Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Measuring Patch-Flow at Google

Michael Dorner

submitted on 01.10.2015

supervised by Maximilian Capraro, M.Sc.

Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 01.10.2015

License

Erlangen, 01.10.2015

Contents

Part I

Front Matter

1 Abstract

In the industrial domain, software development is a highly collaborative work involving different contributing teams. But there is not yet a way to quantify the collaboration between organizational units within a software developing company. However, information about this collaboration is latent in software repositories and has not been defined yet.

We mined Google's internal software repository and identified all commits which are assigned to projects of organizational units the patch author does not belong to. We call this phenomena of collaboration beyond organizational borders *patch-flow*. This work introduces a graph-based metric to quantify this patch-flow. We developed a tool that is able to crawl in Google's repository and collected patches of 2,500 Google developers in the years 2007, 2009, 2011, and 2013. Due to the missing historical information about organizational unit membership of developers, we provided a clustering approach to assign all developers to organizational units. Because the Google internal data has not been released by now, we crawled and analyzed the Chromium project.

Using the Chromium data we were able to apply the patch-flow metric and quantify collaboration over organizational unit boundaries, although the used data source is only suitable to a limited extent. The clustering approach has to be validated.

Keywords: collaboration, mining software repositories, google, orgunit, patch, flow

2 Introduction

Changes To Thesis Goals

There were some major changes in the roadmap of this thesis, which entails some major drawbacks:

- We were not able to determine historical data regarding the assigned organizational units of a developer.
- The internal Google data set is still hanging in the Google internal review process. Therefore, we were forced to adapt the crawler, mine the Chromium project, and use this data for measuring the patch-flow, although we are aware that the Chromium data is of second quality.

When the Google data set will arrive, section 7.5 will become obsolete and chapter 8 and the consequently limitation mentioned in 9 can be adopted to the new data set.

3 Acronyms

CLoC	Comment Lines of Code.
LoC	Lines of Code.
MSR	Mining Software Repositories.
orgunit	organizational unit.
SCM	Source Control Management.
SLoC	Source Lines Of Code.

Part II

Research Chapter

4 Introduction

> *Organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations.*
>
> — **Melvin E. Conway**

Software engineering is a complex engineering activity and often involves team effort. In practice, commercial software development is performed by organizational units (orgunits) consisting of a number of individuals ranging from a handful to thousands.

But as Conway (1968) stated, communication includes also exchange of source code. The goal of this thesis is to measure this communication between orgunits, or how we call it, the patch-flow.

The contributions of this thesis are

- a formal definition of a patch-flow graph and the patch-flow metric,
- a clustering approach to recover historical information about the orgunit membership of developers, and
- a tool mining the necessary data at Google and the Chromium project.

Furthermore, this thesis answers the following research questions:

1. To which extent does patch-flow exist within Google?
2. Which factors do affect or correlate with the quantity of patch-flow within Google?

Google as one of the largest software companies in the world with about 25,000 developers opened its Source Control Management (SCM) system for us to analyze the phenomena of patch-flow: As of January 2015 Google's Perforce-based monolithic code base contains about one billion files (including source files copied into branches, files that are deleted at the latest revision, configuration files, documentation, and supporting data files), nine million source files, two billion lines of code. 45,000 patches are committed every workday. 15,000 commits are committed by humans, 30,000 are committed by automated systems. The depth of the history is 35 million commits. [1]

This monolithic structure allows us to crawl easily and completely.

This thesis is structured as follows: After this introduction, we present the related work in chapter 5. Chapter 6 defines the patch-flow graph and the patch-flow metric from a theoretical perspective.

The data acquisition in chapter 7 considers the tooling, the data model, the constraints, and the necessary preprocessing of the collected data at Google and in the Chromium project, because the information about this code-level collaboration is latent in the SCM.

The results of this thesis are presented in chapter 8. Chapter 9 discusses potential threats to validity and limitations of this work. The follow-up chapter 10 highlights opportunities for future work. In the last chapter 11 we conclude the research part with a short summary.

Because the original requested data was not released until the end of this thesis, we were forced to use the open source project Chromium. So chapters 8, 9, and 11 are Chromium specific.

[1] Rachel Potvin at SCALE 2015, `https://www.youtube.com/watch?v=W71BTkUbdqE`

5 Related Work

This section splits up into two subparts: After a short introduction to the field of Mining Software Repositories (MSR), previous work which considered collaboration in software organizations shall be described.

5.1 Mining Software Repositories

There are many different sources for mining information about the software development process, such as e-mail (Bird, Gourley, Devanbu, Gertz, & Swaminathan, 2006; Rigby & Hassan, 2007), defect tracking systems (Bhattacharya, Iliofotou, Neamtiu, & Faloutsos, 2012; Canfora & Cerulo, 2005), or source code itself (Hassan, 2008). Our work considers SCM, particularly Perforce and git. Kagdi, Collard, and Maletic (2007) present a survey of MSR approaches in the context of software evolution.

There are many research questions regarding mining software repositories. Begel and Zimmermann (2014) collects via survey at Microsoft 145 questions a software repository could answer and Hassan (2008) presents several recent achievements and results of using MSR techniques to support software research and practice. In this paper, the focus is on collaboration on code-level base between orgunits.

But Hassan (2008) showed that mining in software repositories is faced with two problems: Companies are not willing to give external researchers access to such detailed and sensible information about their software systems. Only a few mid size (Colaço Jr, Mendonça, & Rodrigues, 2009) and even less larger companies such as Microsoft (e.g. Bird, Murphy, Nagappan, and Zimmermann (2011), Nagappan, Murphy, and Basili (2008), Pinzger, Nagappan, and Murphy (2008), Bird, Nagappan, Gall, Murphy, and Devanbu (2009), Zimmermann and Nagappan (2008)) or Philips (Vanya, Klusener, Premraj, Van Rooijen, & Van Vliet, 2011) offer an insight in their software repositories to research. Within the scope of this thesis we were able to mine in Google's internal repository.

The second problem for mining software repositories is technical: most repositories are not designed with large-scale data-extraction and mining in mind (Hassan, 2006). Therefore, a bunch of tools were developed to extract these information from CVS, git, and Subversion (e.g. Anbalagan and Vouk (2009), German and Mockus (2003), Gousios and Spinellis (2012), Robles, Gonzalez-Barahona, and Ghosh (2004), Voinea and Telea (2006), Xie, Poshyvanyk, and Marcus (2006)), but none of them for Perforce. We solved this issue by developing a new crawling tool for Perforce and – in the further course of the work – for git.

5.2 Collaboration in Software Organizations

In contrast to mining in protected, closed industry repositories, open source enables research due to its public availability and easy access. Heller, Marschner, Rosenfeld, and Heer (2011), Huang and Liu (2005), Jermakovics, Sillitti, and Succi (2013), Ohira, Ohsugi, Ohoka, and Matsumoto (2005), Xu, Gao, and Christley (2005), Thung, Bissyandé, Lo, and Jiang (2013), Weissgerber, Pohl, and Burch (2007) analyzed collaboration in the context of open source. However, while open source and its research considers individual developers, this thesis focuses on analyzing the collaboration between organizational units within software developing companies in an industrial domain, rather than open source communities. Menzies, Bird, Zimmermann, Schulte, and Kocaganeli (2011) lists more differences, but also similarities between data mining in open source and industry.

Gousios, Kalliamvakou, and Spinellis (2008) presented a model that extract process data from software repositories and combines them in a single contribution factor. Our work differs in three points:

- We are focusing on collaboration of orgunits. They focus on collaboration of individual developers.
- This works considers source code repositories only. They included additionally bug database, mailing lists, wiki and IRC participation.
- They do not provide a real world result for the proposed metric, what we will offer by the data of Google.

Pinzger et al. (2008) used the contribution history to construct the networks of binaries and the developers that contributed to them. They found using measures knowing from social network analysis such as degree centrality, closeness centrality, and Bonacich power in contribution networks had very good predictive power in determining failure-prone binaries. Although the contribution network was the starting point for our model of the patch-flow graph, it differs in four aspects from Pinzger et al. (2008):

- Our pivot is an orgunit. They consider a single developer.

- We do not consider single binaries like they do, but projects.
- Our patch-flow graph is not a undirected bipartite graph as the contribution network, but a directed graph, where projects and users are properties of an orgunit represented by a node. Edges are contribution weighted by a cost function.
- We are able to apply different cost functions to the graph, they consider the amount of contribution only.

Similar to the proposed contribution network, Lopez-Fernandez and Robles (2004) suggests a *commiter* and module network. A *commiter network* is graph where a vertex corresponds to a particular committer, and an edge represents a common module, which both committers contributed to. A vertex in a module network represents a project and there are edges between two vertices if at least one committer contributed to both projects. As mentioned in the previous listing, we focus on groups instead of individual developers. Ignoring the semantic and intention, the module network is close to what we call a patch-flow graph, because of the special case at Google where a project corresponds to an orgunit (see section 6.1). However, this does not hold for the general case, where an orgunit can embrace several projects. Furthermore, their work again applies the proposed approach to open source data only.

Bird et al. (2009) observed that socio-technical network measures combined with dependency measures were stronger indicators of failures than dependency measures alone, but the differing aspects mentioned in paragraph before still remains.

Zerpies (2015) proposed a definition of patch-flow:

> Patch Flow is a model to show and measure code-level collaboration among different units of concern.

Unit of concern can be an individual, open source projects, or organizational unit; *code-level collaboration* refers to Source Lines Of Code (SLoC) changed.

We do not follow this definition due to two important aspects:

1. Large *units of concern* (in our case large orgunits) are able to contribute way more than smaller ones in total; we are missing a normalizing factor.
2. Also other metrics than SLOC could be helpful (e.g. Lines of Code (LoC), Comment Lines of Code (CLoC)).

6 The Patch-Flow Model

6.1 Terminology

Four entities play a central role in this thesis: Patch, User, Orgunit, and Project. But the this terminology is not the only one.

In the following each entity shall be described shortly.

Patch The pivot is the entity patch. It represents a code contribution to a project.

User Users are developers who author patches. A human developer may be assigned to multiple users.

Project A project is a encapsulation of source code files which are belonging to a unique product, service, or result.

Orgunit Users are belonging to and Projects are owned by orgunits.

Table 6.1 lists the different terminology coming from different perspectives, but having the same meaning.

Patch-Flow Terminology	Google's Terminology	Other's Terminology
Patch	Change	Commit, Contribution
User	User	Developer
Orgunit	Project, Orgunit	Team, Group
Project		Component, Package

Table 6.1: *Terminology used in this paper and their synonyms.*

Google does not distinguish between project and orgunit; every project is its own orgunit and vice versa.

6.2 Patch-Flow Graph

Definition 1. A **patch-flow graph** G is a directed graph $G = (O, P)$.

- $O = \{o_1, o_2, \ldots, o_n\}$ is a set of n nodes representing the orgunits. Each orgunit has two attributes:
  - a list of all developers belonging to this orgunit, and
  - a list of all project, the orgunit owns.
- $P = \{(o_i, o_j) \in O \times O\}$ is a set of ordered pairs each representing a patch from a developer of orgunit o_i committed to a project of orgunit o_j. A cost function $c : P \to \mathbb{R}^+$ weights each commit.

Figure 6.1 depicts an example patch-flow graph with three orgunits o_1, o_2, and o_3. All of them are exchanging code represented by weighted edges (e.g. in lines of code), except o_2 which does not contribute to projects of o_3.

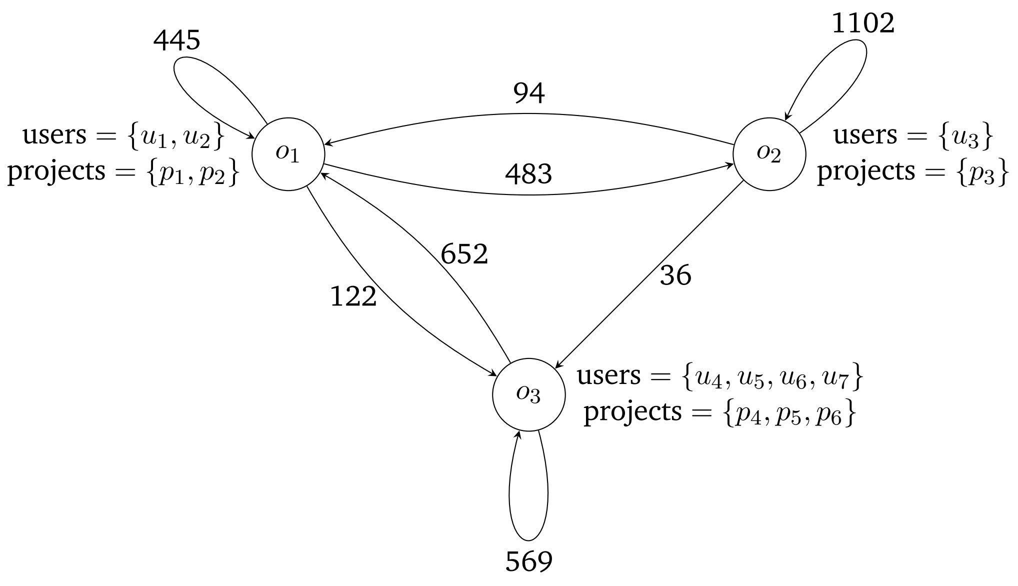

Figure 6.1: *An example patch-flow graph.*

Orgunits can contain subordinate orgunits and subsume all child orgunits' projects and users. So we are able to handle hierarchical organizational structures.

6.3 Patch-Flow Metrics

So we can define the incoming and outgoing patch-flow:

Definition 2. For a given orgunit o_i we define the **incoming patch-flow** as

$$\begin{aligned} f^+(o_i) &= \frac{1}{\sum\limits_{(o_j,o_i)\in P} c[(o_j,o_i)]} \cdot \sum_{\substack{(o_j,o_i)\in P\\ j\neq i}} c[(o_j,o_i)] \\ &= 1 - \frac{c[(o_i,o_i)]}{\sum\limits_{(o_j,o_i)\in P} c[(o_j,o_i)]} \end{aligned} \tag{6.1}$$

The incoming patch-flow is the amount of patches committed by non-orgunit members related to the total amount of patches which are committed by all committers.

Consequently, the definition for the outgoing patch-flow follows:

Definition 3. For a given orgunit o_i we define the **outgoing patch-flow** as

$$\begin{aligned} f^-(o_i) &= \frac{1}{\sum\limits_{(o_i,o_j)\in P} c[(o_i,o_j)]} \cdot \sum_{\substack{(o_i,o_j)\in P\\ i\neq j}} c[(o_i,o_j)] \\ &= 1 - \frac{c[(o_i,o_i)]}{\sum\limits_{(o_i,o_j)\in P} c[(o_i,o_j)]} \end{aligned} \tag{6.2}$$

The outgoing patch-flow is the amount of patches committed by orgunit members to foreign projects related to amount of all patches the orgunit committed.

In this thesis we consider the cost function which sums up all changed, edited, and added lines of code (including comment lines). However, there are many more cost functions c thinkable and applicable.

If $c[(o_i,o_i)] = c[(o_i,o_j)] = 1$, it corresponds to the in- and out-degree of a vertex in graph theory.

From this we can compute the incoming and outgoing patch-flow metric, e.g. for the orgunit o_2 presented in Figure 6.1:

$$\begin{aligned}
f^{+}(o_2) &= 1 - \frac{c[(o_2, o_2)]}{c[(o_1, o_2)] + c[(o_2, o_2)] + c[(o_3, o_2)]} \\
&= 1 - \frac{1102}{483 + 1102} \\
&= 0.3047 \\
f^{-}(o_2) &= 1 - \frac{c[(o_2, o_2)]}{c[(o_2, o_1)] + c[(o_2, o_2)] + c[(o_2, o_3)]} \\
&= 1 - \frac{1102}{94 + 1102 + 36} \\
&= 0.1055
\end{aligned}$$

This means in this certain orgunit o_2 we have about 30% of o_2's patches are coming from developers which are not members of o_2. About 10% of the patches of o_2 are not for the own orgunit, but for o_1 and o_3.

Following the definitions 2 and 3 we can define the patch-flow within a given patch-flow graph:

Definition 4. For a given patch-flow graph $G = (O, P)$ we define the **patch-flow** as

$$\begin{aligned}
F(G) &= 1 - \frac{\sum\limits_{o_i \in O} c[(o_i, o_i)]}{\sum\limits_{\substack{o_i \in O \\ (o_j, o_i) \in P}} c[(o_j, o_i)]} \\
&= 1 - \frac{\sum\limits_{o_i \in O} c[(o_i, o_i)]}{\sum\limits_{\substack{o_i \in O \\ (o_i, o_j) \in P}} c[(o_i, o_j)]} \qquad (6.3)
\end{aligned}$$

Because of the is-a relationship between subordinate and superordinate orgunits, we can apply the patch-flow metric also on organizational trees. However, at this stage we are not able measure vertical patch-flow, meaning patch-flow from a child orgunit to its parent or vice versa, although this could be an interesting metric, too.

7 Data Acquisition

In God we trust; all others must bring data.

— **William Edwards Deming**

We collected the data as follows using an automated tool:

1. **Collect all patches** of a selected year from the Google Perforce repository.
2. **Extract all users** which are authors of these patches.
3. **Sample 2,500 users randomly**.
4. **Filter patches** according to the randomly selected users in step 3.
5. **Enhance patches** and their containing users data with information about orgunit memberships and seniorities coming from an internal Google system.
6. **Anonymize patches** by removing identifying names, etc.
7. **Store patches** in a JSON file.

Figure 7.1 adds the perspective of the data flow between the different processing steps to the workflow explained above.

This was repeated for every year listed in the constraints.

7.1 Constraints

The information to be crawled are restricted to the data model presented in the previous section 7.3.

Only patches are selected, if they

- are committed and accepted in the years 2007, 2009, 2011, and 2013,
- are committed by 2,500 Google developers, randomly selected for each year, and
- are committed to the Google internal Perforce repository.

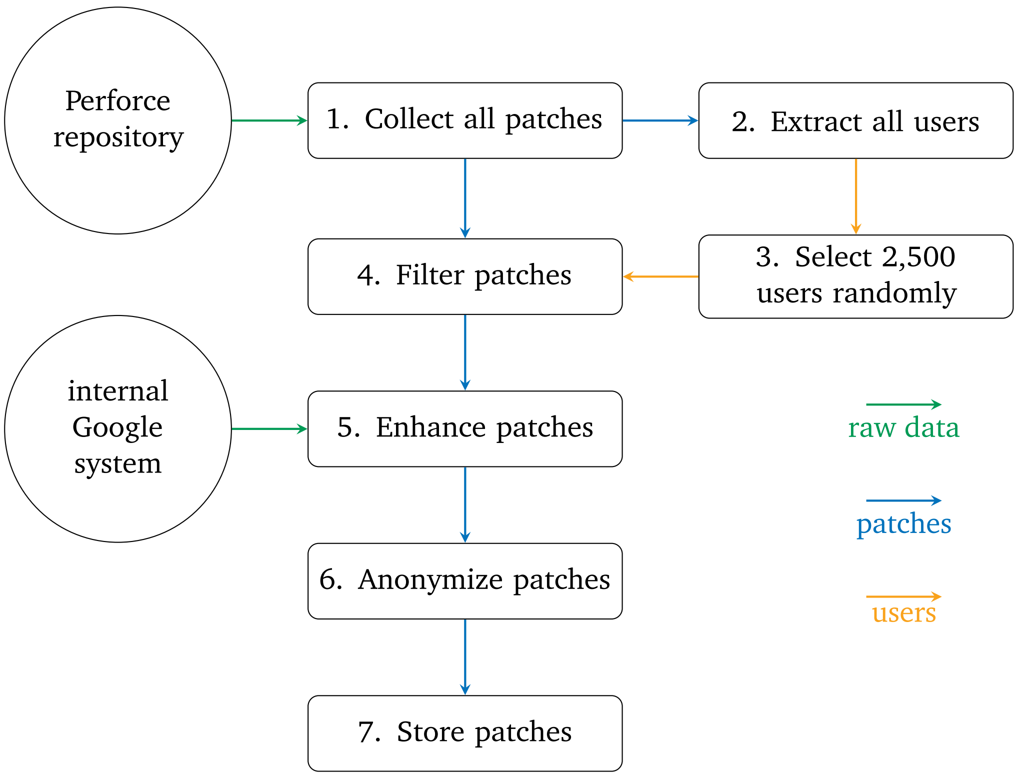

Figure 7.1: *Crawling work flow for one selected year to collect. Circles represent external data sources and rectangles processing steps, encapsulated in modules within the tool.*

Furthermore, only Java, JavaScript, C/C++, Python, and Go source code files in a patch are considered.

7.2 Tool

For collecting these data we developed a crawler, because no crawler for Perforce existed, although there are many open source crawler for git, CVS, or SVN (e.g. Anbalagan and Vouk (2009), German and Mockus (2003), Gousios et al. (2008), Robles et al. (2004), Voinea and Telea (2006), Xie et al. (2006)), and Google uses a non-standard version of Perforce and restricted the standard Perforce API.

The tool which crawls through Google's Perforce repository seizes the workflow presented in previous section and Figure 7.1 and encapsulates each processing step into a component. This leads to an pipes and filters architectural pattern. The tool is implemented in Python and Go. Due to the large amount of data – the Google repository contains about 86 Terabyte of source code – the implementation required a strong focus on performance using parallel computing and as less queries as possible to minimize the server load footprint.

7.3 Data Model

After gathering the data we obtained a data structure, which holds required information about the patch-flow model (patches, users, orgunit/projects) and additional helper container file. In general, Google does not distinguish between a project and an orgunit. So we are not required to elaborate the semantical difference between these entities.

A patch contains one or more files, which encapsulates information for each file about the used programming language as well as lines of code added and removed. Additionally, four relations are stored: *authors*, *reviews*, *contains*, and *belongs to*. The relation *is parent of* can be derived from the path structure.

The tool stores all patches in a JSON file. The resulting data structure is shown in Figure 7.2.

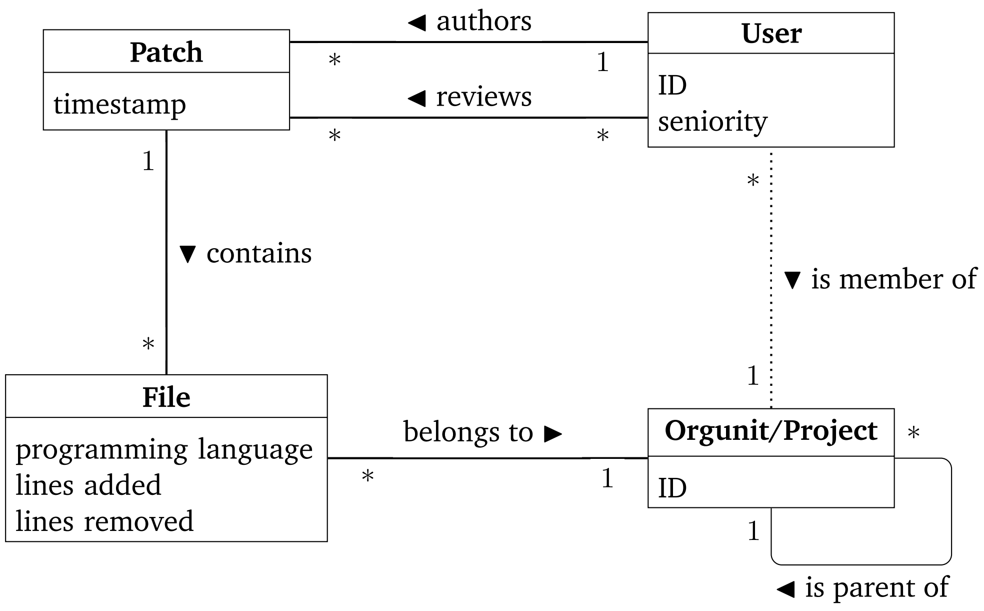

Figure 7.2: *The data model as UML diagram with necessary, but not existing relation between user and orgunit/project.*

7.4 Preprocessing

Information about the membership to orgunits of a user is not available (marked with a dotted lines in Figure 7.2). However, this information is crucial for measuring the patch-flow.

So we formulated the optimization problem for each user with a sampling period of one month

$$\underset{o}{\operatorname{argmax}}\, w_i \cdot c_{u,o} \tag{7.1}$$

where $c_{u,o}$ is the amount of contribution of a user u to a project of orgunit o. So we select the orgunit with the highest weighted costs (e.g. lines of code). The weight w_i can be one of the following:

$$w_0 = (d+1)^0 = 1 \tag{7.2}$$
$$w_1 = (d+1)^{-1} \tag{7.3}$$
$$w_2 = (d+1)^{-2} \tag{7.4}$$
$$w_3 = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \text{ with } \mu = 0, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d-\mu)^2} \tag{7.5}$$

The standard deviation is estimated by the maximum-likelihood estimation (MLE) method. If $\operatorname{argmax}_o w_i \cdot c_{u,o}$ does not obtain a unique orgunit, nodes deeper in the project tree are preferred.

Figure 7.3 illustrates the different weights on a example tree, assuming a test for the orgunit a.

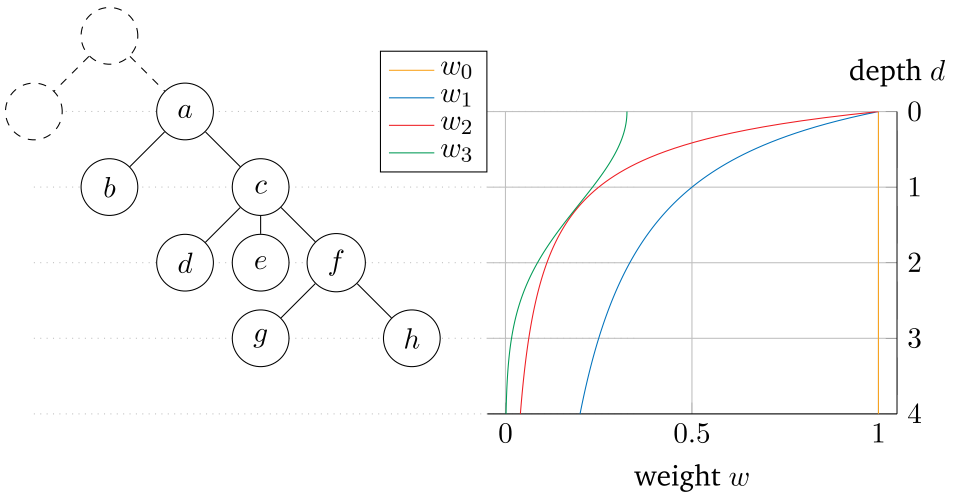

Figure 7.3: *An example segment of a project/orgunit tree weighted by* $w_0, w_1, w_2,$ *and* w_3.

Until now, we were not able to evaluate the accuracy of the cluster approach with these weights, because no test samples are available.

7.5 Mining Chromium

Beyond the direct scope of this thesis we mined in a open source project *Chromium*. This project is led by Google and uses a similar collaboration framework between the developers as Google does internally[1].

Instead of Perforce, Chromium uses a git based system, enhanced by a review tool called *Rietveld*. So we adapted the crawler to be applicable to this new setup. The resulting data model stored in JSON matches to the data coming from Google internally. This enables us to use the same metrics to both data sets.

We constrained the mining to the same restrictions as we did on the Google internal data, except the following aspects, which are now not necessary for publicly accessible data: anonymizing of the dataset, random selection of users, selected years. Instead we crawled all user with their Chromium user name and all patches since the birth of the Chromium open source project in September 2008.

Obviously Chromium is an open source project. Some of the users are voluntaries or are delegated from other companies. The Chromium project offers to their contributors a `@chromium.org` mail address. However, we are interested in the patch-flow within the borders of Google among orgunit borders. So we focus on users with `@google.com` mail addresses.

[1]Personal communication with a Google employee

Results

8

Errors using inadequate data are much less than those using no data at all.

— **Charles Babbage**

After crawling and preprocessing we are now able to apply the patch-flow metric to the Chromium data set.

As explained in detail in chapter 6.3 three different metrics measuring the patch-flow can be evaluated: The patch-flow of the whole Chromium project, the incoming patch-flow and outgoing patch-flow of an orgunit. For the incoming and outgoing patch-flow metric we selected the three largest projects/orgunits with respect to number of patches: `src/`, `src/gpu/`, `src/pdf/`.

In general, we apply a sampling period of one month.

8.1 Chromium Patch-Flow Graph

For the patch-flow in the whole graph we compare the different weights (w_0, w_1, w_2, w_3). As we can see in Figure 8.1 beside one peak in w_3 the results are close: the average range of the patch-flow for a given month m is given by

$$\frac{1}{|M|} \cdot \sum_{m \in M} \underbrace{\max\{f^m_{w_0}, f^m_{w_1}, f^m_{w_2}, f^m_{w_3}\} - \min\{f^m_{w_0}, f^m_{w_1}, f^m_{w_2}, f^m_{w_3}\}}_{R_m} \approx 0.0289 \; . \qquad (8.1)$$

This means, in average all weights differ only in a range of 2.89 %. However, this does not allow us to make any assumptions about the correctness; they all can be wrong.

The large spike in November 2009 requires more investigation. A look into the data shows, that user `sehr@google.com` contributed 2672 LoC to orgunit `src/third_party/npapi/` and 2882 LoC to `src/`. Depending on the weights, this

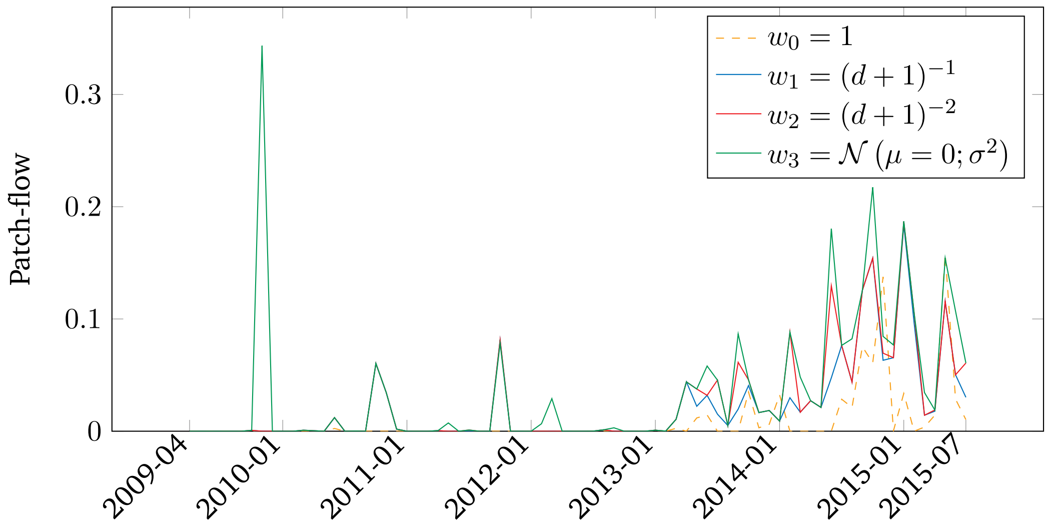

Figure 8.1: *Patch-flow for Chromium considering Google developers only with a sampling period of one month.*

user will be assigned to one of these orgunits – if we are applying weight w_1 we obtain orgunit `src/` for this user.

However, 2882 LoC are 26.33 % of all contributions in this month. So the orgunit assignment of user `sehr@google.com` effects the patch-flow heavily.

This proportion of 26.33 % recurs in Figure 8.5 which depicts the outgoing patch-flow for the orgunit `src/` because of the same reason.

8.2 Incoming Patch-Flow For Selected Orgunits

Figure 8.2 depicts the monthly incoming patch-flow for the three largest orgunits with respect of amount of committed patches: `src/`, `src/pdf/`, and `src/gpu/`. For practical reasons we apply the weight w_1, although there are no evidences that any of these weights are superior.

In 2014 almost 100% of all patches are coming from developers which are not assigned to `src/pdf/`. But Figure 8.3 shows, that there are only 4 contributors for this project at maximum and one developer contributes most in the sampling period (one month). If this contributor is assigned to a different orgunit than `src/pdf/`, we get this large incoming patch-flow. But this large incoming patch-flow suggests that the applied weighting with w_1 could not be appropriated.

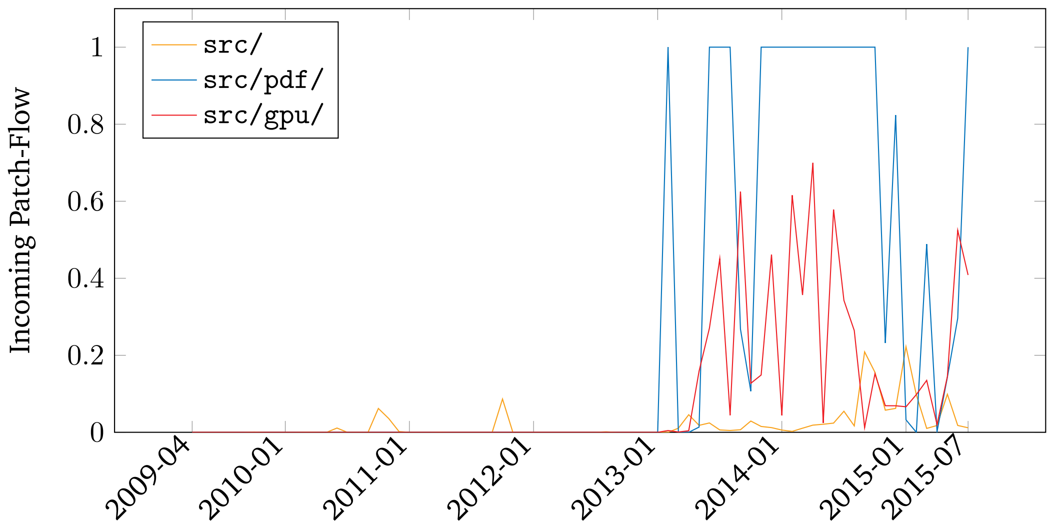

Figure 8.2: *Incoming patch-flow for the selected orgunits `src/`, `src/pdf/`, and `src/gpu/` applying a sampling period of one month.*

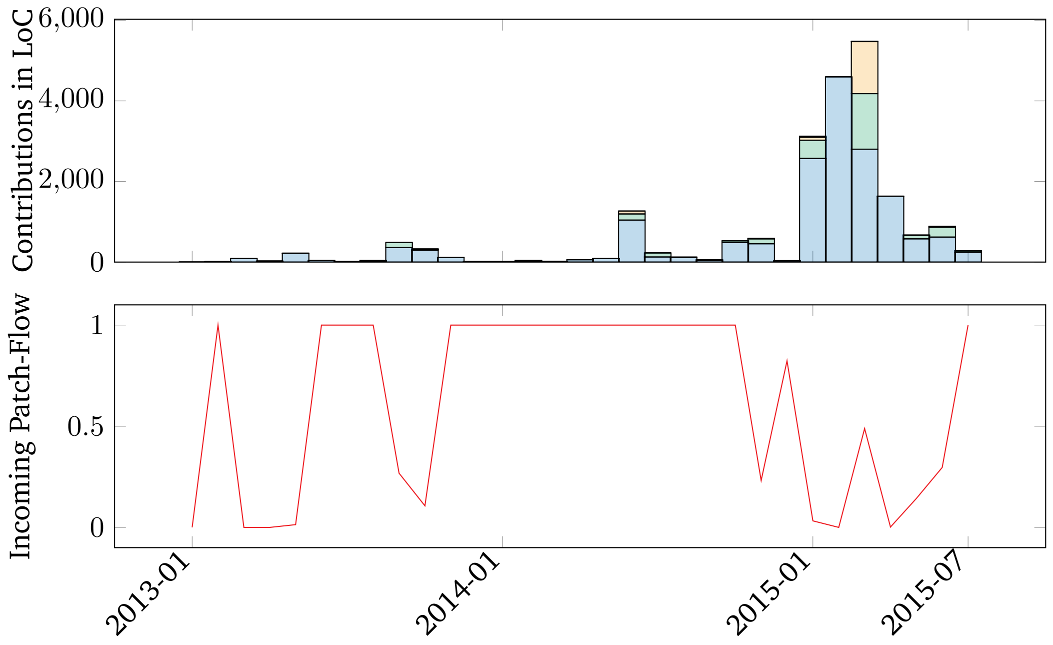

Figure 8.3: *Distribution of the contributed lines of code by all four contributors per month in comparison with the incoming patch-flow for orgunit `src/pdf/`.*

If we apply the same settings to orgunit `src/gpu/`, we see in Figure 8.4 more contributors and contributions reduce the error.

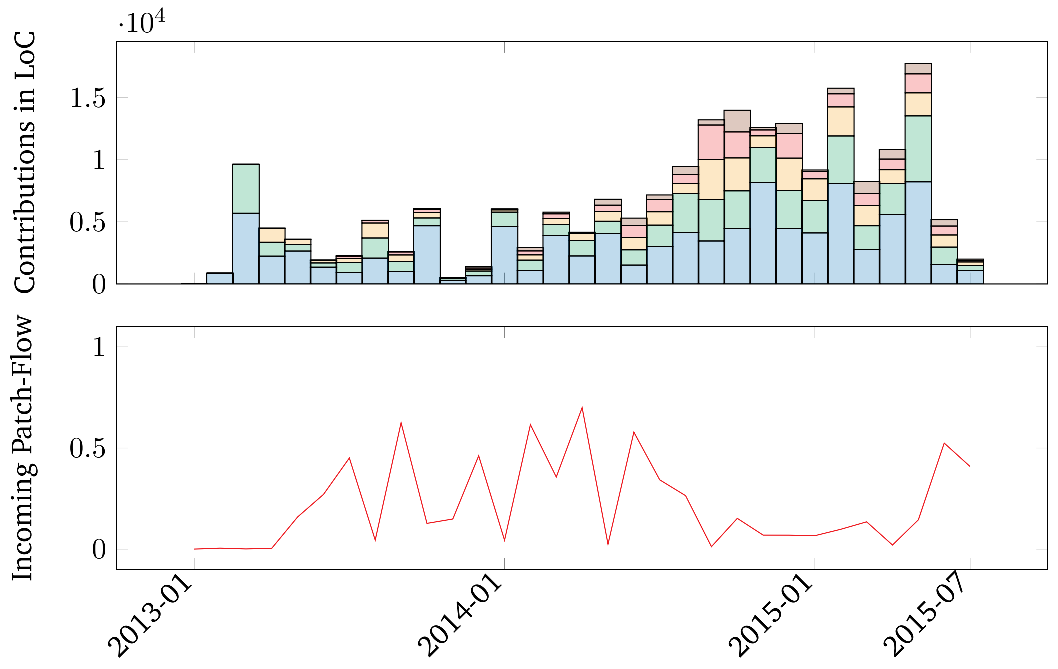

Figure 8.4: *Distribution of the contributed lines of code by top five contributors per month in comparison with the incoming patch-flow for orgunit* `src/gpu/`*.*

8.3 Outgoing Patch-Flow For Selected Orgunits

The outgoing patch-flow is presented in Figure 8.5.

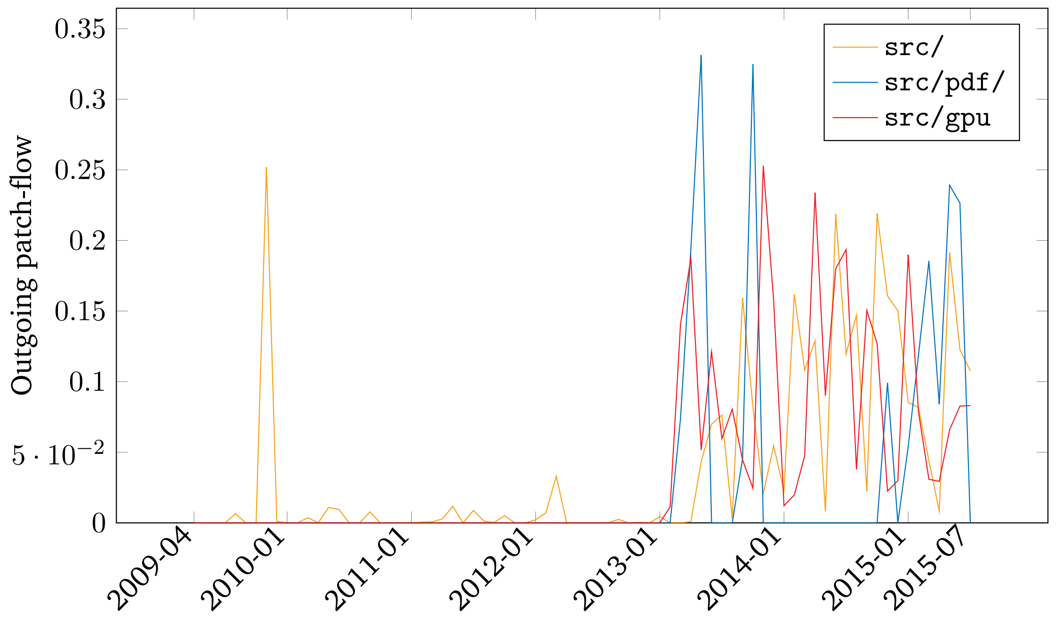

Figure 8.5: *Outgoing patch-flow for the selected orgunits* `src/`*,* `src/pdf/`*, and* `src/gpu/` *applying a sampling period of one month.*

Some effects can be recognized: There is a peak, and as in the patch-flow for the whole graph and the incoming patch-flow we can see an increasing collaboration between different orgunits starting in 2013.

We were not able to find a reason in the Chromium history for this large increase of patch-flow starting in 2013.

9 Limitations

> *With insufficient data it is easy to go wrong.*
>
> — **Carl Sagan**

9.1 Google Data

If there had been the Google data set, we would have had also limitations: No historical information about orgunit memberships of users are available. This information has to be restored by clustering. Depending on the size and quality of the training set this approach has varying accuracy and some samples will be clustered wrong. So an error will be introduced.

The sample of 2,500 Google developers corresponds about 10% of all Google engineers. Selecting them randomly makes this sample representative.

Due to a yearly re-sampling we are not able to observe the evolution of developers within Google.

9.2 Chromium Data

There are technical constraints while mining Chromium:

- Only public patches to review are accessible. There is an unknown number of private patches, which are restricted to reviewers and selected users.
- In Chromium C++ is widely used (95% of all patches), there are no Go source code files. So we would not be able to make any claims about the effect of project's programming language.
- Only Chromium packages are considered. Many patches are directed to packages that are included in Chromium, but they are independent projects. This reduces the amount of data available and we are not able to detect this communication among open source project boundaries.

Furthermore, there are issues with the review software tool Rietvield:

- The code review tool Rietvield has an unexpected behavior where adding a line can lead to a negative number stored (e.g. `https://codereview.chromium.org/api/194883004/1`).
- Some users have usernames with typos (e.g. `caryclark1;reed@google.com`).

Filtering the Google employees by their `@google.com` mail address does not obtain all contributions of the them: there are some `@chromium.org` users which are quite certain Google employees (e.g. `bashi@chromium.org` and `bashi@google.com` or `ager@chromium.org` and `ager@google.com`).

Additionally, this filtering of exclusively Google employees reduces

- the total number of users from 799 to 152,
- the total number of patches from 36271 to 6777,
- the total edited lines from 10889225 to 2618024, and
- the start month from September 2008 to April 2009.

The latter fact implies that there are many Google employees working on Chromium, but using a `@chromium.org` mail address, because it is quite unlikely that no Google developer contributes from 2. September 2008 (where the first patch was approved) to April 2009.

We cannot preclude that Google employees use their `@google.com` mail address and account to contribute as volunteers and not on behalf of their employer. Then any assignment to a Chromium orgunit would be wrong.

Table 9.1 shows that at least 78% of all users can be assigned to one orgunit over the whole period. This implies for the rest, that there is an uncertainty, or the user switched his orgunit. So we have a trade-off between the flexibility of orgunit switches and and overfitting. Investigating this issue requires test samples.

Distance Metric	**Number of users with**		
	one orgunit	**two orgunits**	**$>$ two orgunits**
w_0	136	14	1
w_1	126	22	4
w_2	125	22	5
w_3	119	25	8

Table 9.1: *Number of user with one, two, or more orgunits, applying the clustering algorithm with weights w_0, w_1, w_2, w_3 and a sampling period of one month over the whole period.*

But 78.22% of all patches by Google employees and thereby 88.21% of all changed lines of code are committed to the top level project `src/`. This large a-priori probability has a great effect on the clustering results, because assigning all users to this top level project/orgunit will obtain a recognition rate of 78% and 88%, respectively. A necessary normalization would even more reduce the available amount of data.

The sampling frequency implies that only month-wise orgunit changes can be captured. This introduces fuzziness.

However, we are still not able to estimate the quality of our assigning of users to orgunits. Maybe this also differs from Chromium to Google internal software development.

And even if we would be able to do so, the 152 developers and subset of all patches are neither representative for Google having several ten-thousands developers nor for the Chromium project.

Additionally, Sadowski, van Gogh, Jaspan, Söderberg, and Winter (2015) stated that Chromium (and Android) are developed independently.

10 Future Work

In this chapter we want to outline some of our ideas for the future, we were not able to consider within the scope of this thesis.

10.1 Patch-Flow Model

At this stage the patch-flow metrics does not capture the hierarchical structure of the orgunit structure. It would be helpful to have a second dimension aside from contributions: direction or distance in the orgunit tree.

Not only because we are working on Google data, a comparison to the *PageRank* algorithm applied on the patch-flow graph to measure the importance of a certain orgunit would be interesting.

There is an affinity to network theory, which is not considered in detail in this thesis. Lopez-Fernandez and Robles (2004) presented some interesting properties for their module network, known from the social network theory (degree, distance centrality, betweenness centrality, etc.), which is a feasible starting point.

10.2 Clustering in Trees

Because there is no historical data available about the orgunit membership of users, we need a clustering algorithm, based on the tree structure of the projects/orgunits at Google. The applied approach of different metrics are rudimental steps to fit in the hierarchical structure of orgunits and projects. Schkolnick (1977), Hambrusch, Liu, and Lim (2000), Maravalle, Simeone, and Naldini (1997), Yuruk, Mete, Xu, and Schweiger (2009) suggest solutions for this problem, they are not considered in this paper.

Beside that we have to validate our clustering approach. This can be done by hand for a suitable test sample size of 20 randomly selected users.

Conclusions

11

In this thesis we presented a graph theory based model for measuring the phenomena of patch-flowing among the borders of orgunits.

We mined in two different sources, the Google internal repository and the open source project Chromium.

For both data sets there is a major drawback: No historical information about orgunit membership of any user is available. Therefore, we developed a clustering algorithm. However, because there are also no test samples, we are not able to estimate the error of the applied orgunit clustering.

The Google internal data set was not released in time. So we were forced to use Chromium data set, although it cannot be assumed to be representative, because – beside minor issues –

- the sample size is too small (152 Google developers),
- as already mentioned the orgunit clustering has an error of unknown magnitude,
- the Chromium project is just one specific out of thousands of projects Google is working on, and cannot be assumed to be representative,
- and by intention, Chromium is an open source project.

Due to this limitations we are not able to answer the initial research question to which extent patch-flow exists within Google. Consequently, we are also not able to answer the second research question which factors affect or correlate with the quantity of patch-flow within Google.

Although we are able to measure and quantify the patch-flow for the whole patch-flow graph and selected orgunits, the used data set cannot be assumed to be representative for the internal Google development.

12 Acknowledgement

This work benefits heavily from the input and contributions of three person. Many thanks to

- my supervisor Max Capraro for the uncountable hours and mails of discussions, his remarks, contributions, and expertise,
- my anchor at Google Manuel Klimek for his comprehensive contributions to the crawling and the data structure, the data set itself, his patience, and his respectful, kind way of collaboration, and, last, but not least,
- my professor Prof. Dr. Dirk Riehle for his unconditional support, the pleasant working atmosphere, the cookies, and the freedom for my work I enjoyed.

It was a pleasure to work with them, as a team and individually.

Part III

Elaboration of Research Chapter

13 Metric for Clustering Orgunits without Prior Knowledge

For a dataset without any information about the orgunit membership (number of orgunits, any samples for testing), we implemented a hierarchical clustering algorithm, based on histograms for contributions to different orgunits for all users.

The Jensen-Shannon-divergence (JSD) evaluates the similarity of such two distributions and is given by

$$d_{\text{JSD}}\left(H, H'\right) = \sum_{m=1}^{M} H_m \log \frac{2 \cdot H_m}{H_m + H'_m} + H'_m \log \frac{H'_m}{H'_m + H_m} \tag{13.1}$$

where H and H' are the histograms with M bins to be compared and H_m is the m-th bin of the H (Deselaers, Keysers, & Ney, 2008).

Figure 13.1 illustrates this approach, where H is a histogram of the relative frequency of code contribution to M projects, where 0 indicates no and 1 a high divergence.

What we can see that more than $2/3$ of the users have a very similar code contribution. This main part of users are contributed to the top level project, which is at the same time the largest project.

Beside computational challenges $(O\left(N^2/(2 \cdot f)\right)$ for N users and f sampling frequency), we would be able to cluster user with a similar contribution behavior, but not assign them to a existing orgunit.

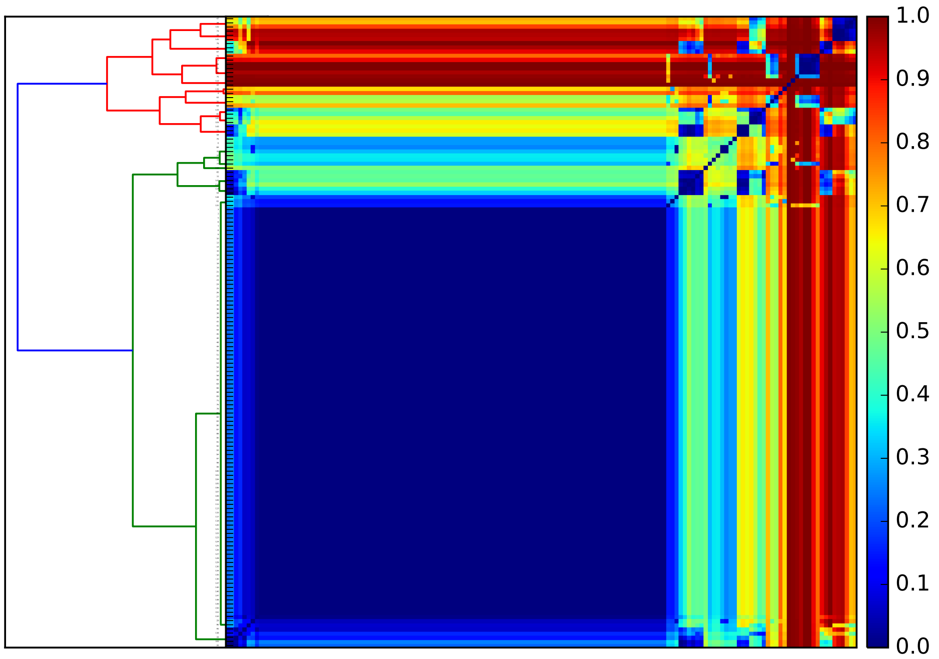

Figure 13.1: *A hierarchical clustering approach using the Jensen-Shannon divergence for all 152 available Chromium contributors having a* `@google.com` *mail address.*

Part IV

References

Anbalagan, P. & Vouk, M. A. (2009). On Mining Data Across Software Repositories. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 171–174.

Begel, A. & Zimmermann, T. (2014). Analyze This! 145 Questions for Data Scientists in Software Engineering. *Proceedings of the 28th International Conference on Software Engineering*, 12–13.

Bhattacharya, P., Iliofotou, M., Neamtiu, I., & Faloutsos, M. (2012). Graph-Based Analysis and Prediction for Software Evolution. In *34th International Conference on Software Engineering* (pp. 419–429). IEEE.

Bird, C., Gourley, A., Devanbu, P. T., Gertz, M., & Swaminathan, A. (2006). Mining Email Social Networks. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 137–143.

Bird, C., Murphy, B., Nagappan, N., & Zimmermann, T. (2011). Empirical software engineering at Microsoft Research. *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, 143–150.

Bird, C., Nagappan, N., Gall, H. C., Murphy, B., & Devanbu, P. T. (2009). Putting It All Together: Using Socio-technical Networks to Predict Failures. *20th International Symposium on Software Reliability Engineering*, 109–119.

Canfora, G. & Cerulo, L. (2005). Impact Analysis by Mining Software and Change Request Repositories. *Software Metrics, 2005. 11th IEEE International Symposium*, 29.

Colaço Jr, M., Mendonça, M. G., & Rodrigues, F. (2009). Mining Software Change History in an Industrial Environment. *XXIII Brazilian Symposium on Software Engineering*, 54–61.

Conway, M. E. (1968). How do committees invent? *Datamation*, *14*(4), 28–31.

Deselaers, T., Keysers, D., & Ney, H. (2008). Features for Image Retrieval: an Experimental Comparison. *Information Retrieval*, *11*(2), 77–107.

German, D. M. & Mockus, A. (2003). Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering* (pp. 63–67). Citeseer.

Gousios, G., Kalliamvakou, E., & Spinellis, D. (2008). Measuring Developer Contribution From Software Repository Data. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 129–132.

Gousios, G. & Spinellis, D. (2012). GHTorrent: Github's data from a firehose. In *9th IEEE Working Conference on Mining Software Repositories* (pp. 12–21). IEEE.

Hambrusch, S. E., Liu, C.-M., & Lim, H.-S. (2000). Clustering in Trees: Optimizing Cluster Sizes and Number of Subtrees. *Journal of Graph Algorithms Applications*, *4*(4), 1–26.

Hassan, A. E. (2006). Mining Software Repositories to Assist Developers and Support Managers. *22nd IEEE International Conference on Software Maintanance*, 339–342.

Hassan, A. E. (2008). The Road Ahead for Mining Software Repositories. In *Frontiers of Software Maintenance* (pp. 48–57). IEEE.

Heller, B., Marschner, E., Rosenfeld, E., & Heer, J. (2011). Visualizing Collaboration and Influence in the Open-Source Software Community. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 223–226.

Huang, S.-K. & Liu, K.-m. (2005). Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1–5.

Jermakovics, A., Sillitti, A., & Succi, G. (2013). Exploring Collaboration Networks in Open-Source Projects. *Open Source Systems*.

Kagdi, H., Collard, M. L., & Maletic, J. I. (2007, March). A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, *19*(2).

Lopez-Fernandez, L. & Robles, G. (2004). Applying Social Network Analysis to the Information in CVS Repositories. *International Workshop on Mining Software Repositories*.

Maravalle, M., Simeone, B., & Naldini, R. (1997). Clustering on trees. *Computational Statistics & Data Analysis*.

Menzies, T., Bird, C., Zimmermann, T., Schulte, W., & Kocaganeli, E. (2011, November). The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining. In *Proceedings of the international workshop on machine learning technologies in software engineering*. ACM.

Nagappan, N., Murphy, B., & Basili, V. R. (2008). The Influence of Organizational Structure on Software Quality: An Empirical Case Study. *Proceedings of the 28th International Conference on Software Engineering*, 521–530.

Ohira, M., Ohsugi, N., Ohoka, T., & Matsumoto, K.-i. (2005). Accelerating Cross-Project Knowledge Collaboration Using Collaborative Filtering and Social Networks. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1–5.

Pinzger, M., Nagappan, N., & Murphy, B. (2008). Can developer-module networks predict failures? *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2–12.

Rigby, P. C. & Hassan, A. E. (2007). What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 23.

Robles, G., Gonzalez-Barahona, J. M., & Ghosh, R. A. (2004). GlueTheos: Automating the Retrieval and Analysis of Data From Publicly Available Software Repositories. In *Proceedings of the International Workshop on Mining Software Repositories* (pp. 28–31).

Sadowski, C., van Gogh, J., Jaspan, C., Söderberg, E., & Winter, C. (2015, May). Tricorder: Building a Program Analysis Ecosystem. In *Proceedings of the 37th International Conference on Software Engineering*. IEEE Press.

Schkolnick, M. (1977). A Clustering Algorithm for Hierarchical Structures. *ACM Transactions on Database Systems*, *2*(1), 27–44.

Thung, F., Bissyandé, T. F., Lo, D., & Jiang, L. (2013). Network Structure of Social Coding in GitHub. *17th European Conference on Software Maintenance and Reengineering*, 323–326.

Vanya, A., Klusener, S., Premraj, R., Van Rooijen, N., & Van Vliet, H. (2011). *Identifying and investigating evolution type decomposition weaknesses*. Springer.

Voinea, L. & Telea, A. (2006). Mining Software Repositories with CVSgrab. In *Proceedings of the 2006 International Workshop on Mining software Repositories*.

Weissgerber, P., Pohl, M., & Burch, M. (2007, May). Visual Data Mining in Software Archives To Detect How Developers Work Together. In *Proceedings of the 29th International Conference on Software Engineering*. IEEE.

Xie, X., Poshyvanyk, D., & Marcus, A. (2006, October). Visualization of CVS Repository Information. In *Proceedings of the 13th Working Conference on Reverse Engineering*. IEEE.

Xu, J., Gao, Y., & Christley, S. (2005). A topological analysis of the open souce software development community. *Proceedings of the 38th Hawaii International Conference on System Sciences*, 198a–198a.

Yuruk, N., Mete, M., Xu, X., & Schweiger, T. (2009). AHSCAN: Agglomerative Hierarchical Structural Clustering Algorithm for Networks. *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 72–77.

Zerpies, M. (2015). *Measuring Patch Flow on GitHub* (Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg).

Zimmermann, T. & Nagappan, N. (2008). Predicting Defects Using Network Analysis on Dependency Graphs. *Proceedings of the 28th International Conference on Software Engineering*, 531–540.