GitHub

Activity

40m+

developers on GitHub, including 10M new users in 2019.*

44m+

repositories created in the last year—and 44% more developers created their first repository in 2019 than in 2018.*

87m+

pull requests merged in the last year—and 28% more developers opened their first pull request in 2019 than in 2018.*

20m+

issues closed in the last year. That's a lot of decisions made, bugs fixed, and boxes checked.*

Corporate use

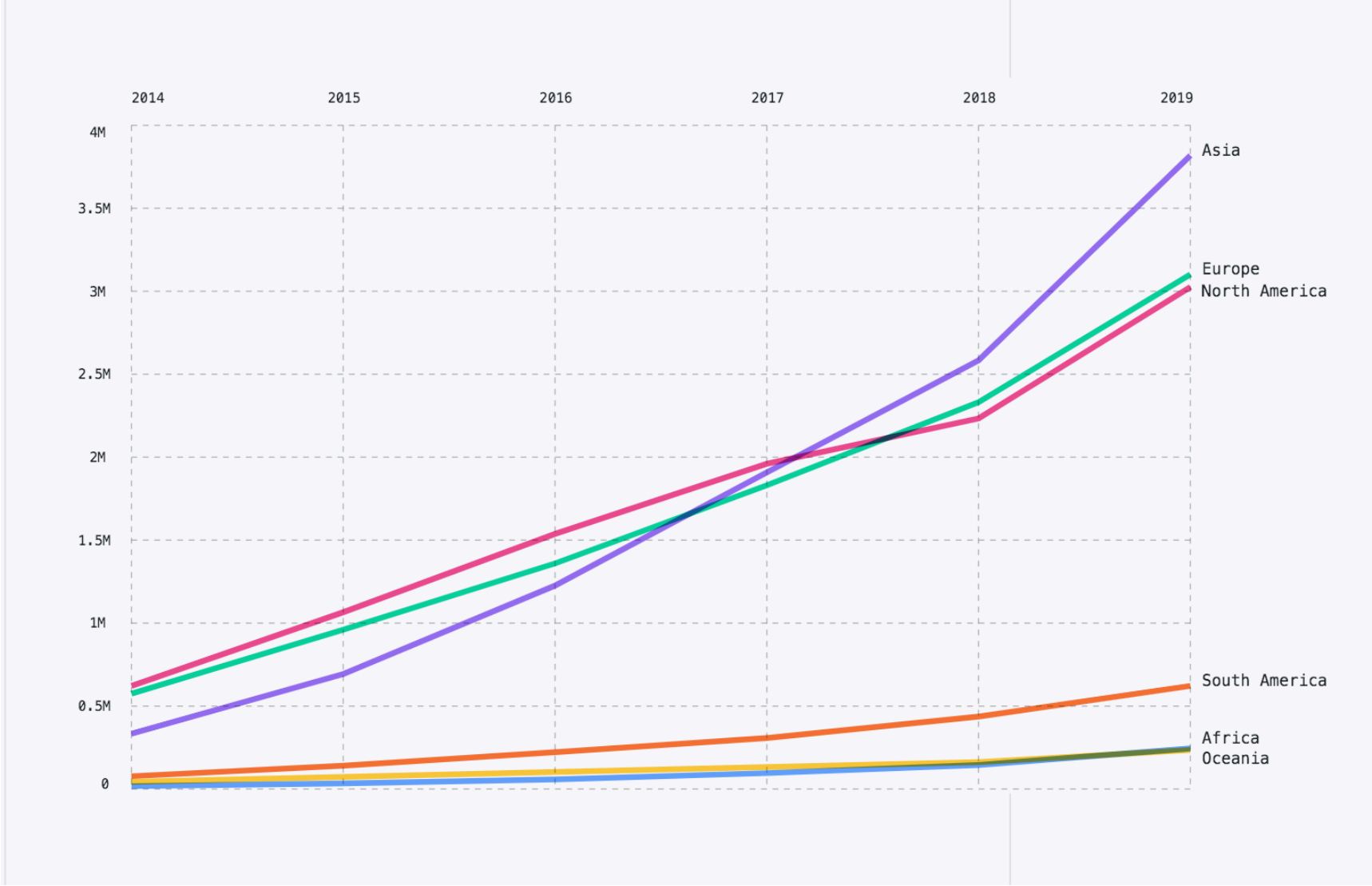
2.9m+

organizations brought people together in public and private repositories. GitHub Enterprise Cloud users alone worked in organizations from 70+ different countries around the world this year.*

35

of Global Fortune 50 companies have made contributions to open source in the last year, and 29 are building the software behind their businesses on GitHub Enterprise.*

Where they come from



CI/CD

Secure development

Code review

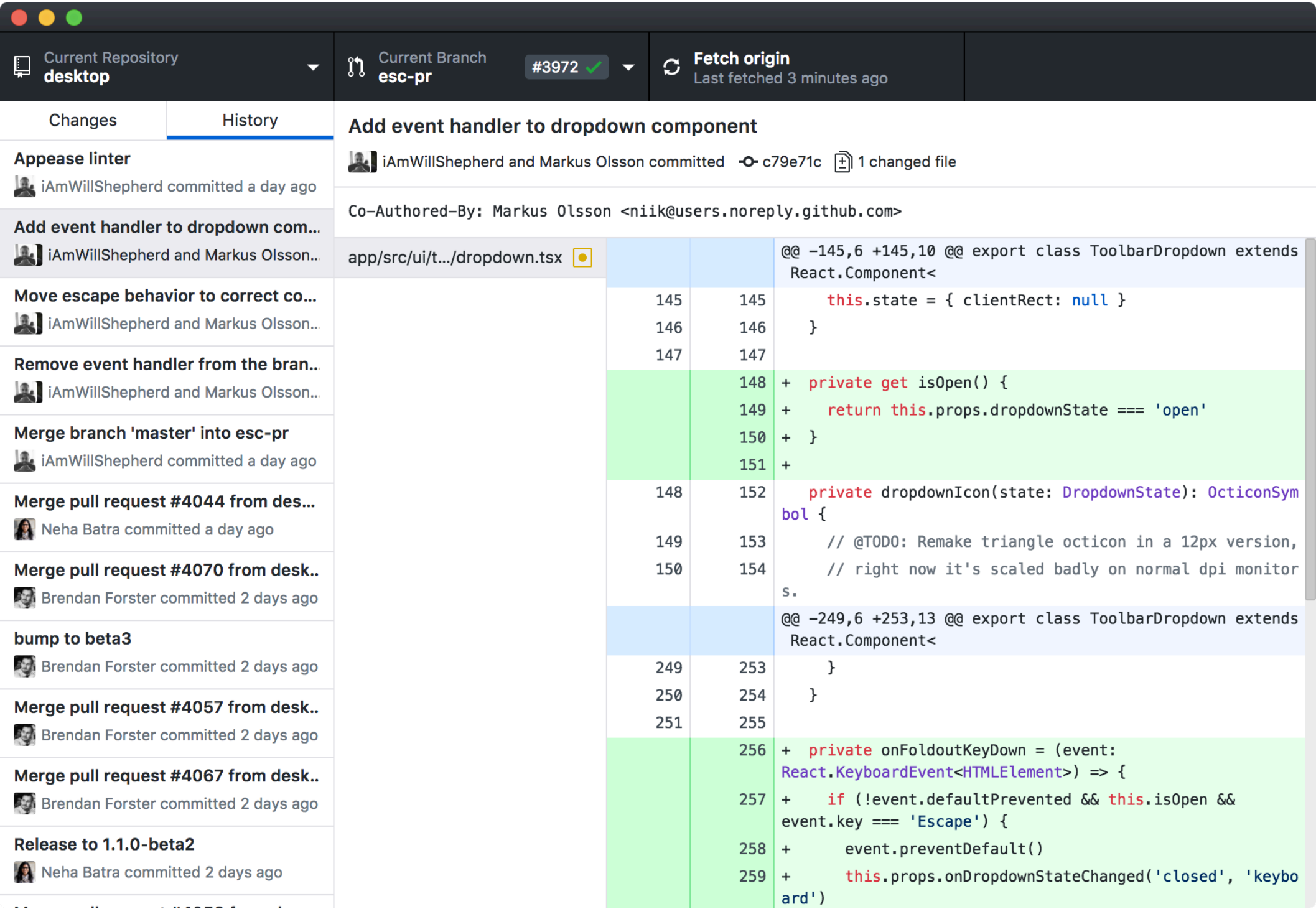
Apps

Hosting

Project management

Team management

Github Desktop



Open Source Style Collaboration using GitHub

Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub

Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer and Daniel M. German
Department of Computer Science, University of Victoria
Email: ikaliam@uvic.ca, danielad@uvic.ca, kblincoe@acm.org, lsinger@uvic.ca, dmg@uvic.ca

***Abstract*—Researchers are currently drawn to study projects hosted on GitHub due to its popularity, ease of obtaining data, and its distinctive built-in social features. GitHub has been found to create a transparent development environment, which together with a pull request-based workflow, provides a lightweight mechanism for committing, reviewing and managing code changes. These features impact how GitHub is used and the benefits it provides to teams' development and collaboration. While most of the evidence we have is from GitHub's use in open source software (OSS) projects, GitHub is also used in an increasing number of commercial projects. It is unknown how GitHub supports these projects given that GitHub's workflow model does not intuitively fit the commercial development way of working. In this paper, we report findings from an online survey and interviews with GitHub users on how GitHub is used for collaboration in commercial**

[21], [26]. Most commercial organizations do not allow public access to their code base, so they use private repositories or deploy GitHub's enterprise version on their own servers. We are not aware of any research studies that have investigated the support GitHub provides to these projects.

Understanding collaboration practices in teams using GitHub to produce proprietary software in commercial organizations is equally interesting and important because traditional practices of closed source projects do not seem tailored for the GitHub development environment which promotes visibility of work and self-assignment of tasks. Traditionally, developer participation in closed source projects is by invitation only - developers are assigned tasks and, in extreme cases, are only

ICSE 2015

RQ: What practices do commercial software projects follow in conjunction with GitHub's features to collaborate? What is the effect on their collaboration?

Method

- Survey:
  - 1000 users, 240 responses
- Interviews
  - 30 interviews (24 commercial projects, 6 OSS)

TABLE I. CODED SURVEY RESPONSES SUMMARY.

Survey item	**Survey responses**	**240**
GitHub use	Primarily for collaboration	148 (62%)
	Primarily for solo projects	90 (37.5%)
	N/A	2 (0.5%)
Reason for adopting GitHub	To contribute to OSS or share code	67 (28%)
	Because they used git already	52 (22%)
	To collaborate with others	35 (15%)
	To host projects and store files	26 (11%)
	GitHub's popularity	24 (10%)
	GitHub's interface / ease of use	24 (10%)
	GitHub was adopted at work	8 (3%)
	Other reasons	4 (2%)
GitHub's effect on develop-ment	Adopt branching workflow	53 (22%)
	Be conscious about writing better code	44 (18%)
	Adopt/learn best practices	43 (18%)
	Write more code / Contribute	41 (17%)
	Same effect as using any DVCS	31 (13%)
	N/A	28 (12%)

TABLE II. INTERVIEWEES' AND PROJECTS' DESCRIPTIVE CHARACTERISTICS.

	Interviewees	**30**
GitHub use	Primarily for job-related projects	24 (80%)
	Primarily for OSS contribution	6 (20%)
Job situation	Professional developers	25 (83%)
	Managers/CTOs (also do development)	4 (13%)
	Developers in internship	1 (4%)
	Job-related projects	**24**
Distri-bution	Distributed	16 (67%)
	Collocated	8 (33%)
	Median team size	**7**

TABLE III. INTERVIEW QUESTIONS CORRESPONDING TO COLLABORATION ELEMENTS AND HOW GITHUB SUPPORTS THEM.

Collaboration element	**Sub-area**	**Question**
Coordination	Definition	What does coordination mean to you? How would you define it?
	Example	Can you give an example of coordination in your team?
	Coordination Needs	What are the occasions that you find it essential to coordinate with your team?
	Issues/Solutions	Can you remember an issue with coordination? How did you resolve it? Do you have any conventions on coordination you follow in your team?
Task division	Criteria	How do you decide how to split the work between team members?
Awareness	Activity	How do you keep track of activity in your project?
	People/Artifacts	Do you prefer to track the activity of people or artifacts? Why?
	Maintaining Awareness	How do you stay aware of actions or changes in the artifacts?
	Challenges/Problems	Does it get too much? Is something missing?
Communication	Communication needs	What are the occasions that you find it essential to communicate with your team?
	Challenges/problems	Does it get too much? Is something missing?
Conflict resolution	Conflicts	Do you come across conflicts? How do you resolve them?
GitHub's support	Role	How does GitHub fit in with all those collaboration pieces?
	Benefits	How has GitHub helped your team collaborate?
	User evolution	Has your use of GitHub changed over time? How?
	Problems	Is there something missing? Does GitHub hinder anything?

Commercial projects on GitHub use a workflow that builds on branching and pull requests to drive independent work.

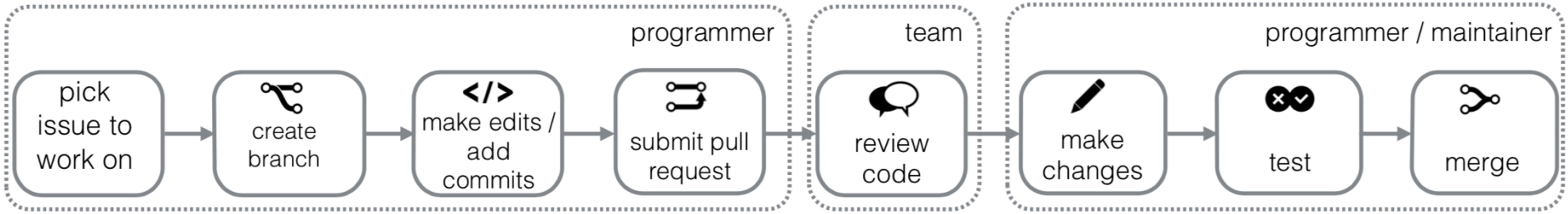


Fig. 1. Branching, pull-based workflow used in 79% of interviewed cases. Another 17% used a more complex variation of this type of workflow.

Commercial projects on GitHub use pull requests in the same way as OSS projects do; to isolate individual development and perform code review before merging.

Commercial projects use GitHub's visibility to focus their communication and coordination needs; mostly on questions and problems during code reviews and merges.

TABLE IV. INTERVIEWEES' PREFERRED METHOD OF KEEPING TRACK OF ACTIVITY.

	Awareness source	
Progress / Status	Issue list	6 (25%)
	Commit list	7 (29%)
	Notification e-mails	5 (21%)
	Chat client integrating with GitHub	6 (25%)

"As far as collaboration and communication, on GitHub most of the communication and collaboration has to do with one of two things: either concerns, or the aftereffects of a change." [P6 - professional developer in commercial project]

TABLE V. PRACTICES REPORTED BY COMMERCIAL PROJECTS, THE CORRESPONDING GITHUB ENABLERS, AND HOW THEY SUPPORTED THE COLLABORATION ELEMENTS IN OUR STUDY

Reported practice	**GitHub enabler**	**Collaboration element effect**
Independent development	Branching workflow (also mentioned as GitHub workflow)	Minimized *coordination* needs
Progress and status monitoring	Timeline, notifications, integration with chat client	*Awareness*
Code reviews	Pull requests with in-line comments	Highlighted *coordination* needs
Code-centric communication	Comments on commits, issues, and pull requests	Targeted *communication*
Self-organization	Public issue tracking	*Task Division*
Automatic testing and deployment	Service hooks in corresponding tools	*Conflict Resolution*

"The reason that we moved to GitHub was because I was using it personally and so at that point I approached the company and I said that we are using git and it'd be nice to have this functionality." [P28 - professional developer in commercial project]

"I decided we should use GitHub and part of it is picking a tool that a lot of people are going to be familiar with" [P21 - CTO in commercial software organization]

The Promises and Perils of Mining Github

Daniel German

The Promises and Perils of Mining GitHub

Eirini Kalliamvakou
University of Victoria
ikaliam@uvic.ca

Georgios Gousios
Delft University of Technology
G.Gousios@tudelft.nl

Kelly Blincoe
University of Victoria
kblincoe@acm.org

Leif Singer
University of Victoria
lsinger@uvic.ca

Daniel M. German*
University of Victoria
dmg@uvic.ca

Daniela Damian
University of Victoria
danielad@cs.uvic.ca

ABSTRACT

With over 10 million `git` repositories, GitHub is becoming one of the most important source of software artifacts on the Internet. Researchers are starting to mine the information stored in GitHub's event logs, trying to understand how its users employ the site to collaborate on software. However, so far there have been no studies describing the quality and properties of the data available from GitHub. We document the results of an empirical study aimed at understanding the characteristics of the repositories in GitHub and how users take advantage of GitHub's main features—namely commits, pull requests, and issues. Our results indicate that, while GitHub is a rich source of data on software development, mining GitHub for research purposes should take various potential perils into consideration. We show, for example, that the majority of the projects are personal and inactive; that GitHub is also being used for free storage and as a Web hosting service; and that almost 40% of all pull requests do not appear as merged, even though they were. We provide a set of recommendations for software engineering researchers on how to approach the data in GitHub.

GitHub is a collaborative code hosting site built on top of the `git` version control system. GitHub introduced a "fork & pull" model in which developers create their own copy of a repository and submit a pull request when they want the project maintainer to pull their changes into the main branch. In addition to code hosting, collaborative code review, and integrated issue tracking, GitHub has integrated social features. Users are able to subscribe to information by "watching" projects and "following" users, resulting in a feed of information on those projects and users of interest. Users also have profiles that can be populated with identifying information and contain their recent activity within the site.

With over 10.6 million repositories[1] hosted as of January 2014, GitHub is currently the largest code hosting site in the world. Its popularity, the integrated social features, and the availability of metadata through an accessible API have made GitHub very attractive for software engineering researchers. Existing research has been both qualitative [4, 7, 16, 17, 19] and quantitative [10, 24, 25, 26]. Qualitative studies have focused on how developers use GitHub's social features to form impressions and draw conclusions on other developers' and projects' activity to assess success, performance, and possi-

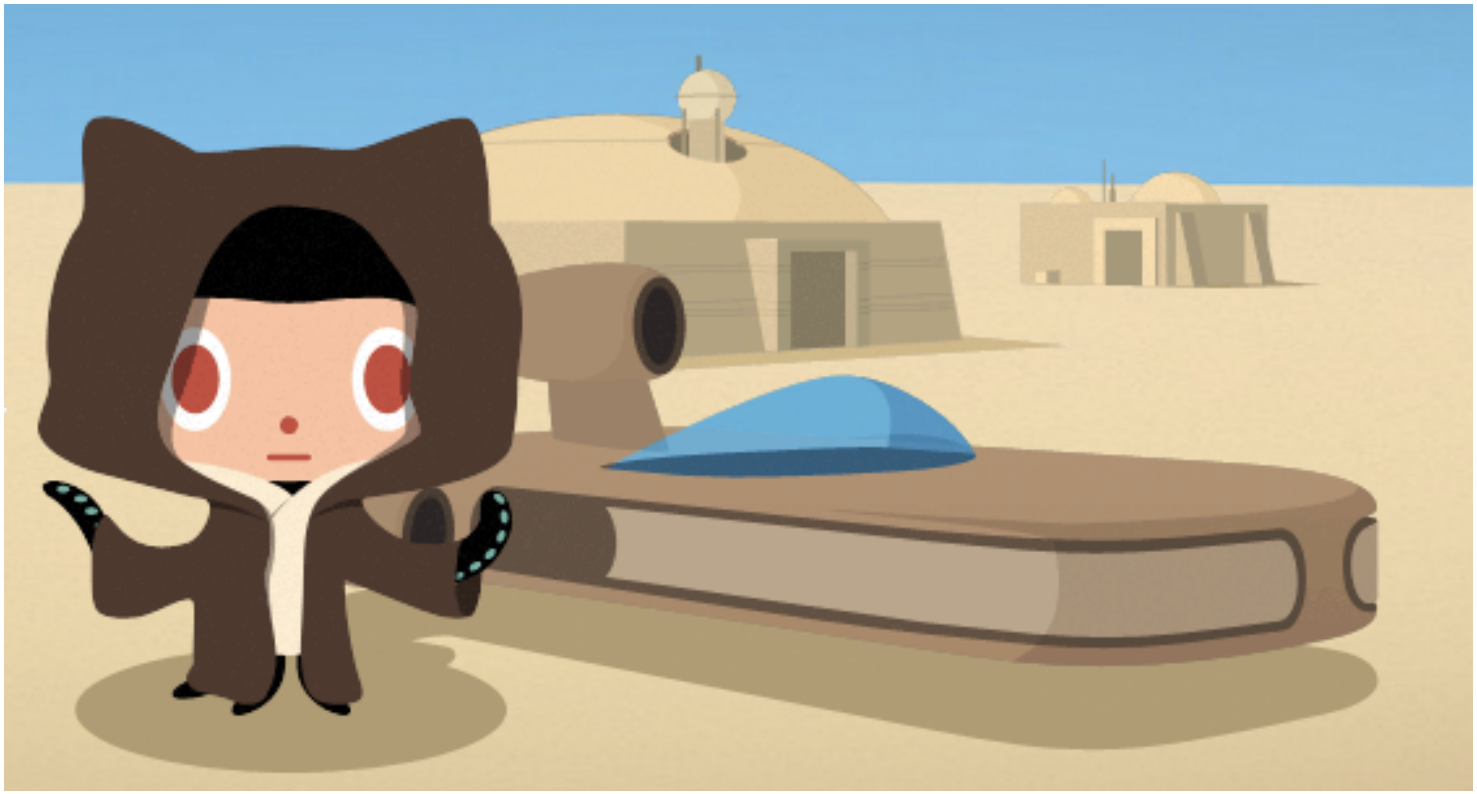
Empir Software Eng
DOI 10.1007/s10664-015-9393-5

An in-depth study of the promises and perils of mining GitHub

Eirini Kalliamvakou[1] · Georgios Gousios[2] · Kelly Blincoe[1] · Leif Singer[1] · Daniel M. German[1] · Daniela Damian[1]

Abstract With over 10 million `git` repositories, GitHub is becoming one of the most important sources of software artifacts on the Internet. Researchers mine the information stored in GitHub's event logs to understand how its users employ the site to collaborate on software, but so far there have been no studies describing the quality and properties of the available GitHub data. We document the results of an empirical study aimed at understanding the characteristics of the repositories and users in GitHub; we see how users take

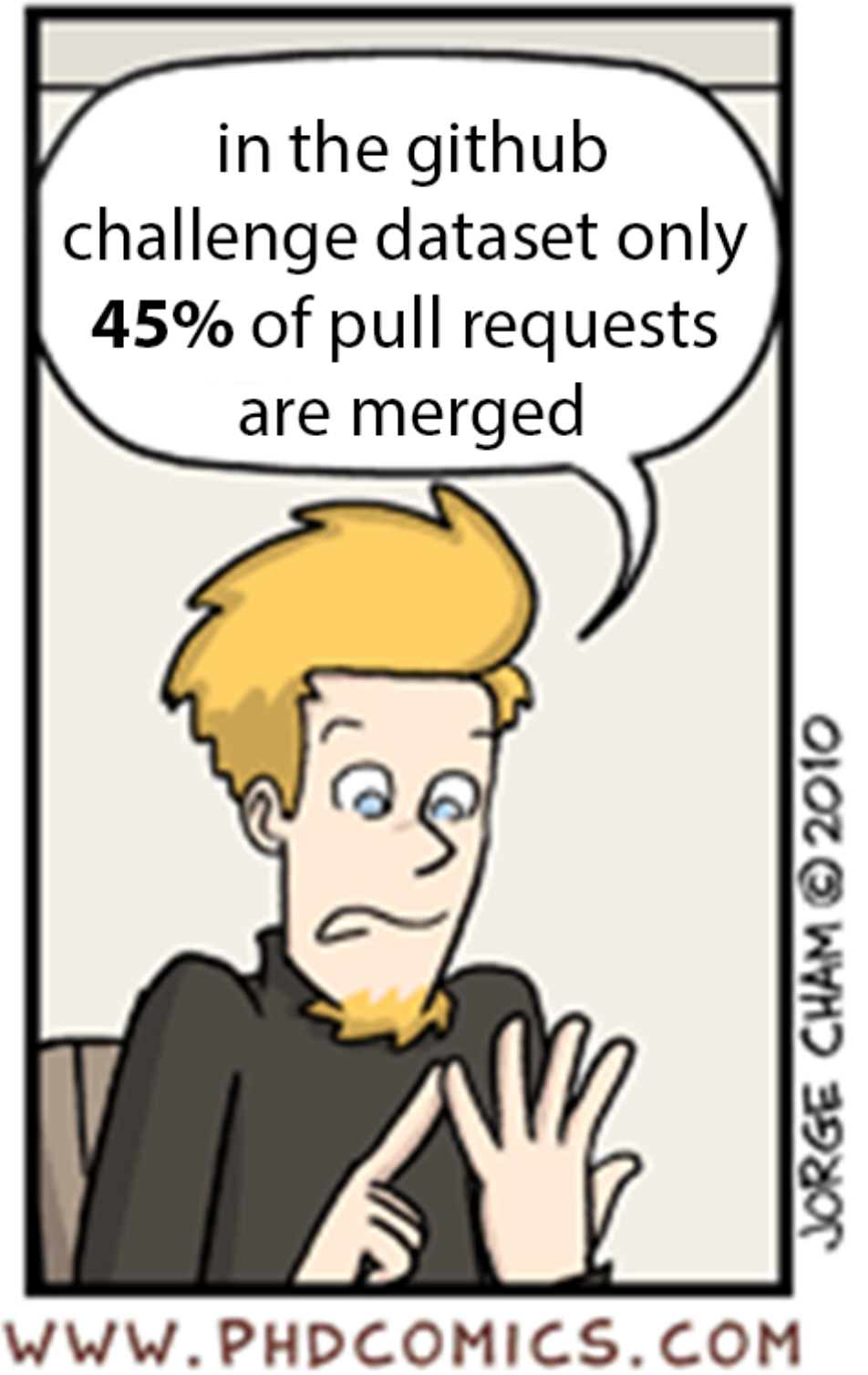
A short time ago, in an office, far far away...



Mining Challenge

This year, the challenge is on the **GitHub** data. We provide the data for the GitHub repository and you should use your brain, tools, computational power, and magic to uncover interesting findings related to it.

a postdoc states:




Trustyourgut


LOOK ME IN THE EYE
SAN DIEGO
I DON'T BELIEVE YOU


IN DATA WE TRUST

An Exploratory Study of the Pull-Based Software Development Model

Georgios Gousios
Delft University of Technology
Delft, The Netherlands
G.Gousios@tudelft.nl

Martin Pinzger
University of Klagenfurt
Klagenfurt, Austria
martin.pinzger@aau.at

Arie van Deursen
Delft University of Technology
Delft, The Netherlands
Arie.vandeursen@tudelft.nl

can only be merged by core team members. The versatility of Git enables pull requests to be merged in three ways, presented below sorted by the amount of preservation of the original source code properties:

1. Through Github facilities. Github can automatically verify whether a pull request can be merged without conflicts to the base repository. When a merge is requested, Github will automatically apply the commits in the pull request and record the merge event. All authorship and history information is maintained in the merged commits.

2. Using Git merge. When a pull request cannot be applied cleanly or when project-related policies do not permit automatic merging, a pull request can be merged using plain Git utilities, using the following techniques:

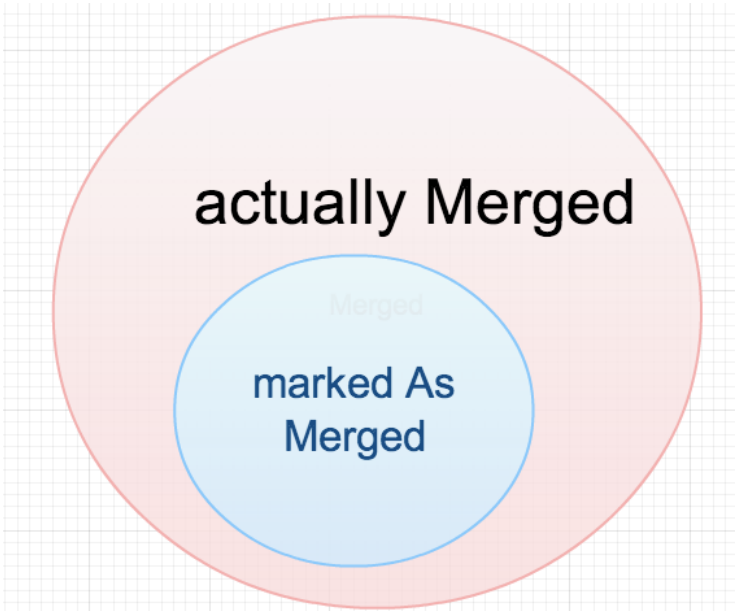
[...]

3. Committing the patch. The merger creates a textual difference between the upstream and the pull request branch, which she then applies to the upstream branch. Both history and authorship information are lost.

A pull request that is not marked as merge
is not necessarily rejected!!

markedAsMerged != merged

!merged != rejected



After creating the data files, we investigated projects where the pull request merge ratio was significantly less than the one we calculated across Github (73%), and in any case less than 40%, as this

Lifetime of pull requests. After being submitted, pull requests can be in two states: merged or closed (and therefore not-merged). In our dataset, most pull requests (84.73%) are eventually merged.

But others will not know about this...

From proceedings in MSR 14...

Of the 75,526 closed pull requests in the data-set, only 34,125 (45.18%) were merged into root projects. We wanted to explore some of the factors that may impact whether or not a pull request is accepted by maintainers of a root project.

Excerpt from a paper at ...

MSR challenge dataset [2] contains 78,955 pull requests (33,910 merged and 45,045 merge failed) made to 88 *base projects* by 20,142 developers. Among them, 9,601 pull requests (4,091 merged and 5,510 merge failed) made to 78 base projects contain *pull request commit comments*. Pull re-

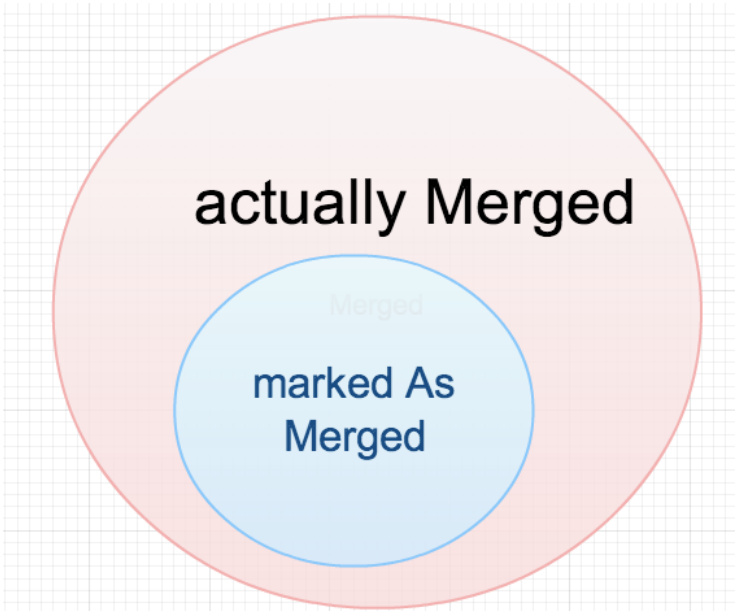
Excerpt from a paper ...

Re-emphasizing

A pull request that is not marked as merge
is not necessarily rejected!!

markedAsMerged != merged

!merged != rejected



Many are already using this data to draw conclusions!

Open source report card

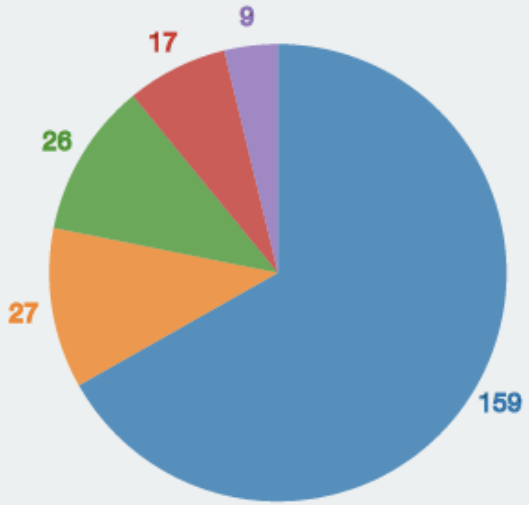
dmgerman

dmgerman is a distinguished low-level hacker who loves pushing code. dmgerman is a fulltime hacker who works best in the afternoon (around 3 pm).

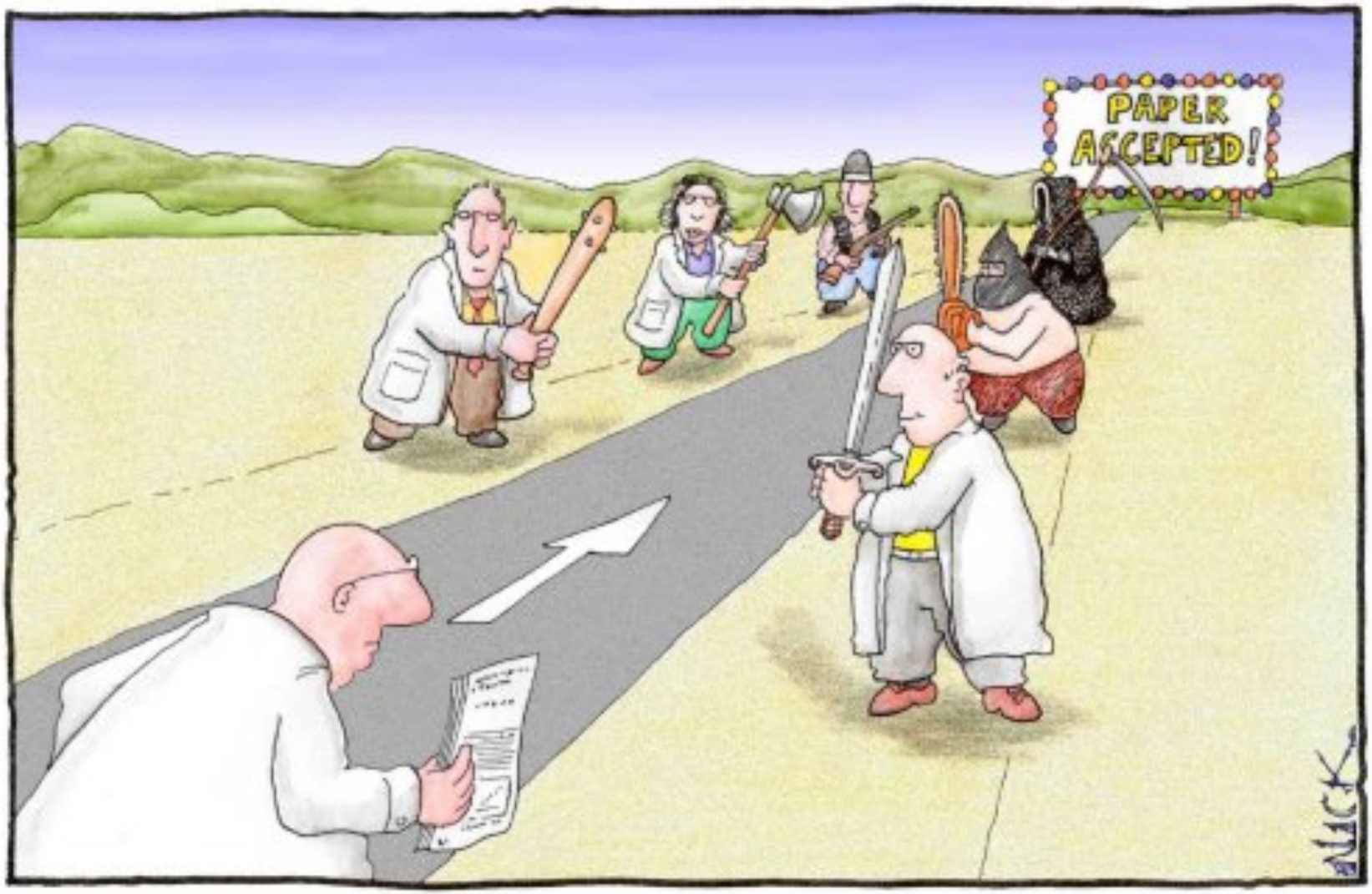
dmgerman's developer personality is very similar to Steven Dee's but Steven is more of a Ruby aficionado. There is also an uncanny similarity between dmgerman's activity stream and those of Henrik Nordström, toadking, Surith Thekkiam, and phillipberndt.

It seems like dmgerman is—or should be—friends with Philippe Ombredanne. With this in mind, it's worth noting that Philippe is far less likely to comment on your commits.

dmgerman has contributed to repositories in 11 languages. In particular, dmgerman seems to be a pretty serious **C** expert. The following chart shows the number of contributions dmgerman made to repositories mainly written in **C**, **JavaScript**, **Emacs Lisp**, **Perl**, and **CSS**.




GOAL



Most scientists regarded the new streamlined peer-review process as 'quite an improvement.'


START

Mixed methods approach

Surveys

Interviews

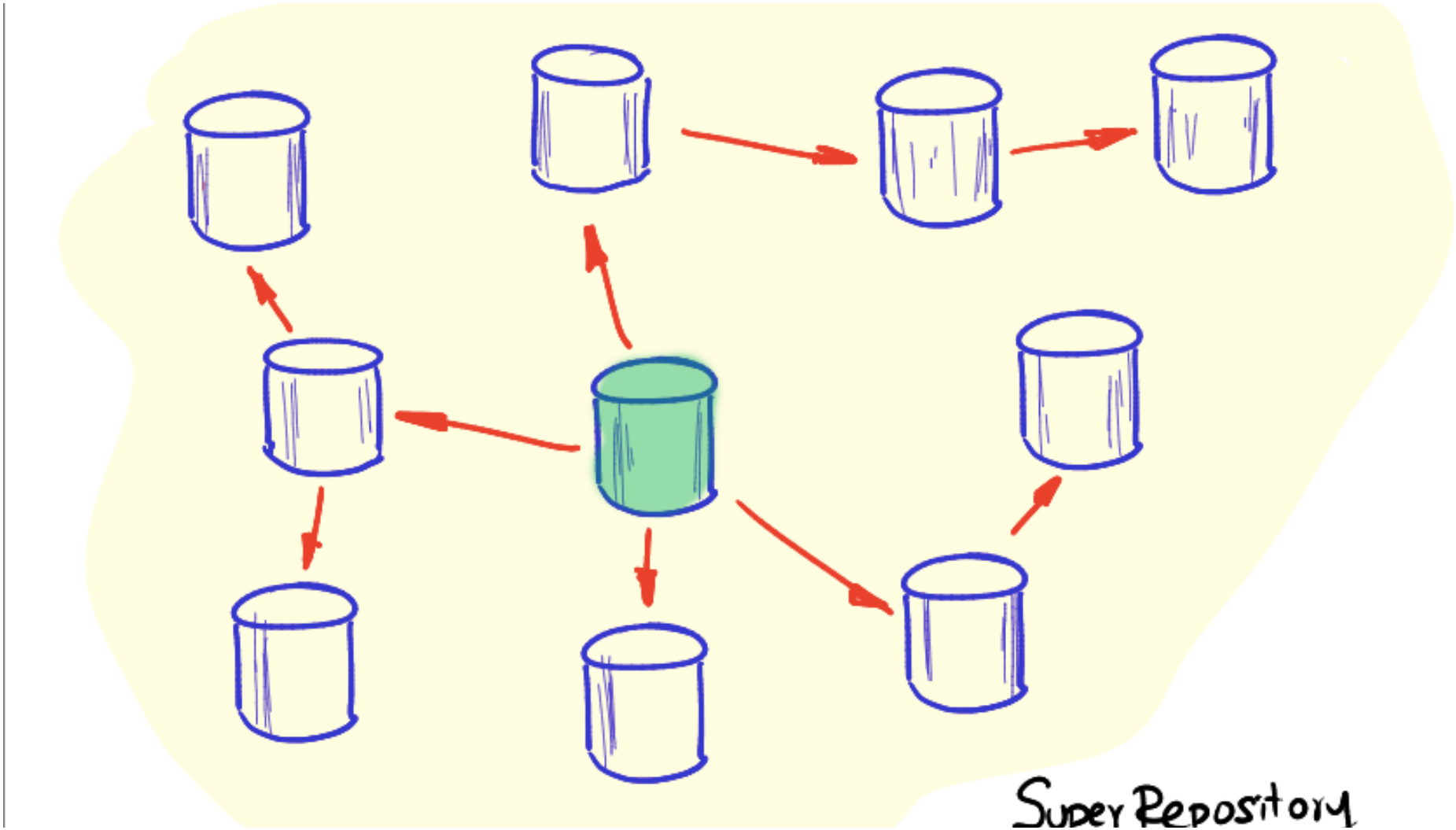
Random Statistical Sampling

The GHTorrent Dataset and Tool Suite

Georgios Gousios
Software Engineering Research Group
TU Delft

Peril	
Project Related	
I	A repository is not necessarily a project
II	Most projects have low activity
III	Most projects are inactive
IV	Many projects are not software development
V	Most projects are personal
VI	Many active projects do not use GitHub exclusively
VII	Few projects use pull requests
Pull Requests Related	
VIII	Merges only track successful code
IX	Many merged pull requests appear as non-merged
User Related	
X	Not all activity is due to registered users
XI	Only the user's public activity is visible
Github Related	
XII	GitHub's API does not expose all data
XIII	Github is continuously evolving

Peril I: A repository is not necessarily a project.



Peril II: Most projects have low activity

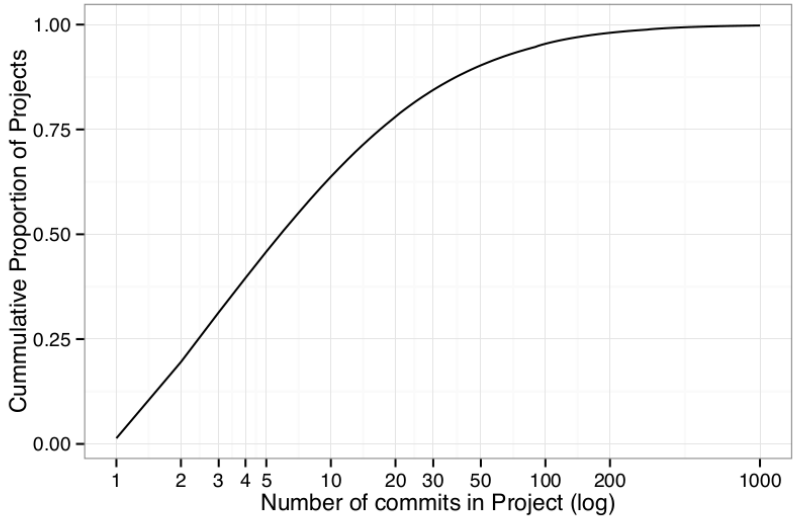


Fig. 1 Cumulative ratio of projects with a given number of commits (includes only pr with at least one commit). Most projects have very few commits. The median numl commits per project is 6 and 90% of projects have less than 50 commits.

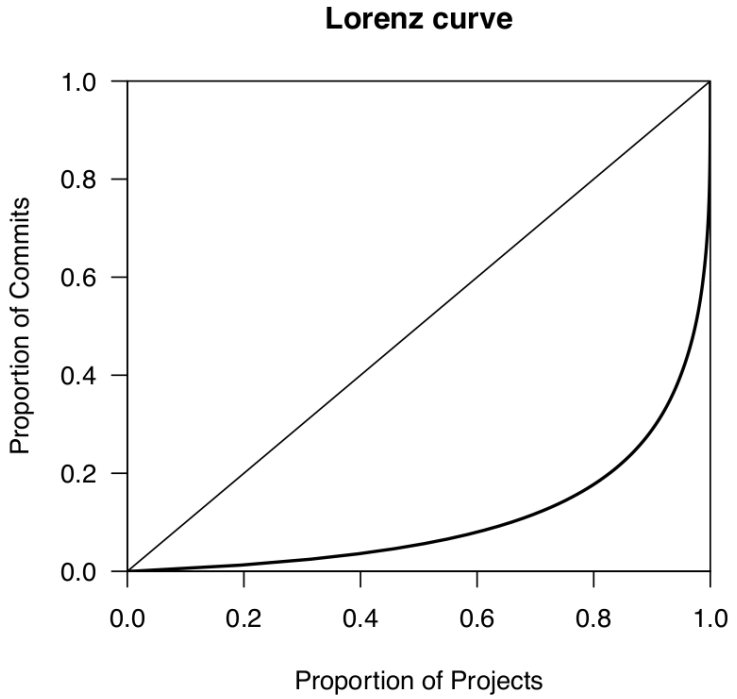


Fig. 2 Lorenz curve showing that a small number of projects account for most of the commits.

Peril III: Most projects are inactive

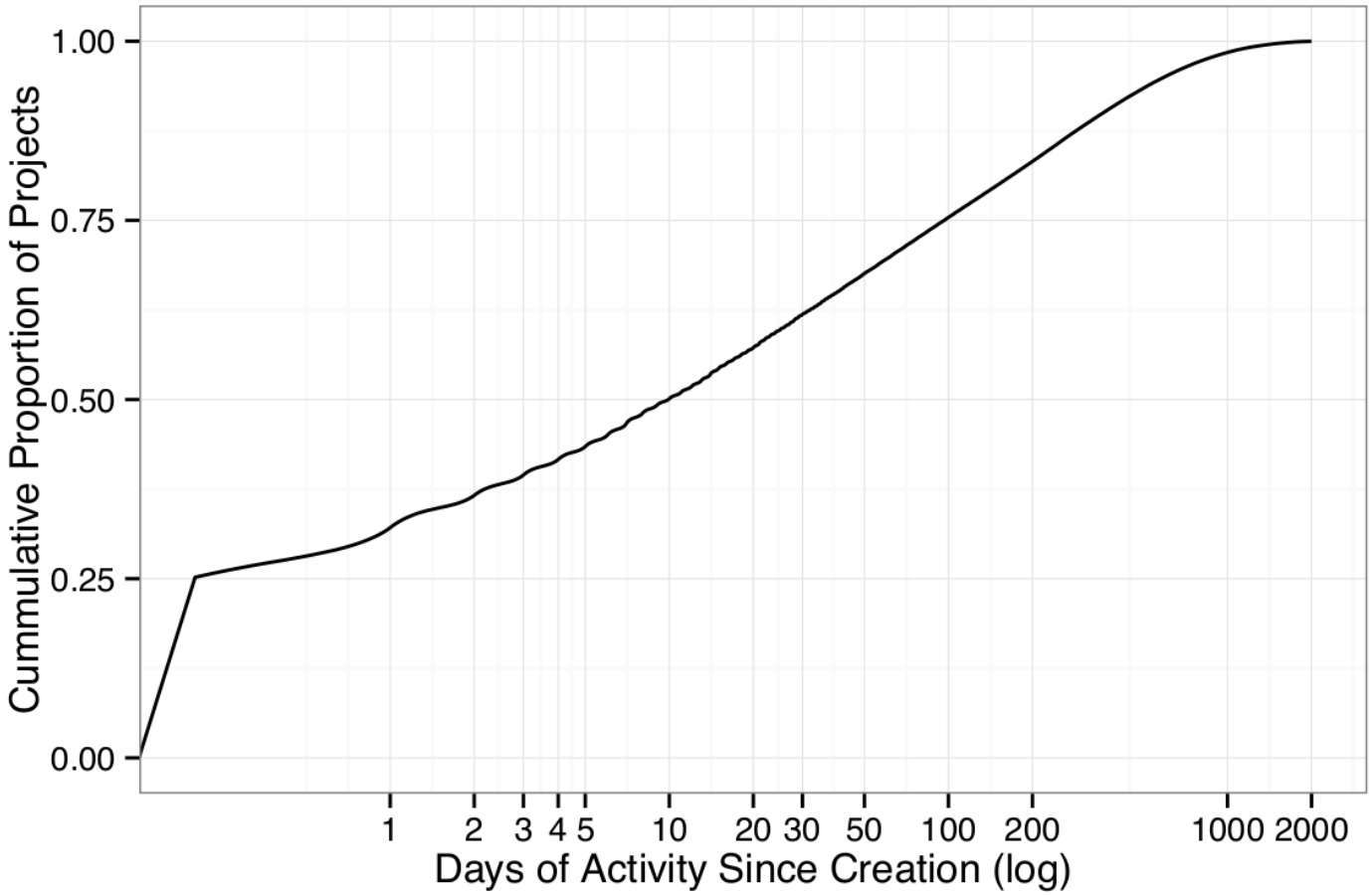
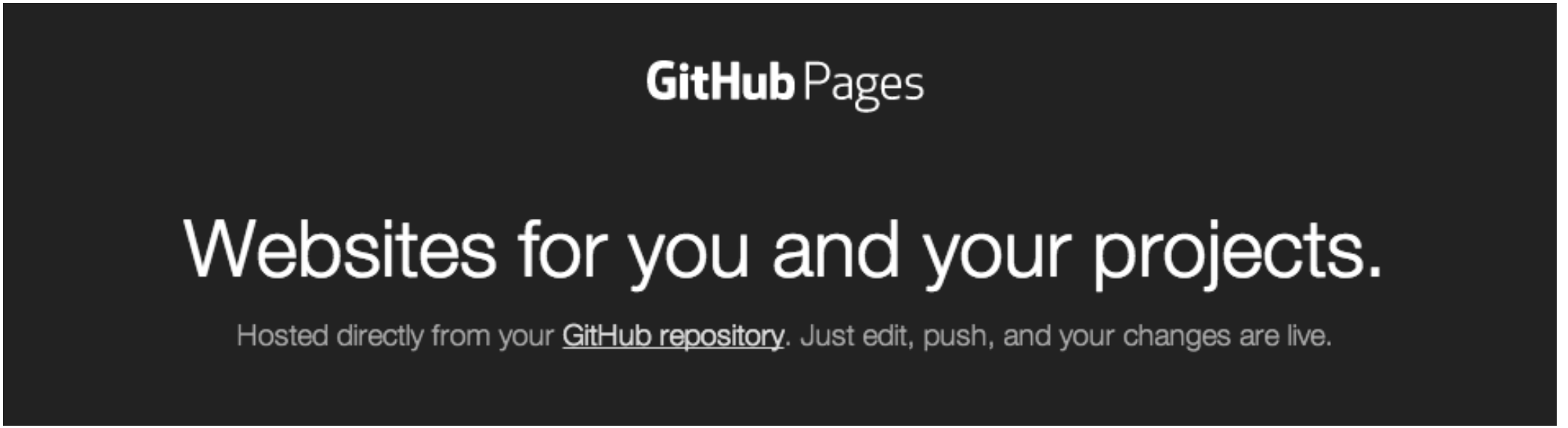


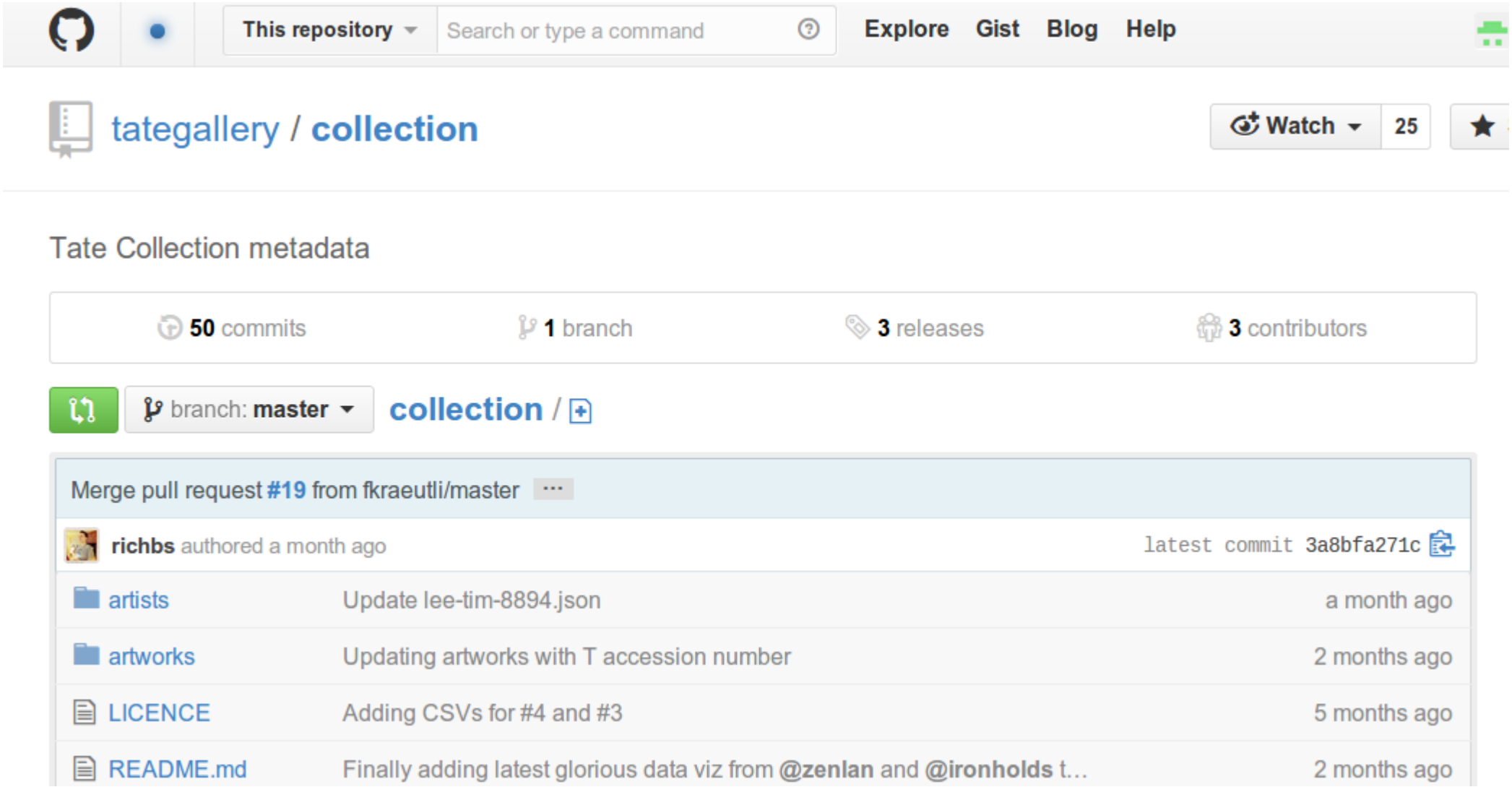
Fig. 4 Cumulative ratio of projects that had activity the last n days since their creation. The median number of days is 9.9, with 25% of projects at 100 or more days; only 32% had activity less than 1 day after being created.

Peril IV: A large portion of repositories are not for software development.

**"I store my presentations in github.
I don't need USB stick anymore!"**

github interviewee


GitHub Pages
Websites for you and your projects.
Hosted directly from your GitHub repository. Just edit, push, and your changes are live.


This repository
Search or type a command
Explore
Gist
Blog
Help
tategallery / collection
Watch
25
Tate Collection metadata
50 commits
1 branch
3 releases
3 contributors
branch: master
collection /
Merge pull request #19 from fkraeutli/master
richbs authored a month ago
latest commit 3a8bfa271c
artists
Update lee-tim-8894.json
a month ago
artworks
Updating artworks with T accession number
2 months ago
LICENCE
Adding CSVs for #4 and #3
5 months ago
README.md
Finally adding latest glorious data viz from @zenlan and @ironholds t…
2 months ago

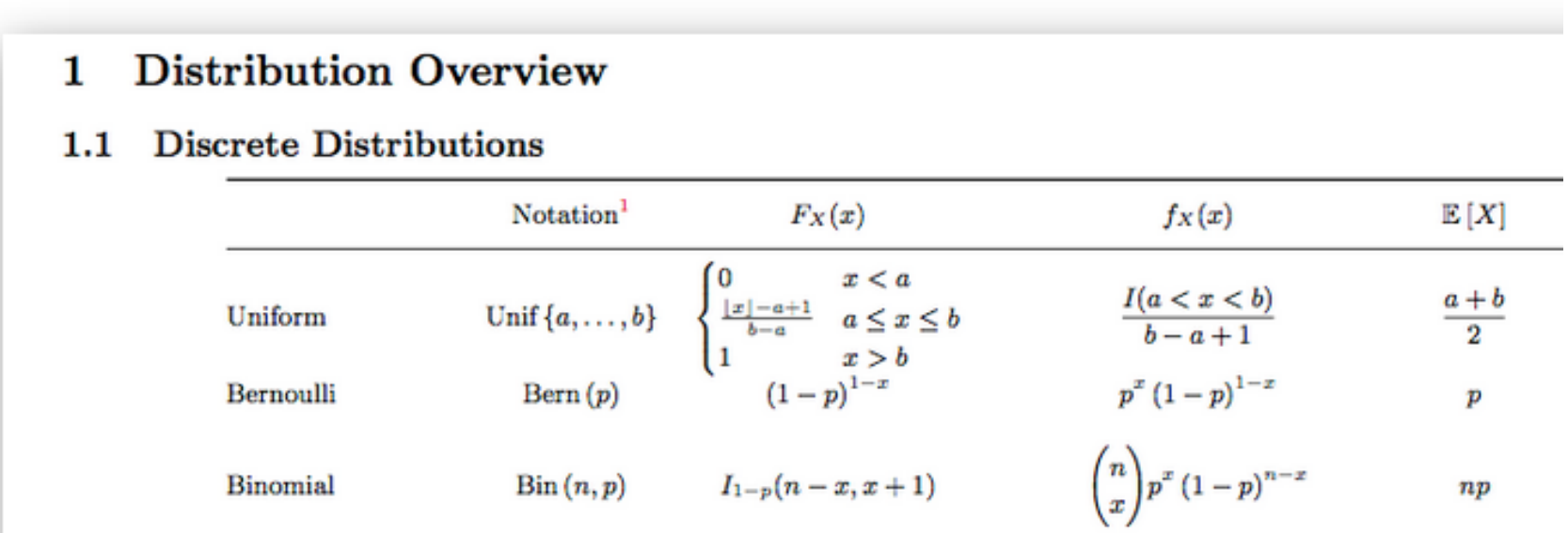
Category of use	**Number of repositories**
Software development	275 (63.4%)
Experimental	53 (12.2%)
Storage	36 (8.3%)
Academic	31 (7.1%)
Web	25 (5.8%)
No longer accessible	11 (2.5%)
Empty	3 (0.7%)

Probability and Statistics Cookbook

Source Code

The LaTeX source code is available on github and comes with a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. Feel encouraged to extend the cookbook by forking it and submitting pull requests.

Screenshots



1 Distribution Overview

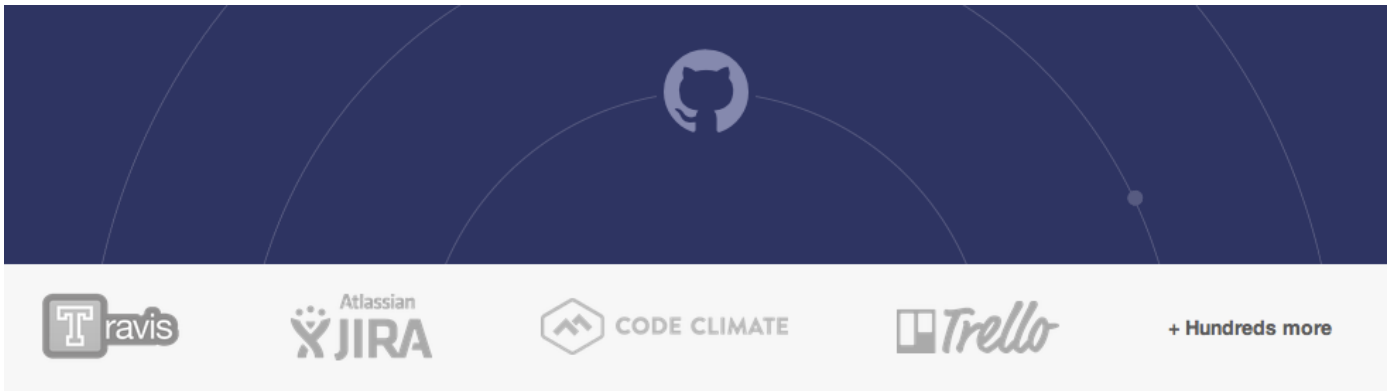
1.1 Discrete Distributions

	Notation[1]	$F_X(x)$	$f_X(x)$	$\mathbb{E}[X]$
Uniform	$\text{Unif}\{a,\dots,b\}$	$\begin{cases} 0 & x < a \\ \frac{\lfloor x \rfloor - a + 1}{b-a} & a \le x \le b \\ 1 & x > b \end{cases}$	$\frac{I(a < x < b)}{b-a+1}$	$\frac{a+b}{2}$
Bernoulli	$\text{Bern}(p)$	$(1-p)^{1-x}$	$p^x (1-p)^{1-x}$	p
Binomial	$\text{Bin}(n,p)$	$I_{1-p}(n-x, x+1)$	$\binom{n}{x} p^x (1-p)^{n-x}$	np

Peril V: Two thirds of projects (71.6% of repositories) are personal.

Peril VI: Many active projects do not use GitHub exclusively

- *"Any serious project would have to **have some separate infrastructure** - mailing lists, forums, irc channels and their archives, build farms, etc. [...] Thus while GitHub and all other project hosts are used for collaboration, **they are not and can not be a complete solution**."*




Coke
zero
Fanta
FUZE
Sprite
TAB

Peril VII: Few projects use pull requests

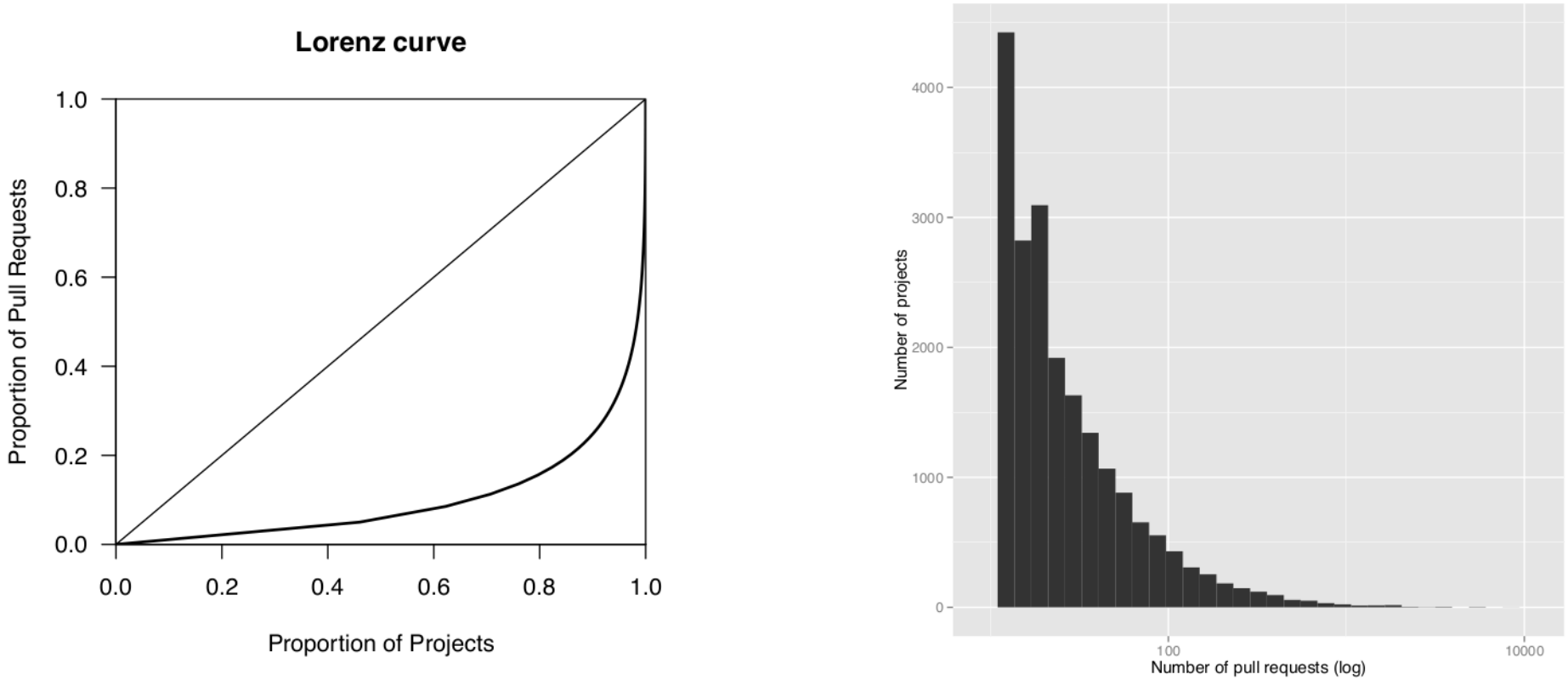


Fig. 5 Lorenz curve for the number of pull requests per project (left) and the corresponding histogram (right). The top 1.6% of projects use 50% of the total pull requests. These plots only include projects with at least one pull request.

Peril VIII: Merges only track successful code

Continuously Mining Distributed Version Control Systems: An empirical study of how Linux uses git

Daniel M German · Bram Adams · Ahmed E. Hassan

Received: date / Accepted: date

Abstract Distributed version control systems (D-VCSs—such as `git` and `mercurial`) and their hosting services (such as Github and Bitbucket) have revolutionalized the way in which developers collaborate by allowing them to freely exchange and integrate code changes in a peer-to-peer fashion. However, this flexibility comes at a price: code changes are hard to track because of the proliferation of code repositories and because developers modify ("rebase")

Table 2 Basic statistics performed on the commits from 2012, based on the two mining methods.

Metric	***cont.***	***snap***	**Ratio *cont./snap***
Active repositories	479	1	479
Unique commits	533,513	64,029	8.3
Unique non-merge commits	485,027	58,953	8.2
Unique merge commits	48,486	5,076	9.5
Unique patches (patch ids)	135,352	58,355	2.3
Unique author email addresses	5,646	3,434	1.6
Unique authors	4,575	2,883	1.6
Unique committer email addresses	1,185	283	4.2
Unique committers	1,058	245	4.3

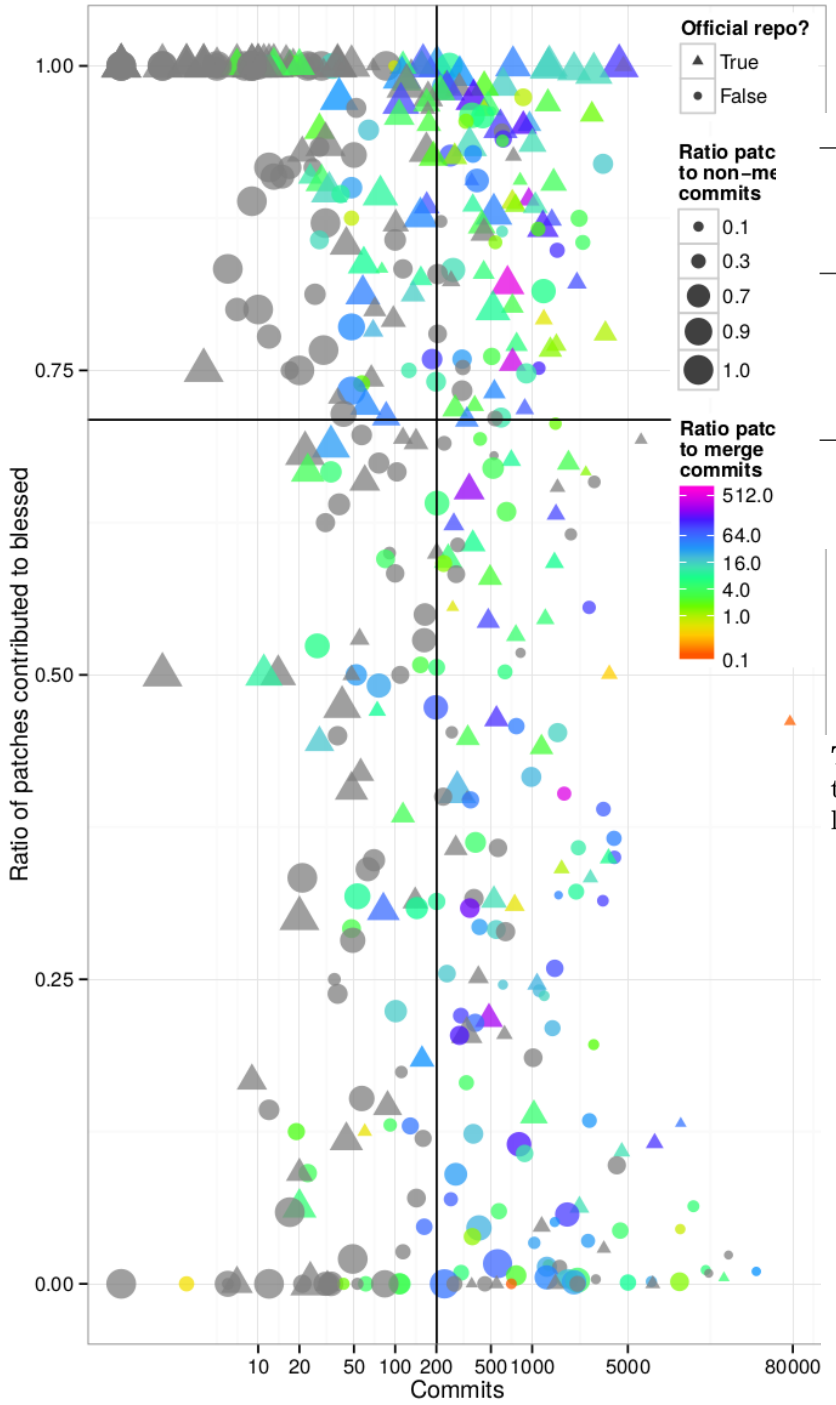


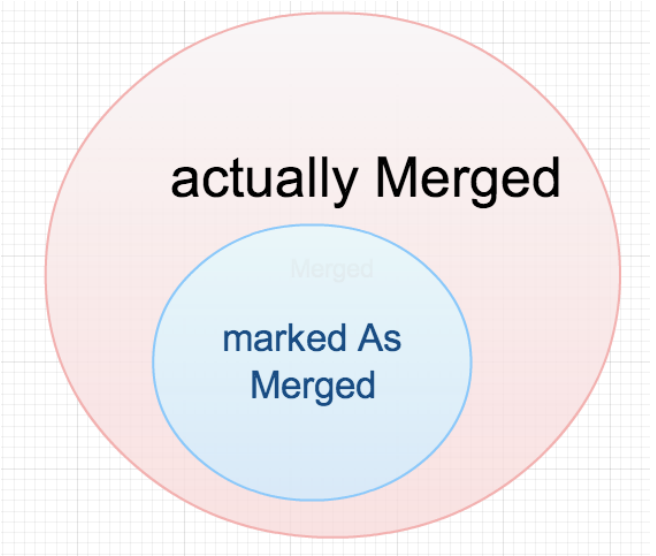
Fig. 6 Scatterplot showing, for each repository, the number of commits in the repository versus the ratio of contributed patches to *blessed*. The size of the points represents the ratio

- **Integration-testers** are people like Stephen Rothwell, who maintain the linux-next integration repository (most of his commits are created automatically as part of the integration testing he is responsible for).
- **Experimentors** are people offering less mainstream versions of the kernel, such as G. Roeck who is the maintainer of Linux Staging (stand-alone drivers and filesystems that are not ready to be merged into *blessed*) and T. Gleixner who maintains Linux RT (real-time version of the kernel).
- **Product-line maintainers** are those who are customizing the code in *blessed* for specific uses in product-line repositories (such as those employed by Android, Ubuntu and Linaro).

Table 6 Most active repositories with a ratio of contributed patches of at most 5%. All these repositories are product-lines of Linux or older releases still being maintained (e.g., linux-stable).

Address	#New Commits	#New Patch.	#Contr. Patch.	Prop. Patches Contr.
linaro.org/.../andygreen/kernel-tilt	43,290	2743	28	1.0%
ubuntu.com/.../**ubuntu-precise**	27,141	1184	28	2.3%
kernel.org/.../rt/**linux-stable-rt**	25,129	407	2	0.4%
ubuntu.com/.../**ubuntu-quantal**	19,459	809	7	0.8%
linaro.org/.../linux-tb-ux500	18,508	1765	20	1.1%
linaro.org/../linux-linaro-tracking	12,058	1291	58	4.4%
android.googlesource.com/.../**msm**	11,922	5911	10	0.1%
kernel.org/.../stable/**linux-stable**	7,591	1116	0	0.0%
github.com/vstehle/linux	7,421	1079	2	0.1%
denx.de/linux-denx	5,005	1915	2	0.1%
opensuse.org/kernel	4,399	1716	75	4.3%

Peril IX: Many merged pull requests appear as non-merged



- In sample projects:
- 44% of pull-requests marked
  - as **merged**
  - Depending on sample
  - Extra 19 to 42% of pull-requests have been merged,
    - but **are not marked as merged** in github

Peril X: Not all activity is due to registered users

- Not all committers or authors of commits are registered GitHub users
- Imported git repos have history predating their github usage
- A committer can make a commit appear as coming from another user
- Do email unification of authors and committers

Peril XI: Only the user's public activity is visible

We found that out of the registered users on GitHub, 53.2% do not have a single public commit. This population can be further divided into two parts: 30% of the registered users do not have a public repository either, while the remaining 23.2% have repositories but no public commits. These repositories fall into two categories: empty repositories (e.g., used for testing) and forks that have no activity.

Peril XII: GitHub's API does not expose all data

- Github does not provide an API to retrieve all events
- Tracking renames
- Deleted entities
- Making a repo private
- Rebasing
- Propagation of commits between repos

Peril XIII: Github is continuously evolving

- “watchers” became “stars”
- Change of terms!!!!

5. Scraping

Scraping refers to extracting data from our Website via an automated process, such as a bot or webcrawler. It does not refer to the collection of information through GitHub's API. Please see Section G for our API Terms. You may scrape the website for the following reasons:

- Researchers may scrape public, non-personal information from GitHub for research purposes, only if any publications resulting from that research are open access.
- Archivists may scrape GitHub for public data for archival purposes.

A New Hope

Promise I: GitHub is a rich source of software engineering research

Promise II: GitHub provides a valuable source of data for the study of code reviews in the form of pull requests and the commits they reference

Promise III: The interlinking of developers, pull requests, issues and commits provides a comprehensive view of software development activities