# REPORT ON OpenDreamKit DELIVERABLE D7.1
## The flow of code and patches in open source projects

DMITRII PASECHNIK

| | |
|---|---|
| Due on | 02/28/2017 (Month 18) |
| Delivered on | 02/28/2017 |
| Lead | University of Oxford (UOXF) |
| Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/148` | |

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #148 ON 2017-02-27

- **WP7:** Social Aspects
- **Lead Institution:** University of Oxford
- **Due:** 2017-02-28 (month 18)
- **Nature:** Report
- **Task:** T7.1 (#144): Social input to design
- **Proposal:** p. 58
- **Final report**

OpenDreamKit builds on top of a large ecosystem of (mostly) academic open-source systems, many of which are large-scale themselves: for example our chosen test system SageMath is the outcome of a decade of work by hundred of contributors; many others are decades old. The social engineering aspects involved in such a large ecosystems are therefore both intricate and central for its long run sustainability. This motivates OpenDreamKit's objective in WP7 of studying the collaborative processes of free open source (mathematical) software development so as to produce guidelines for best practice as well as to develop ideas for extending existing processes to an "ecosystem of systems".

In this deliverable we survey the methodology, data, and tools needed to assess development models of large-scale academic open-source systems, such as the probable correlation between the size of the atomic contribution vs. the speed of the contribution making it into the code, and collect appropriate statistical data, to be published as a report (and possibly a conference publication). While in the proposal it was assumed that the latter might require non-trivial amount of programming work, even only for our test system, great open-source tools to address precisely these kinds of questions were released last year, and we used one of them instead.

Accomplishments:

- ✓ a large number of publications and online sources was reviewed for applicability
- ✓ various analytic tools were tried on a sample of SageMath components
- ✓ results were summarised in a report, with conclusions and pointers to further possible developments

CONTENTS



## 1. QUESTIONS AND OBJECTIVES

Large-scale and to a lesser extent medium-scale open-source software is, as a rule, a product of a collaborative effort spanning many years of development, improvements, bug fixes, ports to new platforms, and partial or even full rewrites. A number of interesting related questions arise in this context.

(1) What are available data, measurement parameters and tools allowing for analysis?
(2) Can one assess the usefulness of the project by estimating how "alive" it is, i.e. how much it is changing over time?
(3) Can one reliably rank the contributors by the effort put into the project?
(4) Can one produce recommendations on the team size and composition to ensure project's well-being?
(5) Does the "openness" of the project matter?
(6) Reliability of the system, stability of APIs etc., reproducibility of computational results, etc.

## 2. DATA, PARAMETERS, AND TOOLS

The main sources of information about the history of a project are versions of its source code and records of various relevant communications, discussions, and test results. Before the wide acceptance of *revision control systems* (RCS) [O'S09] such as CVS [Cvs] and Git [CS14] the only readily available source code data came from regular (often infrequent) public releases. Then it has become more and more widespread, although not universal (cf. e.g. GAP [GAP16], which only in the past few years made its RCS public—and has not released earlier RCS data) to keep the RCS *trees* holding *commits*—code changes accompanied by comments—available online with read access for the public.

Communications on the project take basically three (not totally disjoint) forms: mailing lists/bulletin boards and tracker/code reviewing systems, such as Trac [trac], Redmine [redm], GitHub [github], Bitbucket [bitb], Gitlab [gitlab], etc, and documentation systems/wikis. The latter is open by nature, whereas for the first two the prevailing trend for these, at least in the domain related to the ODK themes, is the ever increased openness of the development process.

The current prevailing form of the analysis of the source code is based on analysing authorship, frequency, and other parameters of commits in the RCS. Most of the tools are in one or another way related to GitHub and its APIs to access RCS trees and collect statistics, see e.g. [Smi16]. A sample of data obtained by such analysis may be found in Appendix below. Communications are analysed using various text mining and analysis tools, such as FOSS Heartbeat [Sha16]; these are not dissimilar to tools used to analyse *social networks* such as Facebook [Rus13]. Interestingly, GitHub as a collaborative social network has been analysed [VFS13; LRM14; Vas+15; Kal+14] and compared to Stack Overflow [stofl]. Note that analysis of Mathoverflow (a sub-site of the latter) in [MP13a] was one of the starting points in adding WP7 into the ODK project.

## 3. Alive or dead? (Cathedral or Bazaar?)

It is not obvious whether extreme stability of the code base, such as e.g. Knuth's TEX[Knu84], with releases numbered by consecutive digits of $\pi$, and only occurring once every 5 or 6 years, see `http://tug.org/texlive/devsrc/Build/source/texk/web2c/tex.web` is a bad sign. Although actively developed OSS projects are certainly showing a different pattern, with dozens of commits per day, etc., see e.g. Figure 1. At this scale the dynamics of the code of TEX will be basically null. Different projects have vastly different cultures in regard of commit

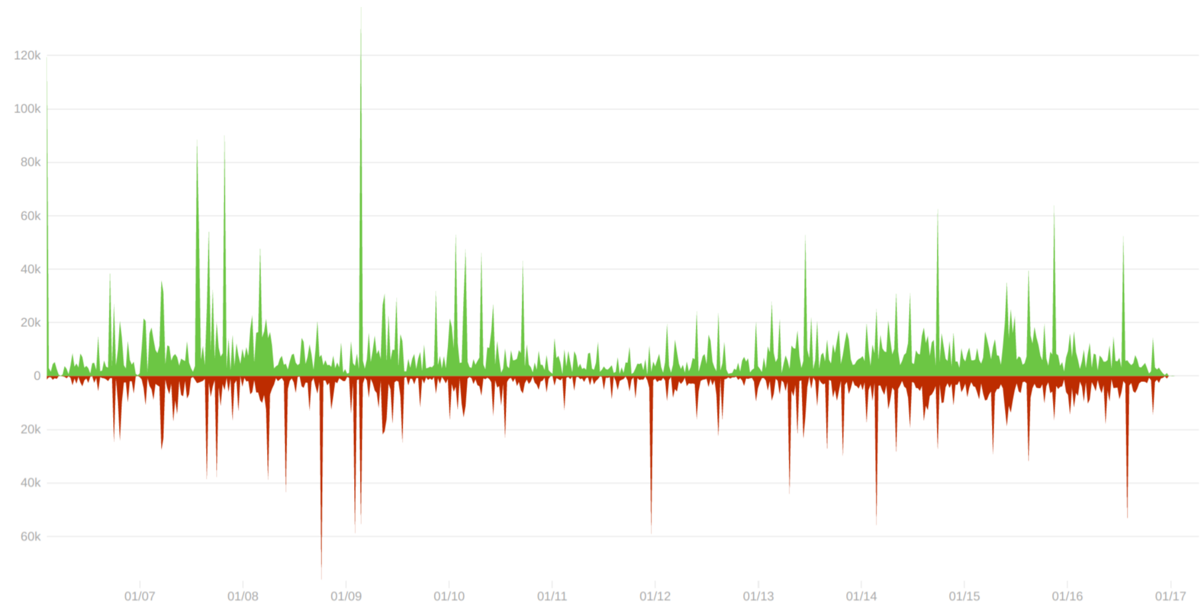


FIGURE 1. Code additions and deletions per week for SageMath.

frequency, commit size, etc., and this makes them hard to compare—something already observed in [Ray99].

## 4. Ranking contributors

This information may be extractable from a number of sources— and headhunters for hi-tech companies publish guides on how to utilise these for the search of talented software developers, cf. e.g. [sota].

### 4.1. **Commits**

Are commits, as advocated by GitHub, a good way to rank contributors by their contributions? Indeed, for GitHub, it is very easy to analyse the commits' (meta)data, computing the size of additions and deletions done by the commit, and not so, as soon as real contributions are involved. To give one a simple example, how does one measure the work done by quality assurance team, or by someone who does complicated debugging and reports its results? Such an activity, if measured in commits, would fall under the radar almost completely. As well, in some projects commits are squashed by release managers, making them disappear all together.

There is also a lot of discrepancy between the commits in the master branch (i.e. released versions) and the development branches— people involved in making releases have more commits in the master branch of mostly administrative kind, and they can be massive in size.

Bug fixing commits are often very short, although might be extremely time consuming to create, compared to adding new code/documentation and unit tests.

### 4.2. **Trackers and other exchanges**

Communications between developers, and to almost the same extent, between users, and between users and developers, is an immense source of information on the state of the project. However, it is often a very loosely coupled textual data spread over a range of different forms and mailing lists, and analysing it needs text-mining tools. More structured data like this is available on various issue/bug trackers. Still, it needs textological analysis for sentiment. Experiments of this kind are known, cf. e.g. [Sha16], where analysis of OSS projects on parameters like negative/positive sentiment in communication, friendliness of the community towards the newcomers, etc., is analysed from the data mined from GitHub, specifically from issue trackers and *pull requests* (i.e. patches proposed for the system, often by external contributors).

An interesting option, not really explored so far, would be ranking issues by their importance, akin to the Stack Overflow [stofl] sites.

## 5. Team composition

What is the optimal team composition for a successful OSS project? Is the Cathedral model [Ray99] better or worse than the Bazaar one? It is hard to make a judgement on this, as this heavily depends upon the application domain and the size of the project. While our test computational system, Sage is developed very much in Bazaar style, many of its components are pretty much Cathedral style ones. Often it is possible to see just from the team size, which one is which.

What is clear from our experience is that it is very important for the project to be open as long as bug tracking and pull requests are concerned, while it's of secondary importance how exactly these are handled, as long as they are handled within a reasonable time frame and in a respectful manner (users and potential contributors must be respected).

## 6. Openness, licensing, etc

We will not argue here about the benefits of being open source for a system, and only remark that this was validated not only empirically, but by looking at the various growth rates before and after open-sourcing, see e.g. [Smi16].

As an example of licenses directly influencing the *flow of code*, until recently Sage and GAP were unable to make full advantage of the Nauty project [MP13b] due to its too restrictive license. It took a lot of convincing on our side to pursuade Nauty's authors to make the license more suitable.

GPL and GPL-style licenses [gpl] are often criticised for being inflexible, too restrictive, etc., compared to less restrictive BSD-like licenses. However, in the opinion of the author of this report (not universally shared across ODK members—and indeed some parts of the project are licensed under a modified BSD license) there is good evidence to them being better in sense of keeping the community together. This can be seen e.g. by noting how many forks of the BSD OS are out there (dozens, including e.g. Apple's macOS, see [bsds]) and how many forks of Linux are out there (none, essentially).

## 7. Reliability, stability of APIs, reproducibility

Whenever a new external component is to be added to an OSS system, a number of questions arise that can potentially be amenable to study through the analysis of the flow of code and patches in the component.

Information on the reliability (i.e. how often critical level bugs pop up, etc) of the system might be obtained from the issue/bug trackers and from user forums and mailing lists, as well as from project's announcements. However it does not look feasible to hope that humans may be replaced, or even helped by, automatic tools to make conclusions about the quality.

Application programming interfaces (APIs) are extremely important parts of large-scale software systems. A very important question is of *stability* of APIs, that is, whether the API was in place for some time already and did not change for a period of time—otherwise it might happen that a change in APIs will require a redesign of the OSS. While some APIs are extremely stable, e.g. famous BLAS's and LAPACK's [blas02; lapack90] APIs for numerical linear algebra, it is much less certain in more modern areas, such as GPU computing, web programming, etc.

In the case of systems making regular (and well-documented) releases it is normally possible to check stability of APIs directly from the documentation. However it is becoming more and more usual that there are no releases as such, the most fresh "master branch" of the RCS tree of the system is all that can be taken as the release (essentially, the commit hash has to be used to give the release a version number). A slightly better situation is where "stable" releases are tagged in the RCS with a version number label. In these cases one could check the history of certain pieces of the code in the RCS manually; however it might be desirable to have automated tools for this. Literature search for the latter did not return anything that does not require actual installation of the software in question and testing its APIs.

## 8. Conclusions and future work

A wide range of tools to pick up the low-hanging fruit— the commit history from the RCS and project communications— and study it are already available and can be readily applied to open-source VREs and to their components. However, it is unclear whether any practically interesting conclusions and recommendations can be derived from such studies—not the least due to their extreme imperfections discussed above. One notable exception might be community sentiment analysis, as discussed in Section 4.2. The latter is only applicable to projects fully hosted on GitHub. While it might be feasible to extend it to other development workflows, e.g. ones using [trac], it appears to be out of scope of ODK.

Better mathematical models to analyse OSS ought to be developed, perhaps by treating them as large-scale discrete dynamical systems [Ant05; MAP11]—whether this is feasible is open to discussion.

APPENDIX

Here we include a sample of statistical data that can be obtained for OSS projects, parts of SAGE fully hosted on GitHub. We used [Sha16] to mine and process data from the latter.

**Some GAP statistics**

Figure 2 illustrates issue reporting per contributor. Currently most active, in these terms, for the reported period, contributors, `@alex-konovalov`, `@fingolfin`, `@markuspf`, and `@ChrisJefferson`, correspond to the four top right points on the graph. Figure 3 illustrates how fast pull requests are dealt with.

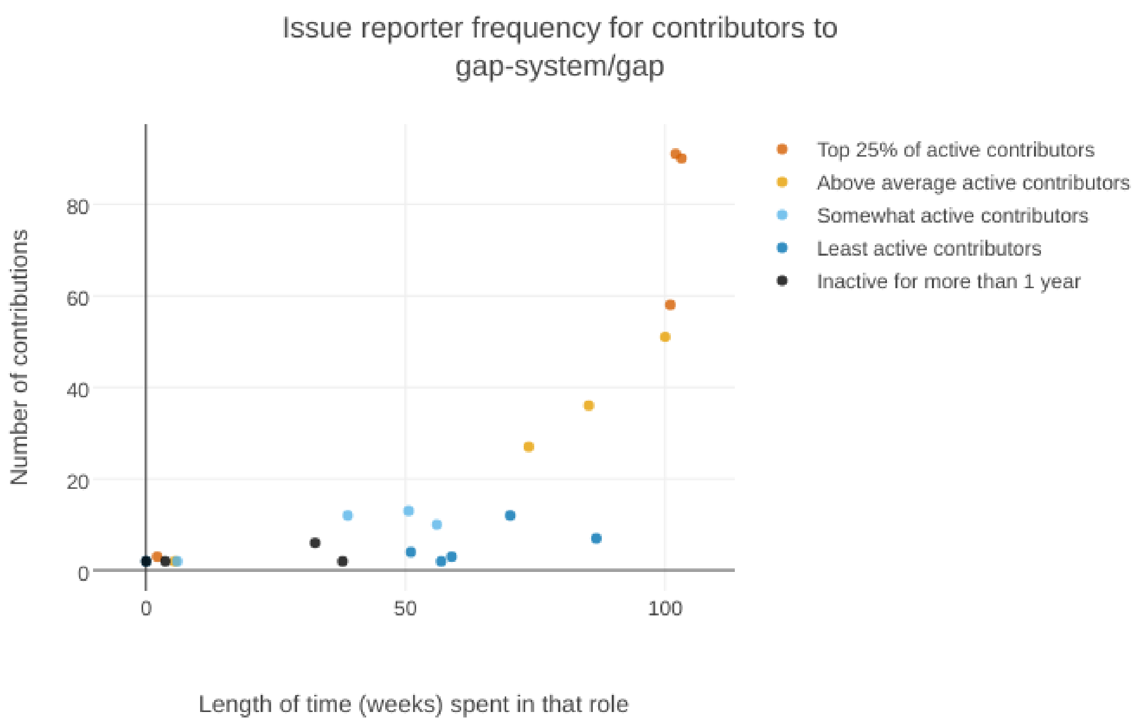


FIGURE 2. Issues reporting for GAP.

**Some SymPy statistics**

Figure 4 illustrates issue reporting per contributor. Currently most active, in these terms, for the reported period, contributors, `@certik` and `@asmeuer`, correspond to the two top points on the graph.

Figure 5 illustrates how fast pull requests are dealt with.

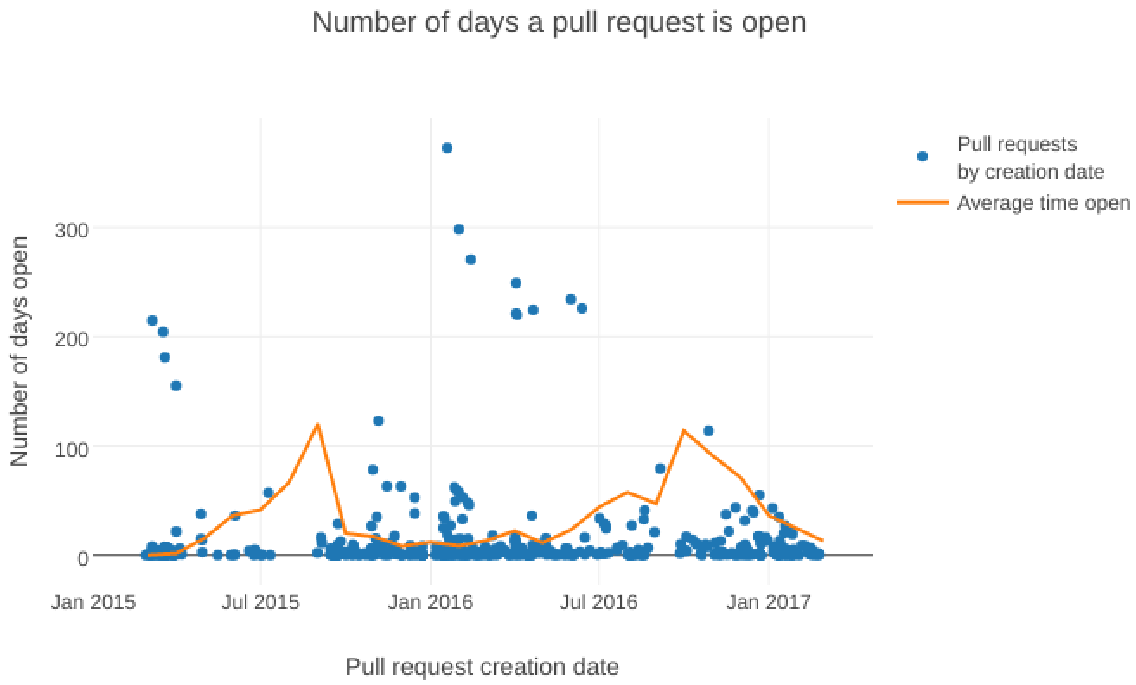


FIGURE 3. Pull requests handling in GAP.

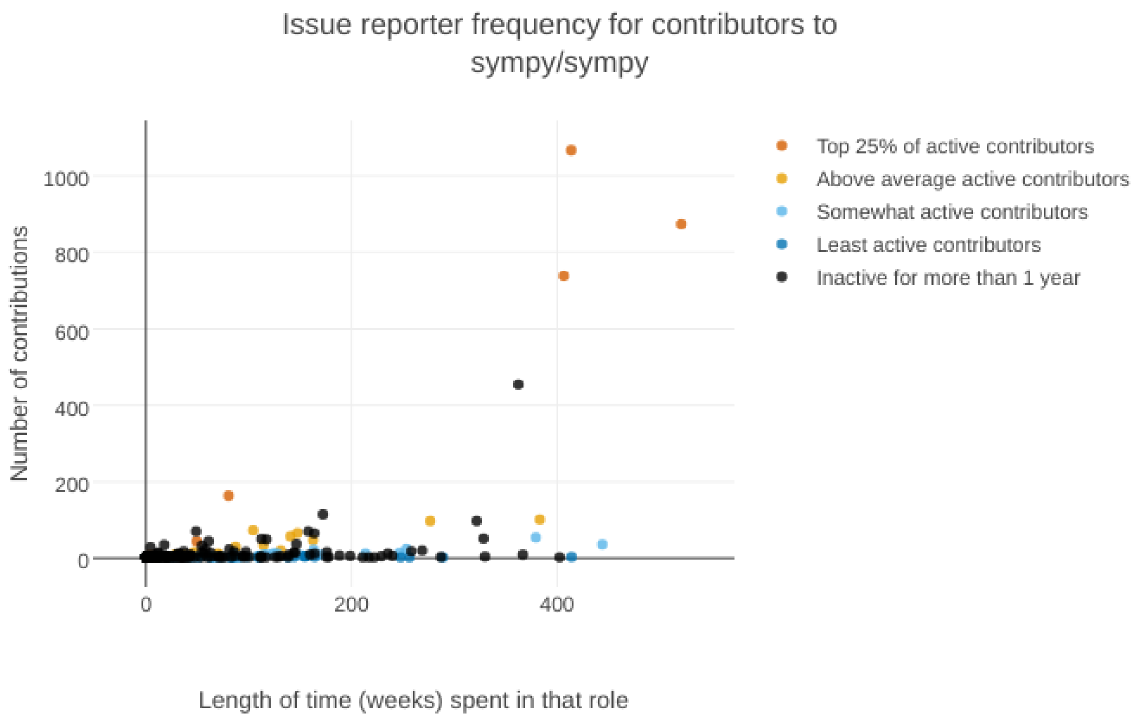


FIGURE 4. Issues reporting for SymPy.

## REFERENCES

[Ant05] Athanasios C. Antoulas. *Approximation of Large-Scale Dynamical Systems (Advances in Design and Control) (Advances in Design and Control)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005. ISBN: 0898715296.

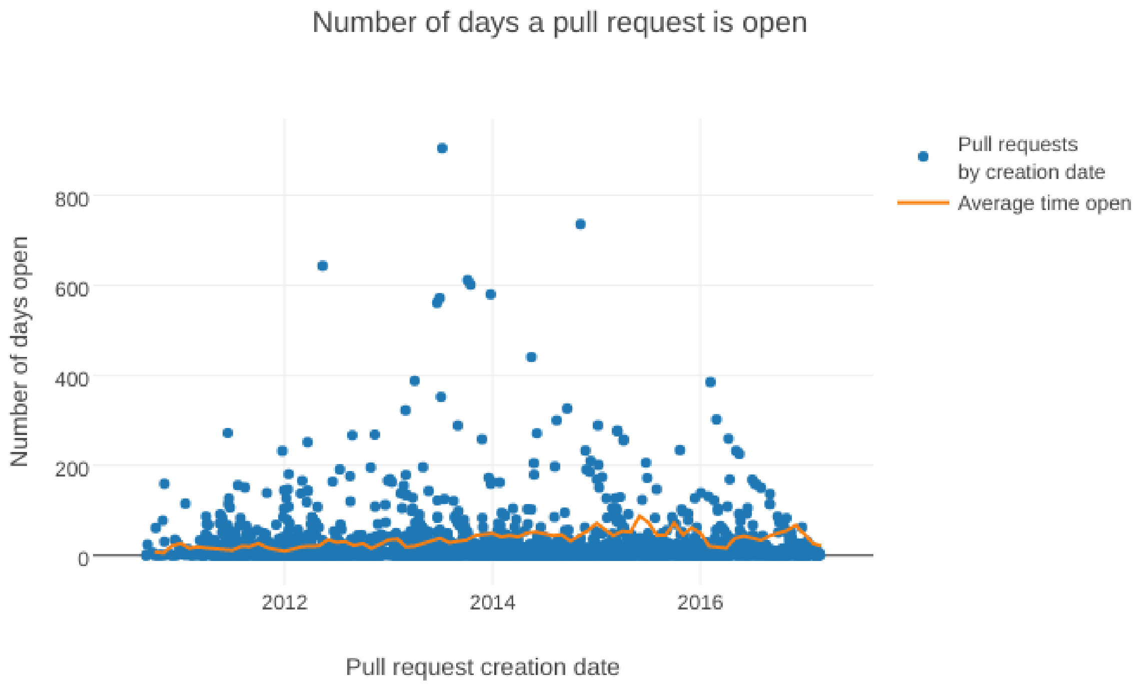


FIGURE 5. Pull requests handling in SymPy.


[bitb] *Atlassian bitbucket*. URL: `https://bitbucket.org`.

[blas02] "An Updated Set of Basic Linear Algebra Subprograms (BLAS)". In: *ACM Trans. Math. Softw.* 28.2 (June 2002), pp. 135–151. ISSN: 0098-3500. DOI: `10.1145/567806.567807`. URL: `http://doi.acm.org/10.1145/567806.567807`.

[bsds] *List of BSD operating systems*. URL: `https://en.wikipedia.org/wiki/List_of_BSD_operating_systems`.

[CS14] Scott Chacon and Ben Straub. *Pro Git*. 2nd Edition. APress, 2014. ISBN: 978-1484200773. URL: `https://git-scm.com/book/en/v2`.

[Cvs] *Concurrent Versions System: The open standard for Version Control*. Web site at `http://www.cvshome.org`. seen August 2005. URL: `http://www.cvshome.org`.

[GAP16] The GAP Group. *GAP – Groups, Algorithms, and Programming*. `http://www.gap-system.org`. [Online; accessed 30 August 2016]. 2016.

[github] *GitHub*. URL: `http://github.com`.

[gitlab] *GitLab*. URL: `http://gitlab.com`.

[gpl] *The GNU General Public License*. URL: `http://www.opensource.org/licenses/gpl-license.php`.

[Kal+14] Eirini Kalliamvakou et al. "The Promises and Perils of Mining GitHub". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 92–101. ISBN: 978-1-4503-2863-0. DOI: `10.1145/2597073.2597074`. URL: `http://doi.acm.org/10.1145/2597073.2597074`.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[lapack90] E. Anderson et al. "LAPACK: A Portable Linear Algebra Library for High-performance Computers". In: *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*. Supercomputing '90. New York, New York, USA: IEEE Computer Society Press, 1990, pp. 2–11. ISBN: 0-89791-412-0. URL: http://dl.acm.org/citation.cfm?id=110382.110385.

[LRM14] Antonio Lima, Luca Rossi, and Mirco Musolesi. "Coding Together at Scale: GitHub as a Collaborative Social Network". In: *CoRR* abs/1407.2535 (2014). URL: http://arxiv.org/abs/1407.2535.

[MAP11] Gianfranco Minati, Mario Abram, and Eliano Pessa. *Methods, Models, Simulations and Approaches Towards a General Theory of Change: Proceedings of the Fifth National Conference on Systems Science*. 1st. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2011. ISBN: 9789814383325, 9814383325.

[MP13a] Ursula Martin and Alison Pease. "What does mathoverflow tell us about the production of mathematics?" In: *SOHUMAN, 2nd International Workshop on Social Media for Crowdsourcing and Human Computation*. 2013.

[MP13b] B. D. McKay and A. Piperno. "Practical graph isomorphism, II". In: *ArXiv e-prints* (Jan. 2013). arXiv: 1301.1493 [cs.DM].

[O'S09] Bryan O'Sullivan. "Making Sense of Revision-Control Systems". In: *Communications of the Association for Computing Machinery (CACM)* 52.9 (2009), pp. 57–62.

[Ray99] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1999. ISBN: 1565927249. URL: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/.

[redm] *Redmine*. URL: http://www.redmine.org.

[Rus13] Matthew A. Russell. *Mining the Social Web, 2nd Edition*. O'Reilly Media, 2013. URL: http://shop.oreilly.com/product/0636920030195.do.

[Sha16] Sarah Sharp. *FOSS Heartbeat analyses the health of a community of contributors*. 2016. URL: https://sarahsharp.github.io/foss-heartbeat.

[Smi16] Arfon Smith. *The shape of open source*. 2016. URL: https://github.com/blog/2195-the-shape-of-open-source.

[sota] Siofra Pratt. *How to: Use GitHub to Find Super-Talented Developers*. URL: https://www.socialtalent.co/blog/how-to-use-github-to-find-super-talented-developers.

[stofl] *StackOverflow*. URL: http://stackoverflow.com/.

[trac] *Trac*. URL: https://trac.edgewall.org/.

[Vas+15] Bogdan Vasilescu et al. "Continuous integration in a social-coding world: Empirical evidence from GitHub. **Updated version with corrections*". In: *CoRR* abs/1512.01862 (2015). URL: http://arxiv.org/abs/1512.01862.

[VFS13] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. "StackOverflow and GitHub: Associations Between Software Development and Crowdsourced Knowledge". In: SocialCom, 2013. DOI: 10.1109/SocialCom.2013.35.