach, as we do on hypothesis *H1*, for the absolute numbers of commits in each developer class as well as for the median number of commits, to get comprehensive results for our hypothesis:

> ***H4:*** *The activity of peripheral developers is significantly higher in the last two weeks of a version development range than in the first weeks.*

Because of the investigation of previous case studies by Joblin et al. [JAHM17], which established evidence of a consistent developer classification as core and peripheral by network-based and count-based operationalizations, we assume that this consistency will also be met by the FIREFOX project.

To analyze the agreement of our operationalizations, we also use the statistic of *Cohen's kappa* as used by Joblin et al. [JAHM17], which incorporates the agreement of the metrics that occurs by chance [LK77]. Thus, we are able to compare the resulting agreement values of this FIREFOX case study with the previously calculated agreement values of the ten different open source projects by Joblin et al. [JAHM17]. The Cohen's kappa in this context is defined as follows:

$$\kappa = \frac{p_0 - p_e}{1 - p_e}, \tag{4.1}$$

where p_0 is defined as the relative agreement of the classifications, i.e., the number of developers classified in the same class by both operationalizations in relation to the sum of all classified developers. p_e, on the other hand, describes the probability of random agreement on the developer roles [JAHM17, LK77].

To calculate the agreement values for all of our four classification metrics, we consider their classifications pairwise on all of the 15 Firefox version development ranges. The ranges for the strength of agreement calculated by Cohen's kappa according to Joblin et al. are the following: "0.81–1.00 almost perfect, 0.61–0.80 substantial, 0.41–0.6 moderate, 0.21–0.40 fair, 0.00–0.20 slight, and < 0.00 poor" [JAHM17].

We expect the strength of agreement of the classifications produced by our chosen operationalizations to be, at least, "slight" in order to accept our last hypothesis:

> ***H5:*** *The classification of developers into core and peripheral obtained from count-based operationalizations will be consistent with the classification obtained from network-based operationalizations.*

5 Results and Discussion

After the description of our study design including the specification of the investigating research questions and hypotheses, we now present and discuss the results of our five hypotheses in relation to their corresponding research questions.

5.1 *RQ1*: Core-Peripheral Classification

> *How many* Firefox *developers can be classified as core or peripheral for each considered development range and how stable is the classification of developers into core and peripheral over time?*

First, we present the results of the hypotheses corresponding to *RQ1*. Those are related to the analysis of the classification of the Firefox developers as core and peripheral on our selected 15 Firefox version development ranges as well as to the derivatively corresponding role stability of core and peripheral developers.

5.1.1 Results for *H1*

> *The most of the* Firefox *developers will be classified as peripheral in each version development range.*

To investigate hypothesis *H1*, we analyze the number of core and peripheral developers on each development range of our selected Firefox versions 37 to 51. As described in Section 4.3, we apply the two count-based operationalizations *commit count* and *LOC count* as well as the two network-based operationalizations *degree centrality* and *eigenvector centrality* individually on the commit data for each version development range to receive the developer classifications as the base for our analysis.

In Figure 5.1, we show the plain numbers of classified core and peripheral developers for each of our 15 Firefox versions based on a classification by commit count. Further, we visualize the resulted numbers of developers in each class, when considering the core developers of the previous three version ranges in each analyzing version development range, i.e., the sliding window approach as defined in Section 4.3. We can observe right at the first glance that more developers are classified as peripheral in each version development range than as core for our plain numbers of classified developers as well as for our sliding window approach.

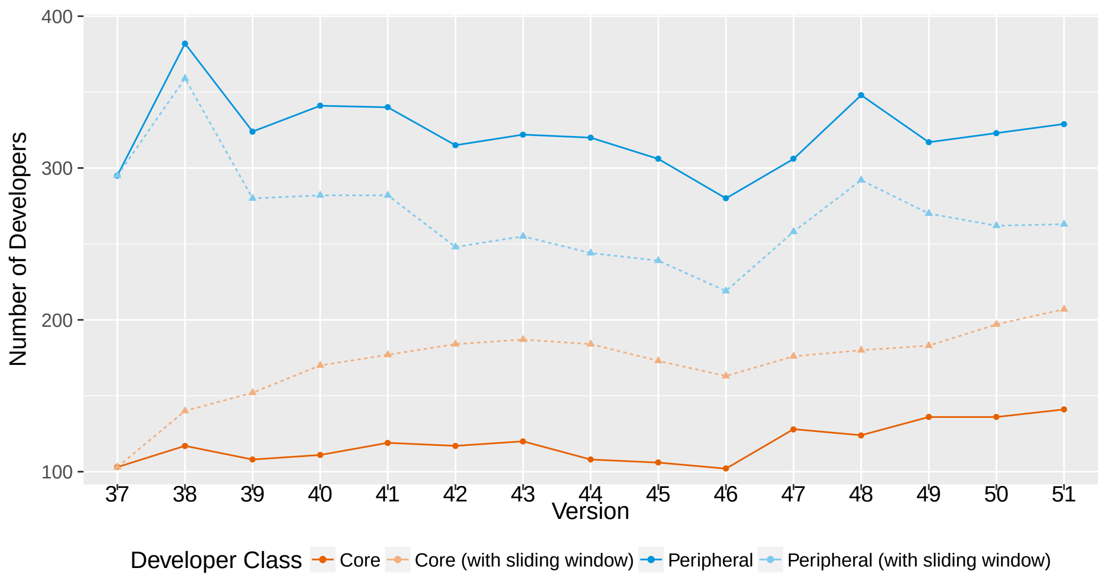


Figure 5.1: Numbers of classified developers in each selected Firefox version based on a classification by commit count

To statistically evaluate this observation in respect to hypothesis *H1*, we additionally conduct a statistical test on the result for both approaches. The first choice is to statistically evaluate the results using a *t-test*, since this parametric test seems to have a higher power than its non-parametric alternatives, at least, on small sample sizes as we have on our analysis [MOD07]. As we compare the numbers of developers in two different developer groups on the same time interval, i.e., one version development range, for each selected Firefox version, we specifically use an *unpaired t-test* for the statistical analysis of this hypothesis. But, since a t-test requires our data to be normally distributed, we firstly need to test the data of our resulted classifications regarding this property using a *Shapiro–Wilk test*. We execute this test against the null hypothesis that the data is normally distributed using an acceptable level of an error that this null hypothesis is rejected even though it is true, designated as the significance level (α) with $\alpha = 0.05$. If our resulting probability value (p–value) is greater than α, we can not reject the null hypothesis, whereby we assume that our data is normally distributed and thus, we can use the unpaired t-test to statistically analyze our data. On the other hand, if the p–value is less than our specified α, we reject our null hypothesis, i.e., the data is not normally distributed, whereby we cannot apply the unpaired t-test [Inc]. In this case, we use an alternative test which does not require the data to be normally distributed, namely referred as *Mann–Whitney U-test* [CF14].

Independently from the necessary choice of the test method, we execute the appropriate statistical test by testing against the null hypothesis H_0 "The number of peripheral developers is not higher than the number of core developers in each version development range" and its corresponding alternative hypothesis H_1 "The number of peripheral developers is higher than the number of core developers in each version development range". To assess the results of our test, we specify the significance level as $\alpha = 0.05$. If our resulting p–value is less than α, we can reject the null hypothesis and, therefore, confirm our hypothesis H_1.

Therefore, we firstly conduct the necessary Shapiro–Wilk test for the plain numbers of developers to check if the numbers of core and peripheral developers for each version development range resulting from a classification by commit count are normally distributed. This test results in a p–value of 0.60 for the 15 plain numbers of peripheral developers and a p–value of 0.33 for the 15 plain numbers of core developers for each of our selected Firefox versions. Thereby, we assume that both data sets are normally distributed, since those values are higher than the specified significance level α of 0.05. Hence, we are able to conduct an unpaired t-test on our data sets in respect to our hypothesis which leads to a p–value of way lower than our specified significance level α for the t-test of 0.05. Therefore, we can reject the null hypothesis of the hypothesis *H1* that the number of peripheral developers is not higher than the number of core developers in each version development range. Following from this, we can accept our hypothesis *H1* for a developer classification by commit count, at least, for the plain numbers of classified developers.

Further, we conduct the statistical tests for our sliding window approach. Here, the Shapiro–Wilk test results in a p–value of 0.1 for the numbers of peripheral developers and 0.0396 for the numbers of core developers. Since the latter p–value is lower than our specified α of 0.05 for the Shapiro–Wilk test, the corresponding null hypothesis that the data sets are normally distributed is rejected. Therefore, we conduct a Mann-Whitney U-test on the data sets leading to a p–value of $1,687e^{-6}$ which is also way lower than our specified significance level α for the test. As on the plain numbers of developers, we can, therefore, accept our hypothesis *H1* for a developer classification by commit count on our sliding window approach.

Similar to a classification by commit count, the numbers of classified core developers resulting from a classification by LOC count, as shown in Figure 5.2, are higher than the numbers of peripheral developers at the first glance. But, in comparison to a classification by commit count, we do not have such a large difference between the plain number of core developers in a version development range to the number of classified core developers using the sliding window approach. However, pretty conspicuous is that less of the half of the developers in a core state on a classification by commit count can be classified as core on a classification by LOC count on every selected version development range. On Firefox version 42, even only two developers could be classified as core.

The execution of the Shapiro–Wilk test on the plain numbers of developers to analyze our data sets regarding normal distribution leads to p–values of 0.67 for the numbers of peripheral developers and 0.33 for the numbers of core developers, whereby we can again state that these data sets should be normally distributed. The p–value of almost zero resulting from our corresponding t-test, which is lower than our specified significance level α of 0.05, verifies our assumption that most of the active developers are classified as peripheral. Therefore, we can confirm our hypothesis *H1* for the plain numbers of developers based on a classification by LOC count.

The Shapiro–Wilk test on the results for the sliding window approach leads to p–values of 0.09 for the numbers of peripheral developers and 0.82 for the numbers

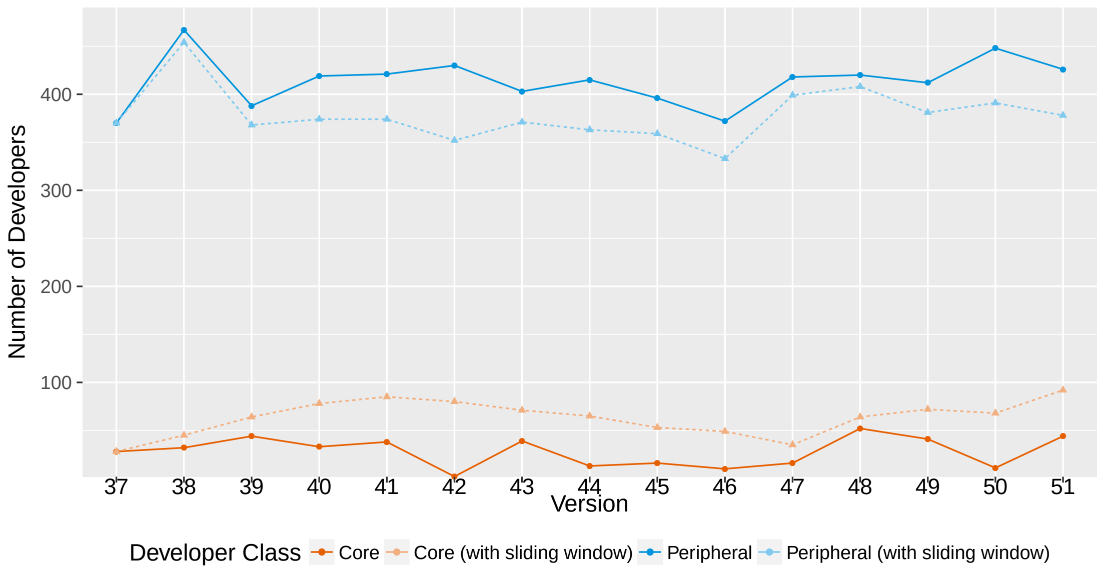


Figure 5.2: Numbers of classified developers in each selected FIREFOX version based on a classification by LOC count

of core developers, whereby we can statistically analyze our findings using a t-test. Because of the low numbers of core developers and the high numbers of peripheral developers, as observed on Figure 5.2, this test also leads to a p–value of almost zero. Hence, we can also accept the hypothesis *H1* for the sliding window approach on a classification by LOC count.

After the investigation of the results for our count-based operationalizations, we now present the numbers of classified developers based on our chosen network-based operationalizations. Therefore, in Figure 5.3, we present the numbers of classified core and peripheral developers for our both approaches based on a classification by degree centrality. As on the classifications by our selected count-based operationalizations, the result shows that clearly more developers are classified as peripheral than as core on each selected version development range. With in general between 50 and 100 core developers on each version development range, the numbers of developers in a core state on each range are right between the numbers of core developers on the classifications by commit count and the classifications by LOC count.

To again statistically analyze our observations on a classification by degree centrality according to our hypothesis *H1*, we firstly execute our statistical tests on the plain numbers of developers. Here, the data sets are with p–values of 0.94 for the numbers of peripheral and 0.89 for the numbers of core developers resulted from the execution of the Shapiro–Wilk test clearly above our specified significance level and should, therefore, be both normally distributed. As already expected, we receive a p–value with $3,557e^{-24}$ way below the specified significance level of our executed t-test, whereby we can accept our hypothesis *H1* for the plain numbers of developers based on a classification by degree centrality.

Since the execution of the Shapiro–Wilk test leads to p–values of 0.46 for the numbers of peripheral and 0.026 for the numbers of core developers resulted from our sliding window approach and are, therefore, not both normally distributed, we have to

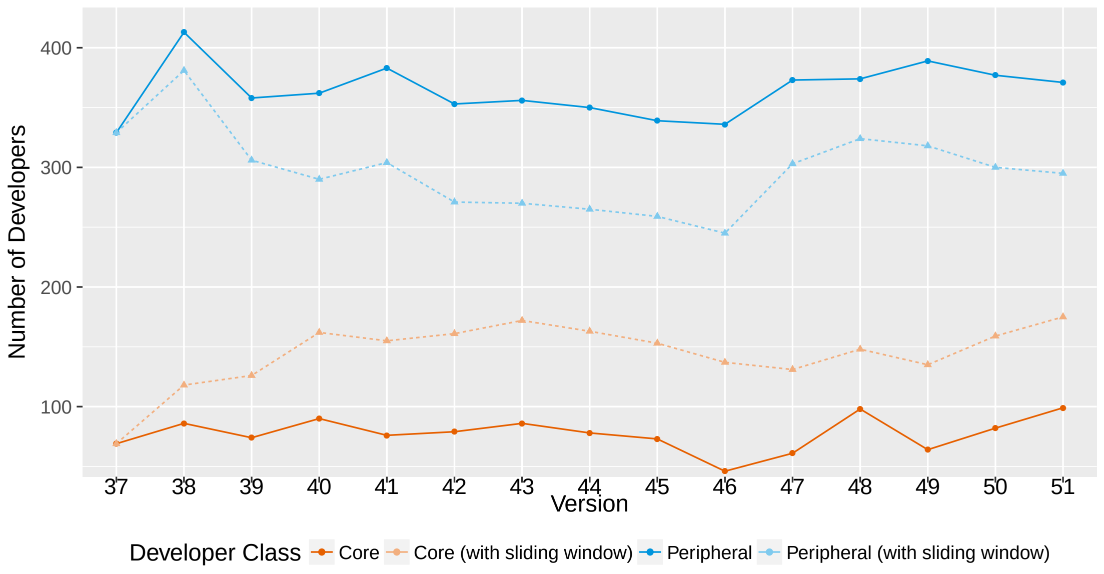


Figure 5.3: Numbers of classified developers in each selected Firefox version based on a classification by degree centrality

conduct a Mann-Whitney U-test on the data sets of this approach. However, with a p–value of $6.447e^{-9}$ resulting from that test, we can also accept our hypothesis *H1* for the sliding window approach on a classification by degree centrality.

Finally, in Figure 5.4, we visualize the result for a developer classification by eigenvector centrality for our two approaches. As on the classifications produced by our other three operationalizations, we can see pretty clearly that the numbers of classified peripheral developers are always higher in each version development range than the numbers of core developers. Further, we can also notice that the sliding window approach results in only slightly more classified core developers, and, therefore, also in only slightly less peripheral developers, until a peak of classified plain core developers in Firefox version 43. After this version, the difference between the classified developers of both approaches increases, at least, until version 46.

The execution of the corresponding Shapiro–Wilk test for the retrieved plain numbers of developers based on a classification by eigenvector centrality results in p–values of 0.87 for the peripheral and $6.789e^{-5}$ for the core developer data sets. Since the data for the numbers of core developers is below our specified significance level, we reject that this data is normally distributed. Therefore, we execute a Mann-Whitney U-test resulting in a p–value of $1.549e^{-6}$ which is lower than the corresponding significance level. Hence, we accept our hypothesis *H1* for the plain numbers of classified developers based on a classification by eigenvector centrality.

The test of normal distribution on the corresponding data of our sliding window approach results in p–values of 0.68 for the numbers of peripheral and 0.0061 for the numbers of core developers, whereby we also conduct a Mann-Whitney U-test on this data, based on our specification in Section 4.3. This test results in a p–value of $1.687e^{-6}$ which also leads to the accept of hypothesis *H1* for the sliding window approach of a classification by eigenvector centrality.

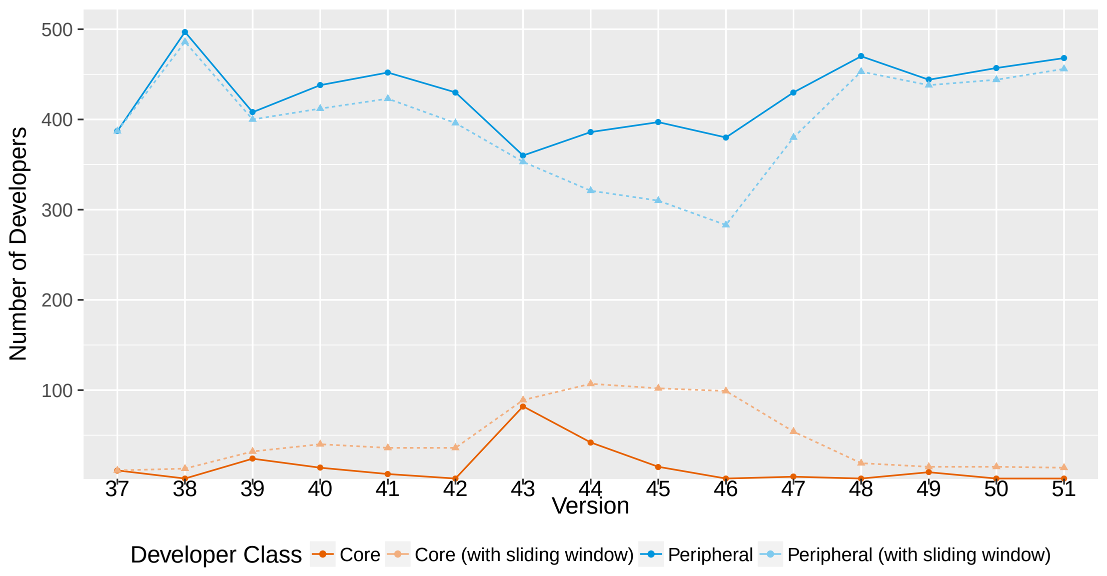


Figure 5.4: Numbers of classified developers in each selected FIREFOX version based on a classification by eigenvector centrality

***H1* (Accepted):** Each classification produced by our selected operationalizations resulted in higher numbers of peripheral developers than of core developers in each version development range for our analysis of the plain numbers of developers as well as for the numbers based on our sliding window approach. Although each classification led to different high numbers of classified core and peripheral developers, we can accept our hypothesis *H1* for each of our selected count-based and network-based operationalizations due to the conducted statistical tests.

5.1.2 Results for *H2*

The majority of FIREFOX *developers classified as core in one version development range will also be classified as core in the following version development range.*

Next, we present the results for our hypothesis *H2*. Therefore, as described in Section 4.3, we firstly investigate the turnover values of the core developers for each version development range to analyze the role stability of those developers regarding their activity in the previous three version development ranges of each version development range. For that purpose, we again conduct an individual analysis for each developer classification based on our four selected operationalizations. As the turnover values on our first three selected FIREFOX versions 37 to 39 do not present a meaningful result in respect to our analysis, since there does not exist the full developer activity data of the corresponding three previous version development ranges, we neglect those three versions in our investigations.

The results for a classification by commit count are shown in Figure 5.5. We can observe that only a pretty stable percentage between 10% and 25% of the core developers in each version development range are not active in the corresponding previous three version development ranges, i.e., they either became firstly active as core developers in the selected version development ranges or they have only been active as core in one or multiple version development ranges before those critical three previous ranges. The latter would tell that they have a very low role stability as core developers, since they were either not active at all or only slightly active as peripheral developers within those critical three previous version development ranges. Because of those findings, we can conclude that, at least, 75% of the core developers in each version development range have a pretty high role stability, since they were active in at least one of those critical three previous version development ranges. The peripheral developers, on the other hand, do have a lower role stability, since, at least, 25% of those developers on each version development range were not active as peripheral in the previous three ranges. Thus, those findings confirm the characteristics of both developer classes in terms of the role stability, as described in Chapter 2.

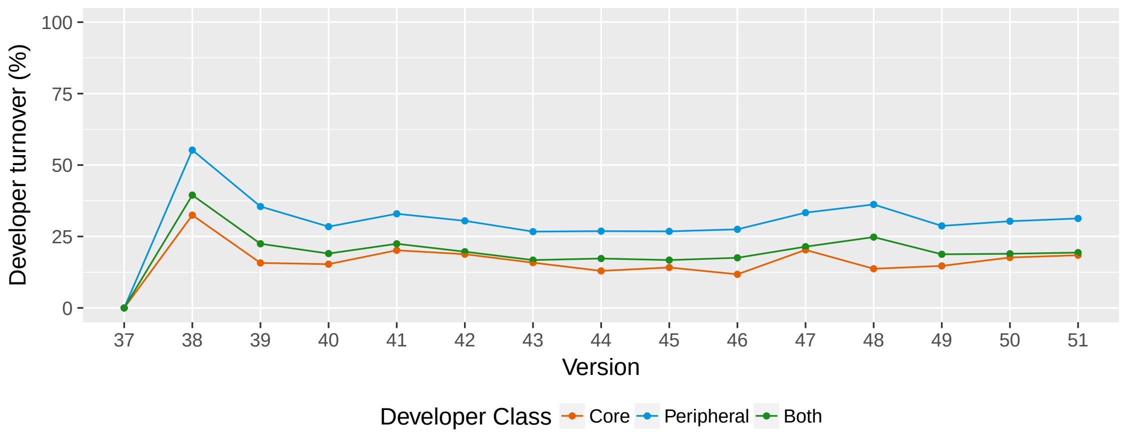


Figure 5.5: Proportion of active developers in each version development range which were not active in the previous three version development ranges (developer turnover values) based on a classification by commit count

Additionally, in Figure 5.5, we visualize the proportion of active developers in each version development range which were neither active as core nor as peripheral developer in the previous three version development ranges. These turnover values are slightly above those of only core developers and below the turnover values of only peripheral developers. Hence, we can conclude that there exist more developers in each version development range which were not active at all in the previous three ranges than developers which were only not active as core in the previous three ranges, but may have been active in a peripheral state in those version development ranges.

In Figure 5.6, we present the corresponding turnover values for core and peripheral developers as well as for developers regardless of the class based on a classification by LOC count. In contrast to a classification by commit count, we receive way higher turnover values for core developers, i.e., there is a way lower role stability of those developers. Further, the instability of the turnover values for core developers in contrast to Figure 5.5 is pretty conspicuous. On Firefox version 42, even all developers which were classified as core on that version were not active as core in the previous three version development ranges. This might be due to the low numbers of developers classified as core on that specific version for a classification by LOC count. As described in Section 5.1.1, a maximum of two developers could be classified as core per week in version 42 using that operationalization, which were obviously both not active as core developers in versions 39 to 41. The peripheral developers, on the other hand, have a way lower and a more stable proportion of developers between 15% and 25% in each version development range which were not active in the previous three version development ranges and, therefore, have a higher and more stable role stability than core developers. This contradicts to our findings on a classification by commit count, as shown on Figure 5.5, and therefore also contradicts to the role stability characteristics of core and peripheral developers.

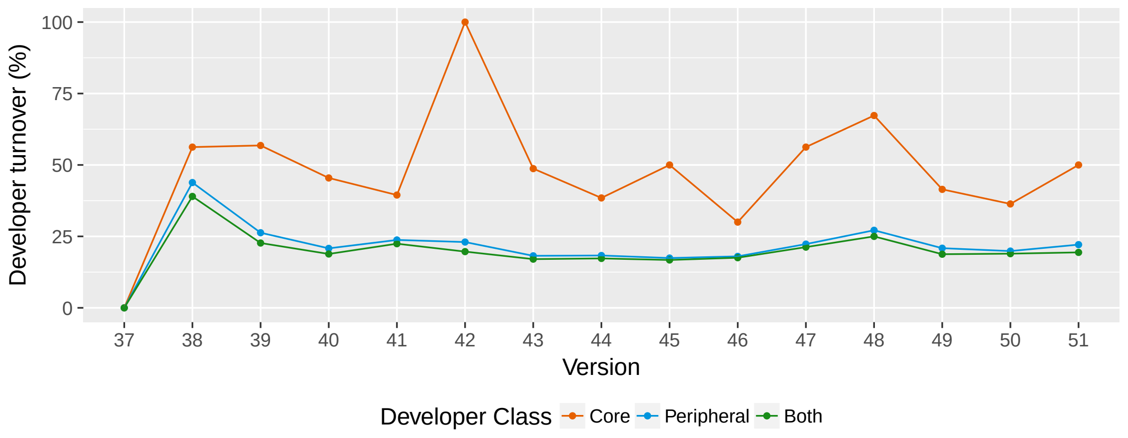


Figure 5.6: Proportion of active developers in each version development range which were not active in the previous three version development ranges (developer turnover values) based on a classification by LOC count

In comparison to the turnover values resulted on a classification by LOC count, we can observe way lower and much more stable turnover values for core developers on a classification by degree centrality (see Figure 5.7). Similar to the turnover values based on a classification by commit count (see Figure 5.5) the turnover values of core developers are lower than those of peripheral developers in the most selected version development ranges. Even though the difference of the turnover values of both classes is not as high as on a classification by commit count, we can still conclude that a classification by degree centrality underlines the characteristically higher role stability of core developers than the stability of peripheral developers, at least, in respect to the turnover values.

Finally, in Figure 5.8, we show the turnover values for the developers classified by eigenvector centrality. As on Figure 5.6, we can notice a very high instability of the turnover values of core developers over our selected Firefox versions right at the first glance. Again, those turnover values are in general way higher than those for peripheral developers on that classification and, therefore, also contradicts to our assumptions of a high role stability of core developers. Excluded from this is the turnover value of the core developers on version 41, which is slightly below the value for peripheral developers.

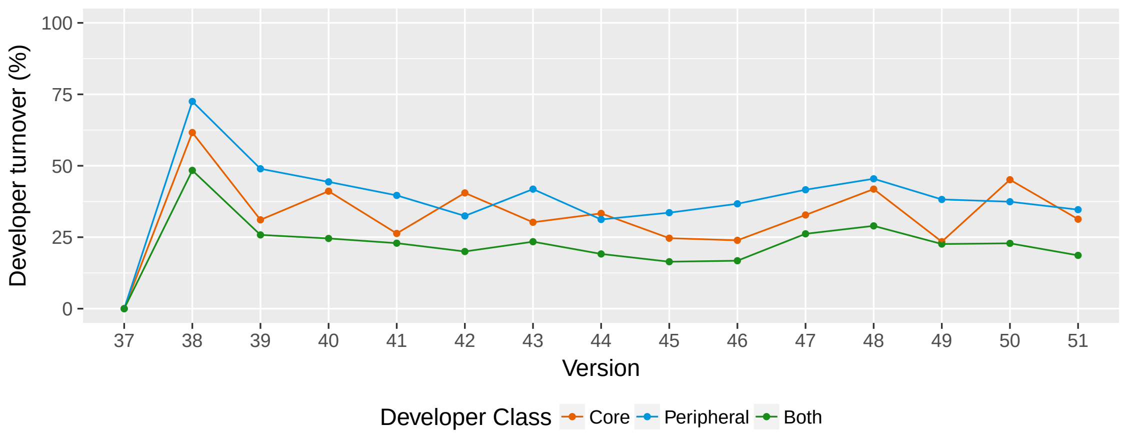


Figure 5.7: Proportion of active developers in each version development range which were not active in the previous three version development ranges (developer turnover values) based on a classification by degree centrality

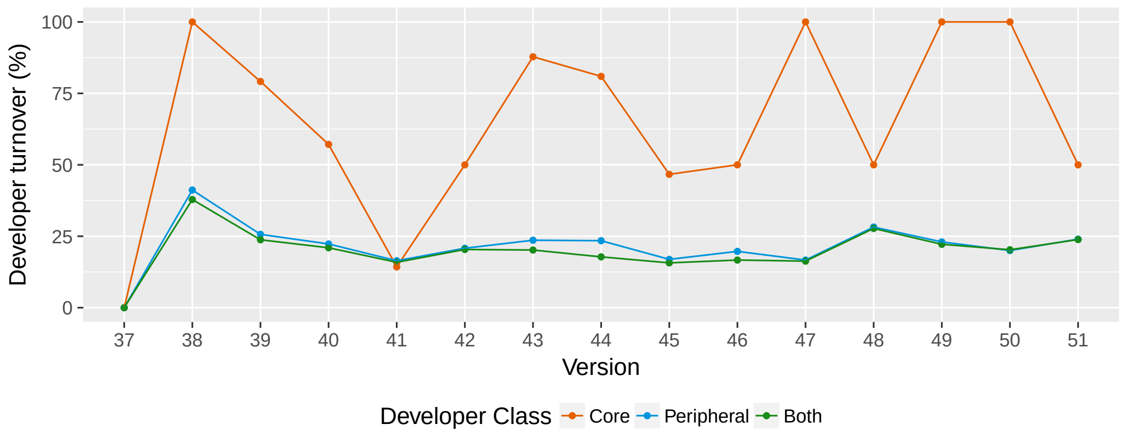


Figure 5.8: Proportion of active developers in each version development range which were not active in the previous three version development ranges (developer turnover values) based on a classification by eigenvector centrality

As described in Section 4.3, the turnover values which we just presented represent the role stability of the core developers regarding their activity in the previous three version development ranges and gave us a coarse look on how the stability of the core developers based on our four classifications looks like. Further, we now present the results of our analysis regarding the probability of developers to be classified as core in a version development range, if they have been classified as core in, at least, the latest version development range with respect to our hypothesis *H2*.

In Figure 5.9, we visualize the role stability for each developer class based on a classification by commit count. As defined in Section 4.3, those values represent the probability of developers to be classified as core, peripheral, or inactive (absent) in a version development range based on their classification of the previous range. We can notice that on average 38% of the developers classified as peripheral on one version development range transition to an absent state, i.e., are not active at all, and only 10% to a core state in the following version development range. Further, with 52% slightly the most of the developers classified as peripheral on one version development range are also classified as peripheral on the following range which only roughly underlines the characteristics of peripheral developers having an irregular, short-term involvement within a OSS project (see Chapter 2). In comparison to the peripheral developers, a clearly lower percentage of 2% of the developers classified as core on one version development range transition to an absent state and only 26% to the peripheral state on the following range. Therefore, with 72% clearly the biggest proportion of developers in a core state on one version development range maintain their state on the following range. So, as expected by the defined characteristics of our both classes, developers in a core state are much more stable within their role than peripheral developers and are likely to maintain their state as core developers, whereby we confirm our hypothesis *H2* for a classification by commit count.

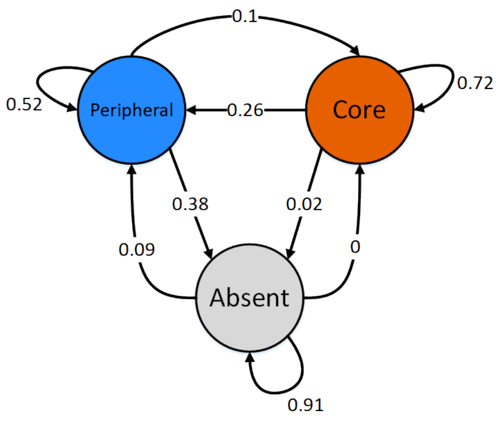


Figure 5.9: Role stability based on a classification by commit count

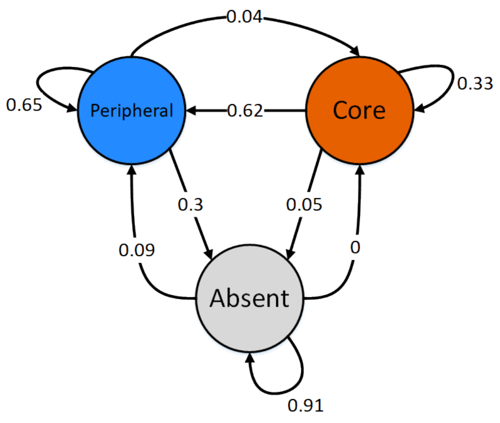


Figure 5.10: Role stability based on a classification by LOC count

As expected by the high turnover values of core developers for a classification by LOC count (see Figure 5.6), we receive a way lower probability of developers to be classified as core, if they have already been classified as core in the previous version development range (33%), i.e., a way lower role stability, for a classification by LOC count, as shown on Figure 5.10. Here, developers in a core state on one version development range are with a probability of 62% more likely to transition to a peripheral state on the next version development range. The most peripheral developers, on the other hand, maintain their peripheral state with a probability of 65%, instead of changing their developer role. Those findings contradict to our general definition of peripheral developers to be likely to transition between roles and core developers to be likely to maintain their role, whereby they also contradict to our assumption made on hypothesis *H2*. Therefore, we can not accept this hypothesis for a classification by LOC count.

The investigation of the role stability based on a classification by degree centrality, on the contrary, lead to a slightly better result, as shown on Figure 5.11. As on a classification by commit count, the role stability of core developer with 47% is higher than the stability of peripheral developers with 36%, as also expected per definition of those developer groups. But there is also a proportion of 22% of the core developer which are likely to be absent on the following version development range and a proportion of 31% to transition to a peripheral state. This does not indicate a high role stability of the core developers as expected, whereby we can not confirm our hypothesis *H2* for a classification by degree centrality.

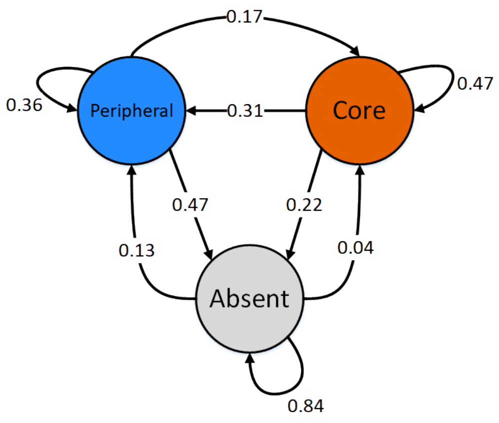


Figure 5.11: Role stability based on a classification by degree centrality

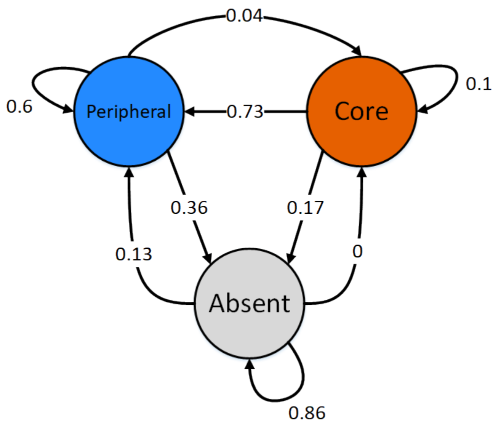


Figure 5.12: Role stability based on a classification by eigenvector centrality

The role stability of core developers resulting by a classification based on eigenvector centrality, as shown on Figure 5.12, with a probability of only 10%, is way lower than the role stability of core developers resulted by the previously described classifications. This was to be expected, since we also have very high turnover values of core developers due to the low numbers of core developers on a classification by that operationalization, as shown on Figure 5.4. Additionally, on average even 17% of the core developers on one selected version development range transition to an absent state and 73% to a peripheral state on the following version development range, which leads to a very low role stability in conflict to our hypothesis *H2*. Further, above the half of the developers classified as peripheral on one of our selected version development ranges are able to maintain there state, whereby 36% of the peripheral developers transition to an absent state on the next version development range. As a conclusion, the role stability of core developers based on a classification by eigenvector centrality is very much distinguished by a high transition of developers between roles, whereby we can not accept our hypothesis *H2* for this operationalization.

***H2* (Partially Accepted):** The investigation of the turnover values led to a pretty low role stability considering the previous three ranges for each version development range for a classification by LOC count and eigenvector centrality. Those findings also reflected themselves by analyzing the role stability of those classifications which led to a very low probability of developers to be classified as core on two consecutive version development ranges. We found slightly better results on a classification by degree centrality, whereby still the most core developers in each version development range are not able to maintain their state. Only the classifications by commit count led to the expected results of a high role stability and low turnover values for core developers, whereby our hypothesis *H2* can only be accepted using the commit count as the classification metric.

5.1.3 Results for *H3*

> *A* Firefox *developer classified as core in one version development range will also be classified as core in the most of all other selected version development ranges.*

We now present the results for our third hypothesis by analyzing the role stability based on all selected Firefox versions.

In Figure 5.13, we show the numbers of core developers with their corresponding number of occurrences as active core developers in each of our 15 selected version development ranges based on a classification by commit count. On this classification, a total of 314 developers are classified as core in, at least, one version development range, whereby even 20 of those developers are classified as core in all of our 15 version development ranges. Even though a large percentage of 26,4% are classified in only one version development range, there are also 95 developers, i.e., approximately 30%, who are classified as core in, at least, eight of the 15 version ranges. But, as the most developers (70%) can only be classified as core in less than eight version ranges, we can not accept our hypothesis *H3* for a classification by commit count.

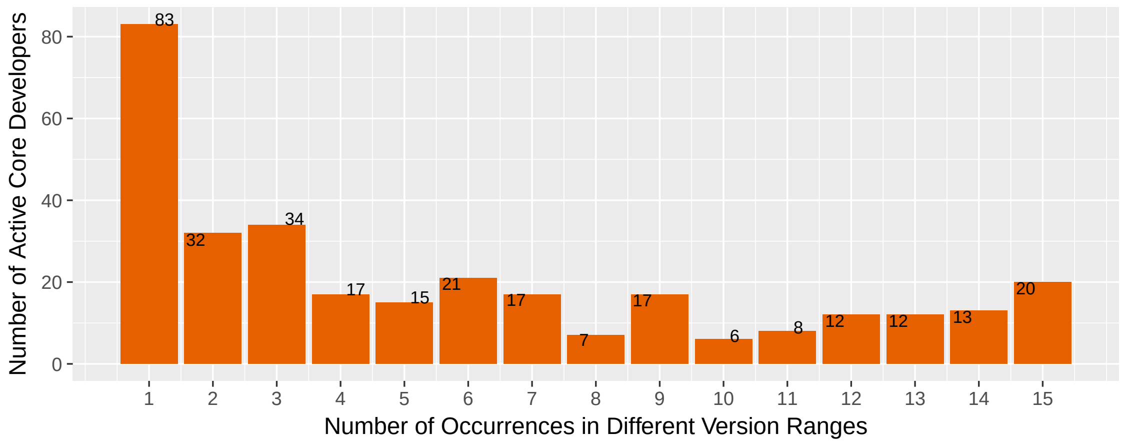


Figure 5.13: Numbers of recurring core developers based on a classification by commit count

As on a classification by commit count, the same number of developers (83) can be classified as core in only one version development based on a classification by LOC count, as shown on Figure 5.14. But as only a total of 177 different developers can be classified as core in any of our 15 selected version development range, and, therefore, just slightly above the half of the developers on a classification by commit count, those 83 developers classified as core on only one of our selected Firefox versions make up almost the half (47%) of our total classified core developers in this classification. By considering the developers who are further classified in a maximum of seven version ranges in respect to our hypothesis, we receive a proportion of 96% of all our core developers. Therefore, only 4%, i.e., 7 developers, can be classified as core in the most of our selected version development ranges, whereby we can also not accept our hypothesis *H3* for a classification by LOC count.

In Figure 5.15, we present the numbers of recurring core developers with their number of occurrences in a core state based on a classification by degree centrality. Even

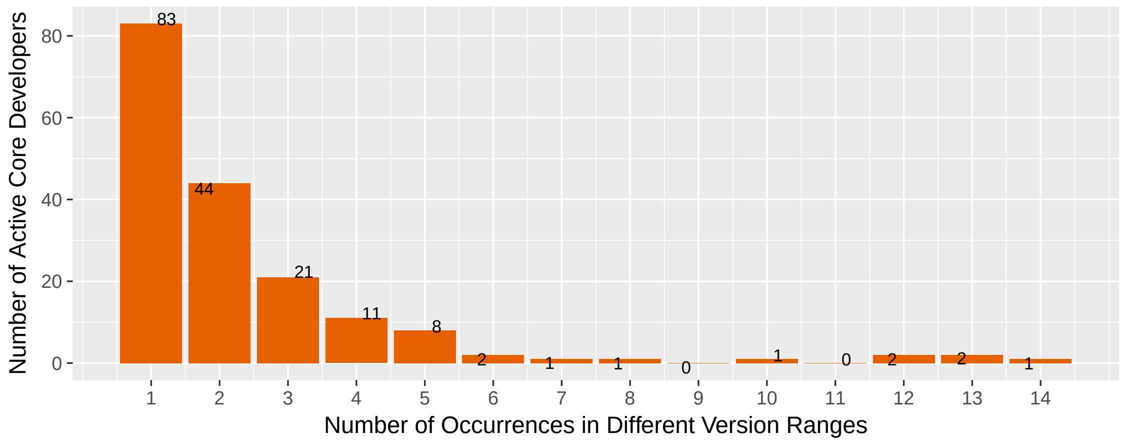


Figure 5.14: Numbers of recurring core developers based on a classification by LOC count

though we also receive a very high proportion of developers being active as core in only one of our selected version development ranges (150 of 357), we further receive 40 developers who can be classified as core in the most of those version development ranges. But, in respect to our hypothesis, those 40 developers make up only 11,2% of all 357 developers classified as core on a classification by degree centrality, whereby we can further also not confirm our hypothesis *H3* for a classification based on this operationalization.

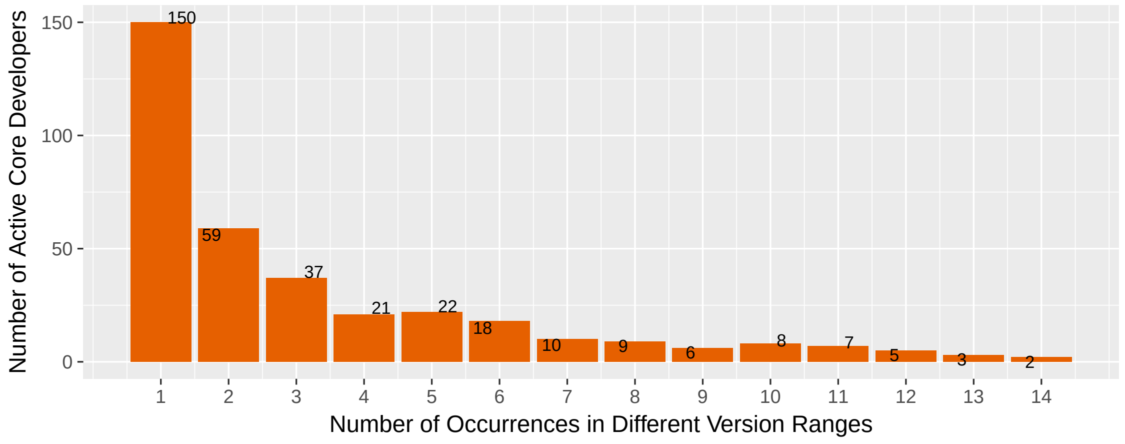


Figure 5.15: Numbers of recurring core developers based on a classification by degree centrality

By analyzing the numbers of core developers with their number of occurrences in the investigating version development ranges based on a classification by eigenvector centrality, shown on Figure 5.16, we can observe that no developer can be classified as core in, at least, eight of our selected version development ranges. Even the most of the 145 developers classified as core in all of our 15 FIREFOX versions can only be classified as core in one single version development range. Therefore, we can not accept our hypothesis *H3* for this operationalization either.

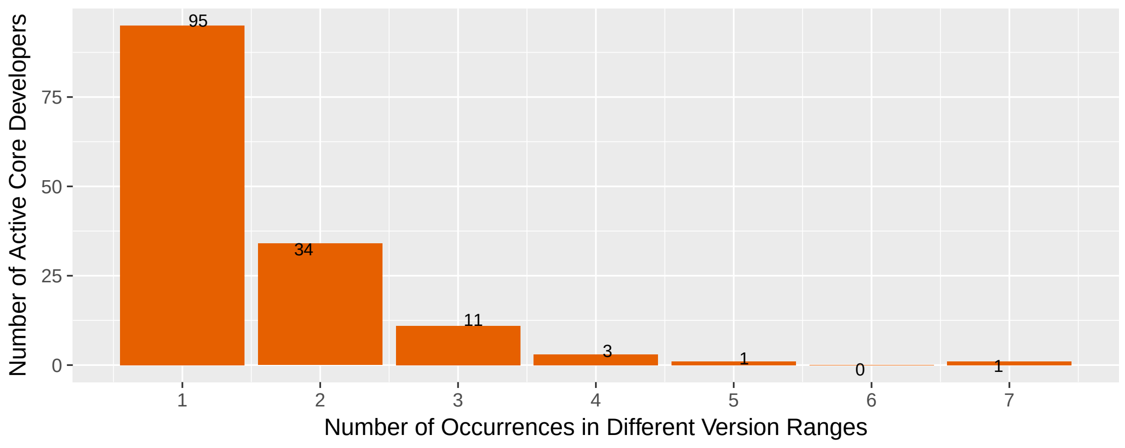


Figure 5.16: Numbers of recurring core developers based on a classification by eigenvector centrality

***H3* (Rejected):** We found that approximately 30% of all core developers are in a core state in, at least, eight of the 15 version ranges on a classification by commit count and over 10% on a classification by degree centrality. However, we did not find the most of the classified core developers to be active in, at least, eight of our 15 investigated Firefox version development ranges for none of our chosen operationalizations. Therefore, we reject our hypothesis *H3* for the classifications produced by any of our four operationalizations.

5.1.4 Discussion

After we presented the results of all our hypotheses regarding our first research question, we now judge and discuss these results in respect to *RQ1*.

On the analysis of hypothesis *H1*, we found that the most of the developers are classified as peripheral in each of our selected version development ranges for all of the classifications produced by our selected count- and network-based operationalizations. However, when considering the commit count of the developers as the classification metric, in each of our 15 selected Firefox versions, at least, 100 developers could be classified as core. Even more developers, between 150 and 210, could be classified as core on that classification, when we additionally considered the developers as core which were in a core state in, at least, one of the previous three version development ranges (sliding window). Thus, a classification by commit count classified more developers as core than any of our other operationalizations. Because of our specified 80% threshold, we can, therefore, conclude that many developers (at least 100) contribute nearly the same amount of commits in each version development range and that there are not only a few developers contributing by far more commits than other developers.

Further, a classification by degree centrality classified the second most developers as core on each version development range (between 50 and 100). Since this operationalization is based on the number of edges a developer has to other developers within our built co-change-based developer network (see Section 4.2.2), we can conclude that also many developers have many technical relationships to other developers by modifying many different functions in multiple files.

In contrast, a classification by LOC count only classified less than the half of the developers as core as on a classification by commit count and even only two developers as core on version 42. Those findings lead to the conclusion that we probably have the same disadvantages when using the LOC count as an operationalization to classify the developers as when using the LOC count as an activity measure (see Section 4.3). Those include that file renames as well as the copying of files in a VCS are considered as a full rewrite of those files. Further, commits do not always contain LOC which were self-written by the author, e.g., when updating an used framework, which can immensely increase the number of changed LOC within a commit. That is also the case for few commits on version 42, whereby we receive those low numbers of core developers. Here, one developer contributed several commits which caused a lot of changed LOC, due to framework updates and file renames. On one single commit, this developer almost changed 6,000 different files with over 3 million added and 207,599 deleted LOC[5]. Hence, this developer has contributed much more changed LOC than any other developer during the development range of version 42, making him/her together with the developer with the second most changed LOC the only two core developers due to our 80% threshold.

Since a classification by eigenvector centrality classified less than 25 developers as core on each version development range and thereby the least developers in comparison to our other classifications, we can conclude that only a few developer have a high eigenvector centrality within our built network. However, this operationalization extends the degree centrality by additionally considering the edges to developers that are themselves in globally central positions. But, since a classification produced by degree centrality resulted in the second most classified core developers, we can further conclude that the limiting factor of a developer not being classified as core on a classification by eigenvector centrality is his/her low number of edges to developers that are in globally central positions. Therefore, obviously only a few FIREFOX core developers have an immense higher number of technical relationships to important other core developers than the most of the developers classified as core on a classification by degree centrality.

On the analysis of the developer turnover values, we observed that the classifications which classified many developers as core and also have a large difference between the number of plain core developers in each version range to the number of core developers based on the sliding window approach, have a way lower turnover value of core developers, i.e., a higher role stability, than the classifications which only classified a few developers as core. This also reflects on the analysis of the role stability based on a classification of developers on two consecutive version development ranges (see Section 5.1.2) as well as on the analysis of the recurring core developers (see Section 5.1.3). Since the most developers have been classified as core on a classification by commit count, the role stability of core developers based on a classification by this operationalization is much higher, than their role stability on a classification

[5]https://github.com/mozilla/gecko-dev/commit/e7aaf06e9c1e5c28c3534be27ce2e48c61829903

which only classified a few developers as core, i.e., the classifications by eigenvector centrality and LOC count. Further, a lot more developers could be classified as core in the most of our selected version development ranges on a classification by commit count than on the classifications with fewer classified core developers.

Based on these findings, we can conclude that the role stability of core developers is very dependent on the number of developers that were classified as core in the same classification. However, there is no noticeable relation between the number of classified peripheral developers and the role stability of peripheral developers.

Furthermore, the characteristically high role stability of core developers could only be confirmed using a developer classification by commit count. The characteristically low role stability of peripheral developers, on the other hand, were represented the best when using a classification by degree centrality.

5.2 *RQ2*: Contribution Patterns of Core and Peripheral Developers

How do the contribution patterns differ among core and peripheral developers?

We now present and discuss the results of our second research question regarding the difference among core and peripheral developers in terms of their contribution activity.

5.2.1 Results for *H4*

The activity of peripheral developers is significantly higher in the last two weeks of a version development range than in the first weeks.

We investigate *RQ2* in the context of hypothesis *H4* by analyzing the contribution activity pattern of peripheral developers at the end of a version development range. Therefore, as described in Section 4.3, we again classify the developers as core and peripheral according to our four operationalizations. Based on these classifications, we individually analyze the developer activity of peripheral developers on a weekly basis for each of our investigating Firefox versions 37 to 51. As a reference and for the sake of completeness and comparison regarding *RQ2*, we additionally illustrate the activity of core developers resulting from our classifications.

In Figure 5.17, we present the weekly developer activity by class based on a classification by commit count for each development range of our investigating Firefox versions. At the first glance, we can see that the numbers of commits, i.e., the developer activity, are pretty unstable within a version development range for both developer classes. Especially the activity in the last week is way lower than in the first weeks on the most investigating version development ranges. Hence, we can further notice a significant increase of the developer activity from the last week of a version development range to the first week of the following version development range. Even though it seems like this finding kind of contradicts with our hypothesis, the activity of the penultimate week of our development ranges is in general not distinguished by a large decrease of the commit count. On eight of those 15 versions the penultimate week is even either the week with the most or the second most peripheral number of commits. In contrast to the last week of a version development range, the activity values of the penultimate weeks therefore might conform to our hypothesis that the peripheral developer activity increases on the last two weeks of a version development range. The analysis regarding the developer activity using the sliding window approach does not lead to different results. Although we receive a higher activity for core developers and a lower activity for peripheral developers due to the additional consideration of developers classified as core in the previous three development ranges on each selected version development range (see Figure 5.1), we can not recognize a significantly different contribution pattern for both developer classes.

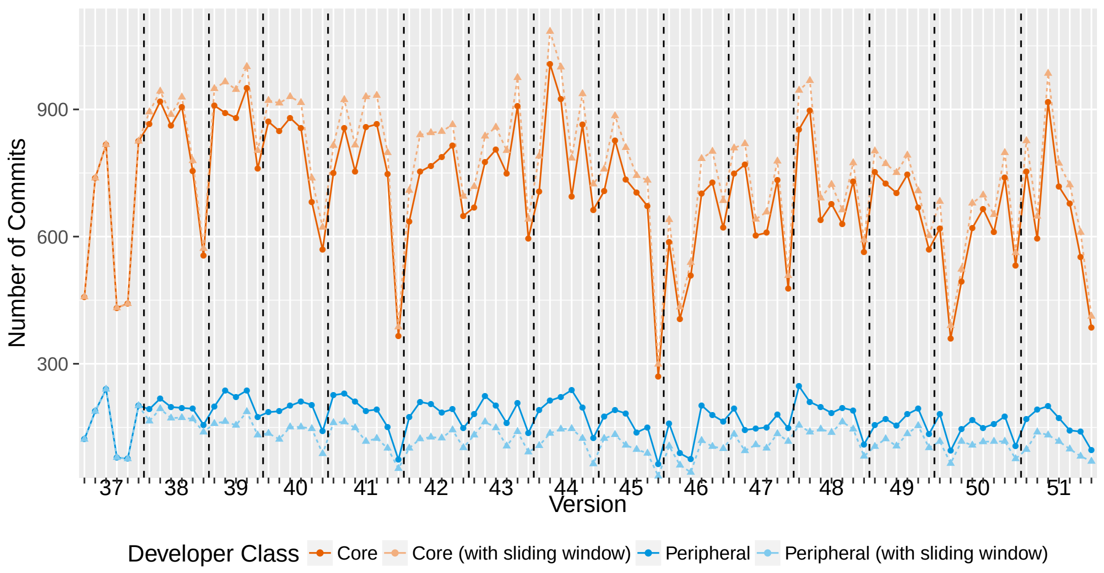


Figure 5.17: Weekly numbers of commits in each developer class based on a classification by commit count

To statistically evaluate the results in respect to our hypothesis, we additionally need to apply a statistical test for both approaches. Therefor, we firstly need to test if the data is normally distributed using a Shapiro–Wilk test, as done on hypothesis *H1*. If so, we can apply a *paired t-test*, since we want to compare the developer activity of the same developer class (peripheral) on two different time intervals, i.e., the first weeks and the last two weeks of each investigating version development range. If the Shapiro–Wilk test states that the data is not normally distributed, we have to execute a *Wilcoxon Signed Rank test* which is the parametric equivalent to the paired t-test [CF14].

To be able to correctly execute the corresponding test, we prepare the activity data of both time intervals in each version development range by dividing the activity measure, e.g., the commit count, by the length of the intervals in weeks, to retrieve the normalized activity values, since both investigating time intervals have an unequal length. We then execute the appropriate statistical test by testing against the null hypothesis H_0 ”The activity of peripheral developers is not higher in the last two weeks of a version development range than in the first weeks" and its corresponding alternative hypothesis H_1 "The activity of peripheral developers is higher in the last two weeks of a version development range than in the first weeks". As for hypothesis *H1*, we specify the significance level as $\alpha = 0.05$ and therefore reject our null hypothesis, i.e., accept our alternative hypothesis, if our resulting p–value is lower than α.

Therefore, we firstly conduct a Shapiro–Wilk test to choose the appropriate test according to the normal distribution property of our peripheral activity data sets resulted from the plain classifications. This test results in p–values of 0.24 for the peripheral developer activity in the last two weeks and 0.56 for their activity in the first weeks of a version development range. Since these values are above our specified significance level α of 0.05, we can conduct a paired t-test on the data sets. This leads to a p–value of 0.997 which is of course higher than our defined significance level α of 0.05 for that test. Therefore, we can not reject our null hypothesis that the activity of peripheral developers is not higher in the last two weeks of a version development range than in the first weeks and thereby can not accept our hypothesis *H4* by analyzing the commit count as the activity measure based on a plain classification by commit count.

Secondly, we need to conduct this statistical test for the peripheral activity data resulted from our developer classifications using the sliding window approach. Since both p–values of 0.92 and 0.96 for the peripheral developer activity in our investigating time intervals are above the specified significance level, we can assume that the data sets are normally distributed and conduct a paired t-test. With the resulting p–value of 0.993, we still can not reject our null hypothesis, whereby we also can not accept our hypothesis *H4* for a classification by commit count using the sliding window approach.

Also the numbers of commits by peripheral developers on each individual week of a version development range based on a classification by LOC count, as shown in Figure 5.18, lead to similar results regarding our hypothesis. As on the classifications by commit count, we can see right at the first glance, that the numbers of commits are in general lower in the last week than in the weeks before for each selected version development range, at least, for the peripheral developers. Further, we can observe that the numbers of commits for peripheral developers are way higher and that the developer activity values for both classes are much more unstable on a classification by LOC count than on a classification by commit count. On the activity values for the sliding window approach, we can notice that the developer activity of core and peripheral developers is in general nearly at the same level. But further, we can still recognize a decrease of the activity on the last week of our selected version development ranges, which contradicts to our hypothesis.

We again statistically analyze our findings regarding hypothesis *H4* by conducting the appropriate statistical tests on our both approaches. The Shapiro–Wilk test for the peripheral activity data based on the plain classifications, therefore, results in p–values of 0.40 for the peripheral developer activity in the last two weeks and 0.015 for their activity in the first weeks of a version development range. Since the latter is below our specified α, we assume that our data is not normally distributed, whereby we conduct a Wilcoxon Signed Rank test on the data sets. This test results

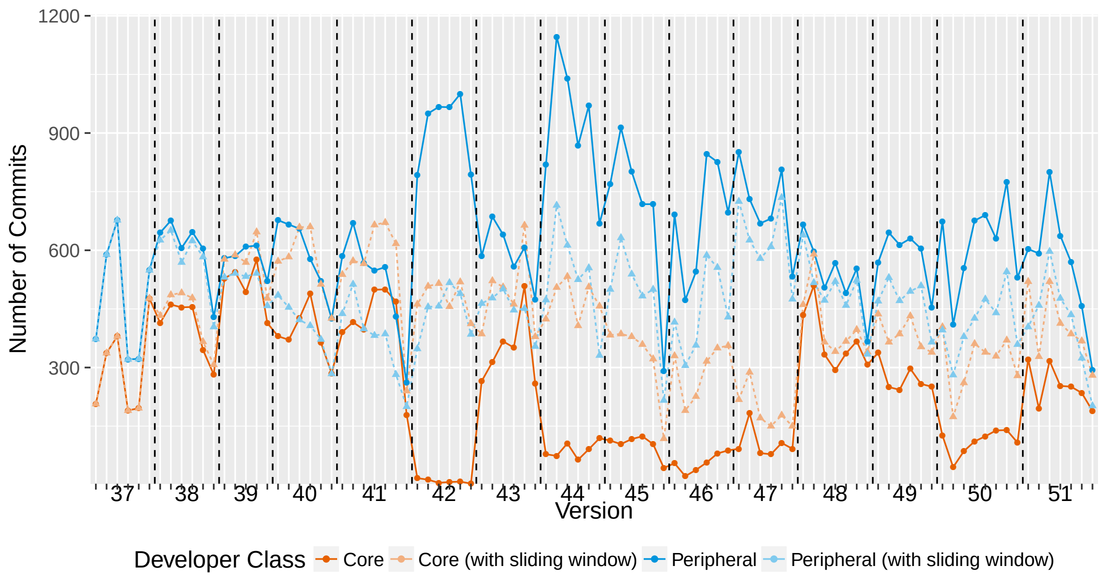


Figure 5.18: Weekly numbers of commits in each developer class based on a classification by LOC count

in a p–value of 0.998, whereby we can also not accept our hypothesis *H4* for a plain classification by LOC count.

In contrary, since the p–values of 0.40 and 0.48 resulting from the execution of the Shapiro–Wilk test on our data sets from the sliding window approach on a classification by LOC count are both above the specified α, we assume that these data sets are normally distributed. But, as on the previous statistical tests regarding this hypothesis, the p–value of 0.999 resulting from the corresponding paired t-test contradicts to our hypothesis *H4*.

The same results can be found by analyzing the peripheral developer activity on the classifications produced by our network-based operationalizations. The results for the classifications by degree centrality are shown in Figure 5.19 and the results for the classifications by eigenvector centrality in Figure 5.20. As on the count-based operationalizations, we can recognize a significant decrease of the number of commits by peripheral developers on the last week of almost every version development range for both network-based classifications. Even though the sliding window approach leads to an increase of the activity of core developers and to a decrease of the activity of peripheral developers in comparison to our plain classifications, we can still observe the same contribution pattern regarding the decrease of the developer activity at the last week of the most of our selected version development ranges.

As on the count-based operationalizations, we conduct appropriate statistical tests on both approaches for both network-based classifications to statistically evaluate our findings according to the hypothesis *H4*. Starting with the plain classifications by degree centrality, the Shapiro–Wilk test leads to p–values of 0.33 and 0.82 for our two data sets, which let us accordingly conduct a paired t-test. But, as on the count-based classifications, we again receive a very high p–value of 0.995 by executing this test, whereby we can also not confirm our hypothesis *H4* for these classifications.

Although the data sets of the corresponding sliding window approach enables us to execute a paired t-test, too, due to p–values of 0.15 and 0.27 resulting from the Shapiro–Wilk test, we also receive a high p–value of 0.99 for that t-test. Thereby, we can not accept hypothesis *H4* for the sliding window approach on a classification by degree centrality either.

Further, the p–values of the Shapiro–Wilk test for the classifications by eigenvector centrality for both approaches, which are 0.95 and 0.52 for the plain classifications and 0.47 and 0.86 for the sliding window approach, enables us to execute a paired t-test on this approach, too. The p–values of 0.9956 for the plain classifications and 0.997 for the sliding window approach resulting from the corresponding t-tests, do also not underline our hypothesis, whereby we can also not accept hypothesis *H4* for a classification by eigenvector centrality.

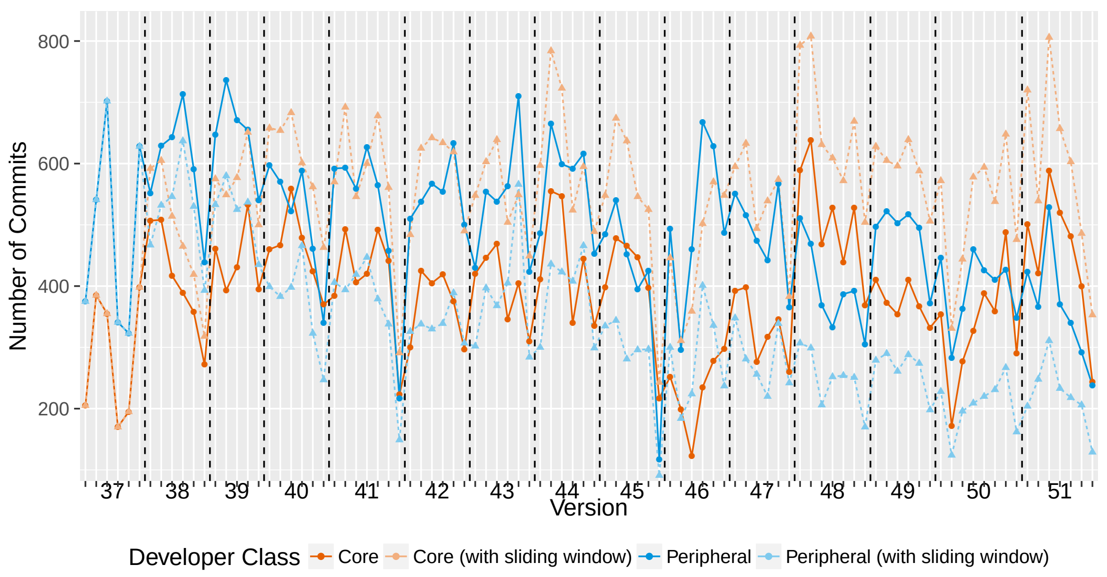


Figure 5.19: Weekly numbers of commits in each developer class based on a classification by degree centrality

Next to the investigation of the absolute numbers of commits per developer class resulting from our classifications, we now further discuss the results of analyzing the median numbers of commits for each developer class. As stated in Section 4.3, this eliminates the impact of a few developers having an extremely high or an extremely low developer activity in comparison to other developers in the same class and should therefore present an alternative view on the data in respect to our hypothesis.

As shown in Figure 5.21 for a classification by commit count, the activity of peripheral developers is much more stable by considering the median number of commits per developer than by considering the absolute number of commits, as shown in Figure 5.17. We can also recognize that the most of the peripheral developers only contributed one commit per week, since the median number of commits per peripheral developers is set at one on average for our investigated version development ranges. Because of the stability of the peripheral developer activity, we can not recognize a significant increase of the activity values at the end of a version development range with respect to our hypothesis. The only development range with such

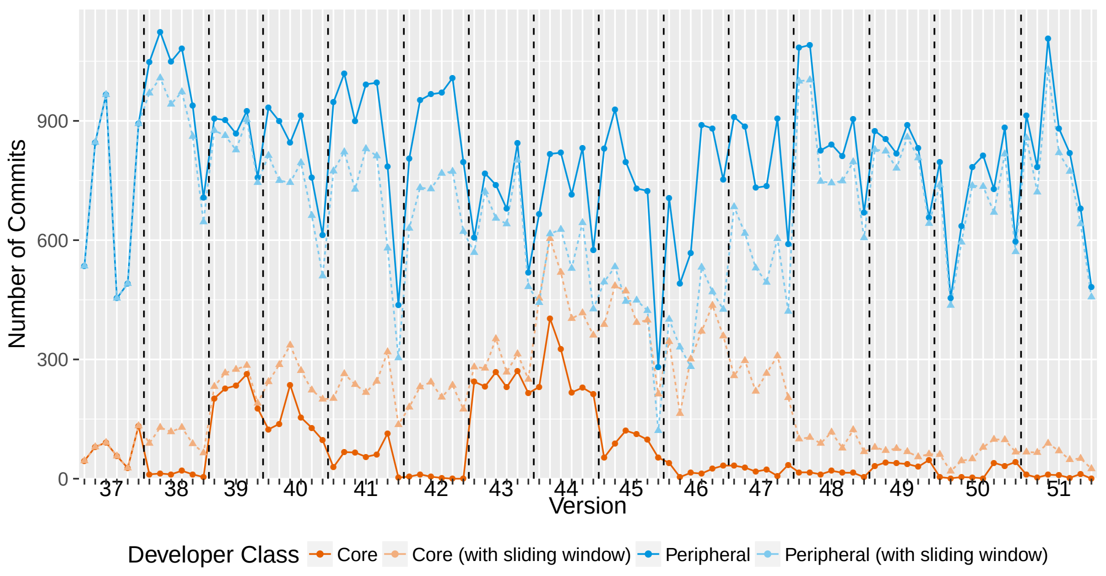


Figure 5.20: Weekly numbers of commits in each developer class based on a classification by eigenvector centrality

an increase is the one for version 40, on which the median number of commits per peripheral developer on the last week is twice as high as on the previous weeks. The contribution pattern of the core developers, on the other hand, partially conforms to our previously findings on the absolute numbers of commits, since we can observe a decrease of the median number of commits in the last week on some version development ranges. Also the analysis of the sliding window approach does not lead to different results of the contribution patterns for both developer classes.

To statistically evaluate hypothesis *H4* for the median peripheral numbers of commits based on a classification by commit count, we also firstly conduct a Shapiro–Wilk test on the data sets of our investigating time intervals. Due to very low p–values of almost zero for the data sets of both investigated approaches, we conduct a Wilcoxon Signed Rank test on these data sets of the peripheral activity data. With the resulting p–values of 0.21 for the plain classifications of the developers and 0.98 for the sliding window approach, our first observations regarding the peripheral contribution pattern gets confirmed, whereby we can not accept our hypothesis *H4* for the median numbers of commits based on a classification by commit count.

Although the analysis of the classifications by LOC count, as shown on Figure 5.22, leads to higher median numbers of commits for both developer classes than on a classification by commit count, we can still not observe an increase of the peripheral developer activity at the end of our selected FIREFOX version development ranges. The median numbers of commits for the core developers, on the other hand, show a significantly increase of the activity at the end of versions 44 and 50. As on the classifications by commit count, the median numbers of commits for a developer classification using a sliding window approach do not lead to significant different results in order to our hypothesis *H4*. Due to this similarity of the median numbers of commits for our both approaches, at least for the peripheral developers, the execution of the statistical tests also leads to similar results. Both result in p–

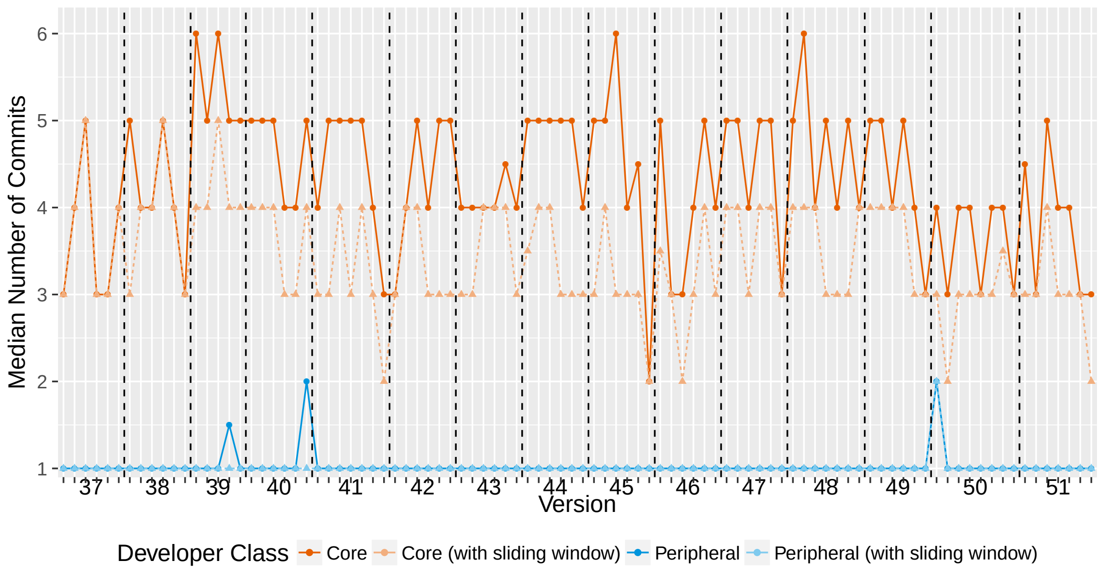


Figure 5.21: Weekly median numbers of commits in each developer class based on a classification by commit count

values of almost zero when executing the Shapiro–Wilk test. Also the corresponding Wilcoxon Signed Rank test for both approaches results in the same p–value of 0.95, which is way above our specified significance level. Therefore, we can not accept our hypothesis *H4* for the median numbers of commits based on a classification by LOC count.

While the median numbers of commits of peripheral developers are pretty stable on a classification by eigenvector centrality (see Figure 5.24), the classifications by degree centrality, as shown on Figure 5.23, are distinguished by some instabilities of the median commit count at different weeks over multiple version development ranges. However, these instabilities are mainly caused by a decrease of the median number of commits at the end of our selected version development ranges, what contradicts to the assumption of an increase of the activity at these points. Also the contribution pattern of core developers regarding the median numbers of commits on a classification by degree centrality has some instabilities which are often caused by a decrease of the activity at the end of a version development range. Although the median number of commits for core developers on a classification by eigenvector centrality decreases at the end of the most selected version development ranges, the number of commits increases at that point on version 46 and 50. As on the classifications produced by the count-based operationalizations, the sliding window approach does not lead to significantly different results on a classification by eigenvector centrality. Even though we can observe some deviations of the median numbers of commits resulting from a plain classification of the developers and those resulting from a classification based on the sliding window approach, these deviations do not play a significant role in respect to our hypothesis.

To further statistically evaluate these findings in respect to our hypothesis, we again conduct the corresponding statistical test for the comparison of the peripheral median numbers of commits in the last two weeks of each version development range

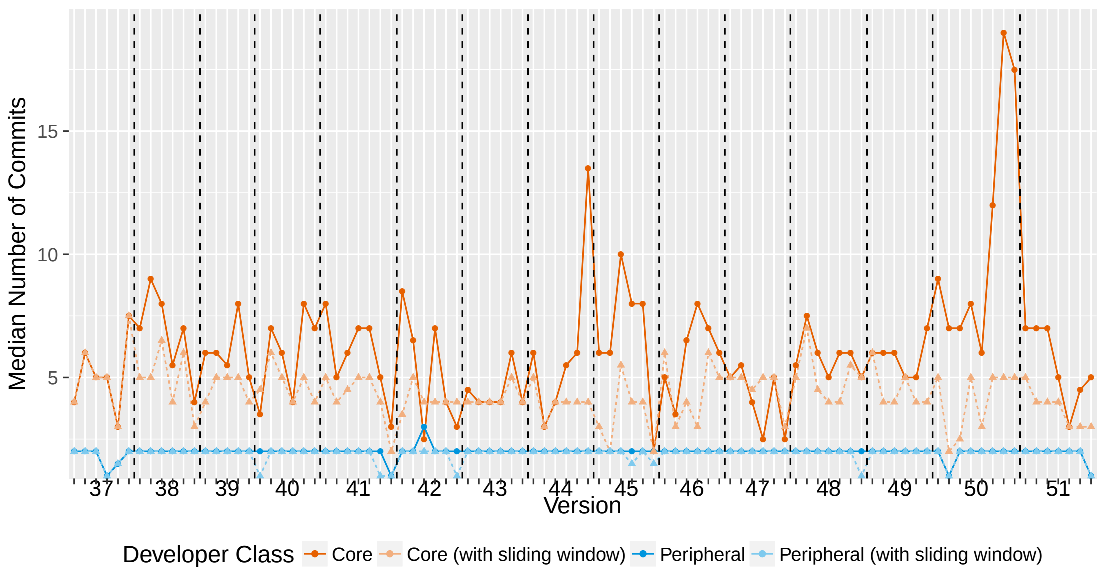


Figure 5.22: Weekly median numbers of commits in each developer class based on a classification by LOC count

and those of the first weeks of each select version range. Therefore, we present the results of the statistical analysis for the classifications by degree centrality first.

The execution of the Shapiro–Wilk test results in p–values of 0.002 for the median numbers of peripheral commits on the last two weeks and 0.018 for those of the first weeks based on our plain classifications. The classifications using the sliding window approach result in p–values of 0.003 and 0.100. Since, at least, one data set of both approaches results in a p–value below our specified α, we assume that these data sets are not normally distributed, whereby we conduct the corresponding Wilcoxon Signed Rank test on the data sets. Both approaches lead to p–values above the specified significance level (0.96 for the plain classifications and 0.32 for the sliding window approach), whereby we can not confirm our hypothesis $H4$ for the median numbers of commits based on a classification by degree centrality.

Since the p–values of the Shapiro–Wilk test on the classification by eigenvector centrality are almost zero for both approaches and, therefore, below our corresponding α, we also conduct a Wilcoxon Signed Rank test on the data sets of these classifications. As on the previous statistical analyses regarding hypothesis $H4$, we receive p–values above our specified significance level by executing this test (0.13 for the plain classifications and 0.86 for the sliding window approach), whereby we can also not accept our hypothesis $H4$ for the median numbers of commits based on a classification by eigenvector centrality.

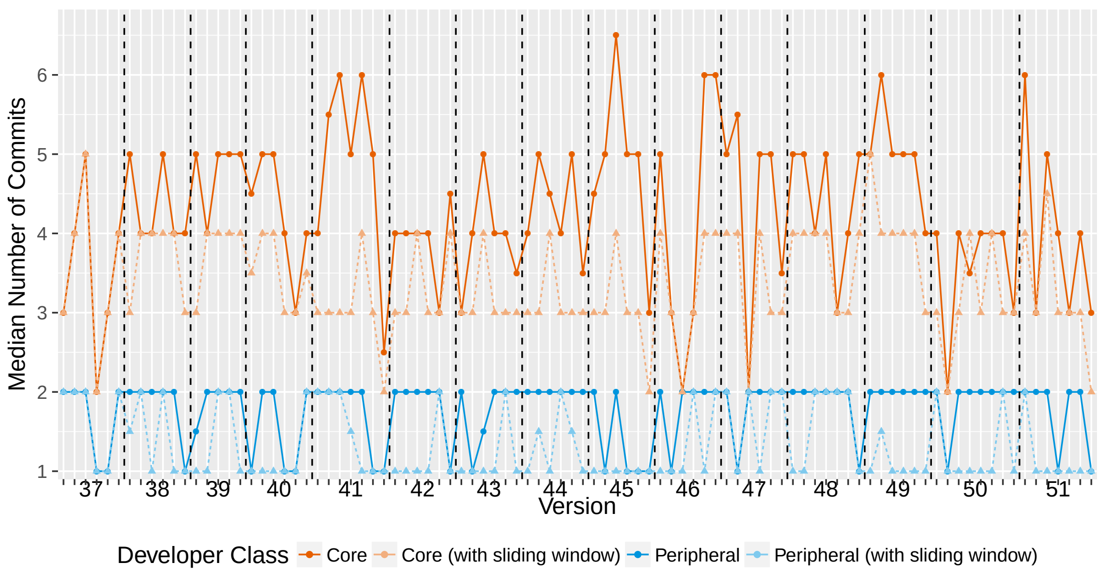


Figure 5.23: Weekly median numbers of commits in each developer class based on a classification by degree centrality

***H4* (Rejected):** Even though the classifications produced by our selected operationalizations led to different activity distributions among our two developer classes according to the numbers of commits, we could statistically evaluate that the activity of peripheral developers does not increase in the last two weeks of a version development range for any of these classifications. We even observed contrary results that the activity of peripheral as well as of core developers decreases especially in the last week of the most selected version development ranges. Further, also the analysis of the median numbers of commits in each version development range did not lead to the expected results. Therefore, we reject our hypothesis *H4*.

5.2.2 Discussion

By taking a closer look at the analyses regarding the developer activity measured by the absolute numbers of commits, we can recognize that the general pattern of the activity of peripheral developers is almost the same as the pattern of the activity of core developers at the end of a version development range for all of our investigated operationalizations. Especially the numbers of commits on the development ranges of version 41, 45, and 51 decrease significantly at the last week for both developer classes in all classifications. Even though the decrease of the contribution activity for core developers conforms to the assumptions made in the context of hypothesis *H4*, the decrease of the peripheral contribution activity strongly contradicts to hypothesis *H4* that the activity of peripheral developers would increase in the last two weeks of a version development range, due to the need of primarily bug fixing work. As we observed a decrease of the contribution activity at the end of a version development range for both developer classes, it is questionable if the reason for these findings

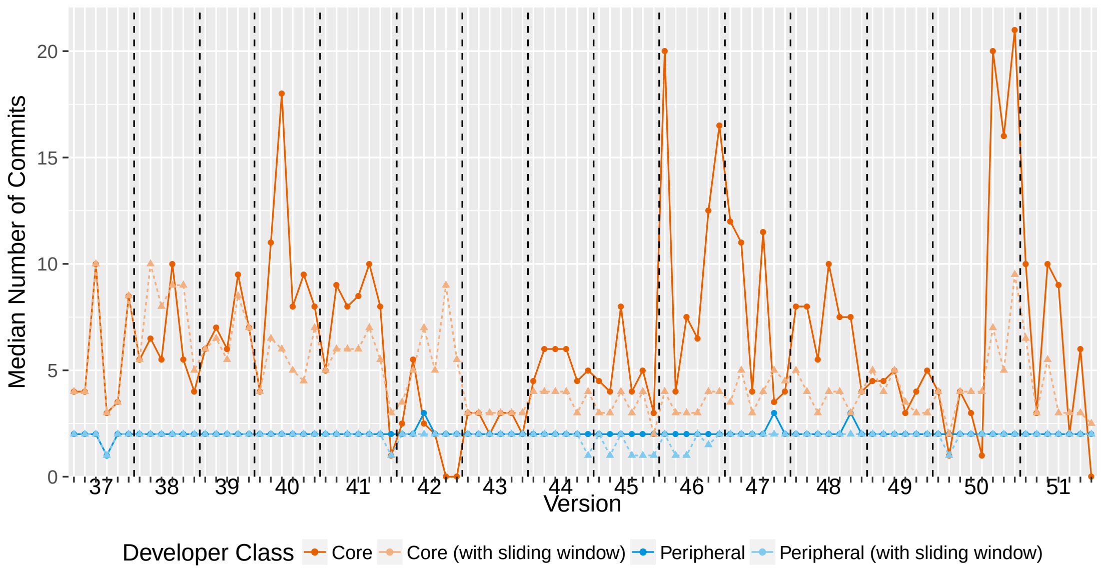


Figure 5.24: Weekly median numbers of commits in each developer class based on a classification by eigenvector centrality

is our wrong assumption on the characteristics and the developer responsibilities of peripheral developers, or if the decreases may rather occur due to the development guidelines of FIREFOX.

FIREFOX developers may most likely urged to not commit any fixes or new features a week before the end of a version development range, as that might cause quality or stability issues which cannot be fixed easily right before the release of a new Firefox version. Therefore, peripheral as well as core developers would try to land their patches either in the penultimate week of the same version development range or in the first week of the following one, which explains the recognizable high developer activity in these two development weeks.

This assumption is underlined by a mail within the developer mailing list of Firefox from May, 2017, in which the release management team of FIREFOX communicated the idea of implementing a soft code freeze on the Nightly channel repository. By increased scrutinies, this soft code freeze shall prevent any patches in the last week, which might regress quality or stability issues [Mozg]. Due to these Firefox development guidelines, it is questionable how representative our results for hypothesis *H4* are.

However, besides the analysis in respect to hypothesis *H4*, we could observe a higher contribution activity, i.e., a higher number of contributed commits, for peripheral developers than for core developers in all our classifications except the one based on the commit count on which the contribution activity is higher for core developers. This can be well explained since this operationalization classifies all developers as core who contributed the most commits to the FIREFOX VCS. Further, the median numbers of commits are in general higher for core developers in every classification produced by our selected operationalizations. Combining these findings with the results of hypothesis *H1*, we can conclude that the most developers are classified as peripheral, but these developers have a way lower number of commits per developer than the ones of core developers. Hence, the sum of the commits of all peripheral developers is in general higher than the total commit count of core developers (see Figure 5.18, Figure 5.19, and Figure 5.20), but since core developers are distinguished by a high contribution activity, they have in general a way higher median number of commits than peripheral developers. Therefore, those findings underline the typical characteristics of core and peripheral developers regarding their contribution patterns.

5.3 *RQ3*: Agreement of the Applied Operationalizations

> *To what extent do the applied count-based operationalizations agree with the network-based operationalizations?*

Finally, we discuss the results of our last hypothesis with respect to *RQ3*. This research question aims at investigating the extent of the agreement of our applied count-based and network-based operationalizations.

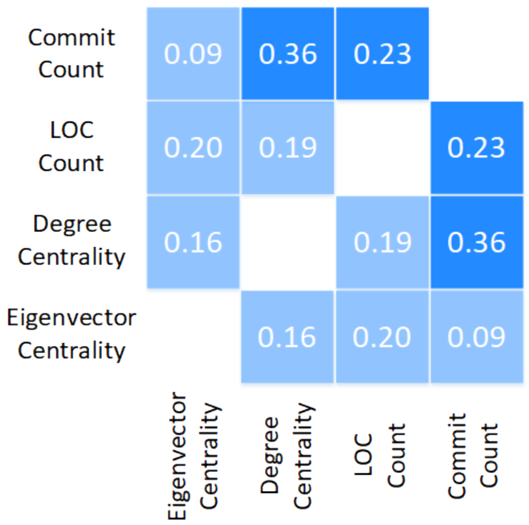
5.3.1 Results for *H5*

> *The classification of developers into core and peripheral obtained from count-based operationalizations will be consistent with the classification obtained from network-based operationalizations.*

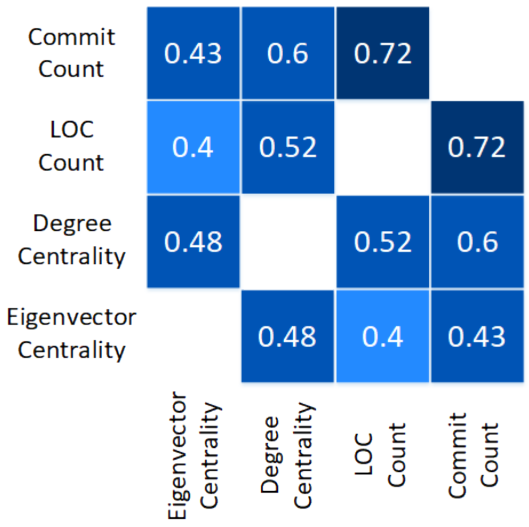
To calculate the agreement values in respect to the hypothesis, we compare the classifications produced by our applied count-based and network-based operationalizations pairwise on all of the 15 Firefox version development ranges using the Cohen's kappa, as described in Section 4.3.

The results, presented in figure 5.25a, show that only the strength of agreement of the operationalizations commit count and LOC count as well as the agreement of commit count and degree centrality can be treated as „fair“ with agreement values of 0.23 and 0.36 according to our defined ranges (see Section 4.3). All other agreements are just stated as „slight“ with values between 0.09 and 0.20. The lowest strength of agreement results from the comparison of the classifications by commit count and the classifications by eigenvector centrality.

Comparing those values with the agreements resulted from the analysis of ten different open-source projects as conducted by Joblin et al. [JAHM17] (see 5.25b), we can observe that the agreement values of the applied operationalizations for the Firefox project are way lower. Even the worst agreement value on the reference, which is 0.4 for the agreement of the classifications by LOC count and eigenvector centrality, is higher than the best agreement of the operationalizations applied on the Firefox project. But, we can also observe that the highest agreement values of the classifications by commit count and LOC count as well as the classifications by commit count and degree centrality on the Firefox project are also the highest values on the reference.



(a) Agreement values resulted from our case study on Mozilla Firefox



(b) Reference agreement values resulted from the study by Joblin et al. [JAHM17]

Figure 5.25: Agreement values of the selected operationalizations in terms of Cohen's kappa

***H5* (Accepted):** The agreement of the classifications produced by our applied count-based and network-based operationalizations are way lower than on the reference study by Joblin et al. [JAHM17]. Although the agreement of the classifications by commit count and degree centrality results in a "fair" consistency, the agreement of the classifications by commit count and eigenvector centrality as well as the agreement of the classifications by LOC count with the classifications of both applied network-based operationalizations only result in a "slight" consistency. But, since none of the agreement values between the classifications produced by our count-based and network-based operationalizations are below zero which would mean a "poor" agreement, we accept our hypothesis *H5*.

5.3.2 Discussion

As described in Section 5.3.1, the agreement values of the classifications produced by our count-based and network-based operationalizations calculated using Cohen's kappa can only be stated as "slight" according to our defined ranges for the strength of agreement. Here, the lowest value is represented by the agreement of the classifications by commit count and eigenvector centrality. This conforms to our findings on the previous research questions and hypotheses. On the analysis of hypothesis *H1*, we found that the classifications by commit count resulted in the most classified core developers compared to the classifications based on the other operationalizations. The classifications by eigenvector centrality, on the other hand, resulted in the lowest number of classified core developers. Further, the role stability as well as the developer activity distributions on the classifications produced by both operationalizations differed significantly. On the contrary, the comparison of the classifications by commit count and by degree centrality resulted in the highest agreement of our applied operationalizations on the FIREFOX project. This relatively high agreement also conforms with our analyses of the previous hypotheses, since the classifications produced by these two operationalizations lead to similar results in respect of these hypotheses. Even though the agreement of the classifications by commit count differs between the selected two network-based operationalizations, the agreement of the classifications by LOC count with those two network-based operationalization is nearly equally stated at the upper boundary of a ”slight” agreement.

6 Threats To Validity

As already indicated in Chapter 5, there are some threats in our conducted analyses which can affect the validity of the results. Further, applying our analyses of the Firefox project on other OSS projects may lead to significantly different results, because of the alternative release model of Firefox. Hence, we now discuss those internal as well as the resulting external threats to the validity of the results in this chapter.

6.1 Internal Validity

As we have seen, the rapid release model of Firefox has obviously a huge impact on the developer activity of core and peripheral developers. Since a release cycle has a length of only six to eight weeks, many new features or larger bug fixes are too complex to be contributed to the VCS within this short time interval. Therefore, some parts of those large contributions are partially developed many months before they get integrated into a release channel of a particular Firefox version. As we only use the author dates to extract the corresponding commits for our analyses regarding the developer activity, we do only consider the commits which are done during the official length of a version development range. Hence, we do not consider any changes that are contributed before the corresponding release cycle length of six to eight weeks. This could be solved by using the commit date of the commits, which does not represent the time when a commit was firstly committed to the VCS as the author date of a commit does, but represents the date of when a commit gets pushed to another repository or branch, i.e., gets pushed to the next release channel. But, by doing so, we would loose the coarse information of when a commit got developed and could, therefore, not assign the commits to their corresponding Firefox version, which we need to correctly analyze the developer activity.

Further, as defined in Section 4.2, our analyses are based on the changes made on the Aurora release channel of Firefox. Even though the most of the changes made on the VCS repository of the Nightly channel are also pushed to the Aurora channel, some commits on Nightly may not be integrated to the next release channels due to stability or security issues, whereby we do not have any guarantee to receive the full developer activity when only analyzing the Aurora channel. Additionally, there might also be some urgent changes, e.g., security fixes, made exclusively to the Beta or Release channel which are also not considered by our analyses due to the focus on the Aurora channel.

An additionally important threat to mention is the exclusive consideration and classification of the authors of the commits. As described in Section 3.3, a commit has to be checked and verified by the corresponding module owner. This is a very experienced developer with a detailed knowledge of the Mozilla codebase and a large involvement in a module of this codebase, i.e., has a strong ownership of this module [Mozk]. As those are also established characteristics of core developers (see Chapter 2), the module owner and reviewer of a commit on the VCS might also be a potential core developer. Further, the push of new verified commits to the Nightly channel as well as the push of stable commits from one release channel to the next one has to be conducted by a community member with commit access, as explained in Section 3.3. Of course, not every developer on the Firefox project has this access, whereby it is pretty common that the author of a commit is not the same person as the *push user*. Community members with commit access are, therefore, developers who already established themselves as trustworthy developers on the project and have a big knowledge of the feature development process [Mozi]. Because of that, those developers might also be potential core developers. Even though module owners and developers with commit access should also be considered as core developers within our analyses, it is technically difficult to extract the advanced information about those individuals, since our data extraction is based on the available commit data on the Firefox Git mirror (see Section 4.2). Although the commit data on this Git mirror is always kept up to date with the official commit data on Mercurial, there can not be extracted any information about the push user using that mirror. Further, the only information about the reviewer on those commits is an alias of the reviewer name on the commit message [Moza, Mozd].

Moreover, we have selected two count-based and network-based operationalizations for the classification of developers as core and peripheral which have already been established as appropriate metrics by a previous study about the classification of developers in OSS projects by Joblin et al. [JAHM17]. However, our investigations about the consistency of the classifications produced by those operationalizations on the Firefox project in context of hypothesis *H5* (see Section 5.3.1) resulted in way lower agreement values in terms of Cohen's kappa than on the reference study. Since those classifications are the base for all of our analyses in this study, the low agreements do also affect the results of those analyses. Thus, we receive contradictory results on the individual investigations of the four classification in respect to our hypotheses. Therefore, the question remains if the chosen operationalizations are good enough to receive valid results on the Firefox project and which operationalization reflects the real classification of developers as core and peripheral the best. As we have seen, especially the analyses based on a classification by LOC count led to anomalous results on the commit data of the Firefox VCS, because of their easily manipulable consideration of changed LOC. Due to those threats to the validity, it might be necessary to conduct a developer survey to receive the perception of multiple Firefox developers about the produced classifications by our chosen operationalizations, as done by Joblin et al. [JAHM17], whereby the validity of those applied operationalizations can be further verified.

6.2 External Validity

It is questionable if the results of our analyses based on the data of the Firefox project are also representative for other OSS projects. Even though the Firefox

Web browser is a well-known typical open-source project, the current release model of FIREFOX with its frequent releases of new versions can strongly affect our investigations on the activity of core and peripheral developers, as described in Section 6.1. Therefore, when investigating the activity of the developers in other OSS projects, the release model and the development process should always be kept in mind. However, the results can clearly give a good insight into the role stability and the collaboration as well as the contribution patterns of OSS projects with a very fast release model like on the FIREFOX project.

Additionally, the applied operationalizations to classify the developers as core and peripheral might have led to good results on other studies of OSS projects, but led to limited results on this FIREFOX study. Therefore, there is no guarantee that those operationalizations lead to valid results on other OSS projects with an alternative release model either.

7 Conclusion

Identifying developers who contribute an essential part to the source code of an OSS project or who have many collaborative relationships to other developers can be crucial for the purpose of coordination within this project [CH13]. Therefore, several consistent operationalizations have been established in previous studies to classify the developers as core and peripheral according to their amount of contributions to the project's VCS as well as to their communication and collaboration patterns. In this case study, we wanted to apply and verify the consistency of these operationalizations on the large and well-known OSS project Mozilla Firefox to reveal the composition of their development team and to analyze the characteristically role stability of core and peripheral developers on that project. Since Firefox is especially known for its rapid release model with frequent releases of new versions every six to eight weeks, a focus of this work was further to investigate the contribution pattern of core and peripheral developers during these short version development ranges.

We conducted our analyses by applying two count-based operationalizations as well as two network-based operationalizations on the VCS commit data of 15 consecutive Mozilla Firefox versions. By counting the number of contributed commits (commit count) and the number of contributed LOC (LOC count) as our selected established count-based operationalizations, we focused on classifying the developers according to their amount of contributions to the VCS of Firefox. The network-based classifications, on the other hand, are based on co-change-based networks consisting of developers as nodes and their technical relationship as edges between the corresponding developers. Here, we counted the number of edges between the developers (degree centrality) and calculated the importance of the developers by either being connected to many other developers or by being connected to developers in globally central positions (eigenvector centrality), to classify the developers according to their collaboration patterns. For each of those chosen operationalizations, we classified the developers as core whose number of contributions to the VCS (count-based) or their centrality value (network-based) together made up 80% of the total number of contributions or respectively 80% of the total centrality value of all developers.

We could confirm our assumption that in each of our 15 version development ranges the most of the Firefox developers are classified as peripheral for each of our four operationalizations and that only a few developers are responsible for the majority of the development work. Further, we could only provide evidence of a characteristically high role stability of core developers for a classification by commit count. Although a classification by degree centrality leads to a higher role stability for core developers than for peripheral developers, the most of the developers in a core state on a version development range still rather tend to transition to a peripheral or absent state on the following version range than maintaining their state. A classification by LOC count and eigenvector centrality even resulted in a higher role stability for peripheral developers than for core developers what contradicts to our specified assumptions as well as to the established characteristics of these two developer classes. Furthermore, our analyses regarding the contribution patterns of core and peripheral developers led to different distributions of the developer activity among our two developer classes for each applied operationalization. In contrast to our applied network-based classifications as well as the classification by LOC count, the classification by commit count led to higher absolute numbers of commits for core developers than for peripheral developers. However, we could not provide evidence of our assumption that the activity of peripheral developers would increase at the end of a Firefox version development range due to the increased need of bug fixes and enhancements at that time period. Instead, we found that the developer activity for both classes decreases significantly at the last week of the version development ranges due to a soft code-freeze in the Firefox development guidelines.

Due to these findings, we could only confirm a "slight" and "fair" consistency of the applied count-based and network-based operationalizations for the classification of the developers as core and peripheral on the Firefox project. Even though these operationalizations led to a higher consistency on previous studies of the classification of developers in OSS projects, the rapid release model of Mozilla Firefox as well as its complex feature development process seem to have a huge impact on the classification of the developers and, therefore, also on the resulting analyses on the developer activity. A further investigation of the project may need to be done with the goal of finding better appropriate operationalizations for such an OSS project by involving developer surveys to verify the validity of the operationalizations. Finally, in reference to future work it should be mentioned that by the time of the creation of this study the Mozilla management announced an adaption of their release cycle by removing their second release channel Aurora after Firefox version 54[6]. This might affect further analyses on the Firefox project with similar research topics.

Additionally, further investigations of the developer activity of core and peripheral developers on OSS projects with a similar release model may be interesting to analyze the consistency of the results. Thus, a statement could be made, if the results of the Mozilla Firefox project are representative for other OSS projects with a similar rapid release model.

[6]https://hacks.mozilla.org/2017/04/simplifying-firefox-release-channels/

A Appendix

Table A.1: Mapping of the Mercurial (*HG*) tags, representing the start of a version development range on the Aurora channel, and their corresponding commits on the Firefox Git mirror (*Git*)

Aurora Version	**Merge Date**	**Aurora Tag (HG)**	**Commit Hash (HG)**	**Commit Hash (Git)**
51	09/19/16	FIREFOX_AURORA_51_BASE	fc69febcbf6c	026552c
50	08/01/16	FIREFOX_AURORA_50_BASE	465d150bc8be	480c026
49	06/06/16	FIREFOX_AURORA_49_BASE	d98f20c25fee	e9707fa
48	04/25/16	FIREFOX_AURORA_48_BASE	1c6385ae1fe7	3c5ee70
47	03/07/16	FIREFOX_AURORA_47_BASE	68d3781deda0	4f4ec17
46	01/25/16	FIREFOX_AURORA_46_BASE	67c66c2878ae	90df109
45	12/14/15	FIREFOX_AURORA_45_BASE	99137d6d4061	a01ac57
44	11/29/15	FIREFOX_AURORA_44_BASE	67a788db9f07	cfd92d9
43	09/21/15	FIREFOX_AURORA_43_BASE	fcef8ded8221	e7a3664
42	08/10/15	FIREFOX_AURORA_42_BASE	7a19194812eb	c084d88
41	06/29/15	FIREFOX_AURORA_41_BASE	312c68b16549	21b3e5c
40	05/11/15	FIREFOX_AURORA_40_BASE	66a95a483d2c	16f43f1
39	03/30/15	FIREFOX_AURORA_39_BASE	1b6bf6612c0f	5df734e
38	02/23/15	FIREFOX_AURORA_38_BASE	98086da94ccd	f7354d5
37	01/12/15	FIREFOX_AURORA_37_BASE	2c951493eef5	3460534
36	11/28/14	FIREFOX_AURORA_36_BASE	b297a6727acf	accfd4e

Table A.2: Tracking of six FIREFOX features and bug-fixes with their push dates to each release channel

Commit	Author Date	Push to Nightly	Push to Aurora	Push to Beta	Push to Release
Version 51		**2016-08-01**	**2016-09-19**	**2016-11-14**	**2017-01-23**
Added support for FLAC (Free Lossless Audio Codec) playback	2016/10/25	2016/10/27	2016/11/10	2016/11/14	2017/16/01
Bug 1288247 - Open synced tabs in a new tab on middle click	2016/08/19	2016/08/25	2016/09/19	2016/11/14	2017/16/01
Version 50		**2016-06-06**	**2016-08-01**	**2016-09-19**	**2016-11-14**
Bug 1273950 - Fire a notification whenever a captive portal check is performed	2016/06/10	2016/06/10	2016/08/01	2016/09/19	2016/10/31
Bug 1281396 - Make URL non-lazy to avoid debuggee would run / recursion issues	2016/07/06	2016/07/08	2016/08/01	2016/09/19	2016/10/31
Bug 1265395 - Implement new low pass filter equation	2016/06/08	2016/06/12	2016/08/01	2016/09/19	2016/10/31
Version 49		**2016-04-25**	**2016-06-06**	**2016-08-01**	**2016-09-19**
Bug 1253284 - Allow reserved attribute without command attribute	2016/05/06	2016/05/18	2016/06/06	2016/08/01	2016/09/05

Bibliography

[CF14] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach.* John Wiley & Sons, 2014.

[CH13] Marcelo Cataldo and James D Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013.

[CWLH06] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 6, pages 118a–118a. IEEE, 2006.

[FA05] Brian Fitzgerald and Par J Agerfalk. The mysteries of open source software: Black and white and red all over? In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 196a–196a. IEEE, 2005.

[Inc] SAS Institute Inc. How do I interpret the Shapiro-Wilk test for normality in JMP®? http://www.jmp.com/support/notes/35/406.html. Accessed August 02, 2017.

[JAHM17] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *Proceedings of the 39th International Conference on Software Engineering*, pages 164–174. IEEE Press, 2017.

[JMA$^+$15] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: a fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 563–573. IEEE Press, 2015.

[KADZ15] Foutse Khomh, Bram Adams, Tejinder Dhaliwal, and Ying Zou. Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*, 20(2):336–373, 2015.

[LC03] Gwendolyn K Lee and Robert E Cole. From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization science*, 14(6):633–649, 2003.

[Leg11] Christian Legnitto. Merge dates vs release dates. https://blog.mozilla.org/channels/2011/05/11/merge-dates-vs-release-dates/, May 2011. Accessed June 12, 2017.

[LK77] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[MFH02] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[MOD07] Gary E Meek, Ceyhun Ozgur, and Kenneth Dunning. Comparison of the t vs. Wilcoxon Signed-Rank Test for Likert scale data and small samples. *Journal of Modern Applied Statistical Methods*, 6(1):10, 2007.

[Moza] Mozilla. Contributing. https://developer.mozilla.org/en-US/docs/Tools/Contributing. Accessed August 06, 2017.

[Mozb] Mozilla. Firefox 0.1 Release Notes. http://website-archive.mozilla.org/www.mozilla.org/firefox_releasenotes/en-US/firefox/releases/0.1.html. Accessed June 10, 2017.

[Mozc] Mozilla. Firefox 1.0 Release Notes. http://website-archive.mozilla.org/www.mozilla.org/firefox_releasenotes/en-US/firefox/releases/1.0.html. Accessed June 10, 2017.

[Mozd] Mozilla. GIT Firefox Repository. https://github.com/mozilla/gecko-dev. Accessed June 07, 2017.

[Moze] Mozilla. Handling Mozilla Security Bugs. https://www.mozilla.org/en-US/about/governance/policies/security-group/bugs/. Accessed July 07, 2017.

[Mozf] Mozilla. History of the Mozilla Project. https://www.mozilla.org/en-US/about/history/details/. Accessed June 07, 2017.

[Mozg] Mozilla. Introducing Nightly soft code freeze. https://mail.mozilla.org/pipermail/firefox-dev/2017-May/005461.html. Accessed July 23, 2017.

[Mozh] Mozilla. Mercurial Firefox Repository. https://hg.mozilla.org/. Accessed June 07, 2017.

[Mozi] Mozilla. Mozilla Commit Access Policy. https://www.mozilla.org/en-US/about/governance/policies/commit/access-policy/. Accessed August 06, 2017.

[Mozj] Mozilla. Mozilla Firefox: Development Specifics. https://mozilla.github.io/process-releases/draft/development_specifics/. Accessed June 11, 2017.

[Mozk] Mozilla. Mozilla Modules and Module Owners. https://www.mozilla.org/en-US/about/governance/policies/module-ownership/. Accessed August 06, 2017.

[Mozl] Mozilla. Our Contributors. https://www.mozilla.org/credits/. Accessed June 07, 2017.

[Moz11] Mozilla. New Channels for Firefox Rapid Releases. https://blog.mozilla.org/blog/2011/04/13/new-channels-for-firefox-rapid-releases/, April 2011. Accessed June 10, 2017.

[Moz16] Mozilla. Update on 2016 Firefox Release Schedule. https://blog.mozilla.org/futurereleases/2016/02/04/update-on-2016-firefox-release-schedule/, February 2016. Accessed June 10, 2017.

[Moz17a] Mozilla. Contributing to the Mozilla code base. https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Introduction, April 2017. Accessed June 11, 2017.

[Moz17b] Mozilla. How to submit a patch. https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/How_to_Submit_a_Patch, January 2017. Accessed June 11, 2017.

[Moz17c] Mozilla. RapidRelease/Calendar. https://wiki.mozilla.org/RapidRelease/Calendar, May 2017. Accessed June 12, 2017.

[NYN$^+$02] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85. ACM, 2002.

[Sha11] Stephen Shankland. Rapid-release Firefox meets corporate backlash. https://www.cnet.com/news/rapid-release-firefox-meets-corporate-backlash, June 2011. Accessed June 10, 2017.

[Sie] Siemens. Codeface. https://siemens.github.io/codeface/. Accessed August 02, 2017.

[Sta] StatCounter. Browser Market Share Worldwide 2016. http://gs.statcounter.com/browser-market-share#yearly-2016-2016-bar. Accessed June 10, 2017.

[TRC10] Antonio Terceiro, Luiz Romario Rios, and Christina Chavez. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *Software Engineering (SBES), 2010 Brazilian Symposium on*, pages 21–29. IEEE, 2010.

[YK03] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation Open Source Software developers. In *Proceedings of the 25th international conference on software engineering*, pages 419–429. IEEE Computer Society, 2003.

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Ferdinand Frank

Passau, den 7. September 2017