Communications of the Association for Information Systems

Volume 49 Article 8

9-21-2021

Episodic Peripheral Contributors and Technical Dependencies in Open Source Software (OSS) Ecosystems

Eunyoung Moon
School of Computing KAIST

Recommended Citation
Moon, E. (2021). Episodic Peripheral Contributors and Technical Dependencies in Open Source Software (OSS) Ecosystems. Communications of the Association for Information Systems, 49, pp-pp.
https://doi.org/10.17705/1CAIS.04908

Communications of the Association for Information Systems

Research Article DOI: 10.17705/1CAIS.04908 ISSN: 1529-3181

Episodic Peripheral Contributors and Technical Dependencies in Open Source Software (OSS) Ecosystems

Eunyoung Moon
School of Computing
KAIST
Daejeon, Republic of Korea

Abstract:

Despite the fact that OSS contributors tend to eschew traditional organizational hierarchies, researchers have found that, in many cases, OSS contributors make tightly coupled system designs and successfully coordinate highly interdependent tasks. Although researchers have explained how OSS contributors make tightly coupled code contributions, we do not know the characteristics of individuals who make such contributions. While previous studies have considered OSS projects as single, independent containers, I note that OSS projects do not constitute independent or standalone entities but reuse and, thus, depend one another. This reuse creates complex networks of interdependencies called "software ecosystems". In this paper, I analyze OSS contributors who have made tightly coupled code contributions using two lenses: the core-periphery lens and the habitual-episodic lens. Based on investigating three volunteer-driven OSS projects, I found OSS contributors who make tightly coupled code contributions to have different code-contribution patterns. Interestingly, I found that half of such contributors made no previous code contributions to the sampled projects but episodically authored patches (or pull requests) that increased software coupling. Based on further investigation, I suggest a multiple-fluid-container view that accommodates software ecosystems in which multiple containers (multiple OSS projects) co-evolve with each container (each OSS project) readily accessible.

Keywords: Open Source Software, Episodic Volunteering, Peripheral Contributors, Software Ecosystems, Software Coupling.

This manuscript underwent peer review. It was received 7/31/2019 and was with the authors for fourteen months for three revisions. Julie Kendall served as Associate Editor.

1 Introduction

Researchers have considered that, when designing development products, organizations should desire correspondence between their structure and the structure of the systems they develop to improve their performance (Baldwin & Clark, 2000; Conway, 1968; Sanchez, 1996). Theorists have argued that a system's technical dependencies will mirror the organizational structure of the organization (e.g., its formal and informal communication and geographic co-location) that develops it (Baldwin & Clark, 2000; Sanchez & Mahoney, 1996). Researchers consider modularity[1] (or loose-coupling) important in open source software (OSS)[2] development because a modular software design lowers coordination requirements among geographically distributed OSS contributors (Benkler, 2002; MacCormack, Rusnak, & Baldwin, 2006). An integral (or tightly coupled[3]) OSS system will require OSS contributors to comprehend enough of the entire system design to make code contributions. Accordingly, researchers have argued that a loosely coupled OSS system will attract more OSS contributors than a tightly coupled OSS system because contributors can make code contributions to the loosely coupled OSS system without having to know the entire system (Baldwin & Clark, 2006). However, researchers have also found that, in many cases, open collaborative groups make tightly coupled system designs and can complete highly interdependent tasks (Colfer, 2010; Colfer & Baldwin, 2010). This finding suggests substantial discrepancies between theoretical predictions and OSS development in practice.

To account for these discrepancies, researchers have investigated how OSS contributors make tightly coupled code contributions. A tightly coupled code contribution refers to a change that one makes to the main branch of a project's source code repository and that increases software coupling. For instance, in a case study on the Rubinius community, Lindberg, Berente, Gaskin, and Lyytinen (2016) investigated how OSS contributors addressed unresolved dependencies (Lindberg, Berente, Gaskin, & Lyytinen, 2016). They found that, as development tasks become more interdependent, OSS contributors generated more activity variations and more contributors became involved. That is, interdependent tasks significantly involved more activities and contributors. In another study, Moon and Howison (2018) investigated 1,033 episodes across three volunteer-driven OSS projects and revealed a more detailed picture of how OSS contributors make tightly coupled code contributions. They operationalized a tightly coupled code contribution as a patch (or a pull request[4]) that increases software coupling. Taking a configurational approach, the authors identified commonly occurring patterns for how contributors make tightly coupled code contributions, which they called "coordination recipes". They found that OSS contributors simultaneously combined rich interpersonal communication and planned labor divisions when making tightly coupled code contributions. In other words, OSS contributors tend to work in a highly coordinated manner when they make tightly coupled code contributions rather than rely on a commonly known open source approach, such as the incremental layering of code around existing modules, the deferral of complex tasks (Howison & Crowston, 2014), and actionable transparency (Colfer & Baldwin, 2010).

Research has provided explanations about how OSS contributors manage technical dependencies in practice. However, we know relatively little about the characteristics of OSS contributors who make tightly coupled code contributions in general and their code-contribution patterns in particular. In investigating these contributors, I consider the following concept from the literature: as the OSS phenomenon has undergone a significant transformation from its free software to more mainstream, the way of developing OSS has taken a whole-product approach (Fitzgerald, 2006). While previous studies consider organizations as "containers" for a work system's elements (Winter et al., 2014), I note that OSS projects do not constitute independent or standalone entities but reuse and, thus, depend on one another. This

[1] Modularity is a design scheme characterized by a high degree of independence between modules based on one-to-one mapping from functional elements to the modules (Ulrich, 1995). Modular designs are loosely coupled because making changes to one module has little impact on other modules (MacCormack, Baldwin, & Rusnak, 2012). I use the terms "loosely coupled" and "modular" interchangeably in this study.

[2] The term "open source" emphasizes the potential benefits that organizations can gain from sharing and collaborating on software source code (Fogel, 2005). It also emphasizes that open source software's distribution terms must abide by the criteria that the Open Source Initiative defines (Chris & Sam, 1999).

[3] An integral structure includes complex mapping from functional elements to modules or coupled interfaces between modules (Ulrich, 1995). Integral designs are tightly coupled in that, at the extreme, changes to one module may mean one has to make changes to every module. I use the terms "tightly coupled" and "integral" interchangeably in this study.

[4] A patch refers to a set of changes that one applies to a codebase that adds new functionalities or fixes a bug (Howison & Crowston, 2014). A pull request, a GitHub feature, lets a person tell others about changes they have pushed to a branch in a repository on GitHub. Once someone opens a pull request, others can review the set of changes. I use the terms patch and pull request interchangeably in this study.

reuse creates complex networks of interdependencies called “software ecosystems” (Valiev, Vasilescu, & Herbsleb, 2018). In software ecosystems, contributors develop a group of software projects, which co-evolve in the same environment. Accordingly, I consider OSS projects in software ecosystems in this study.

By better understanding OSS contributors who make tightly coupled code contributions, we can better understand why they make such code contributions and what those contributions mean. In turn, this new knowledge can help researchers better characterize emerging social structures in software ecosystems because, thus far, researchers have characterized OSS contributors only *in* a particular OSS project and not in the software ecosystem context. It can also provide practical insights. For instance, practitioners can make informed decisions with regard to the proper person to ask about making tightly coupled code contributions to OSS projects and about how to facilitate organizing processes given the complex networks that dependent OSS projects exist in. In addition, coding platform designers may consider improving the development environment to provide a clear sign to contributors who make code contributions to dependent projects. They can also consider the impact that changes in a codebase will have on dependent OSS projects.

In this study, I particularly examine the individuals who contribute to volunteer-driven OSS projects that lack any significant corporate participation. The term volunteer refers to “a participant engaged in any form of OSS contribution, but not directly employed or sponsored by the OSS foundation or vendor for the project to which they contribute” (Barcomb, Kaufmann, Riehle, Stol, & Fitzgerald, 2018, p. 3). Even when volunteers establish an OSS project, if firms have dominated the project, then such a project does not constitute a volunteer-driven project. Corporate OSS projects often have more defined management structures, more explicit coordination mechanisms, and more co-location activity than volunteer-based OSS projects (Dahlander, 2007; Dahlander & Magnusson, 2008; Feller, Finnegan, Fitzgerald, & Hayes, 2008; Germonprez et al., 2016; Germonprez, Kendall, Kendall, & Young, 2014). The characteristics of volunteer-driven OSS projects require researchers to explain such conditions because the characteristics of volunteer-driven OSS projects differ from those of traditional organizations (e.g., hierarchy-based or market-based). Accordingly, in this study, I focus on volunteer-driven OSS projects. I characterize OSS contributors who made tightly coupled code contributions to three volunteer-driven OSS projects (i.e., GNU grep, IPython, and scikit-image) using the core-periphery lens and the habitual-episodic lens. I found that:

- OSS contributors who make tightly coupled code contributions contribute code according in different patterns.
- Across the three projects, half of such contributors made no code contributions to the projects before but made code contributions that increased software coupling.
- A particular OSS project did not bind tightly coupled code contributions that half of such contributors made; rather, the contributions derived purpose, meaning, and structure from their related OSS projects.
- I found that a multiple-fluid-container perspective, which I propose in this paper, can accommodate software ecosystems in which multiple containers (multiple related OSS projects) co-evolve, with each container (each OSS project) readily accessible.
- From a multiple-fluid-container perspective, which I propose in this paper, containers have continually redefined (as opposed to fixed) boundaries since related containers’ work system elements flow into and out from each container at any point.

This paper proceeds as follows: in Section 2, I present and discuss the study’s research questions. In Section 3, I describe the process I followed to select cases. In this study, I consider OSS projects in software ecosystems and investigate the characteristics of OSS contributors who make tightly coupled code contributions and, in particular, their code-contribution patterns. To characterize such OSS contributors, I use two lenses: the core-periphery lens and the habitual-episodic lens. I introduce these lenses in Section 4. In Section 5, I describe how I analyzed the data and report my findings. In Section 6, I discuss the study’s theoretical and practical implications and its limitations. I also conclude the work and suggest future research directions.

2 Research Questions

Software ecosystems have emerged as a means to understand the relationships among software projects, products, communities, and organizations (Franco-Bedoya, Ameller, Costal, & Franch, 2017). OSS projects benefit from reusing other projects that, in turn, reuse yet other projects. This reuse creates complex networks of interdependencies called software ecosystems (Valiev et al., 2018). Accordingly, OSS projects typically create an environment in which software ecosystems emerge.

Considering the complex networks of interdependencies among OSS projects in software ecosystems, loosely coupled software structures allow participants outside an OSS project but active in other OSS projects to make code contributions without having to know about the entire code base. Given an open, fluid membership, any participant can move in and out of an OSS project; they can decide to stop making contributions to a particular project or resume their participation at any time (Moon & Howison, 2014). One has no guarantee that a particular contributor will be available to manage interdependent tasks (Howison, 2009). Despite the fact that OSS contributors eschew traditional organizational hierarchies (Lindberg et al., 2016), researchers have found that, in many cases, OSS contributors make tightly coupled system designs and manage to coordinate themselves to complete highly interdependent tasks (Colfer, 2010; Colfer & Baldwin, 2010).

While researchers have explained how OSS contributors make tightly coupled code contributions (Lindberg et al., 2016; Moon & Howison, 2018), they have not sufficiently explained the contributors' characteristics. Researchers argue that a small number of core contributors make large contributions throughout the entire system, whereas a large group of periphery contributors make only small changes in individual modules (Colfer, 2010; Colfer & Baldwin, 2010). However, such reasoning has been speculative. In some cases, small changes may have effects across modules, while large changes made in one module may have no effect on other modules. Previous studies have not examined contributors who make tightly coupled code contributions. However, some earlier works have indicated the need for such investigations. For instance, one study that focused on large commits that involved changes to a large number of files in the PostgreSQL project found that peripheral contributors occasionally made large commits that added new features to the project (Hindle, German, & Holt, 2008). Other researchers examined casual contributors who made at most one commit to OSS projects and found that 18 percent of such contributors added new features and that six percent updated the version of the software or related dependencies (Pinto, Steinmacher, & Gerosa, 2016). As such, the findings thus far suggest that non-core contributors can at times make non-trivial changes to project codebases. These findings suggest that core contributors do not always make tightly coupled code contributions. Accordingly, in this study, I investigate individuals who make tightly coupled code contributions and the following research question:

> **RQ1:** What characteristics do OSS contributors who make tightly coupled code contributions to volunteer-driven OSS projects have with regard to their code-contribution patterns?

If core or habitual developers make tightly coupled code contributions, it would not be surprising because they know how the entire software system works. However, even when contributors have not participated in a project, if they make tightly coupled code contributions to it, the factors that enable such contributors to make tightly coupled code contributions require further investigation. Accordingly, I investigate:

> **RQ2:** Under what conditions do OSS contributors make tightly coupled code contributions to volunteer-driven OSS projects?

3 Case Selection

In this study, I draw on theoretical sampling rather than random sampling since I focus on uncovering a specific phenomenon of interest. I consider criteria to select cases in order to maximize the range of information and unexpected insights into diverse OSS contributors who make tightly coupled code contributions.

First, I consider volunteer-driven OSS projects in this study. I inspected declared licenses, project histories, and contributors to determine whether volunteers drove an OSS project or not. Second, note that, in this study, I consider OSS projects in the software ecosystem context. Accordingly, OSS projects that have had dependent OSS projects maximize potential opportunities to study code contributions made in the software ecosystem context. One way to sample such projects involves considering an OSS project that also appears in OSS packages, such as the GNU system. For instance, the GNU system provides

various different applications, libraries, and developer tools. One also needs to consider whether an OSS project explicitly lists other projects using it on their official website. Finally, one can also consider library projects in that developers often rely on third-party libraries to use a particular functionality rather than reinvent the wheel (Kula, German, Ouni, Ishio, & Inoue, 2018).

Third, given that I examine individuals who make tightly coupled code contributions, I need to sample OSS projects that have a high software coupling level compared to other OSS projects that build similar software applications. An OSS project with a high software coupling level can provide many opportunities to identify tightly coupled code contributions. In contrast, an OSS project with a low software coupling level may mostly provide opportunities to identify loosely coupled code contributions. No absolute cutoff defines a high or low level of coupling in a codebase. Hence, I use relative comparisons among OSS projects that build similar software applications.

Fourth, one should also consider how many individuals contribute to a codebase for several reasons. Even when an OSS project satisfies the first three criteria, if few developers have mostly developed it, such a project does not maximize the ability to observe diverse OSS contributors who make tightly coupled code contributions. Accordingly, in this study, I looked for OSS projects with the largest number of contributors in their codebase compared to other OSS projects that focus on similar software applications.

To select cases, I began by briefly inspecting OSS projects using openhub.net, which provides basic information about projects, such as how many lines of code they contain and their functionalities. I also visited each prospective project's website to ascertain whether corporations participated in it to a high degree and to examine how it described the project's history and contributors. Once I determined that volunteers drove a project for certain, I looked for other OSS projects that developed similar functionalities to identify the OSS projects with the highest software coupling levels in their respective application domains. To find such projects, I looked for documents that listed a set of related projects or alternatives. In situations where such documents did not exist, I looked for public websites that compared similar projects.

I undertook numerous sampling processes to find samples that met the sampling criteria. Ultimately, through this process, I identified three software applications. Figure 1 depicts the process I followed to select an OSS project with the largest number of contributors in its codebase and the highest software coupling level in its application domain.

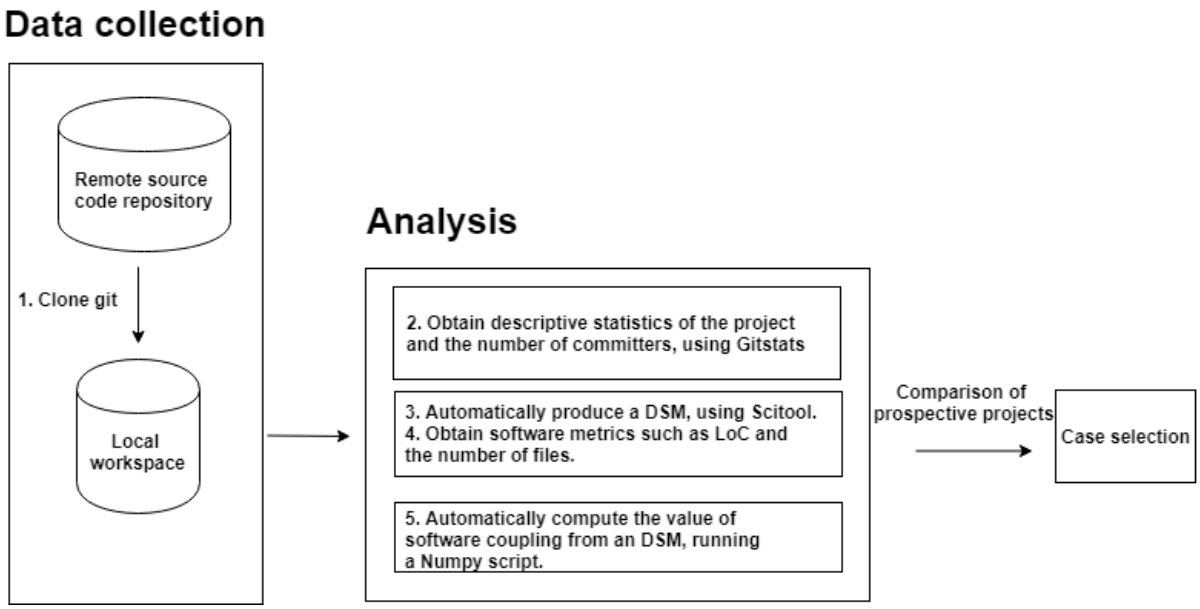


Figure 1. Process I Followed to Select OSS Projects with the Largest Number of Contributors in their Codebase and the Highest Software Coupling Level in the Software Application

In command-line searching utilities, GNU grep had the largest number of contributors and the highest software coupling. GNU grep is one of numerous GNU projects, which suggests that it forms a part of a complex network of interdependencies among various GNU packages. With regard to IPython, a message

that its founder posted via the IPython mailing list provided a way to compare similar projects. Accordingly, I compared these similar projects and found that IPython had the largest number of contributions in its codebase and the highest degree of software coupling. Furthermore, various OSS projects use IPython[5]. For instance, other OSS projects use IPython as an interactive command-line interface or as a library, while some also customize it for their projects. An image-processing library, scikit-image provides APIs for its client applications. Researchers and practitioners have widely used scikit-image, and it constitutes one package in the scientific Python ecosystem. A paper that scikit-image contributors (van der Walt et al., 2014) wrote listed other similar OSS projects. Hence, I also compared scikit-image with these OSS projects. I found that the scikit-image project had the highest level of software coupling and the largest number of contributors in its codebase.

Table 1 provides information about the three OSS projects. The projects sampled in this study range from small (GNU grep) to large (IPython, scikit-image) in terms of how many lines of code (LoC) they contained. In other words, the sampled projects did not constitute large projects (e.g., 100K LoC). However, they represent reasonable projects to study given that only a minor fraction of projects have a large LoC value (Capiluppi, Lago, & Morisio, 2003).

Table 1. OSS Projects Sampled in this Study

	GNU grep	IPython	scikit-image
The latest version †	2.21	4.0	0.12
Software coupling	52.7%	20.6%	14.9%
Lines of code (LoC)	10,669	78,512	72,036
Source files	17	338	433
Functions	181	2,995	2,810
Total number of contributors who made commits	39	527	217
Programming language	C	Python	Python

† As at July, 2016 (i.e., when I performed the sampling process).

4 Characterizing OSS Contributors

4.1 Core-periphery Lens

Previous studies on the OSS phenomenon have characterized OSS contributors into two groups: core contributors and periphery contributors (Crowston, Wei, & Howison, 2006b; Nakakoji, Yamamoto, Nishinaka, Kishida, & Ye, 2002). As Figure 2 illustrates, core developers who contribute most of the code and interact with one another a lot lie at the onion-like structure's center. Many people also refer to these individuals as "committers" because they can directly commit to a source code repository's main branch. Empirical studies have found that a core developer group typically comprises few contributors. For example, for Apache, the top 15 core developers made more than 83 percent of changes in its codebase (Mockus, Fielding, & Herbsleb, 2002). Another study found that contributors outnumber core developers by more than four to one (Dinh-Trong & Bieman, 2005).

In the next ring lies peripheral developers who submit occasional patches. Rullani and Haefliger (2013) define peripheral developers using a residual term in relation to core developers. Researchers have found peripheral developers to be less interconnected, less central, and to have fewer interactions than core developers (Crowston et al., 2006b). However, researchers have found that peripheral developers play an important role in often providing critical input to help solve technical problems (Rullani & Haefliger, 2013). Peripheral contributors also serve as the primary source of new needs and solutions for a community. Furthermore, research has found that, in some cases, contributors in the periphery make substantial changes to a codebase (Hindle et al., 2008; Pinto et al., 2016). Hindle et al. (2008) investigated commits that make changes in a large number of files, which they referred to as large commits. They manually classified large commits made in nine OSS projects and found that peripheral developers can make substantial changes and can even add new features.

[5] For example, see https://github.com/ipython/ipython/wiki/Projects-using-IPython

The onion model considers the process by which users transform from passive users to active users, to peripheral contributors, and, finally, to core contributors (see Figure 2) (Ducheneaut, 2005; Ye & Kishida, 2003). Before moving on to more substantial accomplishments, newcomers should take the following steps: peripherally monitor development activities, report bugs with potential patches, obtain the right to commit, and take charge of a module-sized project. Prior studies have primarily looked at role transformation processes in a single OSS project.

Although researchers have used the core-periphery lens to insightfully characterize OSS contributors, they have noted that it has certain limitations. The core-periphery lens captures a snapshot at one time and primarily focuses on the number of contributions. Researchers have suggested an alternative view that draws on a concept called episodic volunteering (Barcomb et al., 2018). In Section 4.2, I introduce episodic volunteering and discuss how one can characterize participants as habitual or episodic contributors.

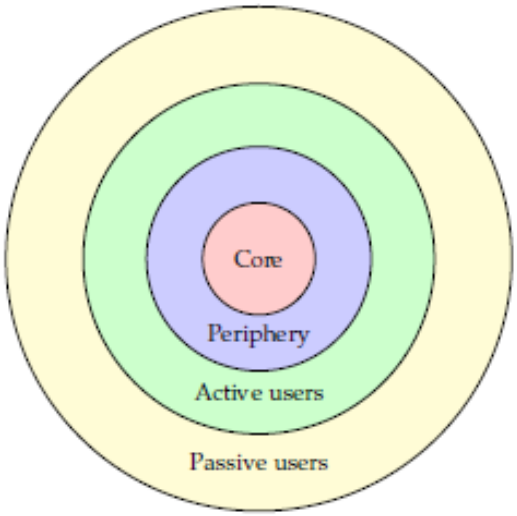


Figure 2. Core-periphery Structure of OSS Contributors (Barcomb et al., 2018)

4.2 Habitual-episodic lens

Episodic volunteering refers to short-term, sporadic participation that ranges from a one-time contribution to infrequent or short-term engagement (Macduff, 2004). Infrequent volunteering activities (once or several times a year) characterize episodic contributors (Hyde et al., 2014). Episodic volunteering constitutes a reflexive volunteering style based on personal preferences or conditional commitment. Episodic volunteers tend to participate in a more individual and non-committal manner; as such, altruistic and social motivations tend not to drive their participation as much (Hustinx & Lammertyn, 2003). Episodic volunteers usually perform one or two tasks and contribute for a short amount of time. In contrast, habitual volunteers make continuous or successive contributions on an ongoing basis (Harrison, 1995).

Episodic volunteering stands in contrast to the collective volunteering style in which volunteering represents an integral part of community life (Hustinx, Haski-Leventhal, & Handy, 2008). Research has found social incentives to drive episodic volunteers. For instance, one may have asked them to volunteer or they followed an example that friends or family set. For episodic volunteers, volunteering tends to be embedded in their social relationships and group membership outside the recipient organization in which they volunteer. Thus, episodic volunteers appear primarily to rely on their existing social relationships in their volunteering activities. In other words, episodic and habitual volunteers significantly differ (Hustinx et al., 2008).

Researchers have drawn the boundary between episodic and habitual volunteers based on how frequently or long they volunteer (Hustinx et al., 2008; Macduff, 2004). In systematically reviewing episodic volunteering, Hyde, Dunn, Scuffham, and Chambers (2014) found that infrequent volunteering activities (once or several times a year), relatively few monthly hours (less than four per month), no board membership, and weak identification with the organization or with volunteering in general characterize episodic contributors. Cnaan and Handy (2005) have proposed a continuum from episodic volunteers to

habitual volunteers. For instance, a one-time volunteer who gives two hours to help organize an event constitutes an episodic volunteer. In contrast, if an individual gives two hours every week, the individual constitutes an ongoing or consistent volunteer. However, if an individual gives two hours to organize an event and six months later returns to do the same, the individual falls between the first two. In studying volunteers at summer festivals, Handy, Brodeur, and Cnaan (2006) defined habitual volunteers as those who volunteer for multiple episodic opportunities on a continual basis, and genuine episodic volunteers as those who volunteer for two or fewer volunteer episodes per year. In other words, episodic volunteers give their time sporadically without an ongoing commitment, which means one can place them on a continuum between one-time volunteers and habitual volunteers.

In the OSS development context, Barcomb et al. (2018) surveyed 13 OSS communities and found widespread episodic volunteering. Lee and Carver (2017) also defined a one-time contributor (or a casual contributor) as an individual who had one patch merged to the source code repository of an OSS project. In their study, they examined 21 mature OSS projects that used the Gerrit code-review tool. Mining the projects' software repositories, they identified core contributors, periphery contributors, and one-time contributors as distinct groups of contributors in terms of the patch size as measured according to the lines of code, the time taken to get the patches accepted, the number of review comments, and the patches' acceptance rate. In another study, Pinto et al. (2016) found that 30 percent of one-time contributors provided (non-trivial in some cases) bug fixes. In addition, 18 percent of one-time contributors added new features, and six percent managed dependencies. They also found that one-time contributors usually participated in other projects. In their study, Barcomb, Stol, Riehle, and Fitzgerald (2019) examined episodic volunteering and found that episodic contributors overwhelmingly contributed habitually in other OSS projects.

In other words, empirical studies have shown that, with regard to OSS development, one-off contributors and sporadic or short-term contributors occasionally make substantial changes to a codebase (see Table 2). However, the existing core-periphery lens does not distinguish those developers in the periphery since it considers all developers in the periphery as a homogenous developer group. In addition, while contributors' commitments may change over time, the core-periphery lens captures a snapshot at a single point in time (Barcomb et al., 2018; van Wesel, Lin, Robles, & Serebrenik, 2017).

Table 2. Empirical Evidence of Peripheral, Episodic Contributors who Make Non-trivial Changes in a Codebase

Study	Project	Findings
Hindle et al. (2008)	Nine OSS projects	Non-core external developers made large commits that added new features, and those large commits made cross-cutting changes in the PostgreSQL project.
Jergensen, Sarma, & Wagstrom (2011)	GNOME	Newly joining contributors and OSS contributors who have been active for between two and five releases have the same level of centrality in terms of contributions in the codebase.
Pinto et al. (2016)	275 mature projects hosted on GitHub	• 30% of one-off contributors fixed (non-trivial in some cases) bugs. • 18% of one-off contributors contributed to adding new features. • 8% of one-off contributors contributed to refactoring. • 6% of one-off contributors updated software or dependencies.
Silva, Wiese, German, Steinmacher, & Gerosa (2017)	367 students who participated in the Google Summer of Code (GSoC)	• 14% of students participating in GSoC had their code merged before, after, and during the program. • 45% of students made no commits after GSoC.

5 Data Analysis and Findings

5.1 Phase 1: Identification of Tightly Coupled Code Contributions

5.1.1 Operationalization of Tightly Coupled Code Contributions

To answer the first research question, I need to characterize OSS contributors who made tightly coupled code contributions regarding their code-contribution patterns. Because OSS projects organize development activities around "pull requests" (or patches) (Lindberg et al., 2016), I identified the pull requests (or patches) that added technical dependencies to a codebase. Researchers typically identify

technical dependencies via analyzing source code or bytecodes. A functional dependency measures a system's structure in terms of highly coupled files via functional relationships; for instance, method X calls method Y. Researchers have used functional dependencies in prior work to measure software coupling (e.g., MacCormack et al., 2012).

A functional dependency focuses on functional relationships (called a "function call"). One statistically extracts function calls from source code, not from code in an execution state. To measure the functional dependency between source files, I used a design structure matrix (DSM) that Steward (1981) initially introduced and Eppinger, Whitney, Smith, and Gebala (1994) later developed for modeling interdependencies between engineered system elements.

5.1.2 Identification of Focal Periods

Rather than randomly sampling OSS system software releases, I tracked the way in which software coupling in the sampled OSS systems evolved over software releases. A software improvement or the implementation of some functionality triggers a software release. "Release often, release early" has emerged as a strong community norm in the OSS development community as frequent releases indicate a tight feedback loop from users and peripheral contributors (Crowston, Howison, & Annabi, 2006a). A version number signifies a release and its properties, such as whether it constitutes a major or minor release. Hence, a software release provides the basis by which to analyze changes in software coupling over time (Gall, Hajek, & Jazayeri**,** 1998).

In this study, I refer to an inter-release period during which software coupling increases significantly as a "focal period" and to the period between the date when someone made the first commit in an OSS project and the date when someone made the last commit right before the focal period as the "whole previous period" (see Figure 3). Accordingly, the whole previous period may comprise few or many inter-release periods. A software release comprises a collection of complete changes in a project's codebase, and contributors typically make such changes made with a patch (or pull request). In this study, I operationalize a tightly coupled code contribution as a patch (or a pull request) that increased the degree of software coupling. Accordingly, I identified the focal periods in the sampled OSS projects and then patches (or pull requests) that increased software coupling during the focal period.

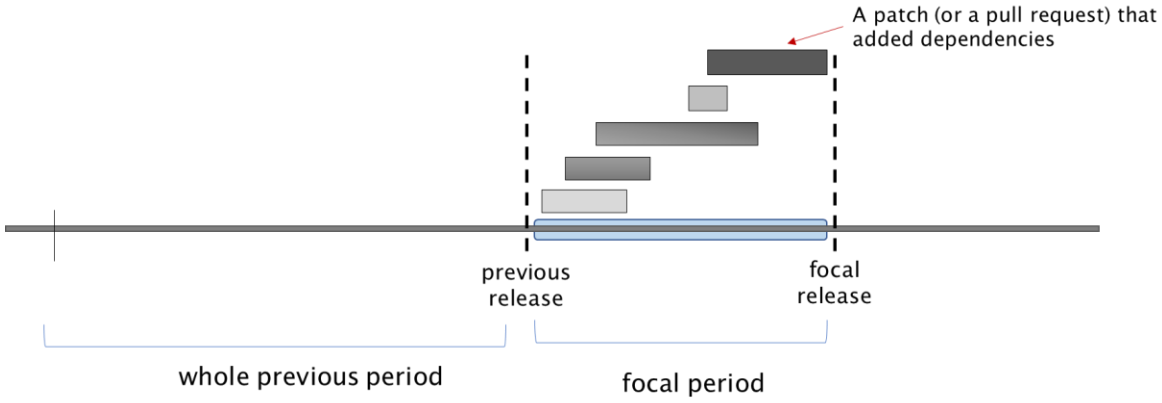


Figure 3. Focal Period and Whole Previous Period[6]

For GNU grep, I obtained 28 releases in total that contributors created over a 20-year period. Using Scitool (see www.scitools.com), I automatically extracted function calls between source files in DSM format for each release. Next, I checked each matrix cell if and only if functional dependencies between source files existed. In this step, I assigned binary numbers to the cells to capture a functional dependency's presence or absence. I then computed the software coupling value via a Numpy script that implemented a propagation cost algorithm (MacCormack et al., 2012; Milev, Muegge, & Weiss, 2009). The computed software coupling value presents the proportion of elements that a change to one element in the software may directly or indirectly affect. Figure 4 shows GNU grep's software coupling over the 28 releases.

[6] Notes that the focal period develops a focal release. The inter-release period right before the focal period develops a previous release.

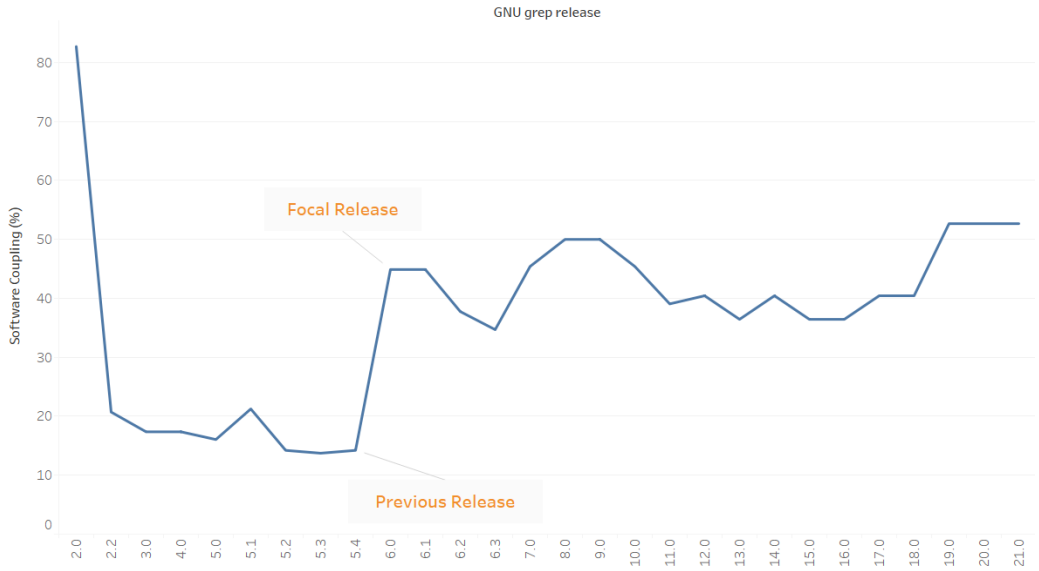


Figure 4. Evolution of GNU grep's Software Coupling Over 28 Releases[7]

To identify focal periods, I conducted a cluster analysis of the 28 software coupling measures for each release. The cluster analysis partitioned the 28 measures into three clusters. The three clusters' underlying distribution did not meet a parametric test's assumptions because the clusters lacked sufficient size and balance. Accordingly, I conducted a non-parametric test, the Kruskal-Wallis H test, which identified the inter-release period between grep 2.5.4 and grep 2.6 as a focal period in which software coupling significantly increased ($p < .05$). This finding indicates that grep became relatively tightly coupled during this focal period. Table 3 summarizes the software metrics that show the difference between the previous release and the focal release.

Table 4. Software Metrics of the Previous Release and the Focal Release for GNU grep

	Previous release (grep 2.5.4)	Focal release (grep 2.6)
Software coupling (%)	14.21	44.89
Lines of code	31,313	8,831
Number of source files	86	22
Number of functions	261	123
Inactive lines	7,096	1,431
Preprocessor lines	5,177	560
Note: the degree of software coupling measured in this study indicates the proportion of files that may be affected, on average, if a change is made to one file in the software system. Accordingly, I report the degree of software coupling in percentage (%).		

For IPython, I obtained 29 releases released over five years using the process that I describe above for GNU grep again to assess software coupling (see Figure 5). Statistical tests revealed three inter-release periods that significantly increased software coupling: between 0.10.2 and 0.11, between 0.13 and 1.0, and between 3.1.0 and 3.2.0. The release notes for IPython 3.2.0 described relatively few commits and contributors—15 authors made 74 commits. These low numbers do not present a viable opportunity to investigate diverse contributors. Accordingly, I did not find it appropriate to investigate the inter-release period between 3.1.0 and 3.2.0. Considering historical validity (Berney & Blane, 1997), I preferred a more recent period. Hence, I chose the inter-release period between 0.13 and 1.0 (from 30 June, 2012, to 8 August, 2013) for IPython (see Table 4).

[7] Note that I omit the major version number two in order to increase readability (for instance, on the Y-axis, 6.0 means release number 2.6.0). The Y-axis indicates the degree of software coupling, and the X-axis shows each release number.

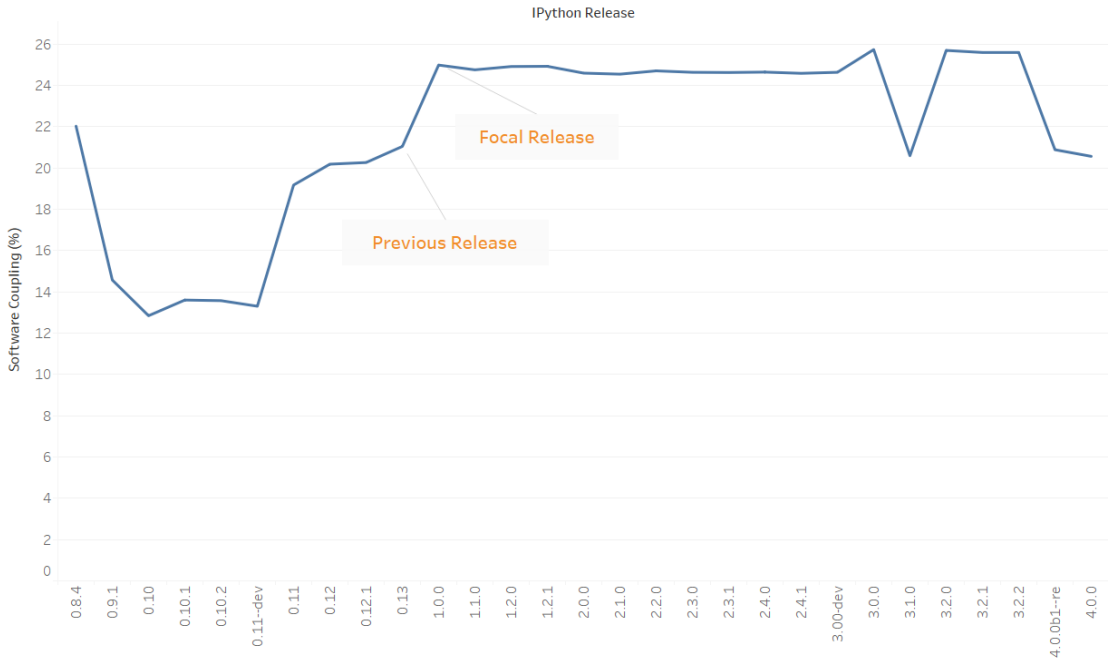


Figure 5. The Evolution of IPython's Software Coupling over Releases

Table 4. Software Metrics of the Previous Release and the Focal Release for IPython

	Previous release (IPython 0.13)	Focal release (IPython 1.0.0)
Software coupling (%)	21.04	24.98
Source lines of code	89,040	87,008
Number of source files	614	662
Number of functions	6,178	6,044

For scikit-image, I obtained 13 releases over seven years and reused the same processes (see Figure 6). From the statistical tests I conducted, I found two focal periods during which the degree of software coupling significantly increased: from release 0.7.0 to 0.8.0 and from release 0.10.0 to 0.11.0. As I state above, I collected and analyzed the retrospective datasets to more deeply explore how contributors made code contributions and how they worked on particular pull requests. To do so, I conducted interviews with contributors. Accordingly, to ensure historical validity, I chose the most recent period from release 0.10.0 to release 0.11.0 (from 28 May, 2014, to 4 March, 2015). Table 5 summarizes the software metrics that show the difference between the previous release and the focal release.

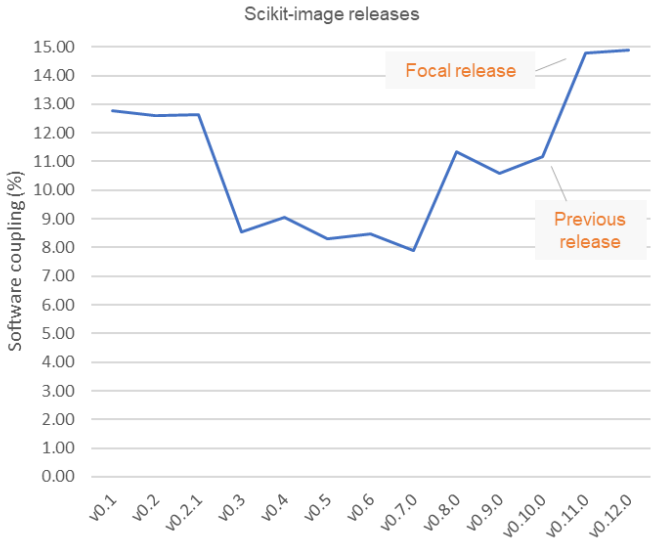


Figure 6. Evolution of scikit-image's Software Coupling over Releases

Table 5. Software Metrics of the Previous Release and the Focal Release for scikit-image

	Previous release (scikit-image 0.10)	Focal release (scikit-image 0.11)
Software coupling (%)	11.16	14.78
Source lines of code	53,565	66,825
Number of source files	351	406
Number of functions	2,148	2,659

5.1.3 Identification of Tightly Coupled Code Contributions

Once I identified the focal period, I then identified tightly coupled code contributions that increased software coupling during the focal period (see Figure 3). By identifying tightly coupled code contributions that occurred during the focal period, I could maximize my opportunity to investigate the contributors who made tightly coupled code contributions. In contrast, inter-release periods with no changes or with decreases in software coupling would likely not allow one to identify tightly coupled code contributions and particularly the OSS contributors that I focus on in this study.

To identify tightly coupled code contributions that occurred during the focal period, I identified a starting commit ID and an ending commit ID for each project's focal period. Next, I extracted commits in that commit ID range, the commit ID, authoring date, the commit's author, and commit log. I also ran the Numpy script to compute the software coupling value per commit. This process resulted in 216 commits for GNU grep, 4,208 commits for IPython, and 1,322 commits for scikit-image. Next, I compared the commits with patches pushed into GNU grep's main repository and identified pull requests that contributors merged into IPython's and scikit-image's master branches during the focal period. As a result, I identified 67 patches for GNU grep, 778 pull requests for IPython, and 188 pull requests for scikit-image.

Next, to identify tightly coupled code contributions, I computed changes in software coupling per patch (or pull request). Through this process, I identified eight among the 67 patches for GNU grep, 88 among the 778 pull requests for IPython, and 24 among the 188 pull requests for scikit-image that increased software coupling during the focal period. Note that only a small fraction of the work (12%) across the three projects increased software coupling during the focal period; in other words, most work done (79%) did not change software coupling. Having identified the tightly coupled code contributions that contributors made during the focal period, in Section 5.2, I describe the method I used to characterize the authors of these patches (or pull requests) that increased software coupling.

5.2 Phase 2: Identification and Characterization of Contributors

In this phase, I identified and characterized the contributors who authored patches (or pull requests) that increased software coupling during the focal period. Git distinguishes between authors and committers. An author refers to an individual who makes changes to files, whereas a committer refers to an individual who has the right to actually commit those files to the repository. The author does not always have the right to commit. In such a case, the committer should commit changes. In this study, I characterize individuals who authored patches (or pull requests) that increased software coupling using both the core-periphery and the habitual-episodic lenses. I also sometimes use authors and contributors interchangeably.

To identify contributors, I needed to address several initial concerns. A single contributor may have used multiple email addresses when making commits to the same project (Zhou & Mockus, 2015). A contributor may use a different email when working for a different organization or company over time or simply when switching to a different email for other reasons. Furthermore, in some cases, the same individual can spell their name differently with the same email. For instance, a contributor may fully spell out their first name, middle name, and last name while later using initials for their middle name or last name but with the same email.

Researchers have developed several identity-matching algorithms (e.g., Bird, Gourley, Devanbu, Gertz, & Swaminathan, 2006; Robles & Gonzalez-Barahona, 2005). However, current approaches can often result in false positives and false negatives. In addition, project-specific or community-specific rules and constraints can influence the name structure and format (Vasilescu, Serebrenik, Goeminne, & Mens, 2014). For these reasons, I matched a contributor's identity to their email address(es). I retrieved all commit logs that the sample projects recorded from their inception and stored this data as a CSV file. By sorting the authors' names and emails, I identified multiple emails associated with the same name. To confirm that the same contributor used those emails, I visited the contributor's GitHub user page, which provides their email and associated organizations (or companies). Moreover, I visited the personal website linked to their GitHub user page.

Over the whole previous period, I found several unique authors and that no single contributor used multiple emails in GNU grep. For the focal period, I also inspected the authors' names and emails. No identical contributors used multiple emails during the focal period in GNU grep, which left eight unique authors. For IPython, after matching identities, 144 unique identities remained from 211 different authors over the whole previous period, and 150 unique identities remained from 165 different authors during the focal period. For scikit-image, 107 unique identities remained from 138 different authors over the whole previous period, and 44 unique identities remained from 47 different authors who authored commits during the focal period.

From analyzing the tightly coupled code contributions, I identified two contributors who made tightly coupled code contributions in GNU grep during the focal period. I also found 25 unique authors for IPython and 11 unique authors for scikit-image (see Figure 7). Next, I inspected the contribution patterns for those contributors who made tightly coupled code contributions during the focal period and examined their overall previous contributions during the whole previous period.

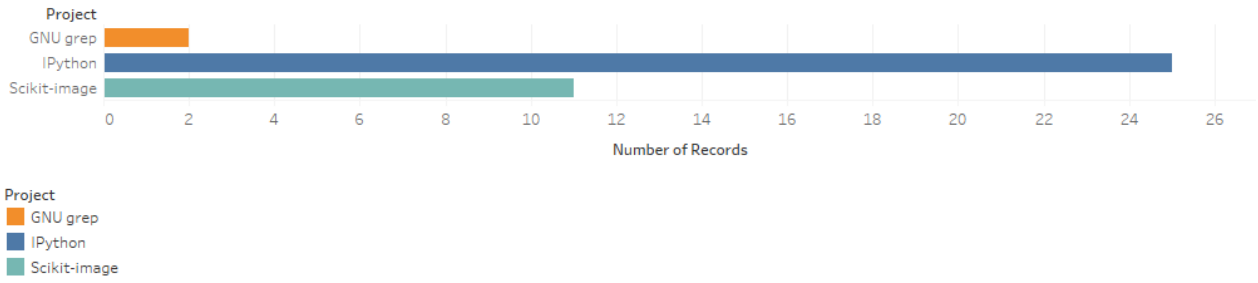


Figure 7. Number of Contributors who Made Tightly Coupled Code Contributions during the Focal Period According to the Project

5.2.1 Core or Peripheral Contributors

Researchers have tended to use arbitrary thresholds to define core or peripheral contributors (Barcomb et al., 2019). For instance, Lee and Carver (2017) empirically chose two percent as the cutoff to identify core contributors and found that the total code contribution percentage dropped significantly under this level. In another study, Setia, Rajagopalan, Sambamurthy, and Calantone (2010) defined peripheral contributors as contributors who made between zero and 12 percent of the total code contributions in terms of lines of code (Setia, Rajagopalan, Sambamurthy, & Calantone, 2010). In their study, Valiev et al. (2018) set the threshold to 90 percent to identify the number of core contributors who performed most of the work (Valiev et al., 2018). It would not be appropriate to apply the cutoffs that previous studies used here since each OSS project had a different number of contributors who engaged in various activities. Moreover, the projects also varied in size when they began and the degree to which corporations participated in them.

A cluster analysis can serve as an exploratory data-analysis method to partition data elements into groups that exhibit relatively homogeneity in themselves and heterogeneity between each other. Such an analysis maximizes the degree of similarity in each cluster while maximizing the dissimilarities between groups. In a cluster analysis, one has no prior knowledge about which data elements belong to which clusters (Burns & Burns, 2009). In the community-based OSS development context, contributors can freely join and leave a project and can self-assign tasks. Accordingly, a cluster analysis represents an appropriate method to segment contributors into two clusters.

For each project, I conducted a cluster analysis of the number of commits to segment contributors into two distinct groups: core contributors and peripheral contributors. Thus, I set the number of clusters to two. The cluster analysis resulted in two significantly different groups of contributors regarding the number of commits. I also conducted a cluster analysis over the whole previous period and during the focal period for each project. As such, I could compare a contributor's previous status over the whole previous period and the status during the focal period.

For GNU grep, seven contributors in total made commits during the whole previous period (from 3 November, 1998, to 11 February, 2009). Looking at the commit logs, I found that, in the early days, grep maintainers made commits to the main source code repository on behalf of patch authors. For instance, the commit logs described information about patch authors as follows: "Patch from XXXX[8] , for recursive" (committed on 28 January, 1999), "Mostly written by XXXX" (committed on 15 February, 1999), "Patch from XXXX for mbs" (committed on 19 February, 2001), and "Speed up patch from XXXX" (committed on 12 June, 2003). That is, those seven contributors were all committers. Accordingly, I considered all seven contributors core contributors. Figure 8 presents the number of commits that these seven core contributors made over the whole previous period.

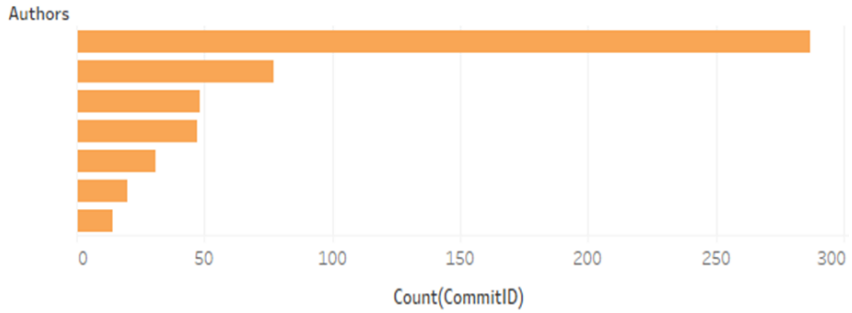


Figure 8. Number of Commits by Seven Contributors who Made Commits in GNU grep over the Whole Previous Period (from 3 November, 1998, to 11 February, 2009)[9]

In comparing the whole previous period and the focal period, I found that two contributors who made tightly coupled code contributions during the focal period made no code contributions over the whole previous period (see Figure 8). In contrast, those two contributors made 52 and 44 percent of all commits during the focal period (from 11 February, 2009, to 23 March, 2010) (see Figure 9). For these contributors, we need to investigate the conditions under which they could make tightly coupled code contributors

[8] I do now show a patch author's name.
[9] Note that the X-axis indicates the number of commits made during the whole previous period. The Y-axis presents the authors' names and emails. I do not show authors' names and emails.

despite no previous code contributions to the GNU grep project. Accordingly, I examine them in greater detail in Section 5.3.

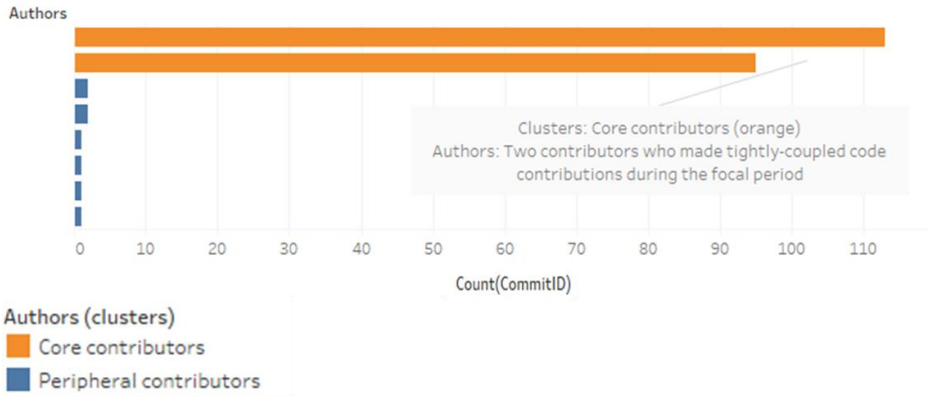


Figure 9. Number of Commits by Contributors who Made Commits in GNU grep during the Focal Period (from 11 February, 2009 to 23 March, 2010)[10]

In IPython, 144 contributors made commits over the whole previous period (from 6 July, 2005, to 30 June, 2012). The cluster analysis in this case partitioned those contributors into two distinct groups based on how many commits they made: core contributors and peripheral contributors (see Figure 10). Due to space limitations, I do not show all 144 contributors in Figure 10. I summarize the cluster analysis in Table 6.

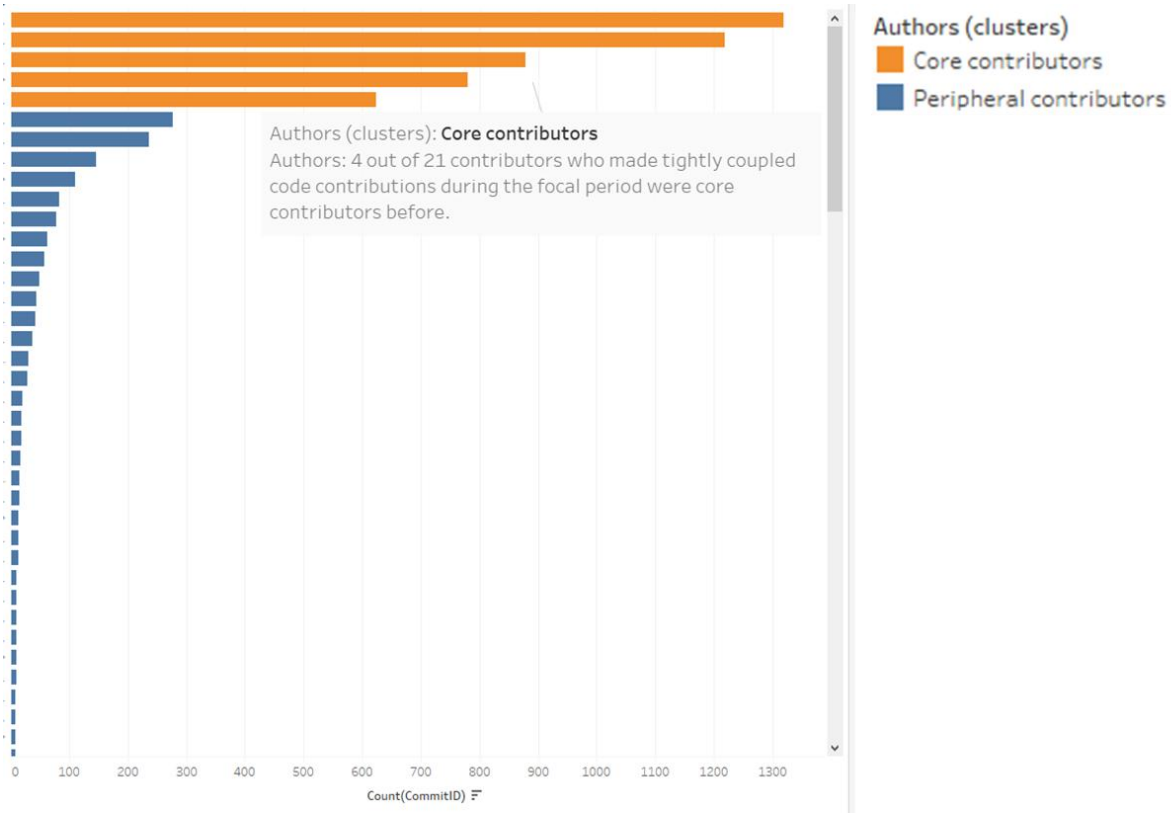


Figure 10. Results of the Cluster Analysis of 144 Contributors who Made Commits over the Whole Previous Period (from 6 July, 2005 to 30 June, 2012) in IPython[11]

[10] Legend: orange depicts core contributors, while blue depicts peripheral contributors. I do not show authors' names and emails.
[11] Legend: orange depicts core contributors, while blue depicts peripheral contributors.
Due to space limitations, I do not show all 139 peripheral contributors (in blue). I also do not show authors' names and emails.

Table 6. Summary of Cluster Memberships of Contributors who Made Commits in IPython over the Whole Previous Period

Clusters	Number of authors	Centers Sum of count (CommitID)
Core (orange)	5	964.0
Peripheral (blue)	139	12.9

Based on those two clusters, I characterized OSS contributors who made tightly coupled code contributions during the focal period as core or peripheral contributors during the whole previous period. The cluster analysis revealed that four out of five core contributors during the whole previous period did make tightly coupled code contributions during the focal period in IPython. That is, four out of 25 contributors who made tightly coupled code contributions during the focal period were core contributors over the whole previous period. Regarding those four core contributors over the whole previous period, they unsurprisingly authored pull requests, which added dependencies in the codebase during the focal period. I found that these contributors had made most of the commits earlier, which suggests that they were familiar with how the system worked in the codebase. From further inspecting the remaining 21 contributors, I found that eight were peripheral and that 13 made no commits over the whole previous period.

I also conducted a cluster analysis of contributors who made commits in IPython during the focal period in this case (from 30 June, 2012, to 9 August, 2013). The cluster analysis partitioned a total of 150 contributors into two distinct groups see Figure 11). I summarize the cluster analysis in Table 7.

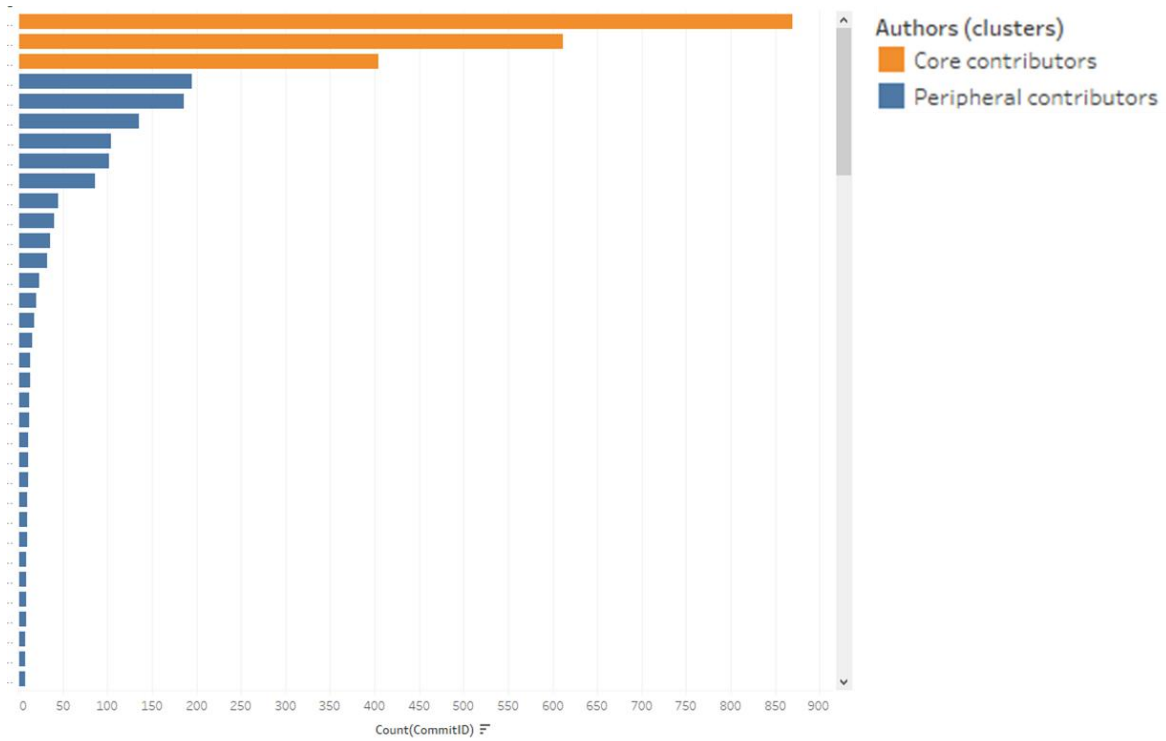


Figure 11. Results of the Cluster Analysis of 150 Contributors who Made Commits in IPython during the Focal Period (from 30 June, 2012, to 9 August, 2013)[12]

The X-axis presents the number of commits that each contributor made over the whole previous period. The Y-axis presents all contributors who made commits over the whole previous period.

[12] Legend: orange depicts core contributors, while blue shows peripheral contributors.
Due to space limitations, I do now show all 147 peripheral contributors (in blue). I do not show authors' names and emails.
The X-axis presents the number of commits that each contributor made during the focal period. The Y-axis presents all contributors who made commits during the focal period.

Table 7. Summary of Cluster Memberships of Authors who Made Commits in IPython during the Focal Period

Clusters	Number of authors	Centers Sum of count (CommitID)
Core	3	628.33
Peripheral	147	9.97

In scikit-image, 107 contributors made commits over the whole previous period (from 22 August, 2009, to 28 May, 2014). The cluster analysis that I conducted in this case segmented those 107 contributors into two distinct groups regarding the number of commits (see Figure 12). I found that two out of 11 contributors who made tightly coupled code contributions during the focal period were core contributors over the whole previous period. Four out of the 11 contributors made no commits over the whole previous period; however, each authored at least one pull request that added dependencies during the focal period.

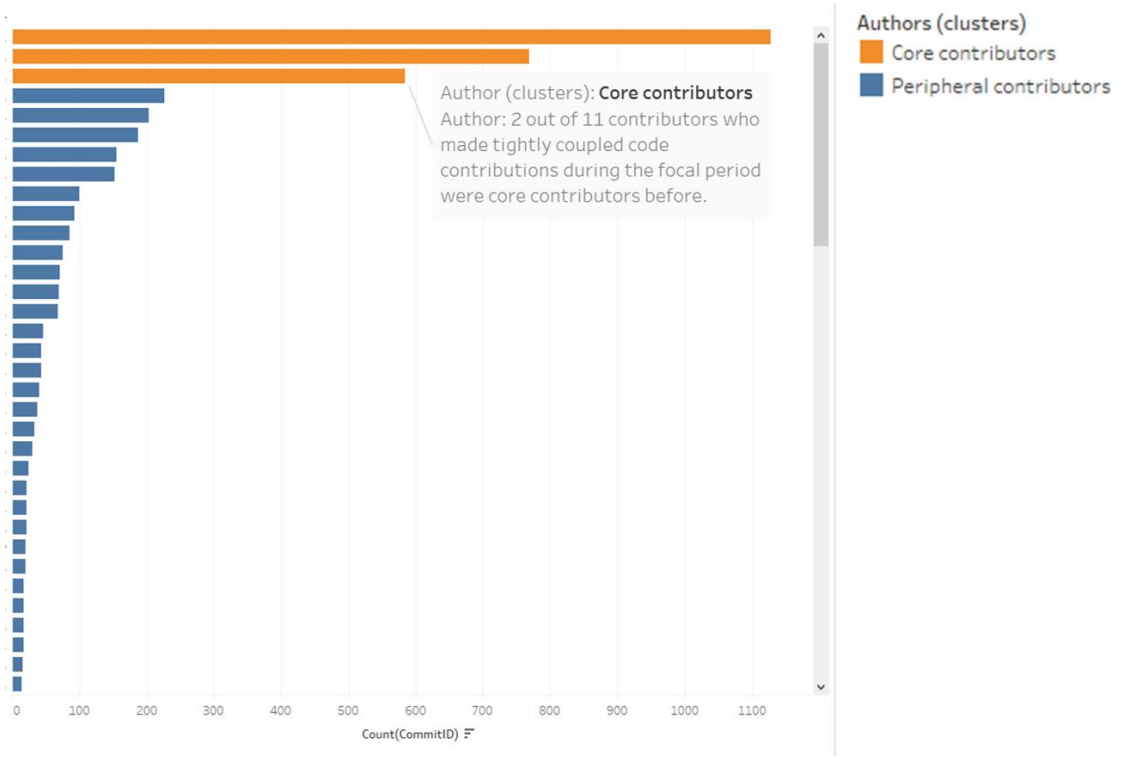


Figure 12. Results of the Cluster Analysis of 107 Contributors who Made Commits in scikit-image Over the whole Previous Period (from 22 August, 2009, to 28 May, 2014)[13]

Table 8. Summary of Cluster Memberships of Authors who Made Commits in scikit-image over the Whole Previous Period

Clusters	Number of authors	Centers Sum of count (CommitID)
Core	3	827.0
Peripheral	104	22.54

[13] Legend: orange depicts core contributors, while blue contributes peripheral contributors.
Due to space limitations, I do not show all 104 peripheral contributors in blue. I do not show authors' names and emails.
The X-axis presents the number of commits that each contributor made over the whole previous period. The Y-axis presents all contributors who made commits over the whole previous period.

I also conducted a cluster analysis of the 44 contributors who made commits during the focal period (from 28 May, 2014 to 4 March, 2015). The analysis partitioned four contributors into a core group (see Figure 13), and I found all four to have made tightly coupled code contributions during the focal period. Seven out of 11 contributors who made tightly coupled code contributions were in the peripheral group during the focal period. Table 9 summarizes the result from the cluster analysis for the number of commits that the contributors made during the focal period.

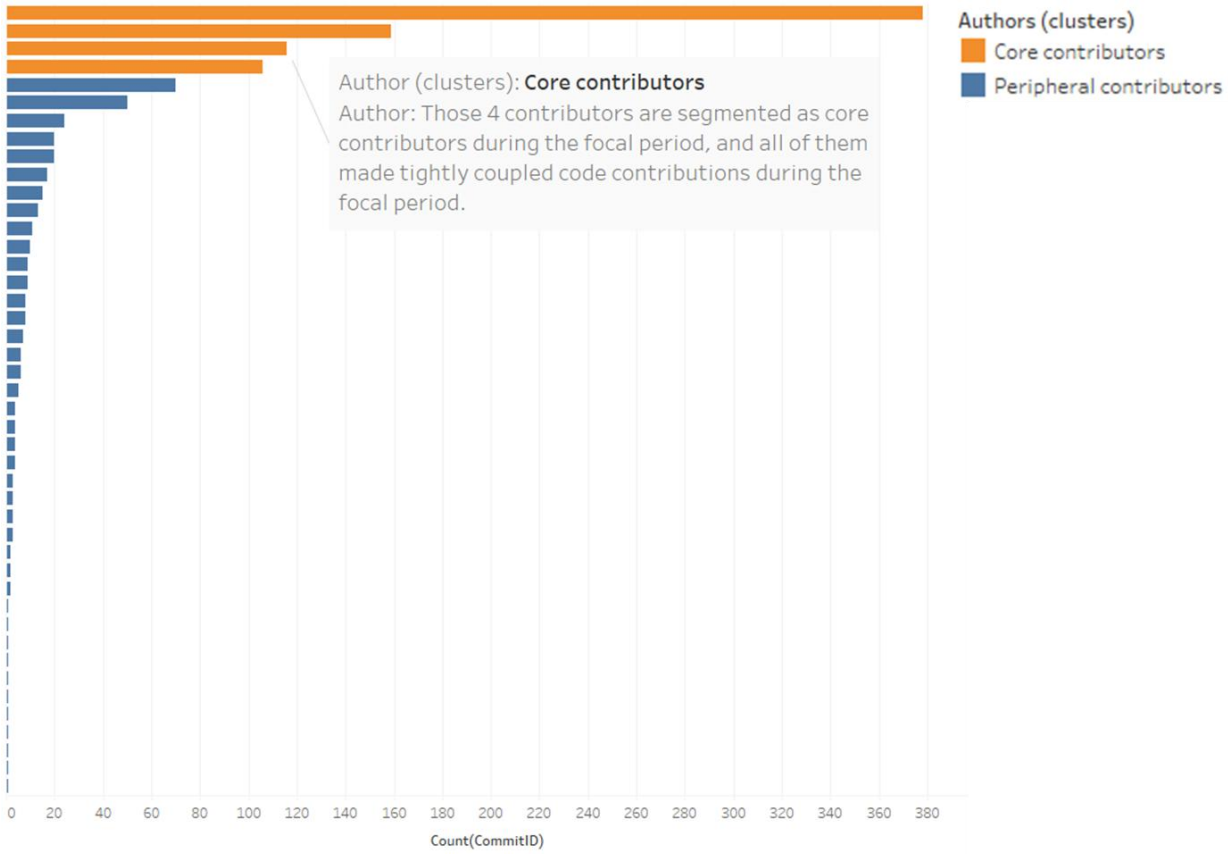


Figure 13. Results of the Cluster Analysis of a Total of 44 Contributors who Made Commits in scikit-image during the Focal Period (from 28 May, 2014 to 4 March, 2015)[14]

Table 9. Summary of Cluster Memberships of Authors who Made Commits in scikit-image during the Focal Period

Clusters	Number of authors	Centers Sum of count (CommitID)
Core	4	189.75
Peripheral	40	8.83

5.2.2 Habitual or Episodic Contributors

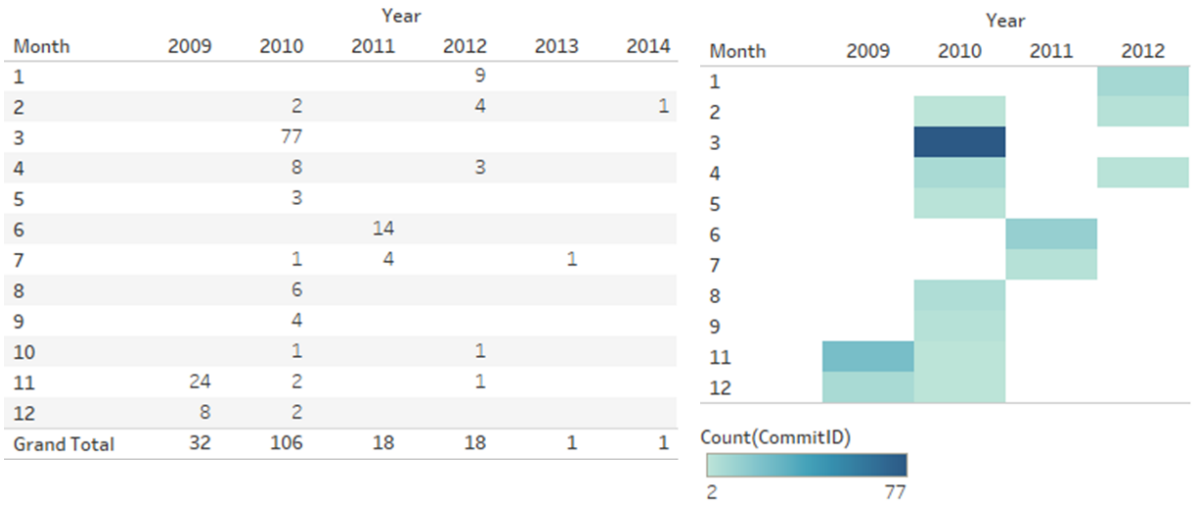
Next, I characterized OSS contributors who made tightly coupled code contributions during the focal period as either habitual or episodic contributors. I defined habitual contributors as individuals who made two or more commits per month for six consecutive months (Barcomb et al., 2018) since the episodic volunteering literature has used this duration and frequency more than any other dimension (Hyde et al., 2014). I defined contributors who did not meet this definition as episodic contributors.

[14] Note that I do not show authors' names and emails.

I visually inspected the contribution duration for contributors who made tightly coupled code contributions during the focal period by visualizing the data in a heat map. In this process, I analyzed all commit logs recorded since the sampled projects began. In the heat map, each cell encodes a single quantitative value with color (Munzner, 2014), and I use a sequential color scheme to map color and continuous quantitative values (Harrower & Brewer, 2003). That is, I show low data values in light colors and high data values in dark colors. I used Tableau software[15] to visualize the data.

The table (on the left) in Figure 14 shows the exact number of commits that a contributor made in one month. The heat map (on the right) in Figure 14 visualizes the number of commits with a sequential color scheme. A darker color signifies more commits. Based on how I define habitual contributors, if a contributor made only one commit during a single month, I did not color-code it in a heat map. I set the minimum number of commits to two to determine a contributor as a habitual one. If a contributor had at least one habitual pattern in six consecutive months over the whole previous period, I considered the contributor habitual even when the contributor had an episodic contribution pattern for the remaining months throughout the previous period. Contributors who had remained with the project for a long time displayed such a pattern. I show an example in the heat map for IPython in Figure 15. The contributor had habitual patterns from January, 2011, to November, 2012, but made no commits in December, 2012. However, the contributor had shown habitual patterns. Accordingly, I considered the contributor a habitual contributor over the whole previous period (from 6 July, 2005, to 30 June, 2012). I present example contributors who made tightly coupled code contributions during the focal period in each of the three projects below.

A GNU grep contributor in Figure 14 made the largest number of commits in March, 2010. I color-coded the cell for this contributor with the darkest color that matched the largest number of commits (i.e., 77). In contrast, other cells in white indicate that this contributor made no commits or one commit per month, such as from January, 2009 to October, 2009. The visual analysis with a heat map shows that this contributor made no previous code contributions before the focal period. However, the contributor started making contributions in November, 2009, and made tightly coupled code contributions during the focal period in GNU grep. After the focal period, the contributor made episodic contributions (i.e., did not make two or more commits per month for six consecutive months). However, the contributor did not make any additional contributions from March, 2014. I also visualized another grep contributor who also made tightly coupled code contributions during the focal period using a heat map (see Figure 18). That contributor's code-contribution pattern shows that the individual did not make code contributions before; however, the individual started making code contributions during the focal period.



	Year					
Month	2009	2010	2011	2012	2013	2014
1				9		
2		2		4		1
3		77				
4		8		3		
5		3				
6			14			
7		1	4		1	
8		6				
9		4				
10		1		1		
11	24	2		1		
12	8	2				
Grand Total	32	106	18	18	1	1

Figure 14. GNU grep: An Example of a Visual Analysis with a Heat Map (right) Matching the Data in the Cells (left) Based on How I Define a Habitual Contributor[16]

[15] See https://www.tableau.com/

[16] Note that I also show the exact number of commits per month in the table (left). The heat map (right) visualizes the quantitative value in each cell of the table based on how I define a habitual contributor (i.e., an individual who makes at least two commits per month).

Figure 15 (at the top) shows the exact number of code contributions from an IPython contributor who made tightly coupled code contributions during the focal period (from 30 June, 2012, to 9 August, 2013). As the cell for July, 2013, has the darkest color in the heat map, this IPython contributor made the largest number of commits in July, 2013. Furthermore, I color-coded every cell for six consecutive months before the focal period since this contributor's first contribution. That is, this contributor made habitual contributions over the whole previous period (from July 6th, 2005 to June 30th, 2012) and during the focal period (from 30 June, 2012, to 9 August, 2013).

	Year								
Month	2010	2011	2012	2013	2014	2015	2016	2017	2018
1		26	28	75	111	148	7	1	2
2		58	27	29	178	56	8	2	1
3	1	30	54	81	79	110	8	3	
4		61	75	118	92	102	12	1	1
5		61	97	112	48	12	2	1	1
6		124	172	82	51	2	7	4	
7		65	38	236	63	5	3		
8	1	62	32	99	31	9	8	5	
9		31	10	95	45	7			
10	62	141	9	135	102	14			
11	12	63	2	75	92	4	8		
12	1	67		57	97	2	1		
Grand Total	77	789	544	1,194	989	471	64	17	5

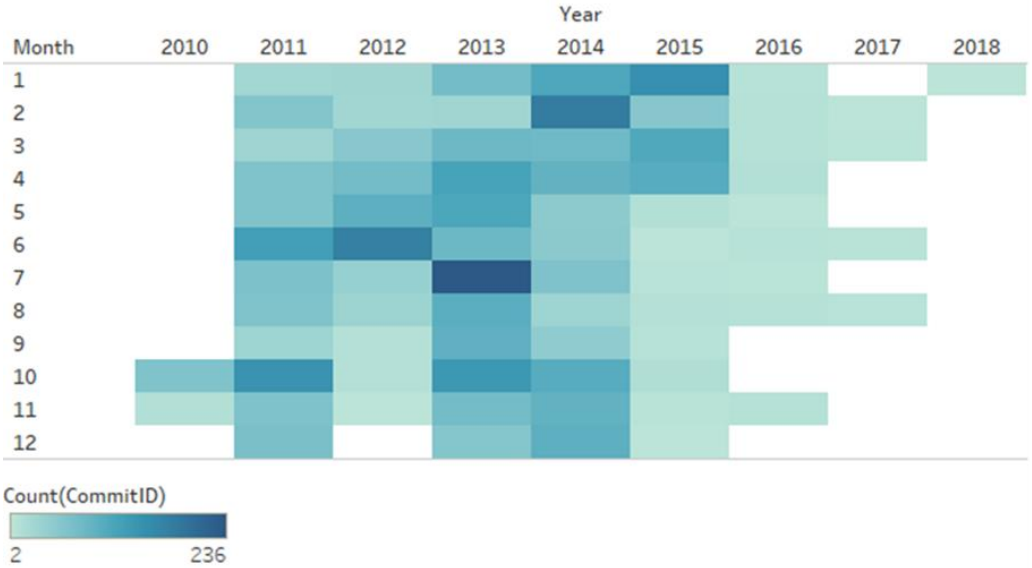


Figure 15. IPython: An Example of a Visual Analysis with a Heat Map (at the bottom) Matching the Data in the Cells (at the top) based on the Definition of a Habitual Contributor[17]

In Appendix A, I visualize the overall contribution duration for all 25 IPython contributors who made tightly coupled code contributions during the focal period. I found that 13 out of 25 contributors made their first code contributions to IPython and that those contributions increased software coupling during the focal period. I investigate these contributors in more depth in Section 5.3.

I also visualized the overall contributions for the 11 contributors who made tightly coupled code contributions during the focal period in scikit-image with heat maps. Figure 16 shows a heat map that visualizes the contributions from a contributor who authored pull requests that increased software coupling during the focal period in scikit-image. For example, the contributor whose code contributions I show in

[17] Note that the figure visualizes the overall contribution duration for a contributor who made tightly coupled code contributions in IPython during the focal period (30 June, 2012 to 9 August, 2013). I show the commits that this contributor made up to June, 2018.

Figure 16 constituted an episodic contributor over the whole previous period but became a habitual contributor during the focal period (from 28 May, 2014, to 4 March, 2015). I visually present the overall contributions from all contributors who made tightly coupled code contributions during the focal period in Appendix A.

	Year				
Month	2012	2014	2015	2016	2017
1			9	11	
2		20	18	1	9
3			19	5	
4				3	
5			8		
6			22		
7		25	36		
8	14	15			
9		44	31		
10	4	179	10		
11		7		1	
12	32	81	13	7	
Grand Total	50	371	166	28	9

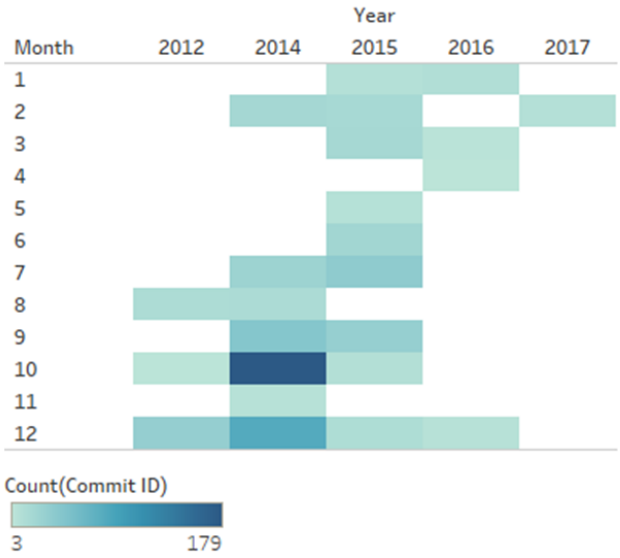


Figure 16. scikit-image: An Example of a Visual Analysis with a Heat Map (Right) Matching the Data in the Cells (Left) Based on the Definition of a Habitual Contributor[18]

5.2.3 Answering RQ1

Characterizing OSS contributors who made tightly coupled code contributions during the focal period into the core, periphery, and habitual, episodic answers RQ1. Figure 17 summarizes how I characterized those contributors over the whole previous period and the focal period from the core-periphery and habitual-episodic perspectives.

In analyzing OSS contributors who made tightly coupled code contributions during the focal period, I found that such contributors had different types of contribution patterns. Interestingly, half of such contributors (19 out of 38) made no previous code contributions to the sampled projects; however, they authored commits that increased software coupling during the focal period. Such contributors contributed code in an episodic and peripheral manner as the last column in Figure 17 shows, which contradicts what the literature has previously proposed (e.g., Colfer & Baldwin, 2016; Ducheneaut, 2005; Ye & Kishida, 2003).

In GNU grep, two contributors who made tightly coupled code contributions during the focal period made no commits to GNU grep before. With regard to IPython, 13 out of 25 contributors who made tightly coupled code contributions during the focal period made no commits before. In scikit-image, four out of 11 contributors who made tightly coupled code contributions during the focal period made no commits before.

One can understand previously peripheral contributors who become core contributors when one views them through the onion model. According to the onion model, contributors progress from the periphery to the core (Ducheneaut, 2005; Ye & Kishida, 2003). In addition, given OSS development's voluntary nature, contributors may spend more or less time depending on how much spare time they have. Accordingly, previously core or habitual contributors may become peripheral or episodic during the focal period. In another case, contributors to OSS projects can decide to stop contributing at any time (Yu, Benlian, & Hess, 2012).

[18] Note that the figure visualizes the overall contribution duration for a contributor who made tightly coupled code contributions in scikit-image during the focal period (from 28 May, 2014, to 4 March, 2015).
Although I set the minimum value for filtering the number of commits to two, this contributor made at least three commits every month. Thus, the color legend for the heat map starts at 3.

Figure 17. The Overall Contributions of OSS Contributors who Made Tightly Coupled Code Contributions during the Focal Period in the Three Projects from the Core-periphery and Habitual-episodic Perspectives

In contrast, the presence of contributors who made no previous code contributions to the sampled projects but made tightly coupled code contributions calls for a more in-depth investigation. The existing onion model and two aforementioned lenses do not provide explanations about the conditions under which these contributors make tightly coupled code contributions as their first code contributions to the projects. To answer the second research question, I investigate this unexpected finding further in Section 5.3.

5.3 Phase 3: Investigation of Contributors

In phase 3, I collected data from multiple sources. Specifically, I collected archival documents from project websites (e.g., a roadmap document) and data pertaining to a project's GitHub page (e.g., comments on pull requests). I also looked for the GitHub user pages of such contributors who made tightly coupled code contributions despite making no previous code contributions to the sampled projects. The GitHub user page shows a user's contribution activity in the projects on GitHub. In addition, some contributors provide a link to associated personal or professional websites on their GitHub user pages. On those websites, I found software they used or had built. I also collected comments on pull requests or patches that such contributors authored. In some cases, their pull requests referenced related email messages. I saved the GitHub user pages, any associated websites, and the related email messages as PDF files.

I also conducted semi-structured artifact-based interviews. I crafted the general interview protocol based on the literature on software ecosystems. The protocol addressed interviews' participation in OSS projects and how they made tightly coupled code contributions. For instance, in the interview, I asked interviewees how they came to participate in the sample project, how long they had participated in the sampled project for, what other OSS projects they participated in, and whether their institution paid for their code

contribution to the sampled project. The interview also included questions about how the interviewees worked on the pull request (or the patch) that increased software coupling. I identified potential interviewees based on contributors who made tightly coupled code contributions during the focal period. Contributors who did so despite no previous code contributions to the sampled projects can provide information about the conditions in which they did so. Moreover, contributors, either core or habitual, can provide a more comprehensive picture of the sampled projects. For instance, the project's founder can account for whether any corporate participation took place during the focal period. For those reasons, I cast a wider net to recruit interviewees. After receiving responses to my recruitment emails, I conducted interviews. I tailored the general interview protocol to each interviewee. I summarize the interviewees in Table 10. I conducted two interviews using video calls. I conducted two other interviews in person and three over email. Except for the email interviews, I transcribed the interviews verbatim with a foot pedal-controlled audio player.

Table 10. Interviewees Who Made Tightly Coupled Code Contributions during the Focal Period

Projects	Over the whole previous period	During the focal period	Number of interviewees
GNU grep	No contribution	Core, episodic	1
IPython	Peripheral, habitual	Core, habitual	1
	Core, habitual	Core, habitual	1
scikit-image	No contribution	Peripheral, episodic	1
	Peripheral, episodic	Core, habitual	1
	Core, habitual	Peripheral, habitual	1

† For GNU grep, I also interviewed the previous grep 2.5.4 maintainer despite the fact that this person did not make tightly coupled code contributions during the focal period because this person communicated with grep 2.6 maintainers.

I organized the data in a tabular display. The unit of analysis was each contributor who made tightly coupled code contributions during the focal period despite no previous code contribution to the sampled projects. For each contributor, I added columns about their contribution patterns during the whole previous period and the focal period from the core-periphery and habitual-episodic perspectives. The next column lists the OSS projects in which the contributor participated. I also created another column to record any remarks on a contributor's previous experience with OSS projects. For instance, one comment stated that one individual maintained two OSS projects that used IPython. The next column accounts for how they worked on tightly coupled code contributions. I analyzed the interviews, comments on pull requests, and messages exchanged via mailing lists while considering how they worked on tightly coupled code contributions (Lindberg et al., 2016; Moon & Howison, 2018). For instance, in that column, I noted that a contributor made the code contribution during a coding sprint or that a participant joined in a rich discussion about managing technical dependencies via pull requests on GitHub.

We know little about how contributors make tightly coupled code contributions when they have made no previous code contributions. Accordingly, I adopted an inductive approach to analyze the data. As I populated and organized the table, I coded the data (Strauss & Corbin, 1990) and looked for conditions that facilitated how contributors could make tightly coupled code contributions when they had made no previous code contributions to the sampled projects. For instance, I coded OSS projects in which those contributors participated as previous experience with related OSS projects in the presence of technical dependencies between those projects and the sampled project. If contributors made tightly coupled code contributions to the sampled project due to other OSS projects using the sampled project, I coded such instances as projects using the sampled project. I also found genuine newcomers to the IPython project. During this process, I attempted to maximize the degree of similarity in identical conditions and the degree of dissimilarity among different conditions. Next, I used this open coding procedure by making connections between those conditions during the axial coding step. I related those conditions across three projects by making connections among them at a higher level. During this step, I merged similar conditions into a single condition. I show the results in Table 11.

5.3.1 GNU grep

For GNU grep, I collected public archival data such as release notes and email messages exchanged via the bug-grep mailing list. In an email message, the grep 2.5.4 maintainer introduced two new grep 2.6 maintainers and discussed how they had been working for grep 2.6 (focal release). Those two grep 2.6 maintainers made tightly coupled code contributions during the focal period (see Figures 18 and 19).

Accordingly, I sent recruitment emails to the grep 2.5.4 maintainer and to two grep 2.6 maintainers to better understand the conditions under which those two contributors became involved in GNU grep and made their tightly coupled code contributions despite making no previous code contributions. I interviewed the grep 2.5.4 maintainer via Skype and one of two grep 2.6 maintainers via email. During this process, the grep 2.5.4 maintainer noted the following pertaining to the two new maintainers:

> *They already had made that contribution to other [GNU] projects, and they were very familiar with how to automate build tools, how to rely on GNU libraries.… We asked some questions, and they came in.… So, we invited them to grep.*

The interview with one of two grep 2.6 maintainers also revealed that this person had been participating in several GNU projects, such as gnulib, Autoconf, and Automake. The GNU grep project invited these experienced GNU developers as new grep maintainers given their experience with other GNU projects. In addition, they were familiar with the conversion process to modern infrastructure and Gnulibification, which refers to switching from built-in libraries to GNU libraries. The previous grep maintainer had planned such tasks in 2005 (as shared via the bug-grep mailing list) but no one had yet conducted them. As those two individuals accepted invitations, they accomplished necessary but challenging tasks, such as Gnulibification (Moon & Howison, 2018).

In the interviews, I also found that two grep 2.6 maintainers worked for the same company during the focal period and had exchanged numerous private emails; however, they did not work at the same site. Moreover, they worked for GNU grep during the focal period as volunteers, not as developers whom their company paid for. In other words, in the case of GNU grep, newly invited maintainers with much experience in other GNU projects could possibly make tightly coupled code contributions even if they made no code contributions to GNU grep before. They transferred skills and familiarity with regard to creating code contributions during other related GNU projects to GNU grep during the focal period.

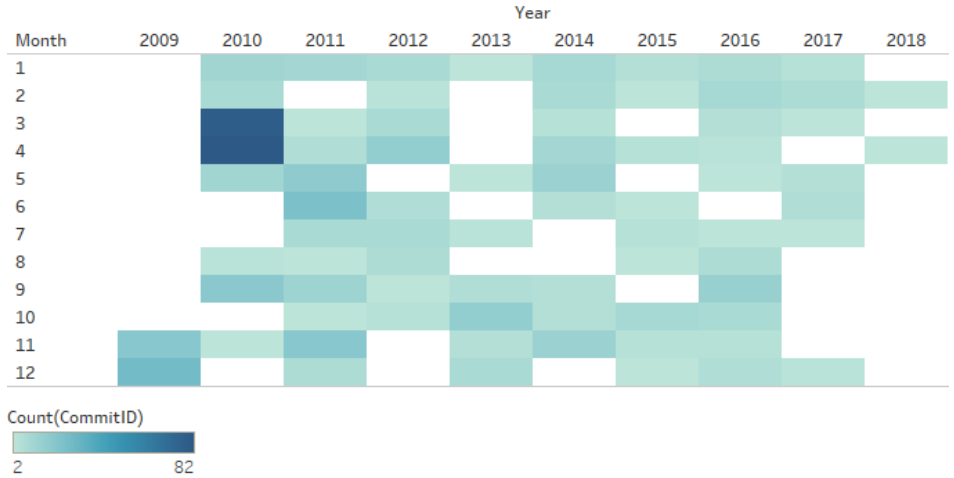


Figure 18. One of Two New Maintainers Became Episodic during the Focal Period (from 11 February, 2009, to 23 March, 2010)[19]

[19] Note that I do not show the author's name and email. I show commits up to June, 2018.

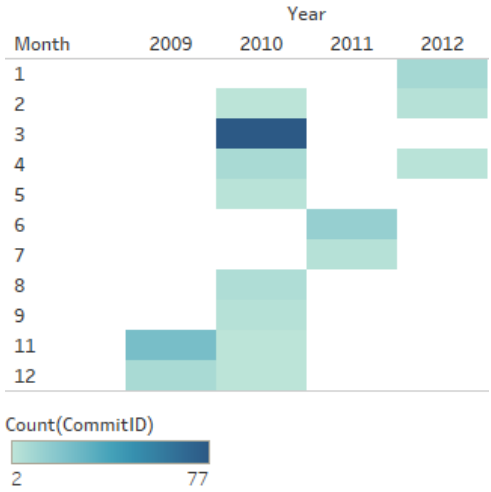


Figure 19. Another New Maintainer Remained Episodic in GNU grep during the Focal Period (from 11 February, 2009, to 23 March, 2010)[20]

5.3.2 IPython

To investigate under what conditions contributors made tightly coupled code contributions to IPython during the focal period, I collected data from multiple sources: archival documents such as release notes, a roadmap document, weekly developer meetings on Google Hangout, meeting notes, comments on pull requests that increased software coupling, and a grant proposal. I also interviewed two IPython contributors in person at the SciPy 2016 conference as I received responses from recruitment emails. The interviewee mentioned that they received grant funding to support core developers' time and resource over two years that began from the focal period. Before the grant funding, everybody participated as a volunteer.

A contributor without previous code contributions made one pull request during the focal period (see Figure 20). The pull request had one commit and added a dependency in the codebase. I examined the commit log and discussions pertaining to that pull request. The pull request concerned another tool, Sympy[21]. Sympy is a Python library for symbolic calculations in Python that one can also use interactively in the IPython notebook[22]. In the interviews, IPython contributors stated that IPython embedded and called other tools such as Sympy.

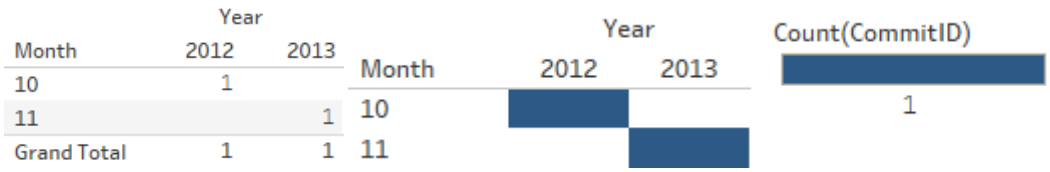


Figure 20. The Overall Contribution of the Contributor who Made no Contributions to IPython before but Added Dependencies during the Focal Period (30 June, 2012, to 9 August, 2013)[23]

The contributor's GitHub user page showed that the contributor had started making code contributions to Sympy in 2011. In this pull request, the Sympy contributor came to the IPython project and made changes in the existing embedded function to move it to Sympy. This type of change in the codebase required changes to both IPython and Sympy; thus, that contributor made two pull requests in IPython and Sympy. Contributors from both IPython and Sympy participated in pull requests as the changes had to reflect how IPython and Sympy work.

[20] Note that I do not show the author's name and email
[21] See https://www.sympy.org/en/index.html
[22] See https://ipython-books.github.io/151-diving-into-symbolic-computing-with-sympy/
[23] Note that I show the exact number of commits in the table on the left. As the contributor did not contribute habitually, I removed the filtering that sets the minimum number of commits in a month to two.
I do not show the author's name and email.

Although the contributor made their first contribution to IPython, the contributor had been participating in Sympy, which IPython used. Accordingly, the contributor could use their transferrable knowledge and skills across related OSS projects and to contribute code that added dependencies in the codebase by making IPython call the function in Sympy.

Another contributor without previous code contributions to IPython made a pull request that increased software coupling and became peripheral and habitual during the focal period (from 30 June, 2012, to 9 August, 2013) (see Figure 21). The pull request this contributor brought concerned QIIME, another tool that uses IPython[24]. The contributor developed the QIIME project and suggested the addition of a new class to simplify interactions between IPython and QIIME. This pull request constituted their first one in the IPython project. The contributor with other contributors what they should name the new class, whether they should define variables as public or private, and how they should design the functions. The contributor went through the comments and finally had the pull request merged.

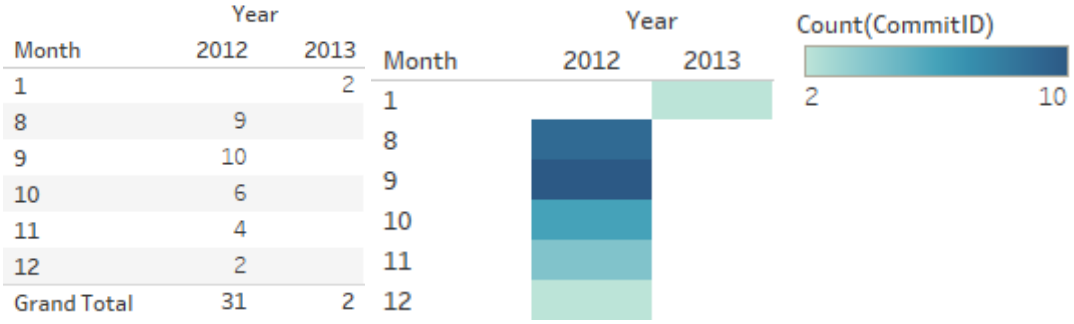


Figure 21. The Overall Contribution of the Contributor who Made no Code Contributions to IPython before but Added Dependencies during the Focal Period (30 June, 2012, to 9 August, 2013)[25]

Another contributor made no previous code contributions to IPython. However, this contributor authored two pull requests that increased software coupling during the focal period. I visualize the contributor's overall code contributions in Figure 22. The contributor's GitHub user page shows that the contributor had reported several issues to other OSS projects, such as PyZMQ[26], csvkit[27], and pandas[28] starting in 2011. That is, the contributor had actively used other related OSS projects. Further, such projects as PyZMQ and pandas relate closely to IPython. Pandas, a Python data analysis library, provides an environment for data analysis in Python when combined with IPython. PyZMQ is a type of Python binding for ZeroMQ[29]—a messaging library that distributed or concurrent applications typically use. As people use IPython for parallel computing[30], core contributors had to rewrite the ZeroMQ architecture[31]. From examining the contributor pages on GitHub, I found that IPython core contributors made most of the commits in PyZMQ[32]. The contributor whose contributions I show in Figure 22 worked on the problems with the parallel code in IPython and had pull requests merged.

[24] See http://qiime.org/
[25] Note that I show the exact count of commits in the table on the left. I do not show the author's name and email.
[26] See https://github.com/zeromq/pyzmq
[27] See https://csvkit.readthedocs.io/en/latest/
[28] See https://pandas.pydata.org/
[29] See http://zeromq.org/
[30] See https://minrk.github.io/scipy-tutorial-2011/
[31] See https://ipython.readthedocs.io/en/stable/whatsnew/version0.11.html#parallel-011
[32] See https://github.com/zeromq/pyzmq/graphs/contributors

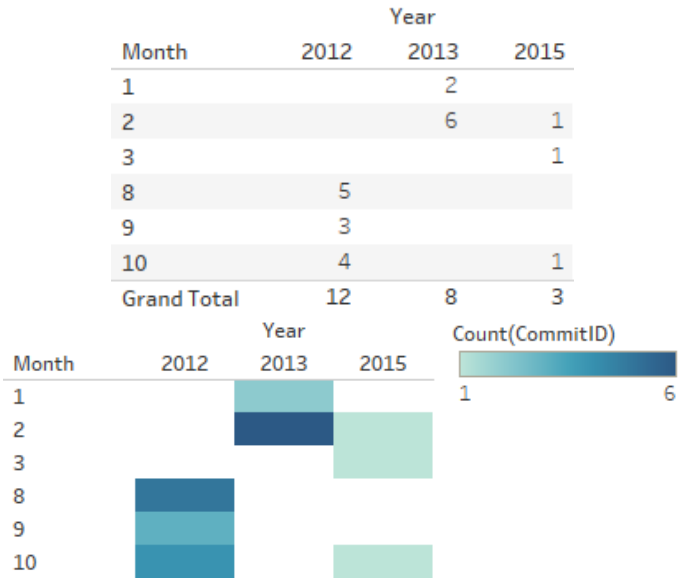


Figure 22. The Overall Contributions of the Contributor who Made no Code Contributions to IPython before but Added Dependencies in the Codebase during the Focal Period (30 June, 2012, to 9 August, 2013)[33]

Another contributor who made tightly coupled code contributions without previous code contributions to the IPython project commented on the issue on GitHub. The contributor asked whether this issue would be a good starting point for their first code contribution. The existing IPython contributor encouraged the contributor to start by fixing that issue. The new contributor opened a pull request, received comments on the code, and fixed the code's original version. As the new contributor worked on the pull request, the existing IPython contributors encouraged the new contributor to fix the code again, and the new contributor expressed the following: "No hard feelings at all! This is the first time I'm playing with the inside of IPython so I'm glad there is someone who can guide me.".

That is, I found that the contributor followed the onion model in that they joined the project and made their first contribution under the existing contributors' guidance and comments. I show this contributor's overall code contributions in Figure 23.

Figure 23. The Overall Contribution of a Contributor who Made no Code Contributions in IPython before but Made Tightly Coupled Code Contributions during the Focal Period (30 June, 2012, to 9 August, 2013)[34]

5.3.3 scikit-image

For scikit-image, I collected data from multiple sources—archival documents on the GitHub Wiki, comments on pull requests that increased the degree of software coupling, messages exchanged via the mailing list, and a research paper by scikit-image contributors. I received three responses from the recruitment emails. I conducted semi-structured artifact-based interviews with one of the scikit-image project's core members[35] via Google Hangout and with two scikit-image contributors via email. The

[33] Note that I show the exact number of commits in the table on the left. As the contributor did not contribute habitually, I removed the filtering that sets the minimum number of commits per month to two.
Note that I do not show the author's name and email.
[34] Note that I show the exact number of commits in the table on the left. As the contributor did not contribute habitually, I removed the filtering that sets the minimum number of commits per month to two. I do not show the author's name and email.
[35] The scikit-image project maintains core members who have the authority to incorporate changes directly into the main development branch.

interview with one of the core members confirmed that they received no corporate participation in supporting contributor's time except the Google Summer of Code (GSoC).

From analyzing the comments on pull requests that increased the degree of software coupling, I found that one out of four contributors without previous code contributions brought in external code the contributor wrote for work because the contributor considered the code as possibly useful for scikit-image. The contributor posted a message that they wrote code to implement a subpixel image registration algorithm (Guizar-Sicairos, Thurman, & Fienup, 2008) using a phase correlation process from outside the scikit-image project. The contributor asked whether the code would suit scikit-image, and several scikit-image contributors stated that they valued the code. Thus, the contributor opened a pull request and the existing contributors commented on the code. As the contributor also wished to generalize the code to n-dimensions, the existing contributors further discussed how to modify the code for higher dimensional support. scikit-image contributors planned to extend higher dimensional support in the project's next release as they indicated in a research paper that they wrote. Because the code implemented an algorithm published in an original research paper (Guizar-Sicairos et al., 2008), the existing contributors emailed the original authors (from outside the scikit-image project) to request a code review and to request licensing permission. As the contributor also worked on the comments from the original authors, the contributors finally merged the pull request into the code repository's main branch.

The aforementioned contributor had substantial changes merged into the code repository as their first code contribution to scikit-image. The contributor's GitHub user page listed a software company that developed software for scientific computing, data processing, and visualization in Python. I also visited the company website, which showed that the contributor had been an engineer in that company. The contributor had domain knowledge and skills in an area related to scikit-image. Accordingly, the contributor could make substantial changes despite no previous code contributions to the scikit-image project (see Figure 24).

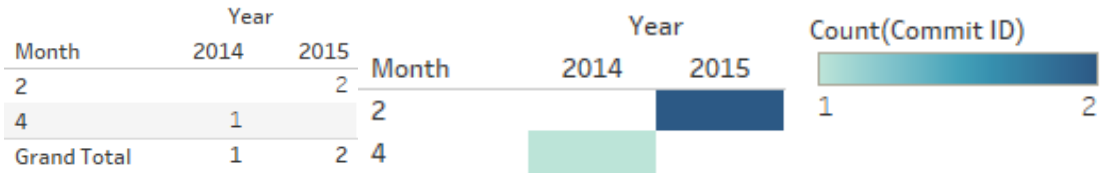


Figure 24. A Contributor Made their First Code Contribution to scikit-image during the Focal Period (from 28 May, 2014, to 4 March, 2015), and the Code Substantially Changed the Codebase[36]

Three out of four contributors without previous contributions worked on pull requests that increased software coupling while attending co-located events—in this case, coding sprints at SciPy 2014 conference (from 11 July, 2014, to 12 July, 2014) and the EuroSciPy 2014 conference (from 27 August to 30 August, 2014). I interviewed a contributor who had made no previous code contribution to the scikit-image project but who made two pull requests that increased software coupling during the focal period. While the heat map for this contributor showed their contributions by year and month (see Figure 25 on the right), I broke these contributions I show in Figure 25 down into the day level to obtain further insight. As Figure 26 shows, the contribution effort occurred primarily on the sprint day (11 July, 2014) and extended around the sprint event.

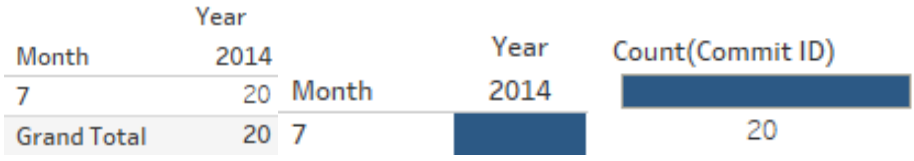


Figure 25. The Overall Contributions of the Contributor by Year and Month

[36] Note that I do not show the author's name and email.

Month	Day	Year 2014
7	11	7
	12	2
	13	5
	15	4
	16	2
Grand Total		20

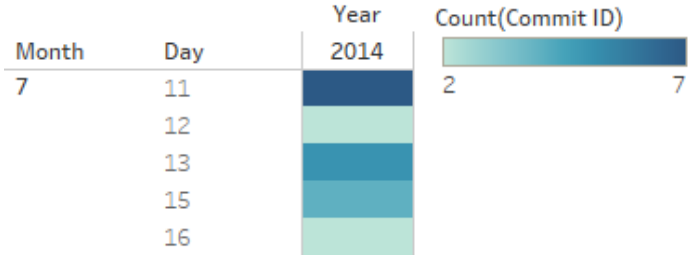


Figure 26. The Overall Contributions of the Contributor by Year, Month, and Day

In the interview, the contributor whose contributions I show in Figures 25 and 26 reported the following:

> *I had used scikit-image in the past, but I did not contribute code until SciPy 2014. After the sprints, I worked on the code for a few more days. I have not been working on scikit-image since then.*

The interview confirmed that the contributor made no contributions to the scikit-image project before and that the contributor worked on code only around the coding sprint period. I also asked about prior experience and the process of working on those pull requests. The contributor stated that they had participated in issue diagnosis and in discussions of many other projects in the past. As to the process, the contributor stated:

> *I asked for specific coding tasks [at the sprint], and the project had a list of several features that needed to be implemented.… XXXX was a good mentor. He explained the necessary features and left me alone to code for them. Occasionally I had questions, and he was nearby to answer.*

The contributor also stated that they had domain knowledge about morphology, which one needed to work on pull requests for scikit-image. The contributor had experience with related OSS projects and domain knowledge. At the co-located event, the contributor and an existing scikit-image contributor had easy access to each other to share design-relevant information and problem-solving activities. Being co-located, they could also have interactive, continuous communication, which allowed them to exchange information much more readily.

Another contributor without a previous contribution to scikit-image made a pull request that added dependencies during the focal period (see Figure 27). As one of the core members participated in the pull request, I asked them about it. The interviewee stated as follows:

> *This case was: the contributor arrived at the sprint with the feature that they used in their own research: "I really like to have this. I have an idea that might easily slot some existing functionality you have.". …A user came in: let's expand the API to integrate my use case, and he sat down and managed to implement that new functionality.*

In the pull request, the contributor described what they attempted based on a discussion at the coding sprint (on 31 August, 2014) in the EuroSciPy conference. A heat map for the contributor shows the overall contributions (see Figure 27 on the right). Figure 28 visualizes the contributor's code contribution by day, month, and year. As Figure 28 shows, the contributor started making commits on the coding sprint day (on 31 August, 2014) and worked more during the next two days. Since the sprint event, the contributor made no other code contribution to the scikit-image project.

The contributor's GitHub user page showed no activities in other OSS projects in the past. However, the contributor's website listed two software development projects on computer vision, which relates to scikit-image. Another Python conference website introduced the contributor as someone who has worked on two computer vision projects in the past. It listed scientific libraries such as scikit-image as the contributor's technical interests. In other words, although the contributor made no previous contribution to the scikit-image project, the individual could make substantial changes given their experience in a domain closely related to scikit-image. In addition, the contributor had rich, interactive discussions at the co-located site.

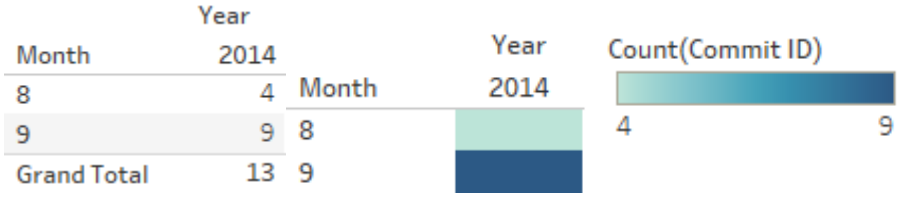


Figure 27. The Overall Contributions of a Contributor who Made their First Contribution during the Focal Period

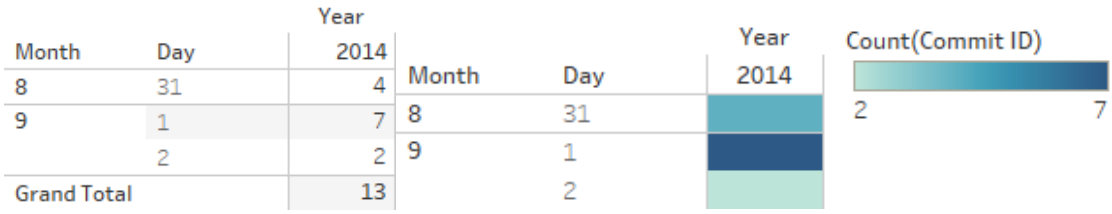


Figure 28. The Overall Contributions of the Contributor by Year, Month, and Day[37]

5.3.4 Answering RQ2

Across three projects, I found that OSS contributors who made tightly coupled code contributions to volunteer-driven OSS projects during the focal period had different types of code-contribution patterns. Half of such contributors made no previous code contributions to the sampled projects; however, they authored commits that increased software coupling during the focal period. These contributors made episodic and peripheral code contributions Figure 17 shows (see Section 6.2.3).

As I discuss in Section 5.2.3, one cannot explain contributors who made no previous code contributions but made tightly coupled code contributions from existing perspectives. In this study, I investigated the conditions under which such contributors made their tightly coupled code contributions. In Sections 5.3.1 to 5.3.3, I illustrate the conditions that facilitated tightly coupled code contributions to three different projects. Table 11 lists those conditions according to the projects and the common conditions across three projects. Across the three projects, one common condition was that such OSS contributors had already participated in other related OSS projects, whereas only two contributors constituted genuine newcomers and asked for help and guidance (see Table 11).

Table 11. Summary: Conditions that Facilitate Contributors to Make Tightly Coupled Code Contributions without a Previous Code Contribution to the Sampled Projects

Projects	Conditions	Number of contributors*
GNU grep	• Previous experience with related OSS projects—especially other GNU projects. • Familiarity with necessary but challenging tasks required in GNU grep	2
IPython	• Contributors to OSS projects that use IPython	4
	• Previous experience with related OSS projects	7
	• Asking for help and guidance as a genuine newcomer	2
scikit-image	• Contributed the code originally written for their work in a domain closely related to scikit-image outside of the scikit-image project	1
	• Previous experience with related OSS projects or software • Attended coding sprints	3
Across three projects	• Contributors already had experience with related OSS (or related software) projects.	17
	• Genuine newcomers ask for help and guidance	2
* Number of contributors without a previous code contribution who made tightly coupled code contributions.		

[37] Note that I do not show the author's name and email.

In conclusion, OSS contributors who made no previous code contributions to the sampled projects made tightly coupled code contributions given their previous experience with related OSS projects. Although the tightly coupled code contribution constituted their first contribution to the sampled projects, they had already participated in other related OSS projects as they had configured and customized related OSS packages. One can better understand my findings in the software ecosystem context rather than at the project level. In Section 6, I discuss the study's findings in detail. In particular, I consider that an OSS project's work system comprises people, resources, activities, goals, information, and technical artifacts (Winter, Berente, Howison, & Butler, 2014).

6 Discussion

6.1 Theoretical Implications

This study's findings challenge a long-held assumption that an OSS project constitutes a single, independent "container". The existing perspective has assumed that a particular project bounds code contributions and characterized contributors based on code contributions made to a particular OSS project. Such a view considers organizations as "containers" for a work system's elements (Winter et al., 2014). The current view does not consider that OSS projects' work system elements can flow across projects. Thus, the current perspective cannot sufficiently account for contemporary OSS development practices.

In this study, I suggest a multiple-fluid-container view that accommodates software ecosystems in which multiple containers (multiple related OSS projects) co-evolve with each container (each OSS project) readily accessible. An OSS project as a fluid container may constantly change its boundary as related projects' work system elements dynamically flow into and out from the container. The multiple-fluid-containers view that I suggest in this study enables one to identify how resources, contributors' joint engagements, and the technical dependencies flow among related projects. The multiple-fluid-containers view differs from inter-organizational systems in which organizational boundaries are "given". Instead, this viewpoint considers a container's boundary as continually redefined as other related containers' work system elements continually flow at any point.

The suggested multiple-fluid-containers view encompasses contemporary OSS development practices; for instance, individuals configure and customize related OSS packages to meet their needs (Valiev et al., 2018). Although an individual may not have been a core or habitual contributor to related OSS projects, the individual, in some cases, makes code contributions to related OSS projects to benefit the way in which they configure a software product, manage technical dependencies among related projects, or help projects stay active. Furthermore, such individuals may also make tightly coupled code contributions to related OSS projects as they have customized and configured related packages. Accordingly, an OSS project provides *an open platform* that helps heterogeneous contributors organize development activities and interactions.

The multiple-fluid-containers view can more fully explain how work related OSS projects' work system elements dynamically flow in software ecosystems. First, code contributions that cross-project contributors make derive purpose, meaning, and structure from multiple related projects' work systems rather than contributions that a particular project solely encapsulates. For instance, OSS projects that use IPython, such as Sympy and QIIME, do not cleanly encapsulate code contributions to them. As I illustrate in Section 5.3.2, such code contributions originate from the related projects and do not initially inherit the IPython project's goals; IPython contributors planned to develop a new version of nbconvert as the main milestone towards IPython 1.0 during the focal period. As cross-project contributors brought code contributions from related projects to the IPython project, they shaped the code in the IPython project's work system during the pull request review process. However, the code initially inherited purpose, meaning, and structure from the projects that used IPython. In the scikit-image project, an individual also brought code initially written for their work outside of the scikit-image project as the individual considered that the code could benefit it. The code did not initially inherit scikit-image's goals, which scikit-image roadmap states. However, during the pull request review process, contributors adapted the code to support one of scikit-image's goals: to extend higher dimensional support. In other words, from the multiple-fluid-containers view, technical artifacts inherit purpose, meaning, and structure from multiple containers.

In other words, while work system elements dynamically flow into the container from outside, a particular container also flexibly shapes and embeds the elements. However, an OSS project may still have recognizable work system elements. Indeed, some code contributions inherit purpose, meaning, and structure from an OSS project. For instance, IPython contributors primarily focused on developing a new version of nbconvert during the focal period by planning and dividing related tasks, and such code contributions increased the degree of software coupling (Moon & Howison, 2018). I also observed that genuine newcomers asked for help and guidance regarding their first code contribution to IPython (see Table 11). That is, merely because an OSS project as a fluid container may constantly change its boundary and shift its shape in the software ecosystem does not mean that one cannot recognize a project or that every work system element is continually emergent.

Furthermore, although each OSS project has project-specific norms or practices, dependent OSS projects in software ecosystems tend to have shared coding practices and norms in the same development environment. For example, two new grep 2.6 maintainers had already participated in other GNU projects, and the GNU ecosystem shared norms and practices. Accordingly, those new grep maintainers could quickly transfer their knowledge and skills with other GNU projects to accomplish necessary but challenging tasks, such as Gnulibification. In the IPython and scikit-image projects, contributors also already had experience with related projects, and such projects co-evolved in the same development environment. That is, as contributors develop related projects and as these projects co-evolve in the same development environment, their technical infrastructure, coding practices, and norms can converge.

6.2 Practical Implications

My findings empirically evidence the capability of peripheral, episodic contributors despite their limited availability. Although OSS contributors play an important role in sustaining OSS projects in the long term (e.g., Zhou & Mockus, 2015), OSS contributors do not necessarily intend to become core or habitual contributors. The literature on general volunteering also highlights the crucial roles that specialist episodic volunteers who can provide certain capabilities play despite the fact that they do not have much time (Bryen & Madden, 2006; Meijs & Brudney, 2007). In this section, I discuss how to attract such contributors and how one may code open coding platforms in the software ecosystem context.

Regarding how to attract specialist episodic contributors to OSS development, one could consider social incentives. I found that the GNU grep project invited experienced GNU developers as new GNU grep maintainers given their experience with other GNU projects and GNU libraries. Subsequently, they joined and made substantial code contributions to the GNU grep project as I illustrate in Section 5.3.1. Other studies have found that developers tend to participate in multiple projects (Vasilescu et al., 2014), and GitHub contributors are more likely to join projects with which they have had previous social connections (Casalnuovo, Vasilescu, Devanbu, & Filkov, 2015). Research has also found personal invitations to be a common way to recruit episodic OSS contributors (Barcomb et al., 2018). The general volunteering literature has found episodic volunteers to emphasize social incentives such as social relationships and group memberships that exist outside the organization to which they volunteered (Hustinx et al., 2008). Accordingly, in this study, I suggest that OSS projects may strategically consider inviting individuals who have contributed to other related OSS projects in an ecosystem. However, given OSS development's voluntary nature, contributors can indeed come and go from an OSS project (Germonprez, Allen, Warner, Hill, & McClements, 2013) and do not necessarily intend to become long-term, habitual contributors. Accordingly, rather than trying to reinforce them to engage in OSS projects, social incentives such as invitations by contributors with whom they previously worked or from close friends or colleagues may motivate these individuals to consider making code contributions to OSS projects.

However, episodic contributors face several challenges. Hence, one should consider strategies that can mitigate such challenges. For instance, a study of one-time contributors who successfully contributed a single code commit found that much time was required for a review of the patches created by one-time contributors and that the documentation pertaining to making a contribution was not easy to follow (Lee & Carver, 2017). In addition, one-time contributors faced low patch-acceptance rates and difficulties with projects' social aspects (Pinto et al., 2016; Steinmacher, Gerosa, Conte, & Redmiles, 2019). To mitigate such challenges, I suggest that OSS projects should consider co-located events in which contributors to software ecosystems can work together. For instance, in the scikit-image case here, three out of four contributors without previous code contributions to the scikit-image project successfully had their first pull request merged into the main branch, while their code contributions made substantial changes. The scikit-image contributors planned a list of tasks for co-located events (in this case, coding sprints at two SciPy

conferences). Anyone interested in scientific computing in Python, not limited to scikit-image, could come and go, and existing contributors provided explanations about the tasks (specifically the technical skills required and domain areas).

In addition, individuals who design coding platforms such as GitHub may consider improving features to facilitate development activities related to dependent OSS projects. For instance, a new feature can provide visual displays so that contributors can quickly recognize that their code contributions have made an impact on multiple dependent OSS projects. Such signals can help contributors identify reviewers from related projects and can facilitate problem-solving activities.

6.3 Limitations and Future Work

In this study, I measured software coupling based on the existence of functional dependencies; for instance, method X calls method Y. I used these functional dependencies to identify code contributions that led to highly coupled files. However, this measure does not consider whether dependencies occur between files in the same modules or different modules. One could use another measure, such as the relative clustered cost (Milev et al., 2009), to discern changes in or across modules. Alternatively, one could use logical coupling (Gall et al., 1998) to identify indirect or semantic relationships between files. Other technical dependency measures may help further distinguish between different types of changes that contributors make in a codebase. Hence, such measures may help identify contributors with different technical expertise.

To date, research on OSS development has tended to take a project-centric perspective and study phenomena that occurs in an OSS project. We require more studies that look outward at the OSS phenomenon in the software ecosystem context. For instance, studies that examine the factors that motivate individuals to make contributions to a particular OSS project may investigate individuals' motivations regarding the related OSS projects or their participation in the software ecosystem. Individuals may contribute to a particular project due to technical dependencies among related projects or their familiarity with a particular project's work system elements. It would also be insightful to examine OSS contributors' contribution patterns as related OSS projects co-evolve.

Software ecosystems research has focused more on technical aspects than on social or organizational aspects. Individuals configure software systems related to one another and manage technical dependencies among dependent projects. Hence, we also need to understand how heterogeneous groups of individuals dynamically engage in problem-solving activities and converge around a common goal across projects. Ecosystems evolve over time based on code contributions that manage technical dependencies in a project and across dependent projects. Accordingly, future work could investigate the emerging social structure in a software ecosystem. It would also be useful to investigate the relationships between cross-project code contributions and the technical dependencies among related projects over time. In a similar vein, it may be insightful to explore an OSS project's capability to attract and sustain contributors in relation to technical dependencies among related projects.

Future studies with a multiple-fluid-containers view that I suggest this study can help researchers shift their focus from an OSS project's internal elements to its external environment and their relations and interactions with related projects in software ecosystems. Such a shift can better explain the various types of software ecosystems and how contributors continually flow to and from related projects and how they transfer knowledge, share information, and develop common problem-solving approaches across related OSS projects. We can more holistically understand software ecosystems with a cross-discipline perspective, such as business strategies, technology and innovation management, and economics. Hence, moving this argument across disciplines should also help more richly explain how software ecosystems are sustained and how they evolve.

References

Baldwin, C. Y., & Clark, K. B. (2000). *Design Rules: The power of modularity*. Cambridge, MA: MIT Press.

Baldwin, C. Y., & Clark, K. B. (2006). The architecture of participation: Does code architecture mitigate free riding in the open source development model? *Management Science*, *52*(7), 1116-1127.

Barcomb, A., Kaufmann, A., Riehle, D., Stol, K., & Fitzgerald, B. (2018). Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities. *IEEE Transactions on Software Engineering*, *46*(9), 962-980

Barcomb, A., Stol, K.-J., Riehle, D., & Fitzgerald, B. (2019). Why do episodic volunteers stay in FLOSS communities? In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering.*

Benkler, Y. (2002). Coase's Penguin, or, Linux and "the nature of the firm". *The Yale Law Journal*, *112*(3), 369-446.

Berney, L. R., & Blane, D. B. (1997). Collecting retrospective data: Accuracy of recall after 50 years judged against historical records. *Social Science & Medicine*, *45*(10), 1519-1525.

Bird, C., Gourley, A., Devanbu, P., Gertz, M., & Swaminathan, A. (2006, May). Mining email social networks. In *Proceedings of the International Workshop on Mining Software Repositories*.

Bryen, L. M., & Madden, K. M. (2006). *Bounce-back of episodic volunteers: What makes episodic volunteers return?* Retrieved from https://core.ac.uk/download/pdf/10875803.pdf

Capiluppi, A., Lago, P., & Morisio, M. (2003). Characteristics of open source projects. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering.*

Casalnuovo, C., Vasilescu, B., Devanbu, P., & Filkov, V. (2015). Developer onboarding in GitHub: The role of prior social links and language experience. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*.

Chris, D., & Sam, O. (Eds.). (1999). *Open sources: Voices from the open source revolution* (1st ed.). Beijing: China: O'Reilly Media.

Cnaan, R., & Handy, F. (2005). Towards understanding episodic volunteering. *Vrijwillige Inzet Onderzocht*, *2*(1), 29-35.

Colfer, L. (2010). *Three essays on the structure of technical collaboration* (doctoral dissertation). Graduate School of Arts and Sciences, Harvard University, Boston, MA.

Colfer, L. J., & Baldwin, C. Y. (2010). The mirroring hypothesis: Theory, evidence and exceptions. *Harvard Business School.* Retrieved from https://hbswk.hbs.edu/item/6361.html

Colfer, L. J., & Baldwin, C. Y. (2016). The mirroring hypothesis: Theory, evidence, and exceptions. *Industrial and Corporate Change*, *25*(5), 709-738.

Conway, M. E. (1968). *How do committees invent?* Retrieved from http://melconway.com/Home/Conways_Law.html

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the ACM Workshop on Interdisciplinary Software Engineering Research*.

Crowston, K., Howison, J., & Annabi, H. (2006a). Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, *11*(2), 123-148.

Crowston, K., Wei, K., Li, Q., & Howison, J. (2006b). Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences.*

Dahlander, L. (2007). Penguin in a new suit: A tale of how de novo entrants emerged to harness free and open source software communities. *Industrial and Corporate Change*, *16*(5), 913-943.

Dahlander, L., & Magnusson, M. (2008). How do firms make use of open source communities? *Long Range Planning*, *41*(6), 629-649.

Dinh-Trong, T. T., & Bieman, J. M. (2005). The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, *31*(6), 481-494.

Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work, 14*(4), 323–368.

Eppinger, S. D., Whitney, D. E., Smith, R. P., & Gebala, D. A. (1994). A model-based method for organizing tasks in product development. *Research in Engineering Design*, *6*(1), 1-13.

Feller, J., Finnegan, P., Fitzgerald, B., & Hayes, J. (2008). From peer production to productization: A study of socially enabled business exchanges in open source service networks. *Information Systems Research*, *19*(4), 475-493.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587-598.

Fogel, K. (2005). *Producing open source software: How to run a successful free software project.* Beijing, China: O'Reilly Media.

Franco-Bedoya, O., Ameller, D., Costal, D., & Franch, X. (2017). Open source software ecosystems: A Systematic mapping. *Information and Software Technology*, *91*, 160-185.

Gall, H., Hajek, K., & Jazayeri, M. (1998). Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance.*

Germonprez, M., Allen, J. P., Warner, B., Hill, J., & McClements, G. (2013). Open source communities of competitors. *Interactions*, *20*(6), 54-59.

Germonprez, M., Kendall, J. E., Kendall, K. E., Mathiassen, L., Young, B., & Warner, B. (2016). A theory of responsive design: A field study of corporate engagement with open source communities. *Information Systems Research*, *28*(1), 64-83.

Germonprez, M., Kendall, J. E., Kendall, K. E., & Young, B. (2014). Collectivism, creativity, competition, and control in open source software development: Reflections on the emergent governance of the SPDX ® working group. *International Journal of Information Systems and Management*, *1*(1), 125-145.

Guizar-Sicairos, M., Thurman, S. T., & Fienup, J. R. (2008). Efficient subpixel image registration algorithms. *Optics Letters*, *33*(2), 156-158.

Handy, F., Brodeur, N., & Cnaan, R. A. (2006). Summer on the Island: Episodic volunteering. *Voluntary Action*, *7*(1), 31-42.

Harrison, D. A. (1995). Volunteer motivation and attendance decisions: Competitive theory testing in multiple samples from a homeless shelter. *Journal of Applied Pscyhology*, *80*(3), 371-385.

Harrower, M., & Brewer, C. A. (2003). ColorBrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, *40*(1), 27-37.

Hindle, A., German, D. M., & Holt, R. (2008). What do large commits tell us? A taxonomical study of large commits. In *Proceedings of the International Working Conference on Mining Software Repositories*.

Howison, J. (2009). *Alone together: A socio-technical theory of motivation, coordination and collaboration technologies in organizing for free and open source software development* (doctoral thesis). Syracuse University, Syracuse, New York.

Howison, J., & Crowston, K. (2014). Collaboration through open superposition: A theory of the open source way. *MIS Quarterly*, *38*(1), 29-50.

Hustinx, L., Haski-Leventhal, D., & Handy, F. (2008). One of a kind? Comparing episodic and regular volunteers at the Philadelphia Ronald McDonald House. *The International Journal of Volunteer Administration*, *25*(3), 50-66.

Hustinx, L., & Lammertyn, F. (2003). Collective and reflexive styles of volunteering: A sociological modernization perspective. *International Journal of Voluntary and Nonprofit Organizations*, *14*(2), 167-187.

Hyde, M. K., Dunn, J., Scuffham, P. A., & Chambers, S. K. (2014). A systematic review of episodic volunteering in public health and other contexts. *BMC Public Health*, *14*, 1-16.

Jergensen, C., Sarma, A., & Wagstrom, P. A. (2011). The onion patch. In *Proceedings of the 13th European Conference on Foundations of Software Engineering*.

Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K. (2018). Do developers update their library dependencies? *Empirical Software Engineering*, *23*(1), 384-417.

Lee, A., & Carver, J. C. (2017). Are one-time contributors different? A comparison to core and periphery developers in FLOSS repositories. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.*

Lindberg, A., Berente, N., Gaskin, J., & Lyytinen, K. (2016). Coordinating interdependencies in online communities: A study of an open source software project. *Information Systems Research*, *27*(4), 751-772.

MacCormack, A., Baldwin, C., & Rusnak, J. (2012). Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. *Research Policy*, *41*(8), 1309-1324.

MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, *52*(7), 1015-1030.

Macduff, N. L. (2004). *Episodic volunteering: Organizing and managing the short-term volunteer program*. Walla Walla, WA: MBA Publishing.

Meijs, L. C. P. M., & Brudney, J. L. (2007). Winning volunteer scenarios: The soul of a new machine. *International Journal of Volunteer Administration*, *24*(6), 68-79.

Milev, R., Muegge, S., & Weiss, M. (2009). Design evolution of an open source project using an improved modularity metric. In C. Boldyreff, K. Crowston, B. Lundell, & A. I. Wasserman (Eds.), *Open source ecosystems: Diverse communities interacting* (pp. 20-33). Berlin: Springer.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions of Software Engineering and Methodology*, *11*(3), 309-346.

Moon, E., & Howison, J. (2014). Modularity and organizational dynamics in open source software (OSS) production. In *Proceedings of the Americas Conference on Information Systems.*

Moon, E., & Howison, J. (2018). Coordination recipes for tightly-coupled software work in open source software (OSS) communities. In *Academy of Management Proceedings.*

Munzner, T. (2014). *Visualization analysis and design*. New York, NY: CRC Press.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*.

Pinto, G., Steinmacher, I., & Gerosa, M. A. (2016). More common than you think: An in-depth study of casual contributors. In *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering.*

Robles, G., & Gonzalez-Barahona, J. M. (2005). Developer identification methods for integrated data from various sources. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1-5.

Rullani, F., & Haefliger, S. (2013). The periphery on stage: The intra-organizational dynamics in online communities of creation. *Research Policy*, *42*(4), 941-953.

Sanchez, R. (1996). Strategic product creation: Managing new interactions of technology, markets, and organizations. *European Management Journal*, *14*(2), 121-138.

Sanchez, R., & Mahoney, J. T. (1996). Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal*, *17*(S2), 63-76.

Setia, P., Rajagopalan, B., Sambamurthy, V., & Calantone, R. (2010). How peripheral developers contribute to open-source software development. *Information Systems Research*, *23*(1), 144-163.

Silva, J. D. O., Wiese, I. S., German, D. M., Steinmacher, I., & Gerosa, M. A. (2017). *How long and how much: What to expect from summer of code participants?* In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution.*

Steinmacher, I., Gerosa, M., Conte, T. U., & Redmiles, D. F. (2019). Overcoming social barriers when contributing to open source software projects. *Computer Supported Cooperative Work*, *28*(1), 247-290.

Steward, D. V. (1981). The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management, EM-28*(3), 71-74.

Strauss, A., & Corbin, J. M. (1990). *Basics of qualitative research: Grounded theory procedures and techniques*. Thousand Oaks, CA: Sage.

Ulrich, K. (1995). The role of product architecture in the manufacturing firm. *Research Policy*, *24*(3), 419-440.

Valiev, M., Vasilescu, B., & Herbsleb, J. (2018). Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

Vasilescu, B., Serebrenik, A., Goeminne, M., & Mens, T. (2014). On the variation and specialisation of workload—a case study of the Gnome ecosystem community. *Empirical Software Engineering*, *19*(4), 955-1008.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). Scikit-image: Image processing in Python. *PeerJ*. Retrieved from https://peerj.com/articles/453/

van Wesel, P., Lin, B., Robles, G., & Serebrenik, A. (2017). *Reviewing career paths of the OpenStack developers.* In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution.*

Winter, S., Berente, N., Howison, J., & Butler, B. (2014). Beyond the organizational "container": Conceptualizing 21st century sociotechnical work. *Information and Organization*, *24*(4), 250-269.

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*.

Yu, Y., Benlian, A., & Hess, T. (2012). An empirical study of volunteer members' perceived turnover in open source software projects. In *Proceedings of the 45th Hawaii International Conference on System Sciences*.

Zhou, M., & Mockus, A. (2015). Who will stay in the FLOSS community? Modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, *41*(1), 82-99.

Appendix A: Visually Analyzing Overall Code Contributions

By visually analyzing OSS contributors who made tightly coupled code contributions using heat maps, I could characterize those contributors as habitual or episodic contributors. I visualized the data using Tableau software. To identify habitual contributors, I set the minimum number of commits in a month as two (Barcomb et al., 2018). If a contributor did not have a habitual pattern, I removed the filter for further investigation into their contribution patterns. I collected and inspected commits that contributors made up to June, 2018.

GNU grep

The GNU grep project invited two new maintainers after the grep 2.5.4 release. They started working on GNU grep from November, 2009, as maintainers and made tightly coupled code contributions during the focal period (11 February, 2009, to 23 March, 2010). I visualize each maintainer's contribution pattern by year and month based on how I define a habitual contributor.

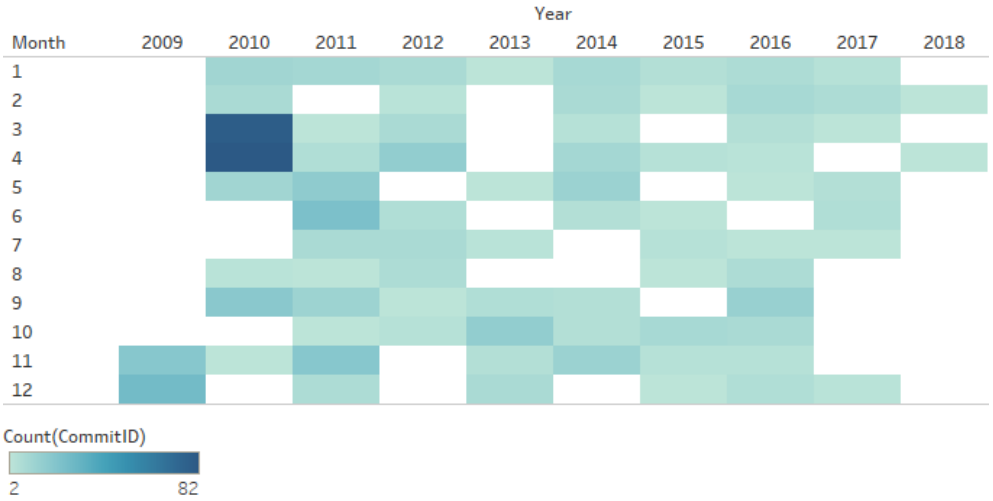


Figure A1. The Overall Contributions of One of Two New Maintainers who Made Tightly Coupled Code Contributions to the GNU grep Project

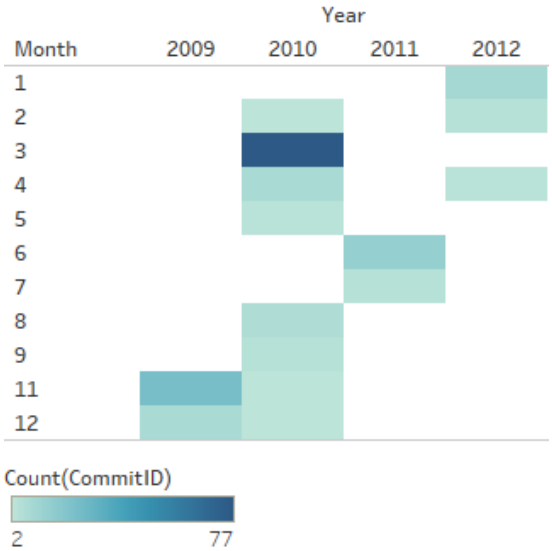


Figure A2. The Overall Contribution of Another New Maintainer who Made Tightly Coupled Code Contributions to the GNU grep Project

IPython

I visualize the overall contributions from OSS contributors who made tightly coupled code contributions during the focal period (30 June, 2012 ~ 9 August, 2013) using heat maps based on how I define a habitual contributor.

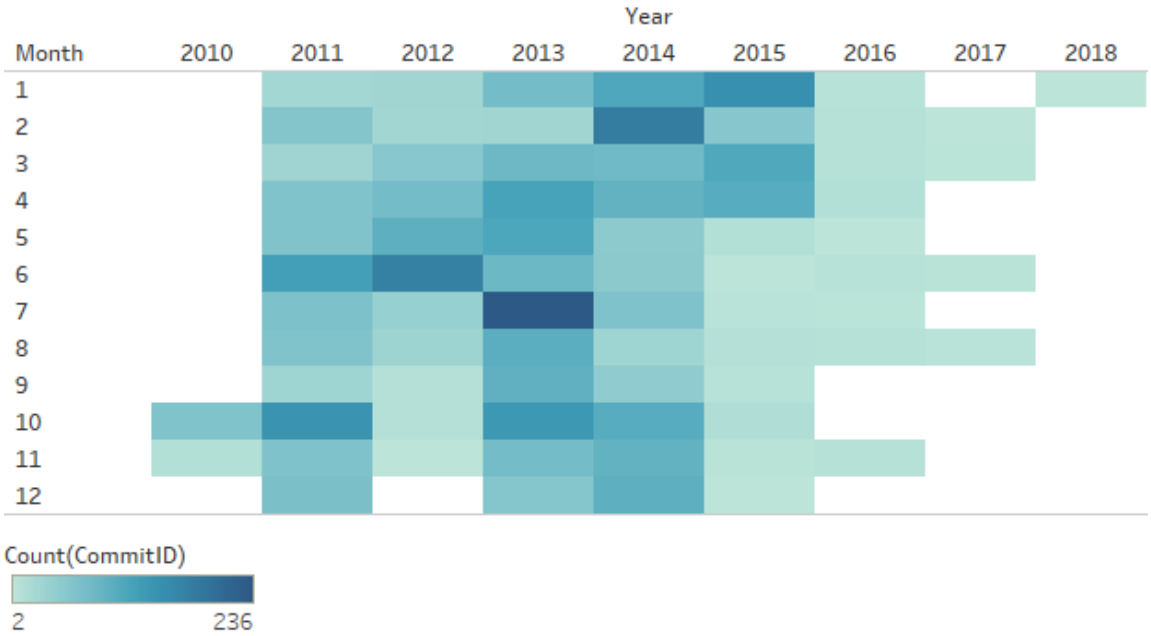


Figure A3. The Overall Contribution by an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

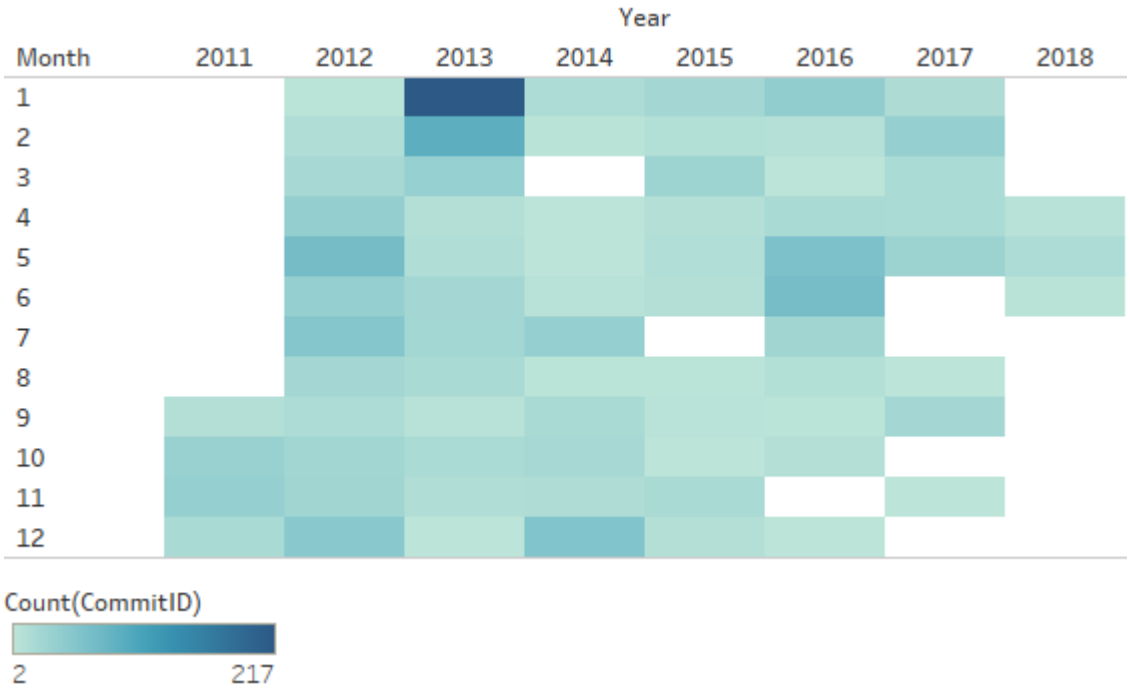


Figure A4. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

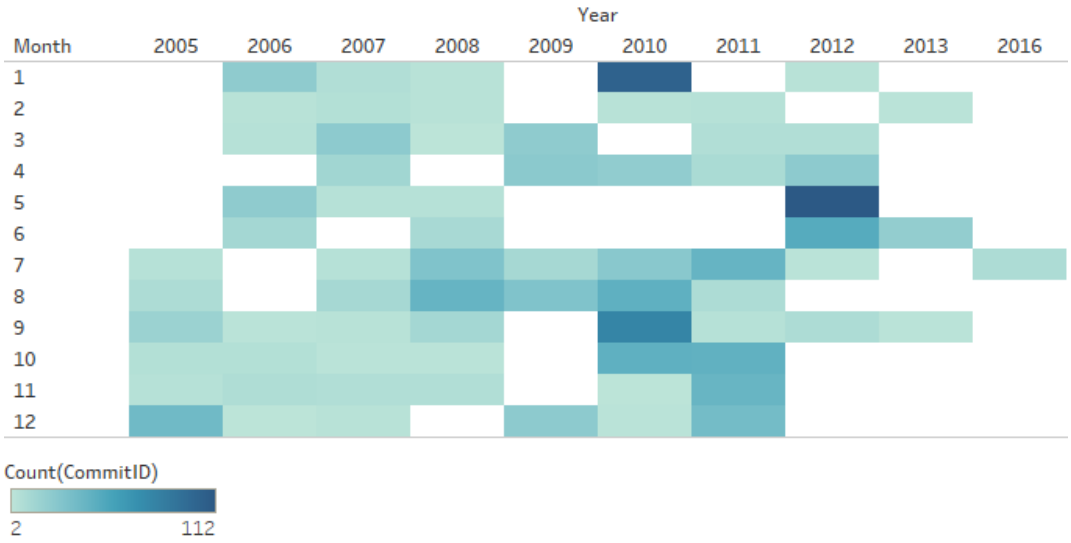


Figure A5. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

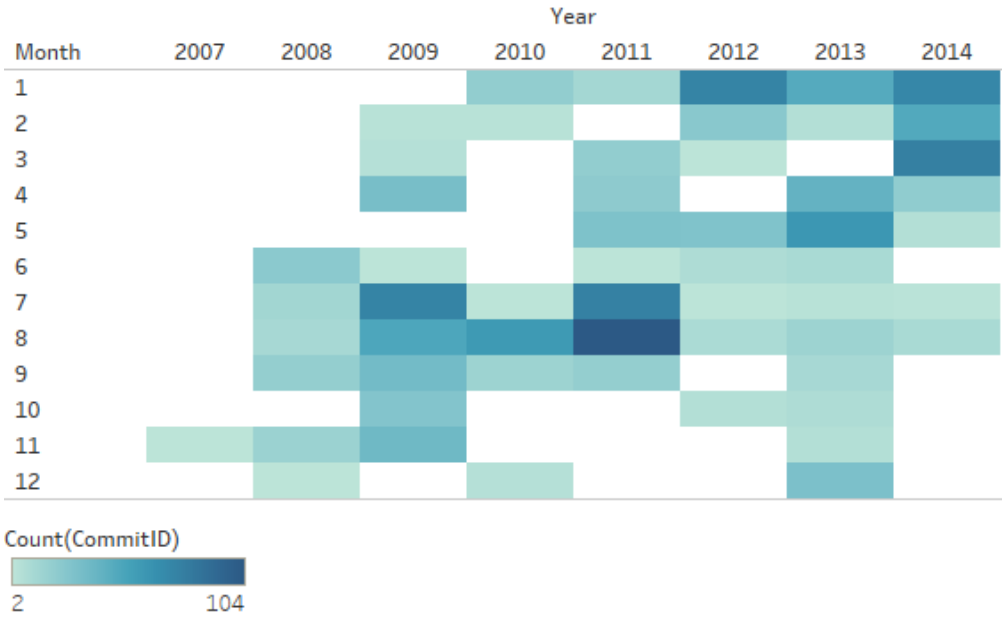


Figure A6. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

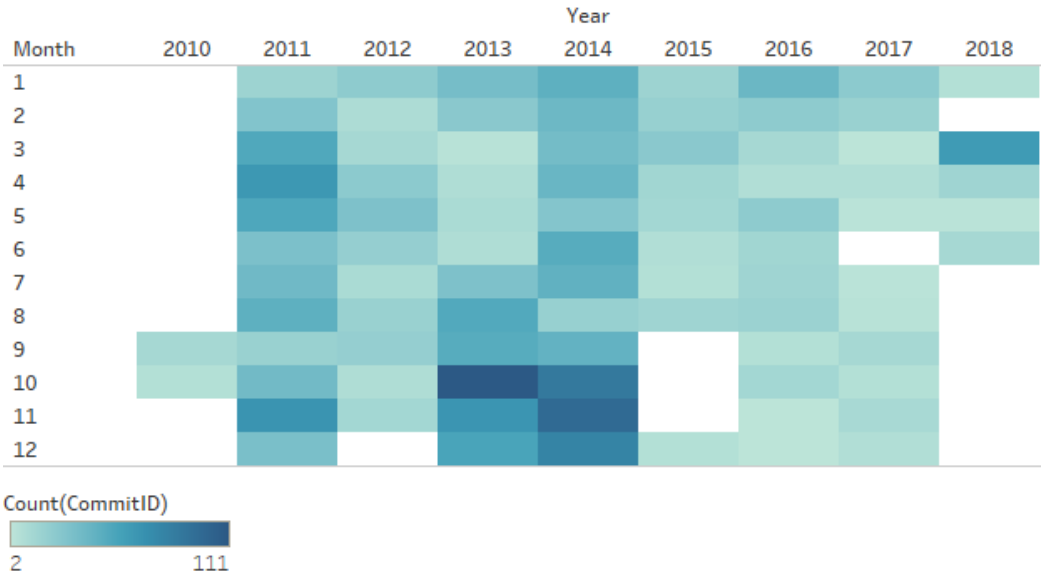


Figure A7. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

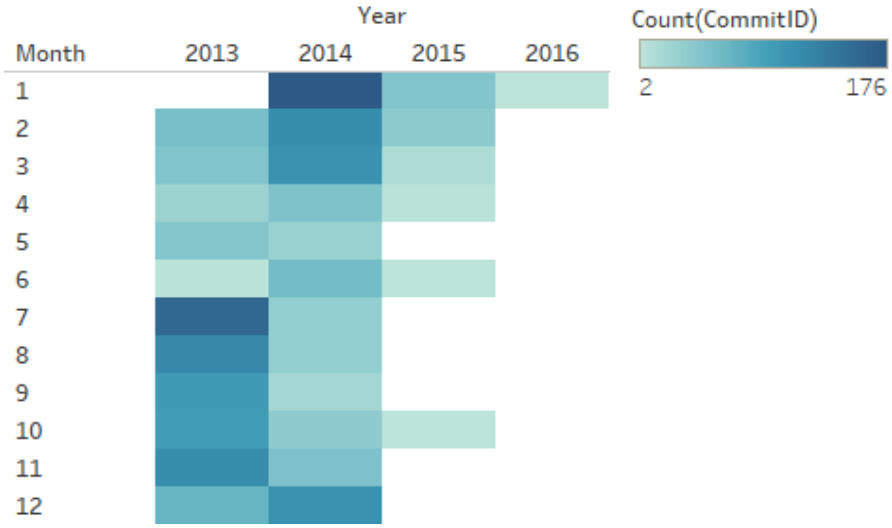


Figure A8. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

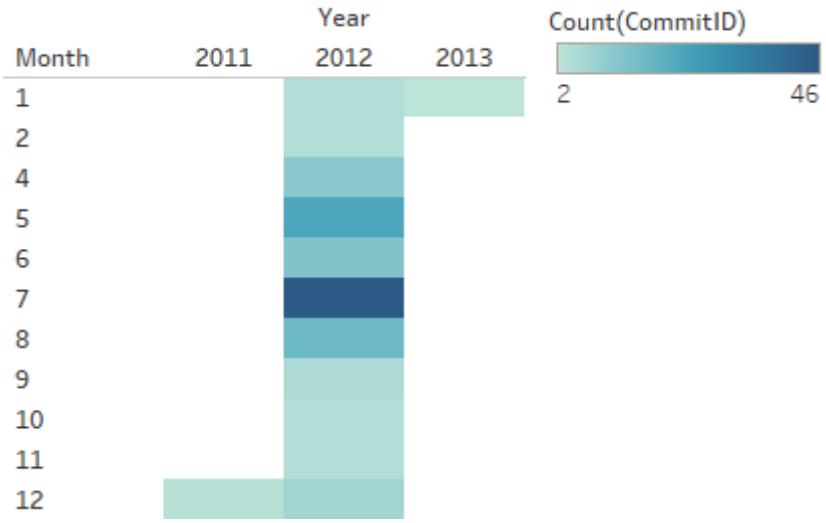


Figure A9. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

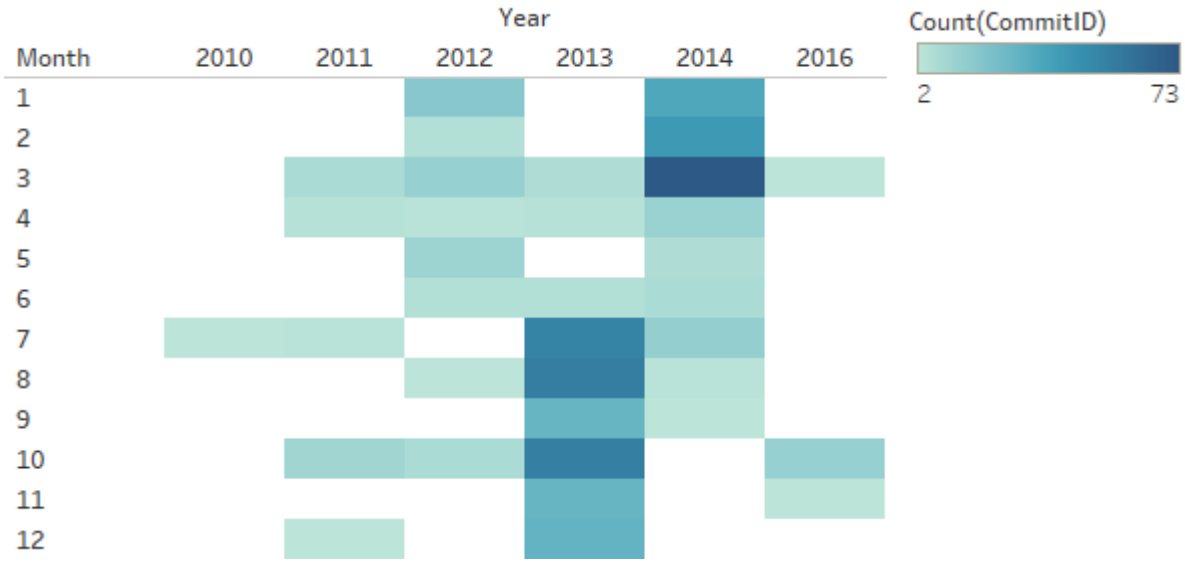


Figure A10. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contributions during the Focal Period

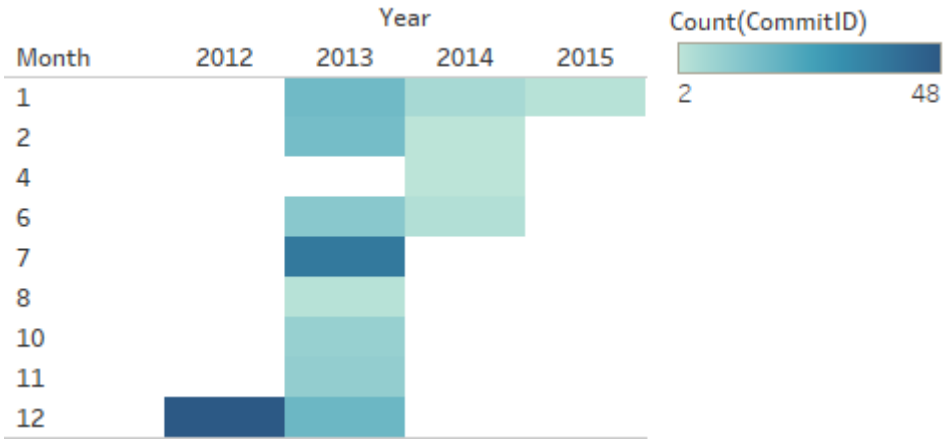


Figure A11. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

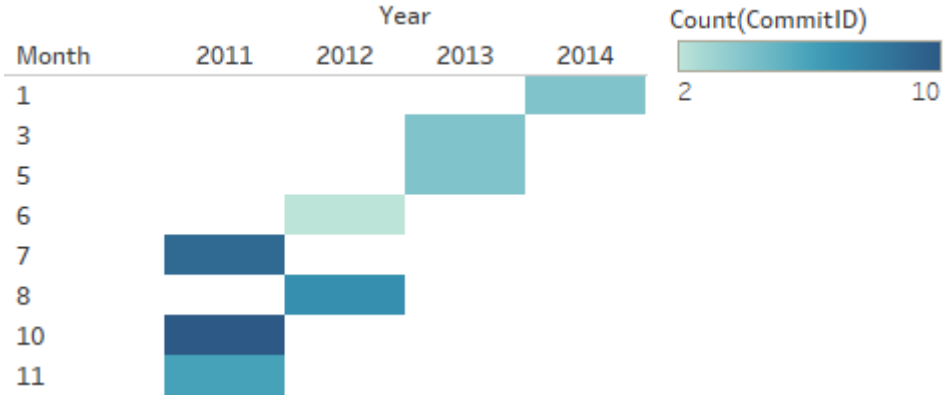


Figure A12. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

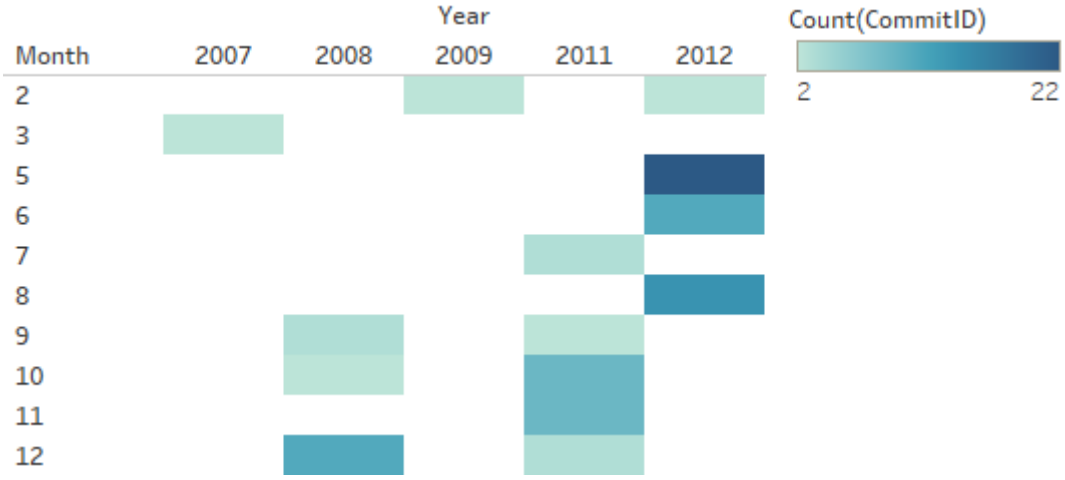


Figure A13. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

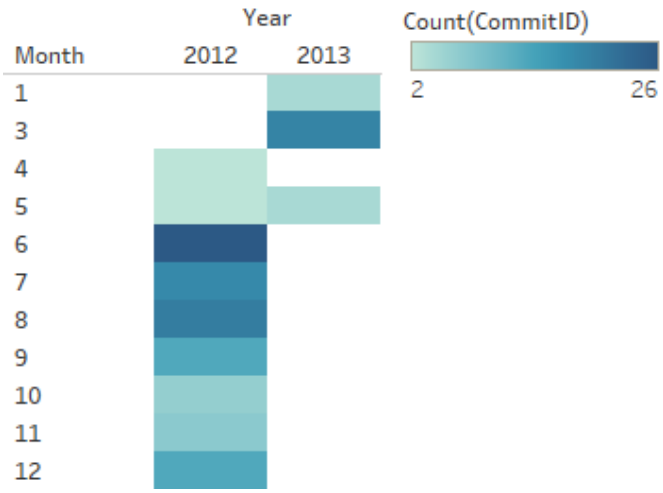


Figure A14. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

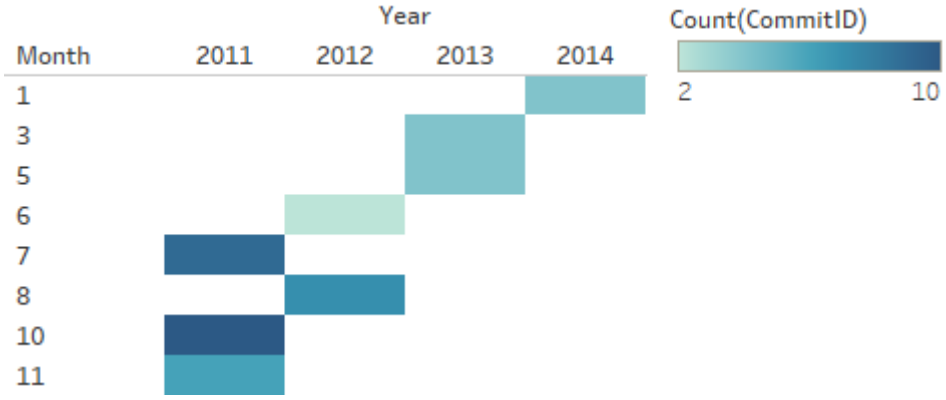


Figure A15. The Overall Contribution of an IPython Contributor who Made Tightly Coupled Code Contribution during the Focal Period

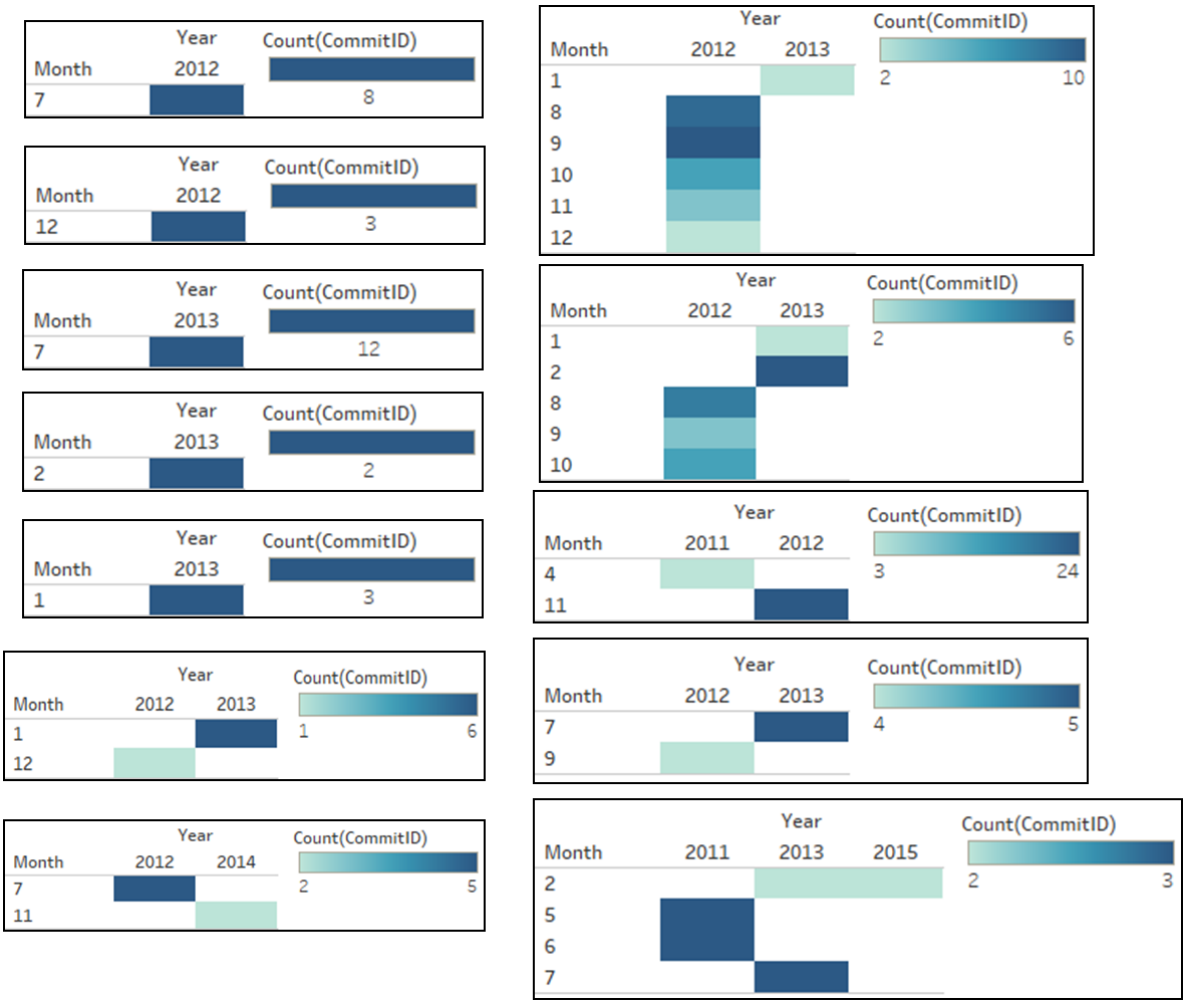


Figure A16. The Overall Contributions by IPython Contributors who Made Tightly Coupled Code Contributions during the Focal Period[38]

[38] Each heat map visualizes each contributor's overall code contribution to the IPython project. I merged these smaller heat maps to save spaces.

scikit-image

I visualize the overall code contributions from scikit-image contributors who made tightly coupled code contributions during the focal period (28 May, 2014 to 4 March, 2015) using heat maps.

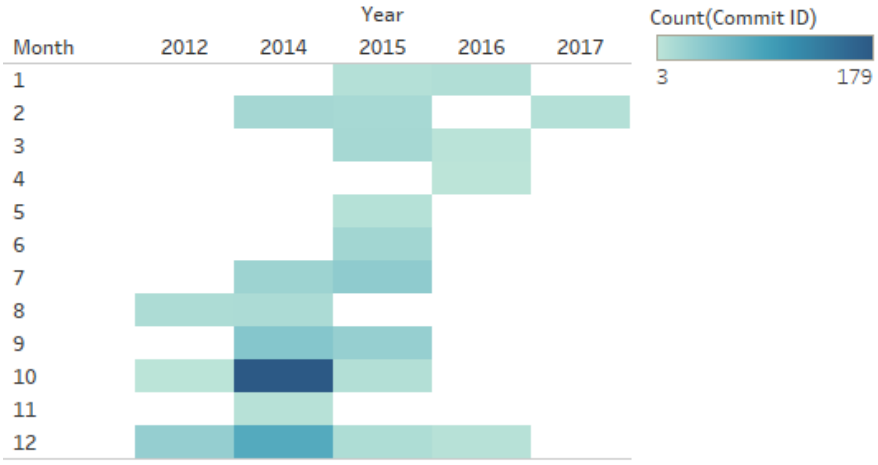


Figure A17. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

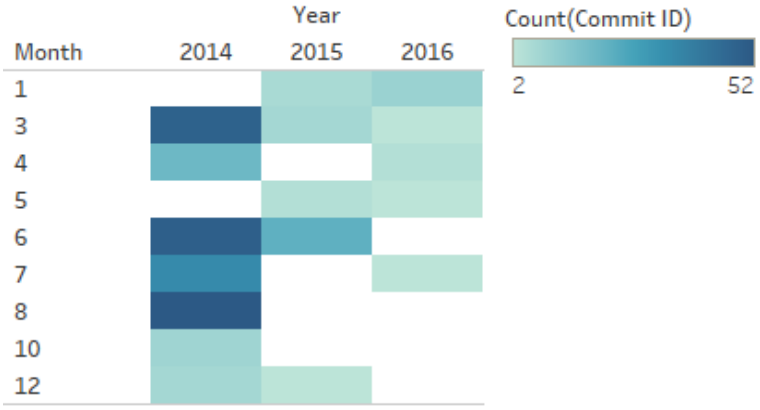


Figure A18. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

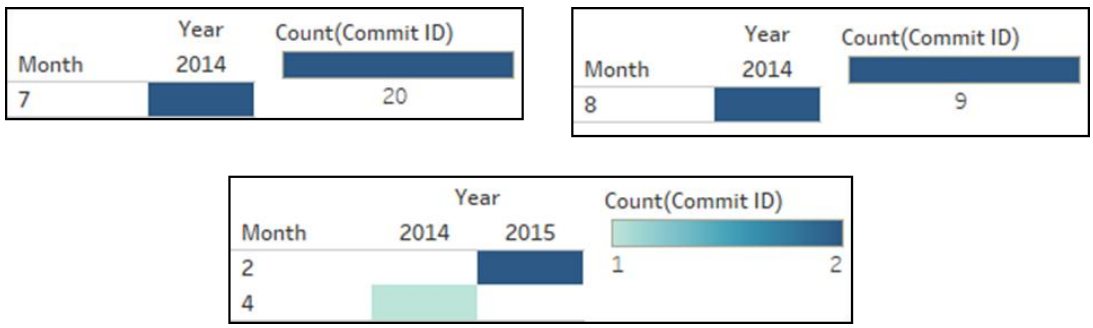


Figure A19. The Overall Contributions by scikit-image Contributors who Made Tightly Coupled Code Contributions during the Focal Period[39]

[39] Each heat map visualizes each contributor's overall code contribution to the scikit-image project. I merged these smaller heat maps to save spaces.

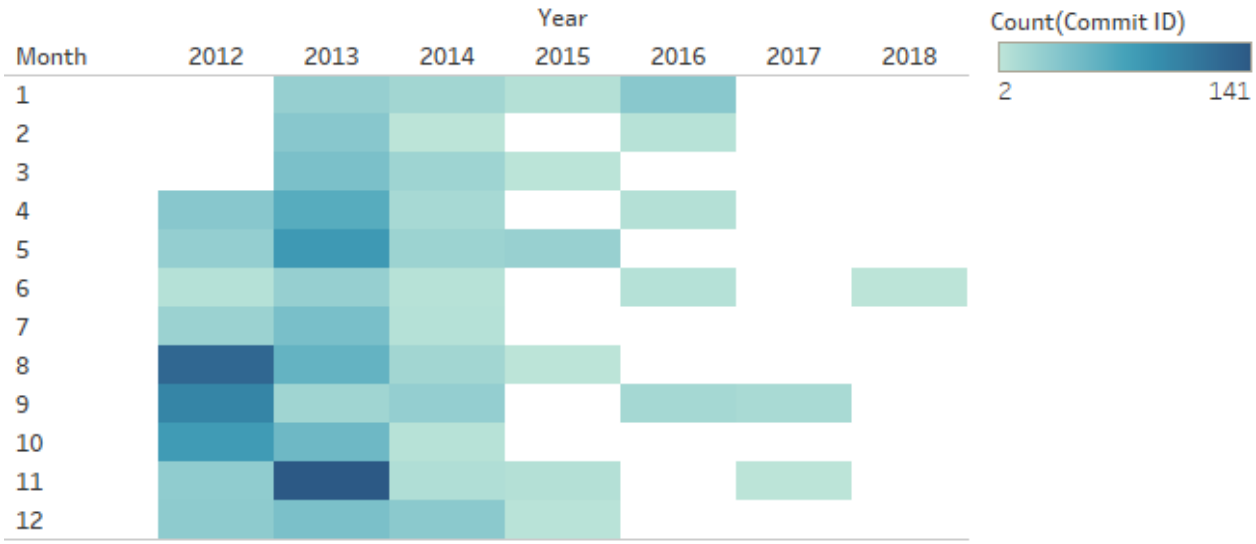


Figure A20. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

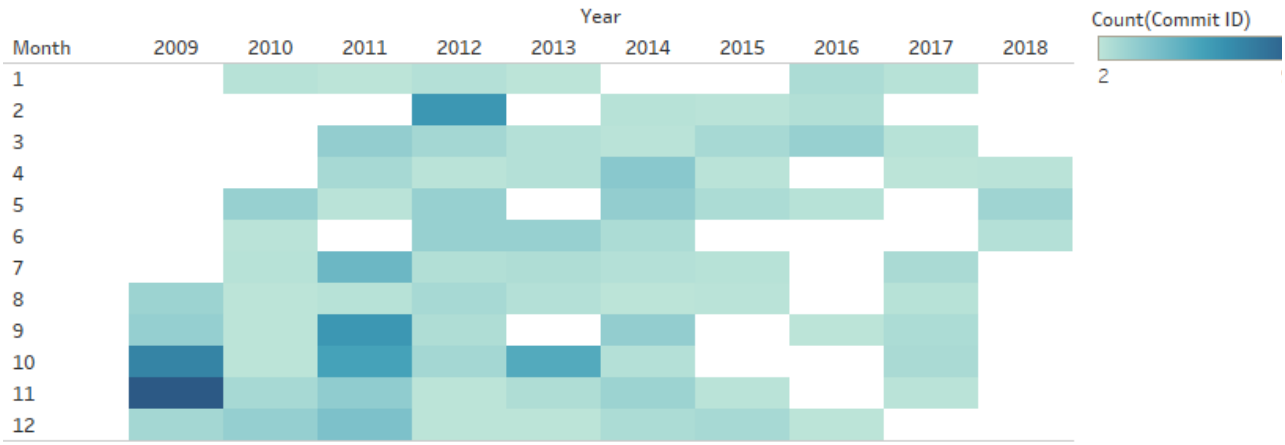


Figure A21. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

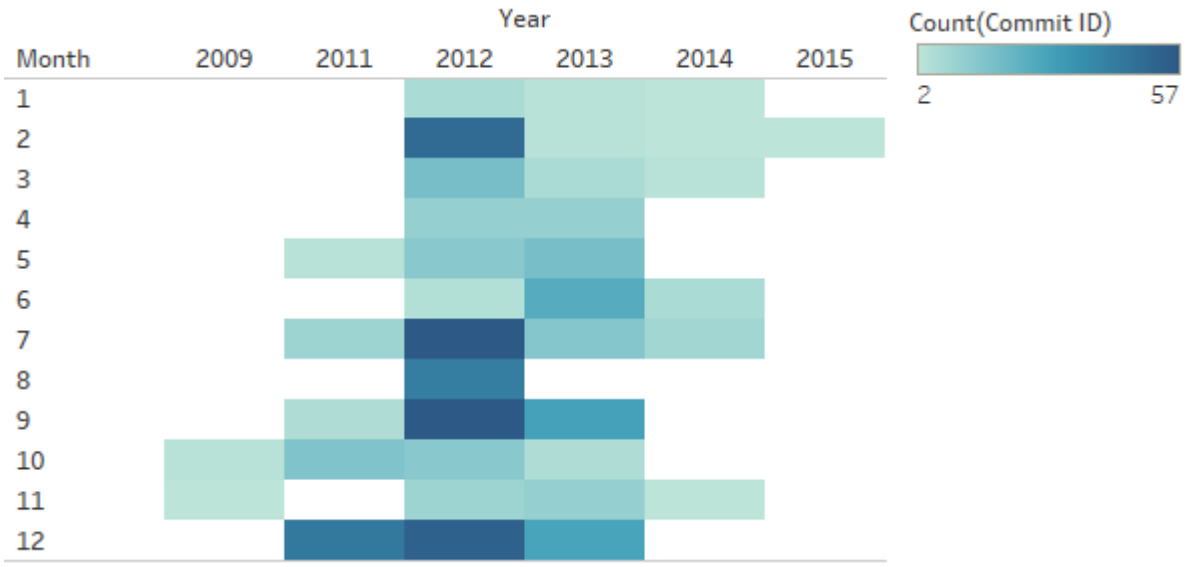


Figure A22. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

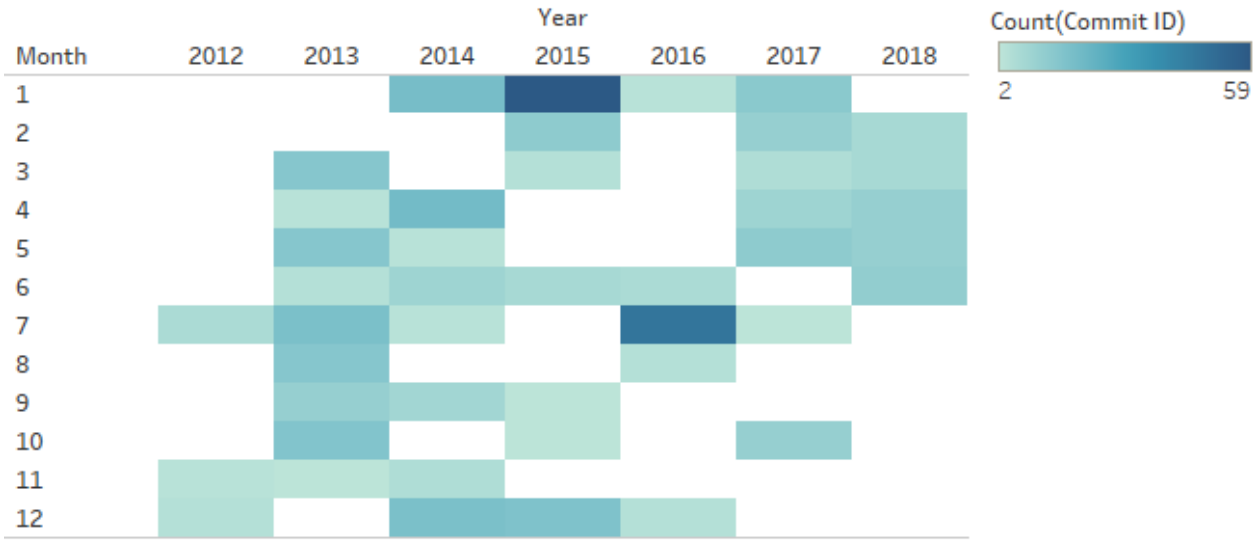


Figure A23. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

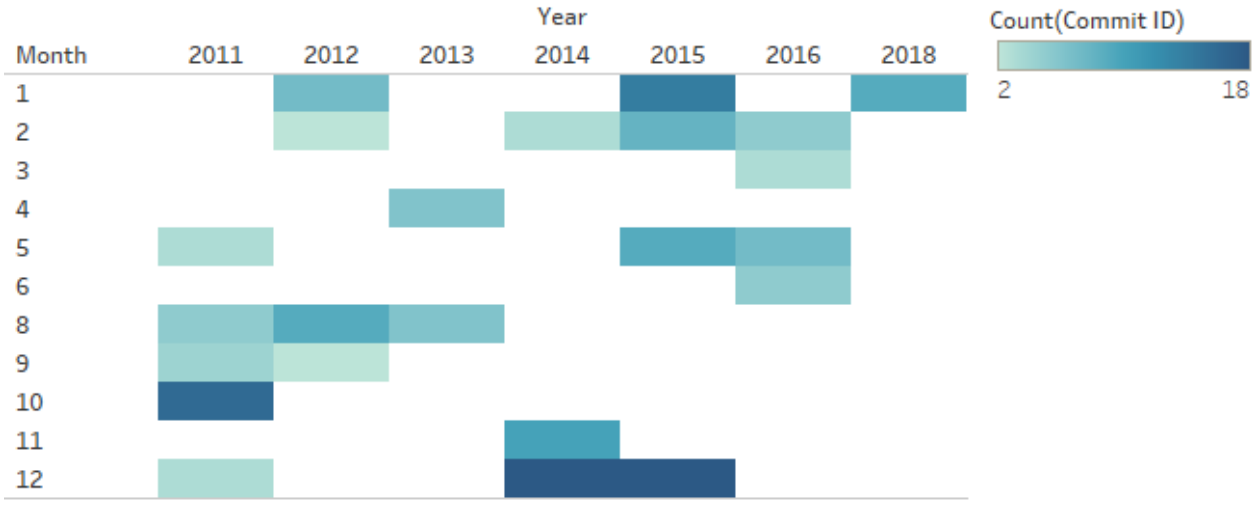


Figure A24. The Overall Contribution of a scikit-image Contributor who Made Tightly Coupled Code Contributions during the Focal Period

About the Author

Eunyoung Moon is currently a Research Assistant Professor in the School of Computing at the Korea Advanced Institute of Science and Technology (KAIST). She earned her PhD in the School of Information at the University of Texas at Austin and Master of Science at KAIST. Her research interests include collaborative work and technology, and Open Source Software (OSS) communities.