Metrics

Release 2021-10

https://chaoss.community/metrics

Contents

Common Metrics WG

Focus Area	Goal
Contributions	Identify what contributions are made (e.g., code commit, wiki edit, documentation)
Time	Identify when contributions are made (e.g., time to respond)
People	Identify organizations and individuals who make contributions (e.g. contribution type, demographics)
Place	Identify where contributions occur in terms of physical and virtual places (e.g., GitHub, Chat Channel, Forum, conferences)

Focus Area - Contributions

Goal: Identify what contributions are made (e.g., code commit, wiki edit, documentation)

Metric	Question
Clones	How many copies of an open source project repository have been saved on a local machine?
Programming Language Distribution	What are the different programming languages present in an open source project(s), and what is the percentage of each language?
Technical Fork	What are a number of technical forks of an open source project on code development platforms?
Types of Contributions	What types of contributions are being made?

Clones

Question: How many copies of an open source project repository have been saved on a local machine?

Description

A clone is a copy of a repository saved to a local machine.

Note: Many times clone and technical fork are used interchangeably, but there is a difference between the two. A technical fork is a copy of a repository on the same platform, whereas a clone is a copy on a local machine.

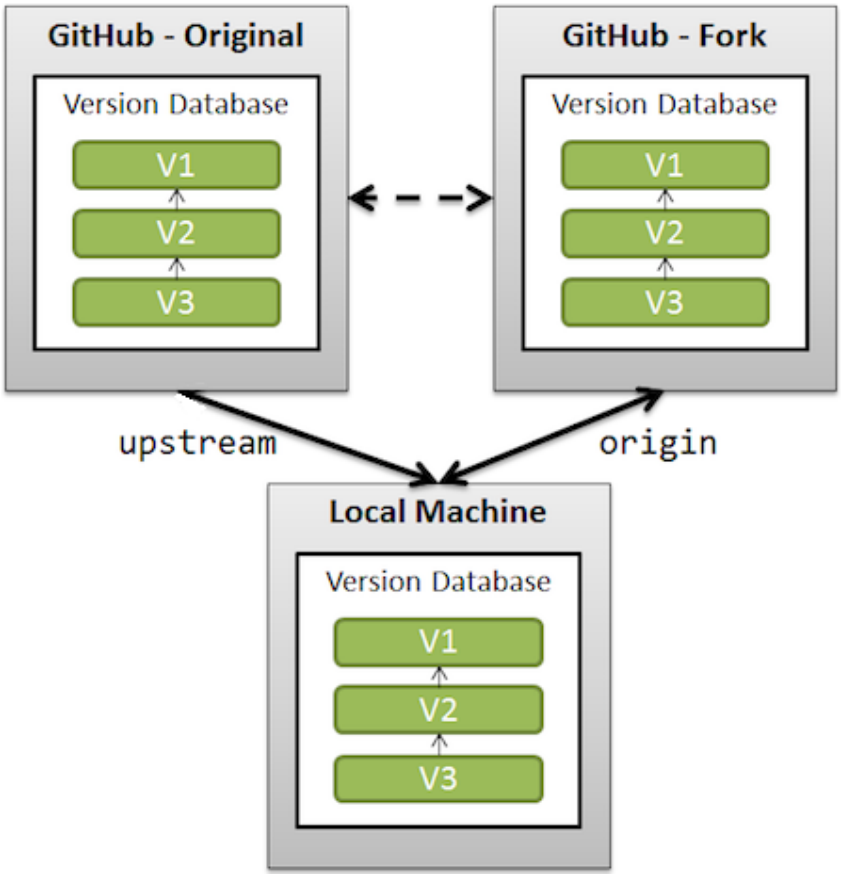


Image is sourced from Stakeoverflow

Objectives

The objective of the Clones metric is to ascertain how many copies of a project are downloaded. Analysis of clones may provide insight into the project's popularity and usage.

Implementation

Filters

- Time Period (e.g., clones by week, month, year)
- Partial clones (offered by gitlab https://docs.gitlab.com/ee/topics/git/partial_clone.html#partial-clone)

Visualizations

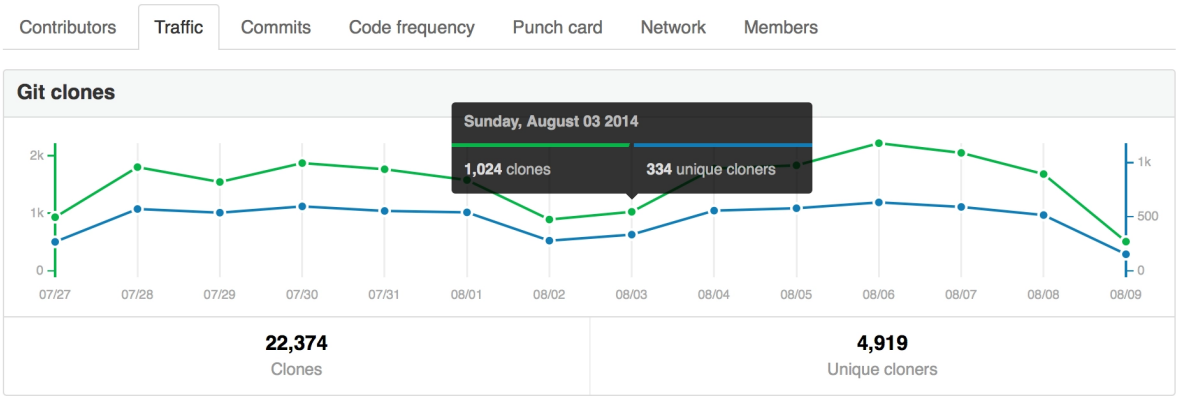


Image is source from Github Blog

References

- What is the difference between forking and cloning a repository?
- What's the difference between a fork and clone?
- [Github API](https://docs.github.com/en/rest/reference/repos#get-repository-clones)

Contributors

- Vinod Ahuja
- Sean Goggins
- Matt Germonprez
- Kevin Lumbard
- Dawn Foster
- Elizabeth Barron

Programming Language Distribution

Question: What are the different programming languages present in an open source project(s), and what is the percentage of each language?

Description

The number of programming languages and the percentage of each language in a project provides some understanding of the skills required from code contributors, as well as the nature of the project itself.

Objectives

This metric will aid newcomers to a particular open source project, as well as provide open source program managers with a perspective on the project's profile, in the context of their own experience and organization. In Value: This metric may be used by developers for identifying projects that rely heavily on languages they use, as part of a job search.

In Evolution: This metric can be useful for identifying changes in the number of files, or lines of code in each language over time. For example, a project may be X% Python, and Y% Javascript at a point in time. Perhaps a year later, the amount of Javascript measured by files, or lines of code, may be greater as a result of a project's shift in focus toward user experience.

In Risk: This metric can be combined with dependency metrics to determine if there is a prominent language used in a project, but for which a dependency scanner is not yet identified.

In DEI: When inclusive, diverse, and equitable communities are identified, they will have some degree of language distribution.

As an individual looks for new projects to work on, knowing which projects rely on languages they already know, or want to learn, can be one of a number of "personal filters".

In general, this metric is useful for OSPOs, and community managers aiming to understand which languages are most prominent, and perhaps which languages are little used, but critical.

Implementation

Language distribution takes into account different properties of each file in a repository with an a priori identifiable computer programming language. As new languages emerge, there may be initial periods where counting tools do not recognize their extensions, in which case they may be counted as "other". Such periods are typically brief.

Filters

- Time
- Number of Files - The number of files of each language.

Programming Language	Language Files
Python	246
JSON	28
BASH	26
Plain Text	14
Shell	12
gitignore	8
YAML	6
Makefile	4
R	2
Docker ignore	2
Dockerfile	2
Markdown	2

- Lines of Code - The percentage of lines of code for each language.

Language	Percentage of LOC
Python	94.60%
BASH	2.48%
Shell	1.08%
JSON	0.52%
R	0.42%
Plain Text	0.38%
Makefile	0.16%
YAML	0.15%
gitignore	0.12%
Dockerfile	0.05%
Docker ignore	0.03%
Markdown	0.00%

Either lines of code, or files, could be presented as absolute numbers, or percentages, depending on the application of the metric. In many cases, a simple count of files is useful, while the absolute number of lines of code can be difficult to differentiate because the numbers are much larger.

Tools Providing the Metric

The Augur-Community-Reports repository provides this metric currently

GrimoireLab provides this information through the proxy of file extensions

Augur provides this information in its frontend, as well as through an API endpoint.

Data Collection Strategies

The contents of a repository can be counted by iterating through each file, though several libraries exist, including the one used by Augur: https://github.com/boyter/scc

File extensions for some languages, like Jupyter Notebooks, might be excluded because they obscure the actual language used.

References

- https://github.com/boyter/scc

Contributors

- Dawn Foster
- Beth Hancock
- Matt Germonprez
- Elizabeth Barron
- Daniel Izquierdo
- Kevin Lumbard
- Sean Goggins

Technical Fork

Question: What are a number of technical forks of an open source project on code development platforms?

Description

A technical fork is a distributed version control copy of a project. The number of technical forks indicates the number of copies of a project on the same code development platform.

Note: Many times technical fork and clones are used interchangeably, but there is a difference between the two. A technical fork is a copy of a repository on the same platform, whereas a clone is a copy on a local machine.

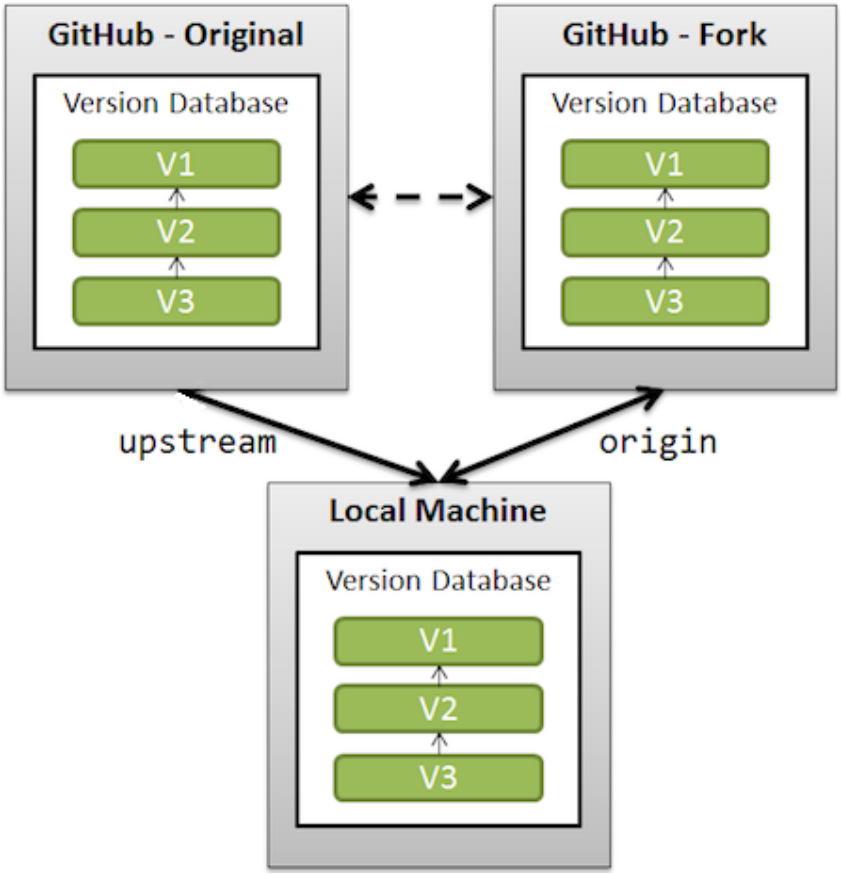


Image is sourced from Stakeoverflow

Objectives

The objective of the Technical Fork metric is to ascertain how many copies of a project exist on a code development platform. Analysis of technical forks may provide insight into forking intentions (different types of forks such as contributing, and non-contributing forks).

Implementation

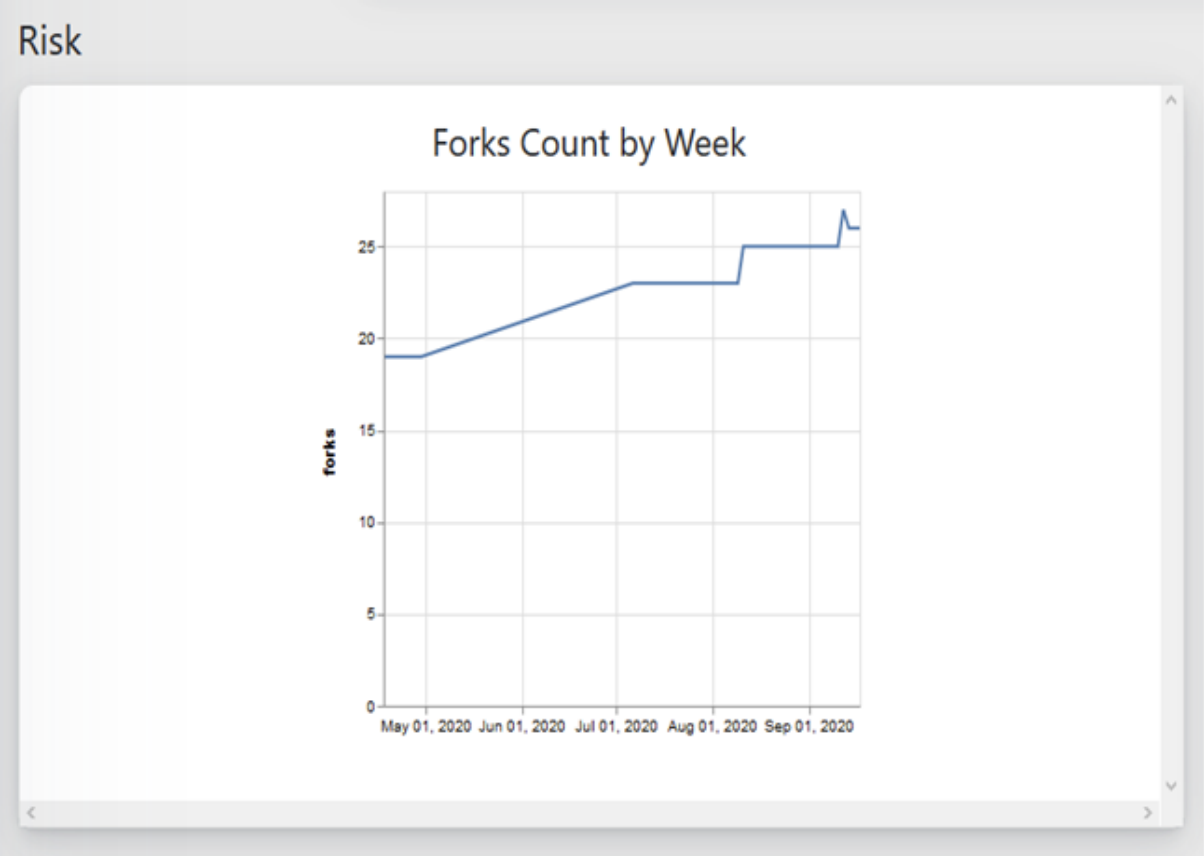
Filters

- Time Period (e.g., Weekly, Monthly, Annually)

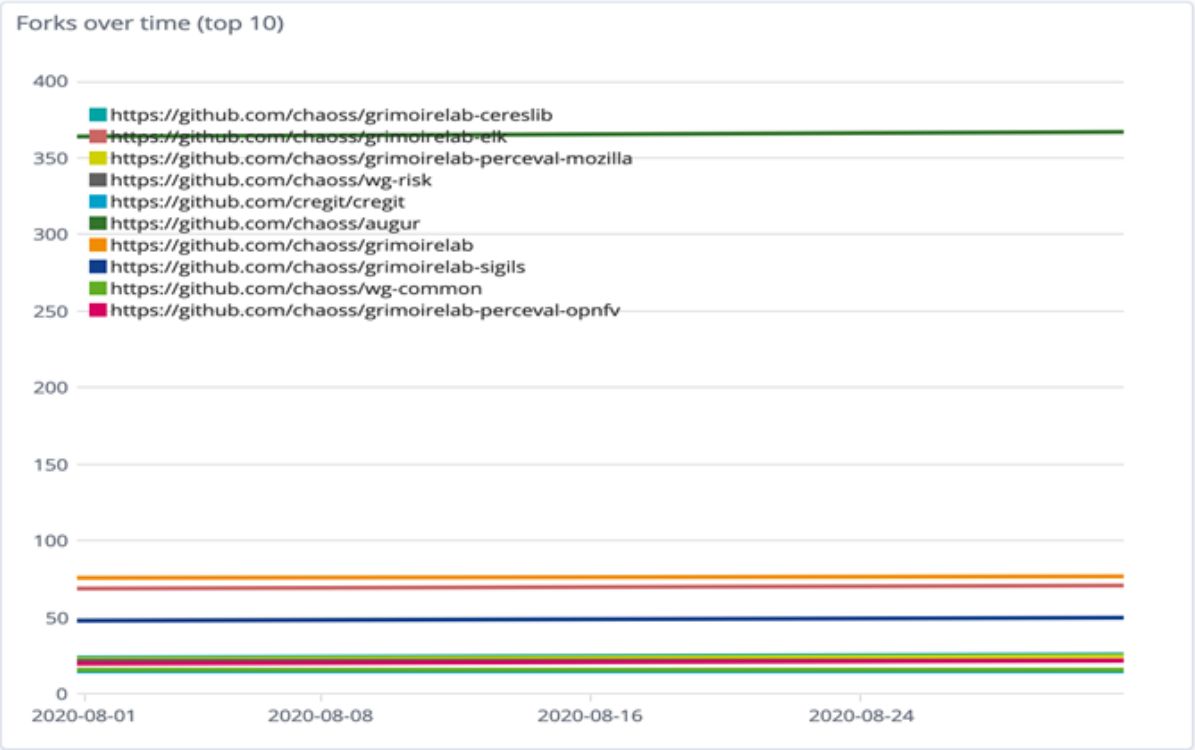
- Ratio of contributing fork to total forks (A contributing fork is a fork that has opened a change request against the original repository.)
- Ratio of non-contributing fork to total forks (A non-contributing fork is a fork that has never opened a change request against the original repository.)

Visualizations

Augur Implementation



GrimoireLab Implementation



Tools Providing the Metric

- Augur
- GrimoireLab

References

- https://help.github.com/en/enterprise/2.13/user/articles/fork-a-repo
- https://opensource.com/article/17/12/fork-clone-difference
- Github API (**https://developer.github.com/v3/repos/forks/#list-forks**)
- GitLab API (**https://docs.gitlab.com/ee/api/projects.html#list-forks-of-a-project**)
- Bitbucket API (**https://developer.atlassian.com/bitbucket/api/2/reference/resource/repositories/%7Bworkspace%7D/%7Brepo_slug%7D/forks**)

Contributors

- Vinod Ahuja
- Sean Goggins
- Matt Germonprez
- Kevin Lumbard
- Dawn Foster
- Elizabeth Barron

Types of Contributions

Question: What types of contributions are being made?

Description

Multiple, varied contributions make an open source project healthy. Many projects have community members who do not write code but contribute in equally valuable ways by managing the community, triaging bugs, evangelizing the project, supporting users, or helping in other ways.

Objectives

A variety of contribution types can demonstrate that a project is mature and well-rounded with sufficient activity to support all aspects of the project, and enable paths to leadership that are supportive of a variety of contribution types and people with varying expertise beyond coding.

Implementation

How contributions are defined, quantified, tracked and made public is a challenging question. Answers may be unique to each project and the following suggestions are a starting point. As a general guideline, it is difficult to compare different contribution types with each other and they might better be recognized independently.

- The following list can help with identifying contribution types:
    - Writing Code
    - Reviewing Code
    - Bug Triaging
    - Quality Assurance and Testing
    - Security-Related Activities
    - Localization/L10N and Translation
    - Event Organization
    - Documentation Authorship
    - Community Building and Management
    - Teaching and Tutorial Building
    - Troubleshooting and Support
    - Creative Work and Design
    - User Interface, User Experience, and Accessibility
    - Social Media Management
    - User Support and Answering Questions
    - Writing Articles
    - Public Relations - Interviews with Technical Press
    - Speaking at Events
    - Marketing and Campaign Advocacy

  - Website Development
  - Legal Council
  - Financial Management

Data Collection Strategies

- **Interview or Survey:** Ask community members to recognize others for their contributions to recognize contribution types that have previously not been considered.
  - Who in the project would you like to recognize for their contributions? What did they contribute?
- **Observe project:** Identify and recognize leads of different parts of the project.
  - What leaders are listed on the project website or in a repository?
- **Capture Non-code Contributions:** Track contributions through a dedicated system, e.g., an issue tracker.
  - Logging can include custom information a project wants to know about non-code contributions to recognize efforts.
  - Proxy contributions through communication channel activity. For example, If quality assurance members (QA) have their own mailing list, then activity around QA contributions can be measured by proxy from mailing list activity.
- **Collect Trace Data:** Measure contributions through collaboration tool log data.
  - For example, code contributions can be counted from a source code repository, wiki contributions can be counted from a wiki edit history, and email messages can be counted from an email archive
- **Automate Classification:** Train an artificial intelligence (AI) bot to detect and classify contributions.
  - An AI bot can assist in categorizing contributions, for example, help requests vs. support provided, or feature request vs. bug reporting, especially if these are all done in the same issue tracker.

Other considerations:

- Especially with automated reports, allow community members to opt-out and not appear on the contribution reports.
- Acknowledge imperfect capture of contribution types and be explicit about what types of contributions are included.
- As a project evolves, methods for collecting types of contributions will need to adapt. For example, when an internationalization library is replaced with a different one, project activity around localization conceivably produces different metrics before and after the change.
- Account for activity from bots when mining contribution types at large scale.

References

- https://medium.com/@sunnydeveloper/revisiting-the-word-recognition-in-foss-and-the-dream-of-open-credentials-d15385d49447
- https://24pullrequests.com/contributing
- https://smartbear.com/blog/test-and-monitor/14-ways-to-contribute-to-open-source-without-being/
- https://wiki.openstack.org/wiki/AUCRecognition

- https://www.drupal.org/drupalorg/blog/a-guide-to-issue-credits-and-the-drupal.org-marketplace

Focus Area - Time

Goal: Identify when contributions are made (e.g., time to respond)

Metric	Question
Activity Dates and Times	What are the dates and timestamps of when contributor activities occur?
Burstiness	How are short timeframes of intense activity, followed by a corresponding return to a typical pattern of activity, observed in a project?
Review Cycle Duration within a Change Request	What is the duration of a review cycle within a single change request?
Time to First Response	How much time passes between when an activity requiring attention is created and the first response?
Time to Close	How much time passes between creating and closing an operation such as an issue, change request, or support ticket?

Activity Dates and Times

Question: What are the dates and timestamps of when contributor activities occur?

Description

Individuals engage in activities in open source projects at various times of the day. This metric is aimed at determining the dates and times of when individual activities were completed. The data can be used to probabilistically estimate where on earth contributions come from in cases where the time zone is not UTC.

Objectives

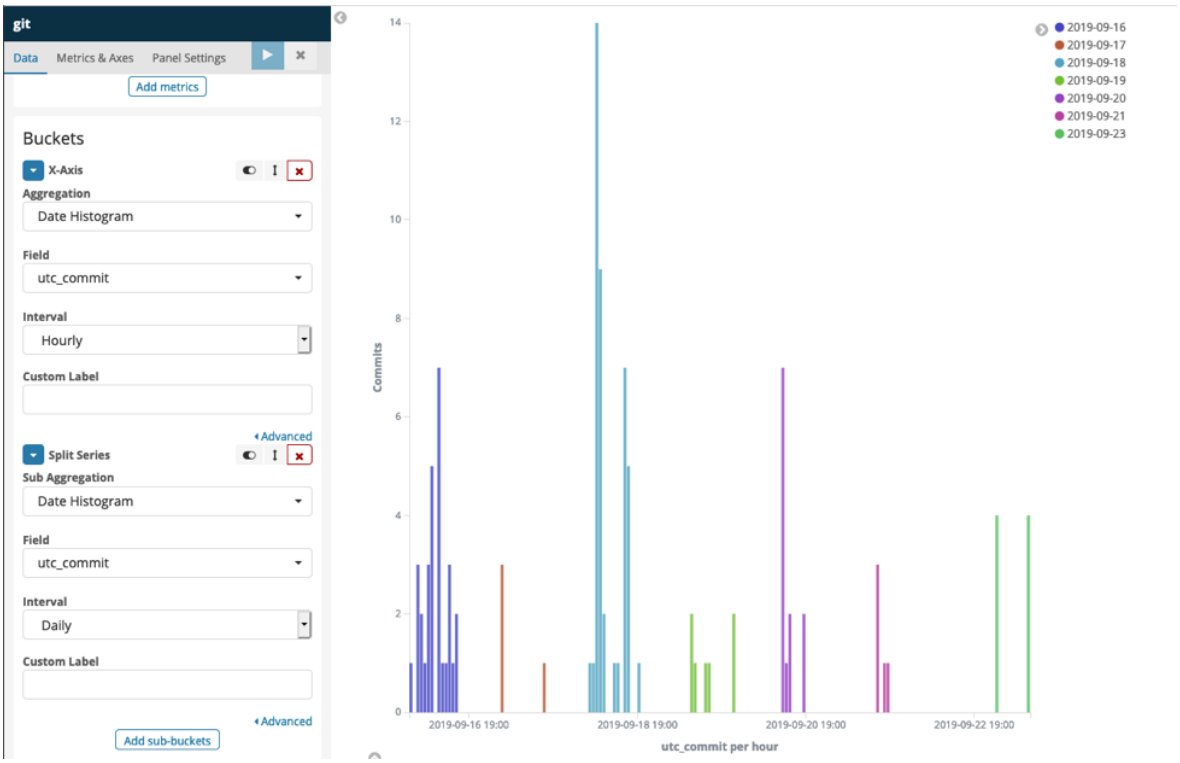
- Improve transparency for employers about when organizational employees are engaging with open source projects
- Improve transparency for open source project and community managers as to when activity is occurring

Implementation

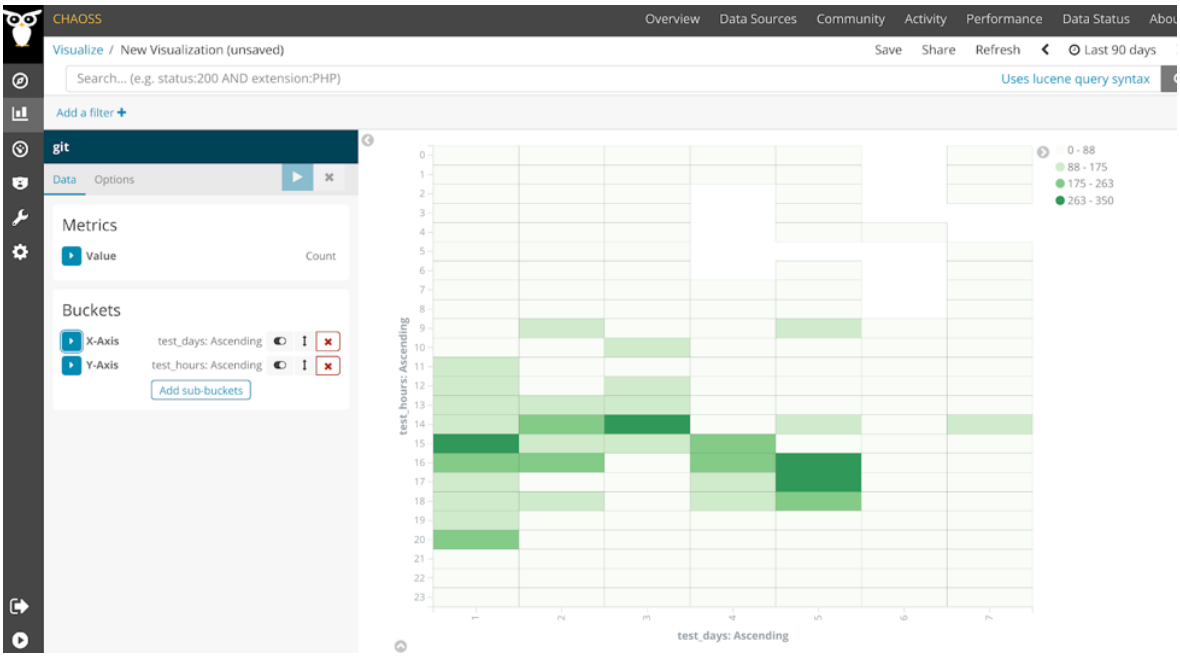
Filters

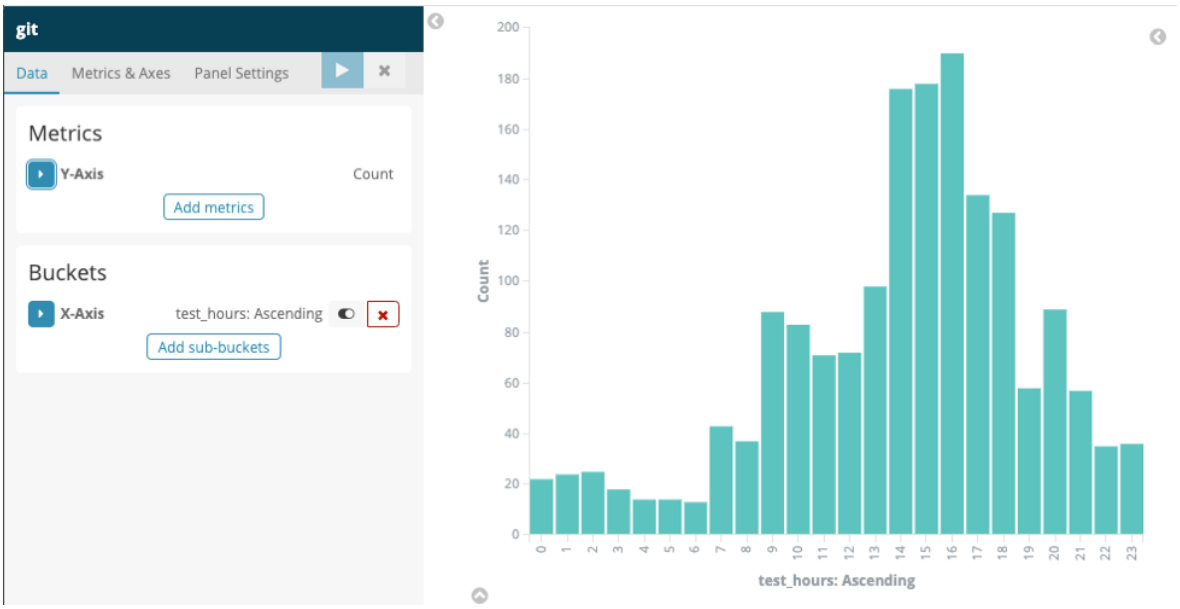
- Individual by Organization
- Aggregation of time by UTC time
    - Can show what times across the globe contributions are made; when the project is most active.
- Aggregation of time by local time
    - Can show what times of day in their local times they contribute. Conclusions about the If contributions are more during working hours, or if contributions are more during evening hours.
- Repository ID
- Segment of a community, (e.g., GrimoireLab has more EU time zones activity and Augur more US time zones activity)

Visualizations



Time	author_name	repo_name	tz	utc_author	utc_commit	commit_date
Sep 23rd 2019	Carter Landis	https://github.com/chaoss/augur	-5	September 23rd 2019, 15:42:16.000	September 23rd 2019, 15:42:16.000	September 23rd 2019, 10:42:16.000
Sep 23rd 2019	Matt Snell	https://github.com/chaoss/augur	-5	September 23rd 2019, 13:21:56.000	September 23rd 2019, 13:21:56.000	September 23rd 2019, 08:21:56.000
Sep 23rd 2019	Gabe Heim	https://github.com/chaoss/augur	-5	September 23rd 2019, 12:31:49.000	September 23rd 2019, 12:31:49.000	September 23rd 2019, 07:31:49.000
Sep 23rd 2019	Santiago Dueñas	https://github.com/chaoss/grimoirelab-perceval	2	September 23rd 2019, 12:30:18.000	September 23rd 2019, 12:30:18.000	September 23rd 2019, 14:30:18.000
Sep 23rd 2019	Valerio Cosentino	https://github.com/chaoss/grimoirelab-perceval	2	September 23rd 2019, 11:26:04.000	September 23rd 2019, 11:26:30.000	September 23rd 2019, 13:26:30.000
Sep 23rd 2019	Valerio Cosentino	https://github.com/chaoss/grimoirelab-perceval	2	September 23rd 2019, 11:24:21.000	September 23rd 2019, 11:24:21.000	September 23rd 2019, 13:24:21.000





Tools Providing Metric

GrimoireLab

Augur Date/Timestamps

References

Coordinated Universal Time

Burstiness

Question: How are short timeframes of intense activity, followed by a corresponding return to a typical pattern of activity, observed in a project?

Description

There are a number of reasons that may prompt a sudden increase or decrease in the amount of activity within a repository. These increases and decreases appear both as a sudden change in activity against the average amount of activity. Burstiness is a way of understanding the cycle of activity in existing metrics, like issues, merge requests, mailing lists, commits, or comments. Examples of root causes for bursts in activity include:

- Release cycles
- Global pandemics
- Hackathon activities
- Mentorship programs
- Conferences, meetups, and other events where tools are presented
- Conventional and social media announcements and mentions
- Critical bugs as raising awareness and getting people's attention
- Community design meetings or brainstorming meetings to address a particular issue
- Community members show up from another community that is relying on your project (e.g., dependencies)

Objectives

- To identify impacts of root causes of a burst in activity
- To provide awareness when project activity unknowingly goes up
- To help capture the meaningfulness of increases or decreases in project activity
- To help the community and maintainers prepare for future bursts that follow a pattern
- To help measure the impact of influential external activities
- To differentiate skewed activity versus normal activity

Implementation

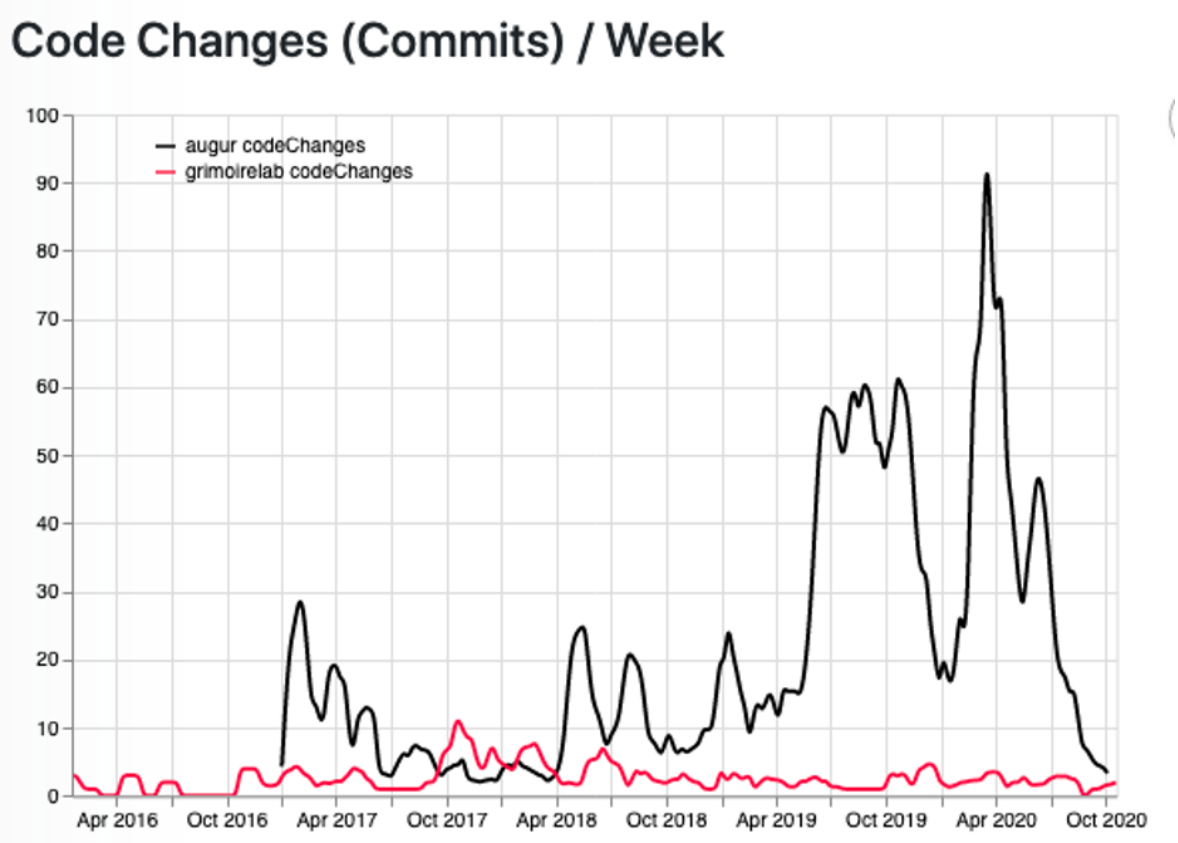
Filters

- Stars
- Forks
- Issues or bug reports
- Labels
- Downloads
- Release Tags

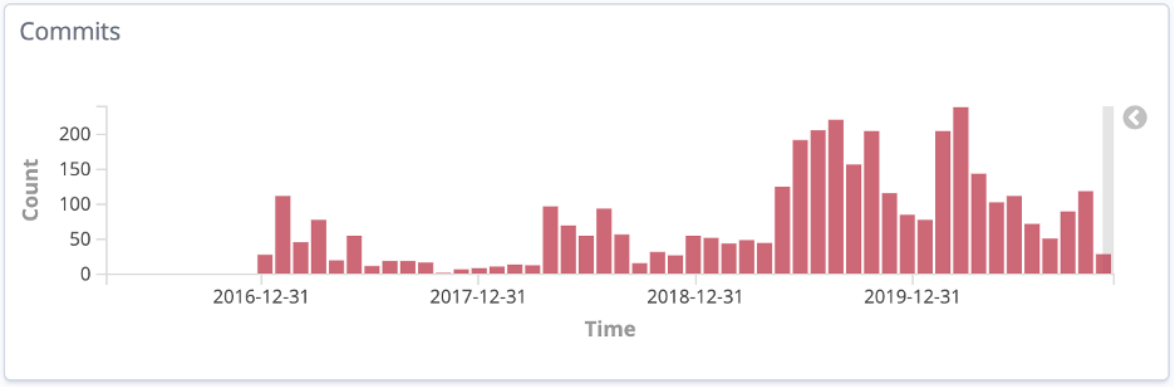
- Change Requests
- Mail List Traffic
- Documentation additions or revisions
- New Repositories
- Feature Requests
- Messaging Conversations
- Conventional and Social Media Activity
- Conference Attendance and Submissions

Visualizations

Augur:



GrimoireLab:



Tools Providing the Metric

- Grimoire Lab
- Augur

Data Collection Strategies

- Quantitative
  - Time box activities identifying deviations away from some norm
  - Outliers for certain thresholds, using statistics like Bollinger Bands to measure stability or volatility: https://en.wikipedia.org/wiki/Bollinger_Bands
- Qualitative Interview Questions
  - Why do you contribute more during a period of time?
  - What do you believe to be the root cause for particular bursts?
  - What impact do different events (e.g., hackathons, mentorship program, or conferences) have on project activity?

References

This metric was inspired by the work of Goh and Barabasi (2008): https://arxiv.org/pdf/physics/0610233.pdf

Review Cycle Duration within a Change Request

Question: What is the duration of a review cycle within a single change request?

Description

A change request is based on one or more review cycles. Within a review cycle, one or more reviewers can provide feedback on a proposed contribution. The duration of a review cycle, or the time between each new iteration of the contribution, is the basis of this metric.

Objectives

This metric provides maintainers with insight on: Code review process decay, as there are more iterations and review cycle durations increase. Process bottlenecks resulting in long code review iterations. Abandoned or semi-abandoned processes in the review cycles, where either the maintainer or the submitter is slow in responding. Characteristics of reviews that have different cyclic pattern lengths.

Implementation

Review Cycle Duration is measured as the time length of one review cycle within a single change request. The duration can be calculated between: The moment when each review cycle begins, defined as the point in time when a change request is submitted or updated. The moment when each review cycle ends, either because the change request was updated and needs a new review or because it was accepted or rejected.

Filter

Average or Median Duration, optionally filtered or grouped by: Number of people involved in review Number of comments in review Edits made to a change request Project or program Organization making the change request Time the change request was submitted Developers who contributed to a change request Change request Number of review cycle on a change request (e.g., filter by first, second, … round)

Visualizations

Tools Providing the Metric

References

Example of data that could be used to develop the metric: https://gerrit.wikimedia.org/r/c/mediawiki/core/+/194071

Time to First Response

Question: How much time passes between when an activity requiring attention is created and the first response?

Description

The first response to an activity can sometimes be the most important response. The first response shows that a community is active and engages in conversations. A long time to respond to an activity can be a sign that a community is not responsive. A short time to respond to an activity can help to engage more members into further discussions and within the community.

Objectives

Identify cadence of first response across a variety of activities, including PRs, Issues, emails, IRC posts, etc. Time to first response is an important consideration for new and long-time contributors to a project along with overall project health.

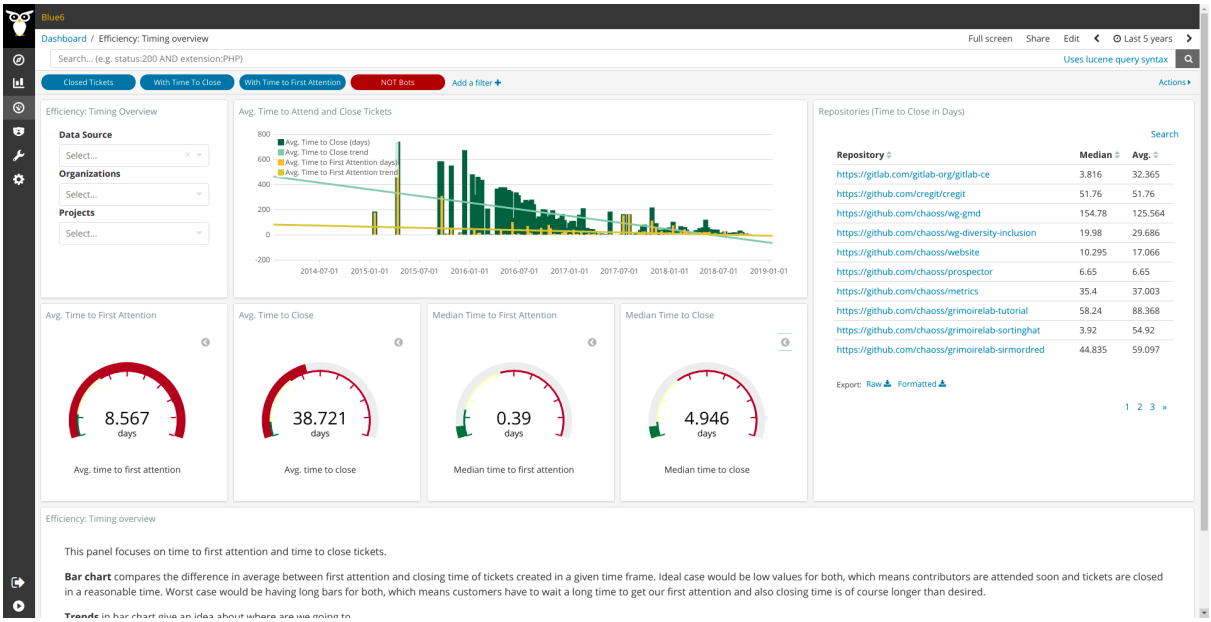
Implementation

Time to first response of an activity = time first response was posted to the activity - time the activity was created.

Filters

- Role of responder, e.g., only count maintainer responses
- Automated responses, e.g., only count replies from real people by filtering bots and other automated replies
- Type of Activity, e.g., issues (see metric Issue Response Time), emails, chat, change requests

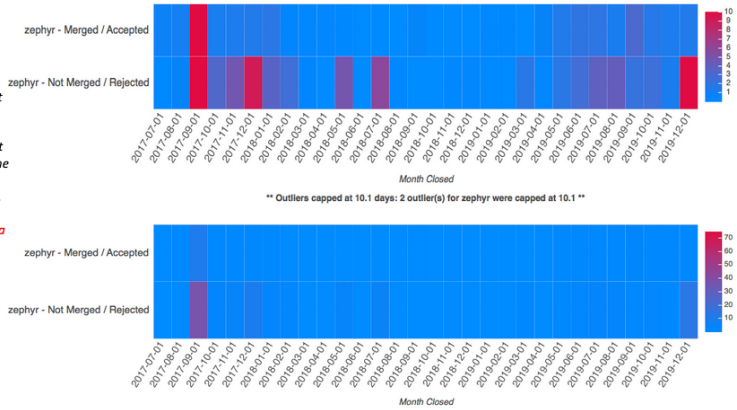
Visualizations



Mean Days to First Response for Closed Pull Requests

Some Internal Slowing, But Outperforming Other Repositories

The heat maps illustrate variance in mean pull request duration to the first response for Zephyr over time. The left side compares Zephyr against itself, and the heat map on the right side is on a scale more aligned with the competition. Note especially that the internal Zephyr gradient has a max of 10 days, while the external comparisons have a max of 75 days.

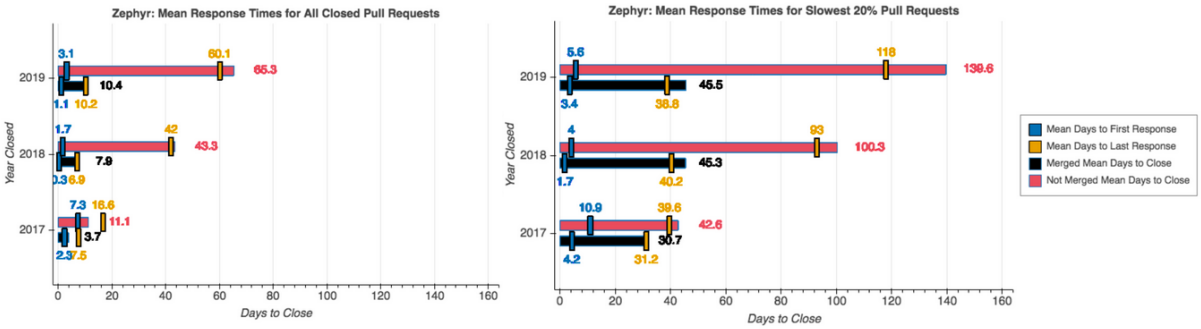


Internal Zephyr PR Performance (0 - 10 scale)

Zephyr PR Performance Compared to Others (0 - 75 scale)

Mean Response Times (Days) For Closed Pull Requests

Long Running Pull Requests Are Usually Rejected



The length of the black bar illustrates the total number of days that rejected and merged pull requests were open. The blue bar shows that, for the Zephyr project, we see the mean time to first response improving significantly for both rejected and merged pull requests from 2017 - 2019. The orange bar shows the last response or event associated with a pull request.

Tools Providing the Metric

- GrimoireLab Panel: Efficiency Timing Overview
- Kata Containers dashboard efficiency panel

References

Time to Close

Question: How much time passes between creating and closing an operation such as an issue, change request, or support ticket?

Description

The time to close is the total amount of time that passes between the creation and closing of an operation such as an issue, change request, or support ticket. The operation needs to have an open and closed state, as is often the case in code review processes, question and answer forums, and ticketing systems.

Related metric: Issue Resolution Duration

Objectives

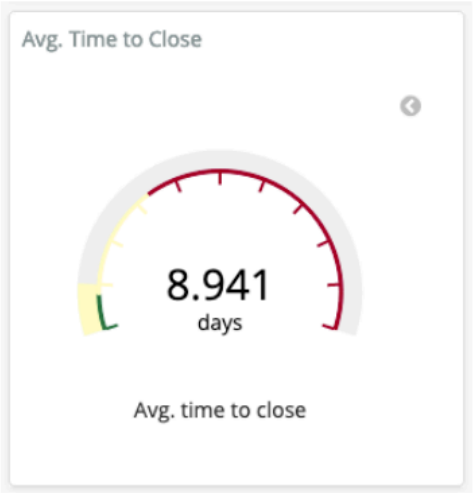
1. Determining how responsive a community is can help efforts to be inclusive, attract, and retain new and existing contributors.
2. Identifying characteristics of operations that impact an operation closing quickly or slowly (e.g., finding best practices, areas of improvement, assess efficiency).
3. Identifying bias for timely responses to different community members.
4. Detecting a change in community activity (e.g., to indicate potential maintainer burnout, reduction in the diversity of contributions)
5. Understand how the time to close an issue or change request is related to merge success or failure.

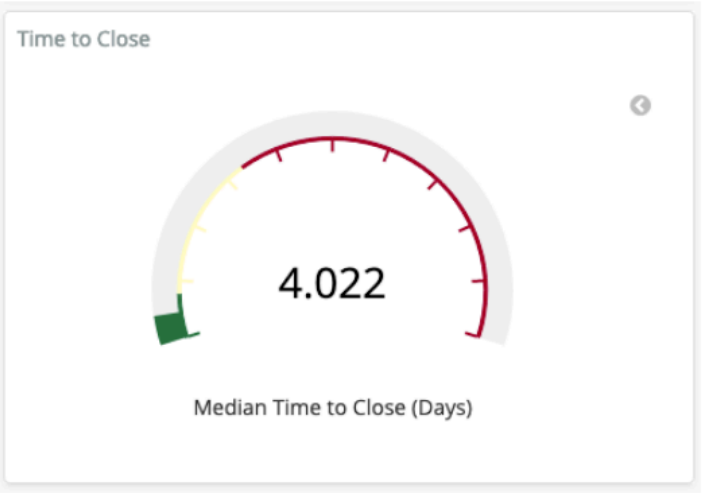
Implementation

Filters

- Creator of operation (e.g., new contributor vs. maintainer)
- First closed, final closed
- Labels (e.g., bug vs. new feature)
- Change Request Merge Status (e.g. Time to Merge or Time to Close without Merge)

Visualizations





Tools Providing the Metric

Augur implementation:

- Issue Close Duration
- Issue Duration
- Issue Response Time

GrimoireLab implementation:

- Pull Requests Efficiency
- Issues Efficiency
- Efficiency:TimingOverview

Data Collection Strategies

The time to close metric may be contextual based on the project activity and objectives. For example, the time to close a bug report may provide different information than the time to close a new feature request. Data collection strategies should address different project objectives. Other variables that may influence these processes are:

- Issue Tracking Systems: the type of issue such as bug report, blueprint (OpenStack nomenclature), user story, feature request, epic, and others may influence how long this event takes to be closed. Other variables, such as the priority or severity may help to advance how quickly this event will be closed.
- Change Request Processes: this depends on the change request infrastructure, as Gerrit, GitHub or mailing lists (as in the Linux Kernel) and may differ depending on how complicated the process is. For example, newcomers or advanced and experienced developers will proceed in different ways and with more or less time required.
- Question and Answer Forum: this depends on the quality of the answer and the opinion of the person asking the question. A valid answer is marked, and the process is closed once the person questioning has successfully found a correct answer to their question.

References

- “Practice P.12: Respond to all submissions” from “Appendix to: Managing Episodic Volunteers in Free/Libre/Open Source Software Communities” by Ann Barcomb, Klaas-Jan Stol, Brian Fitzgerald and Dirk Riehle: https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/13519

Focus Area - People

Goal: Identify organizations and individuals who make contributions (e.g. contribution type, demographics)

Metric	Question
Bot Activity	What is the volume of automated bot activity?
Contributors	Who are the contributors to a project?
Contributor Location	What is the location of contributors?
Organizational Diversity	What is the organizational diversity of contributions?

Bot Activity

Question: What is the volume of automated bot activity?

Description

A bot is a piece of software that supports project activities. Bots provide an increasingly important role in open source projects, helping coordinate open source project work. Bots support various workflows including issue and merge request management and contributor agreements. Bots give open source software projects numerous options in deciding how to effectively manage project work.

Objectives

The Bot Activity metric helps differentiate project activity between people and automated applications (e.g., bots). In doing so, community managers can better seperate such things as increases in new contributors, CLA bot activity, and bot-related access control. This metric may provide additional context for:

- Time to first response
- Build process timing
- Social considerations -- it's not just all robots
- An indication of maturity
- Indication of community size
- Indication of project transparency

Implementation

- Ratio of bot to human activity over time
- Average number of bots over time

Filters

- Bots that have user profiles
- Bots that require human interaction
- Bots that function by themselves (automated)
- Bots that assist with software development
- Bots that assist with communication
- Bots that assist with access control
- Bots that assist with inclusivity
- Platform where the bot is used (e.g., GitHub, Slack)

Visualizations

image

From: https://k8s.devstats.cncf.io/d/5/bot-commands-repository-groups?orgId=1&var-period=w&var-repogroup_name=Kubernetes&var-repo_name=kubernetes%2Fkubernetes&var-commands=All

References

- 6- Bots to Better Your Open Source Project
- GiHub's Hubot

Contributors

- Sean Goggins
- Matt Germonprez
- Kevin Lumbard
- Dawn Foster
- Elizabeth Barron
- Vinod Ahuja
- Matt Cantu

Contributors

Question: Who are the contributors to a project?

Description

A contributor is defined as anyone who contributes to the project in any way. This metric ensures that all types of contributions are fully recognized in the project.

Objectives

Open source projects are comprised of a number of different contributors. Recognizing all contributors to a project is important in knowing who is helping with such activities as code development, event planning, and marketing efforts.

Implementation

Collect author names from collaboration tools a project uses.

Aggregators:

- Count. Total number of contributors during a given time period.

Parameters:

- Period of time. Start and finish date of the period. Default: forever. Period during which contributions are counted.

Filters

By location of engagement. For example:

- Commit authors
- Issue authors
- Review participants, e.g., in pull requests
- Mailing list authors
- Event participants
- IRC authors
- Blog authors
- By release cycle
- Timeframe of activity in the project, e.g, find new contributors
- Programming languages of the project
- Role or function in project

Visualizations

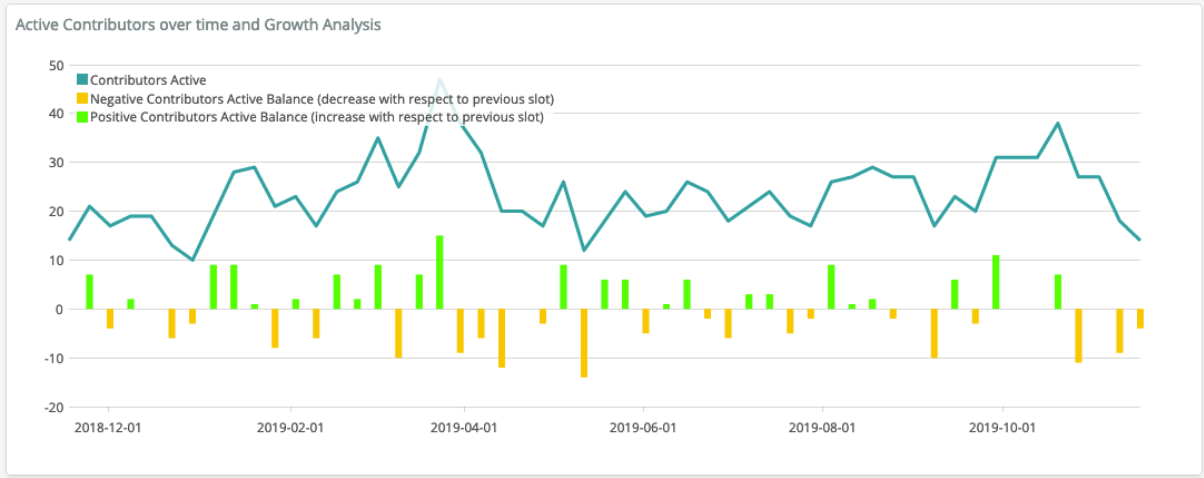
1. List of contributor names (often with information about their level of engagement)

Lines of code added by the top 10 authors

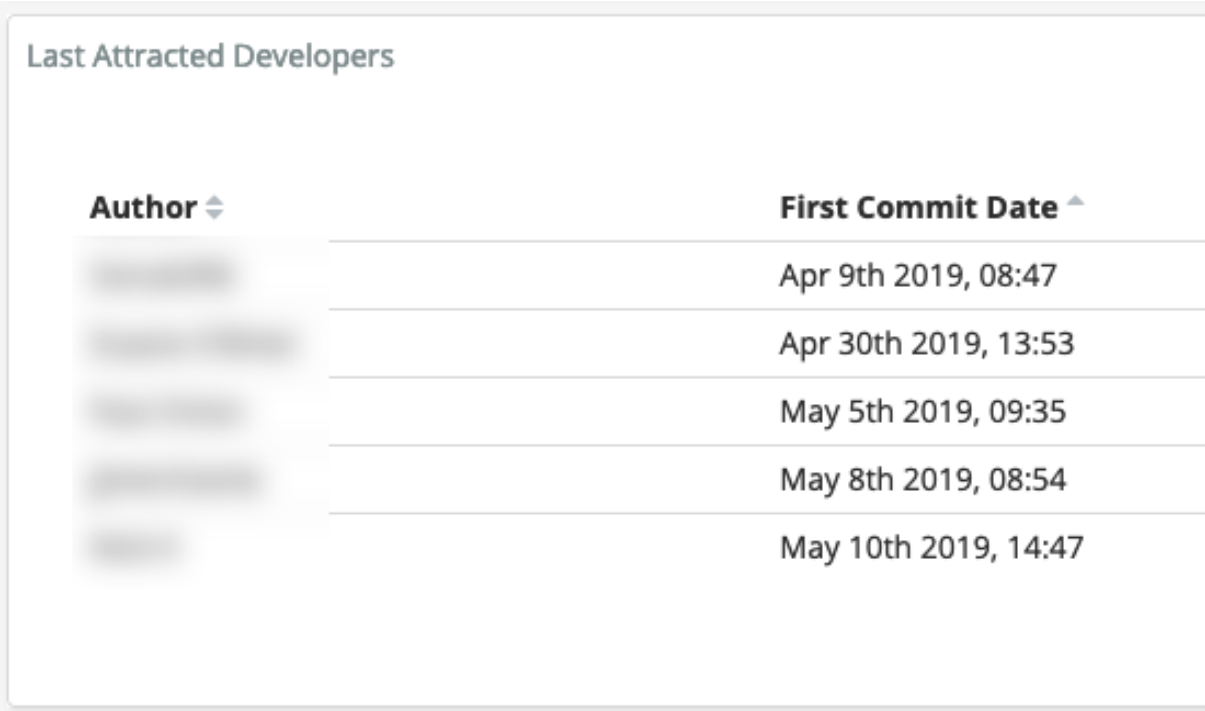
Author	2012	2013	2014	2015	2016	2017	2018
[redacted]	0	133	0	3444	37	12905	1361
[redacted]	0	0	0	0	0	0	59
[redacted]	0	0	0	33	0	0	0
[redacted]	0	0	0	0	0	0	33
[redacted]	0	0	0	0	0	17	0
[redacted]	0	0	0	7	0	0	0
[redacted]	0	0	0	1	0	0	0

1. Summary number of contributors

1. Change in the number of active contributors over time



1. New contributors (sort list of contributors by date of first contribution)



Tools Providing the Metric

- GrimoireLab
- Augur

Data Collection Strategies

As indicated above, some contributor information is available via software such as GrimoireLab and Augur. However, some contributor insights are less easily obtained via trace data. In these cases, surveys with community members or event registrations can provide the desired information. Sample questions include:

- Interview question: Which contributors do not typically appear in lists of contributors?
- Interview question: Which contributors are often overlooked as important contributors because their contributions are more “behind the scenes”?
- Interview question: What other community members do you regularly work with?

Additionally, surveys with community members can provide insight to learn more about contributions to the project. Sample questions include:

- Likert scale [1-x] item: I am contributing to the project
- Matrix survey item: How often do you engage in the following activities in the project?
    - Column headings: Never, Rarely(less than once a month), Sometimes (more than once a month), Often(once a week or more)
    - Rows include: a) Contributing/reviewing code, b) Creating or maintaining documentation, c) Translating documentation, d) Participating in decision making about the project’s development, e) Serving as a community organizer, f) Mentoring other contributors, g) Attending events in person, h) Participating through school or university computing programs, i) Participating through a program like Outreachy, Google Summer of Code, etc., j) Helping with the ASF operations (e.g., board meetings or fundraising)

References

Contributor Location

Question: What is the location of contributors?

Description

Geographical location from which contributors contribute, where they live, or where they work.

Objectives

To determine global locations of contributors in an effort to understand work practices and times zones. To identify where contributions do not come from in an effort to improve engagement in these areas.

Implementation

Filters

Filter contributions by:

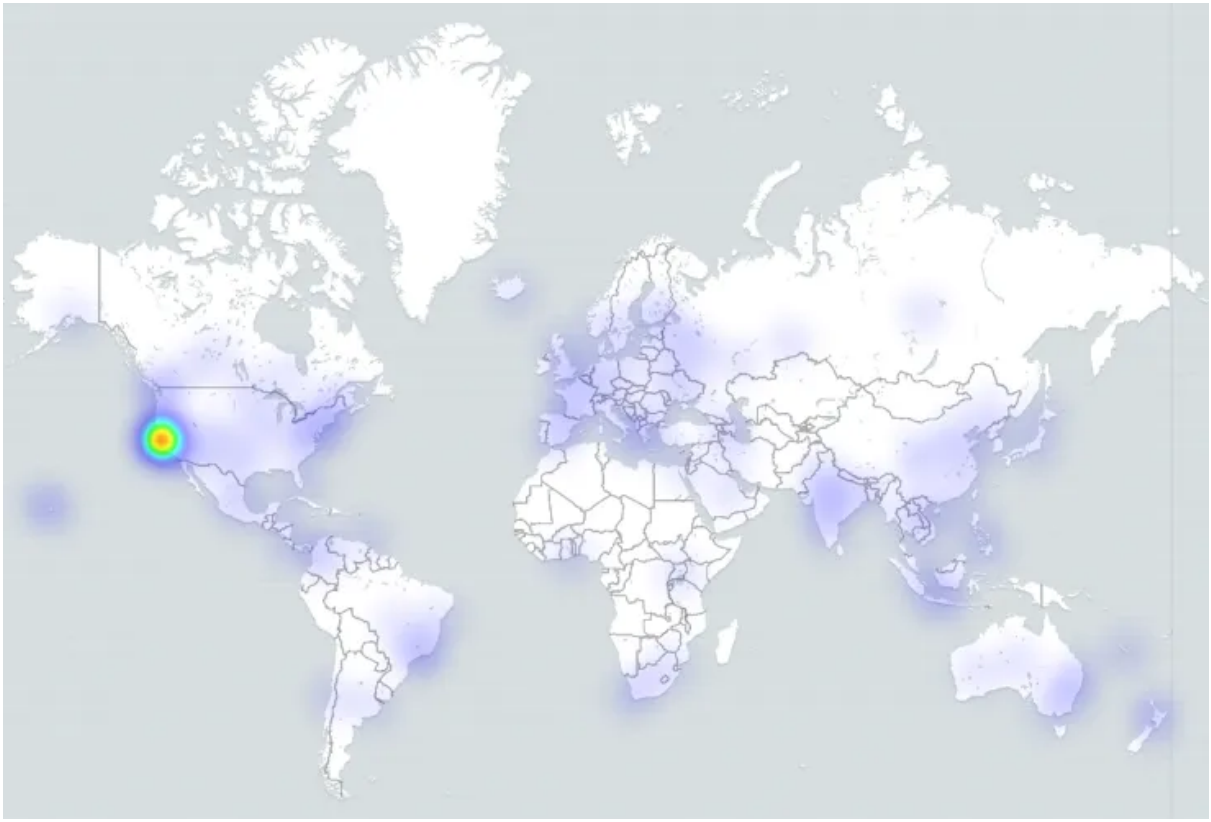
- **Location.** Attempt to group locations in regions to have multiple levels of reporting. Location is a purposely ambiguous term in this context, and could refer to region, country, state, locale, or time zone.
- **Period of time.** Start and finish date of the period. Default: forever. Period during which contributions are counted.
- **Type of contributor**, for example:
    - Repository authors
    - Issue authors
    - Code review participants
    - Mailing list authors
    - Event participants
    - IRC authors
    - Blog authors
    - By release cycle
    - Programming languages of the project
    - Role or function in project

Visualizations

Dot Density Map:

Source: https://chaoss.biterg.io/goto/a62f3584a41c1c4c1af5d04b9809a860

Visual heat map:



Source: https://blog.bitergia.com/2018/11/20/ubers-community-software-development-analytics-for-open-source-offices

Tools providing the metric

- GrimoireLab
- Augur

Data Collection Strategies

Different approaches can be used to collect information about location:

- Collect the location information from a contributor's profile in the system of engagement.
- Use IP address geolocation of the most frequent locations that contributions are made.
- Infer geographical location from the timestamp in contributions.
- Survey contributors.

The key challenge for collecting data is determining the location of the contributor. Best practice would be to leverage any profile information available from the system of engagement, and if that is not available then use IP geolocation to determine the most frequent location of contribution from that individual. Note that contributors may enter in their profile information false or nonsensical location information (e.g., "Earth" or "Internet"). Note that IP geolocation can provide large numbers of false positives due to use of VPNs or other IP masking tools.

An additional consideration would be the use of external data collection tools such as community surveys or event registration data that could cross reference systems of engagement profiles. Contributor location data could be collected inline with event attendee demographics and speaker demographics.

References

- Gonzalez-Barahona, J. M., Robles, G., Andradas-Izquierdo, R., & Ghosh, R. A. (2008). Geographic origin of libre software developers. *Information Economics and Policy*, *20*(4), 356-363.

Organizational Diversity

Question: What is the organizational diversity of contributions?

Description

Organizational diversity expresses how many different organizations are involved in a project and how involved different organizations are compared to one another.

Objectives

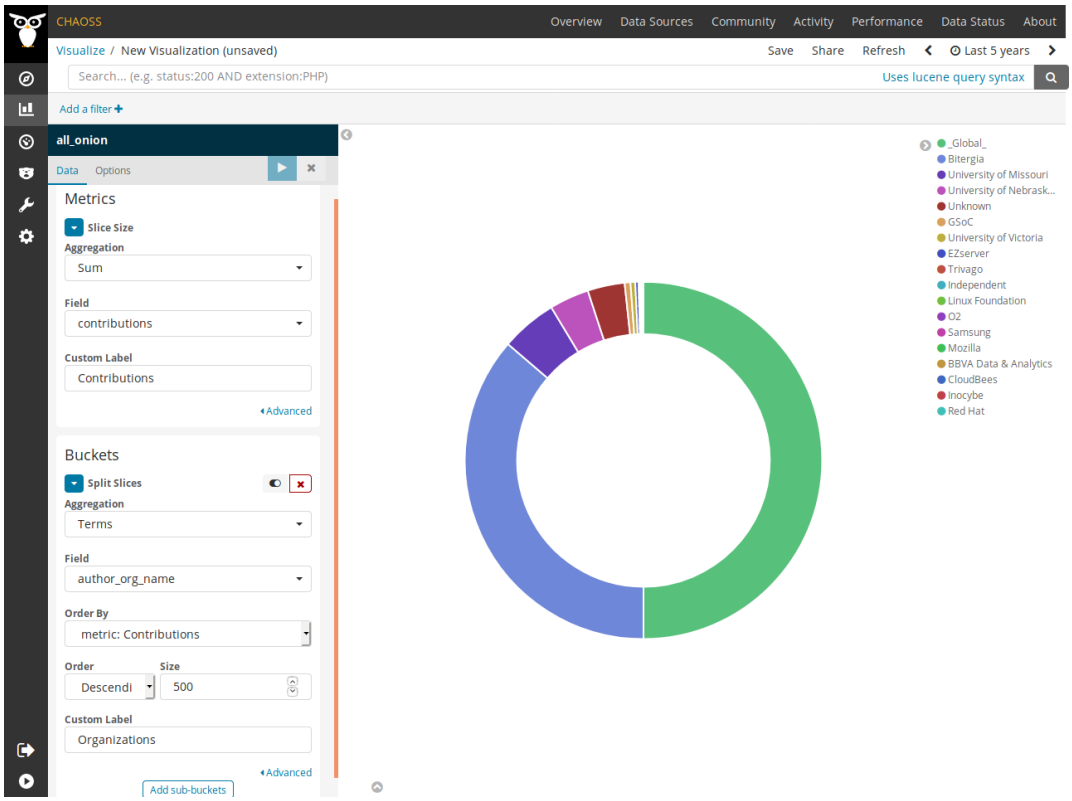
- Get a list of organizations contributing to a project.
- See the percentage of contributions from each organization within a defined period of time.
- See the change of composition of organizations within a defined period of time.
- Get a list of people that are associated with each organization.

Implementation

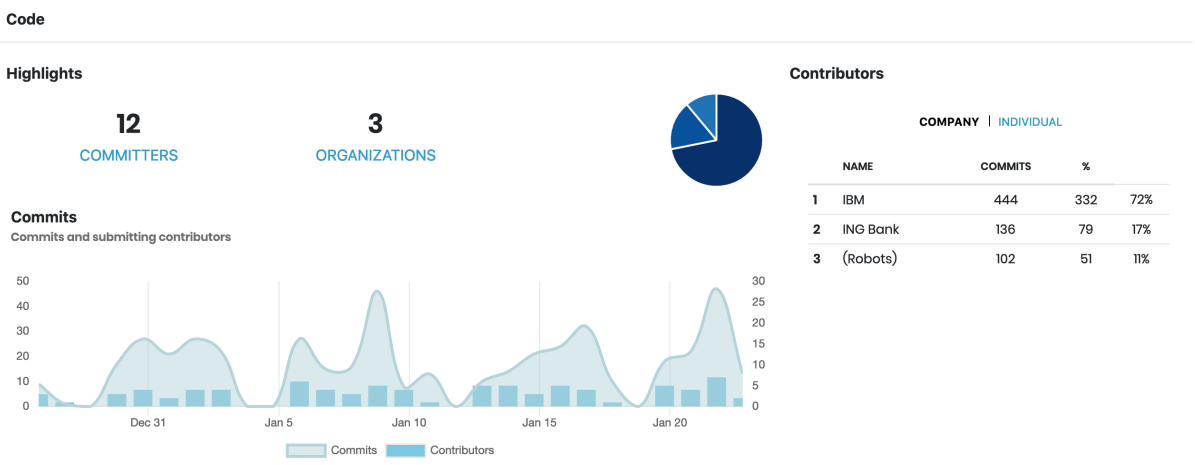
- Collect data from data sources where contributions occur.
- Identify contributor affiliations to get a good estimate of which organizations they belong to.
- Correlate information about contributions, assigning each to appropriate organization.
- Depending on the needs of the project, you may want to consider such issues as how to handle multiple email addresses, affiliation changes over time, or contractor vs. employee.

Tools Providing the Metric

- GrimoireLab supports organizational diversity metrics out of the box. The GrimoireLab SortingHat manages identities. The GrimoireLab Hatstall user interface allows correcting organizational affiliation of people and even recording affiliation changes.
    - View an example visualization on the CHAOSS instance of Bitergia Analytics.
    - Download and import a ready-to-go dashboard containing examples for this metric visualization from the GrimoireLab Sigils panel collection.
    - Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
        - Create a new Pie chart
            - Select the `all_onion` index
            - Metrics Slice Size: `Sum` Aggregation, `contributions` Field, `Contributions` Custom Label
            - Buckets Split Slices: `Terms` Aggregation, `author_or_name` Field, `metric: Contributions` Order By, `Descending` Order, `500` Size, `Organization` Custom Label
        - Example Screenshot



- LF Analytics provides organization diversity metrics in the primary view for commits, issues filed, and communication channels (current support for Slack and groups.io)



Data Collection Strategies

Qualitative

- Footprint of an organization in a project or ecosystem
- Influence of an organization in a project or ecosystem
- Affiliation diversity in governance structures.

Quantitative

- % of commits by each organization

- % of merges/reviews from each organization
- % of any kind of contributors from each organization
- % of lines of code contributed by each organization
- % issues filed by each organization
- Contributing Organizations - What is the number of contributing organizations?
- New Contributing Organizations - What is the number of new contributing organizations?
- New Contributor Organizations - New organizations contributing to the project over time.
- Number of Contributing Organizations - Number of organizations participating in the project over time.
- Elephant Factor - If 50% of community members are employed by the same company, it is the elephant in the room. Formally: The minimum number of companies whose employees perform 50% of the commits
- Affiliation Diversity - Ratio of contributors from a single company over all contributors. Also described as: Maintainers from different companies. Diversity of contributor affiliation.
- In projects with the concept of code ownership, % of code owners affiliated with each organization weighed by the importance/size/LoC of the code they own and the number of co-owners.

References

- Potential implementations and references:
  - https://bitergia.gitlab.io/panel-collections/open_source_program_office/organizational-diversity.html
  - Kata Containers dashboard entry page (bottom of this)
  - Augur

Focus Area - Place

Goal: Identify where contributions occur in terms of physical and virtual places (e.g., GitHub, Chat Channel, Forum, conferences)

Metric	Question
Collaboration Platform Activity	What is the count of activities across digital collaboration platforms (e.g., GitHub, GitLab Slack, email) used by a project?
Event Locations	Where are open source project events located?

Collaboration Platform Activity

Question: What is the count of activities across digital collaboration platforms (e.g., GitHub, GitLab Slack, email) used by a project?

Description

Open source projects use many different digital communication and collaboration platforms. These platforms may include email, social media, chat applications, and code management technologies (e.g., GitHub, GitLab). This metric is a measure of where and how much message activity is occurring across collaboration platforms.

Objectives

Understand where the community is collaborating. To develop insights related to the processes followed by each project, which are more likely to be accurate if our review of the communication logs is as complete as possible. Demonstrate how open and transparent a project is. Help people find the appropriate place to make contributions and connect with the project. Help project maintainers determine the optimal number of channels to effectively and efficiently share information while allowing contributors to connect in the way that works best for them Find the lowest barrier channels for engagement Inform other metrics like: Burstiness, Project Velocity, Social Listening, Activity Dates and Times, Chat Platform Inclusivity

Implementation

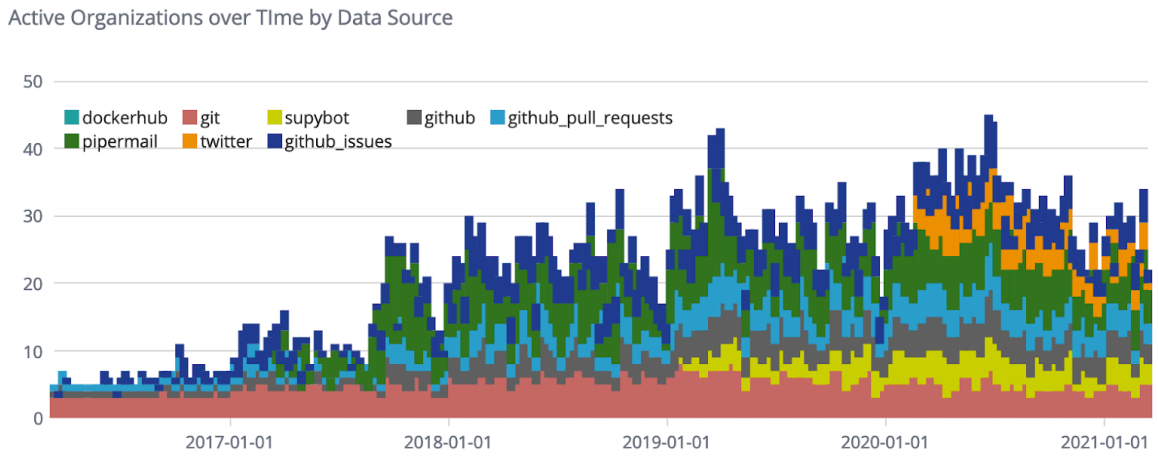
The unit of data collection is the individual activity on a platform. Metadata related to the metric can include:

- Timestamp
- Sender
- Threaded/Non-Threaded Platform
- Data Collection Date
- Platform message identifier

Filters

- Number of people
- Number of messages
- Number of comments on Issues and Change Requests
- Type of channel (mailing list, irc, and so on)
- Activity per day of the week
- Contribution attributions (e.g., people or organizations)

Visualizations



https://chaoss.biterg.io/app/kibana#/dashboard/ab68fe20-17f2-11e9-872f-e17019e68d6d

Tools Providing the Metric

- Orbit.love is a community management platform that captures this for a few channels: https://docs.orbit.love/docs/adding-activities
- GrimoireLab
- Augur

References

- Related metric: https://chaoss.community/metric-chat-platform-inclusivity/
- Related metric: https://chaoss.community/metric-issues-new/

Contributors

- Elizabeth Barron
- Sean Goggins
- Matt Germonprez
- Danial Izquierdo
- Dawn Foster
- Beth Hancock
- Kevin Lumbard
- Vinod Ahuja

Event Locations

Question: Where are open source project events located?

Description

This metric helps understand where events for a project are located to better understand the project's commitment to engage a global audience. Project events may include conferences, hackathons, meetups, and other synchronous get togethers. Project events may be embedded or co-located with other open source events.

Objectives

To give insight as to where project events and meet-ups are happening globally.

Implementation

Filters

- Virtual versus in-person or hybrid
- Attention to time zones
- Collocated with another conference
- Geographic Region
- Type of event (local meetup vs. global conference)

Tools Providing the Metric

- Shared or publicly available calendars on project websites
- Event lists on Foundation or project websites

Data Collection Strategies (Optional)

- Ask event organizers
- Look on project or foundation websites. For example: https://events.linuxfoundation.org/
- Aggregated Open Source event sites
- IEEE SA Open's Community Calendar: https://saopen.ieee.org/events/

References

Contributors

- Elizabeth Barron
- Dawn Foster
- Vinod Ahuja
- Kevin Lumbard
- Matt Germonprez

Diversity, Equity and Inclusion WG

Focus Area	Goal
Event Diversity	//github.com/chaoss/wg-diversity-inclusion/issues/375). Following a comment period, this metric will be included in the next regular release
Governance	Identify how diverse and inclusive our governance is.
Leadership	Identify how healthy our community leadership is.
Project and Communities	Identify how diverse and inclusive our project places, where community engagement occurs, are.

Focus Area - Event Diversity

Goal: //github.com/chaoss/wg-diversity-inclusion/issues/375). Following a comment period, this metric will be included in the next regular release

Metric	Question
Code of Conduct at Event	How does the Code of Conduct for events support diversity and inclusion?
Diversity Access Tickets	How are Diversity Access Tickets used to support diversity and inclusion for an event?
Event Diversity - Family Friendliness	How does enabling families to attend together support diversity and inclusion of the event?
Event Demographics	To what extent does an event consider and pay attention to attendee, speaker, and volunteer demographics?
Inclusive Experience at Event	To what extent does an event organizing team commit to an inclusive experience at an event?
Time Inclusion for Virtual Events	How can organizers of virtual events be mindful of attendees and speakers in other time zones?

Code of Conduct at Event

Question: How does the Code of Conduct for events support diversity and inclusion?

Description

A code of conduct describes rules of good behavior between event participants and what avenues are available when someone violates those expected good behaviors. An event with a code of conduct sends the signal that the organizers are willing to respond to incidences, which helps people from all backgrounds feel included and comfortable attending the event. Event participants are empowered to report incidences and proactively help mitigate situations of unwanted behavior.

Objectives

- An event organizer wants to make sure they have effective processes in place to deal with misbehaving attendees.
- An event organizer wants to make sure that participants have a positive experience at the event.
- Event participants want to know how to report offensive behavior.
- Event participants want to know that they will be safe at an event.

Implementation

Data Collection Strategies

- Interview and/or survey participants to understand more about why the event code of conduct did or did not meet their expectations.
    - What can this event do to improve the code of conduct at this event?
    - What are some examples of how this event met or exceeded your code of conduct expectations?
    - Are participants required to accept the code of conduct before completing registration?
- Observe event website for a code of conduct.
- Observe whether a code of conduct is posted at an event.
- Observe that code of conduct has a clear avenue for reporting violations at the event.
- Observe that code of conduct/event website provides information about possible ways to provide support victims of inappropriate behaviour, eventually links to external bodies?
- Browse the event website. If code of conduct is posted and there is a clear avenue for reporting violations at the event, this criteria is fulfilled. (Note: ideally, the code of conduct would be discoverable)
- As an attendee or event staff, observe whether participants will have an easy time finding a code of conduct posted at the event. Having a code of conduct prominently posted at a registration site may be useful.
- Survey participants about the code of conduct:
    - Likert scale [1-x] item: How well did the event meet your code of conduct expectations.
    - On registration, and during the event were you made aware of the code of conduct and how to report violations? [i]

    - Did the existence of the code of conduct make you feel safer, and more empowered to fully participate at this event? [i]
    - If you reported a violation of the code of conduct, was it resolved to your satisfaction? [i]

References

- Attendee Procedure For Incident Handling
- 2018 Pycon Code Of Conduct
- Conference anti-harassment
- Women In Tech Manifesto

[i] Some sample questions re-used from the Mozilla project.

Diversity Access Tickets

Question: How are Diversity Access Tickets used to support diversity and inclusion for an event?

Description

Inviting diverse groups of people may be made explicit by offering specific tickets for them. These tickets may be discounted to add further incentive for members of the invited groups to attend. A popular example are reduced tickets for students.

Diversity access tickets can enable community members to attend events and encourage new members to join. Especially unemployed, underemployed, or otherwise economically disadvantaged people may otherwise be excluded from attending an event. Furthermore, diversity access tickets can encourage additional contributions and ongoing contributions.

Objectives

- Demonstrate effectiveness of increasing diversity at events.
- Enable attendees to attend who could not without a discount due to their economic situation.
- Encourage members from underrepresented groups to attend.
- Track effectiveness of outreach efforts.

Implementation

Data Collection Strategies

- Observe website for availability and pricing of diversity access tickets.
    - How many (different) diversity access tickets are available?
    - What are the criteria for qualifying for a diversity access ticket?
    - What is the price difference between regular and diversity access tickets?
    - Are regular attendees encouraged to sponsor diversity access tickets?
    - Are sponsors of diversity access tickets named?
    - Are numbers from previous conferences displayed on website about how many diversity access tickets were used?
- Interview organizers:
    - How were the diversity access tickets allocated?
    - How many (different) diversity access tickets are available?
    - What are the criteria for qualifying for a diversity access ticket?
    - How many attendees used diversity access tickets?
    - Where did you advertise diversity access tickets?
    - If any, who sponsored diversity access tickets?
- Survey participants about perception of diversity access tickets.

    - Likert scale [1-x] item: The availability of discounted student [replace with whatever group was invited] tickets was effective in increasing participation of students [or other group].
    - True/False item: I was aware that [this conference] provided diversity access tickets.
    - True/False/Don't Know item: I qualified for a diversity access ticket.
    - True/False item: A diversity access ticket enabled me to attend [this conference].
- Track attendance numbers based on diversity access tickets.
    - Count use of different discount codes to track outreach effectiveness and which groups make use of the diversity access tickets. Requires conference registration system that tracks use of discount codes.
    - Count at each event, how many diversity access tickets were sold or given out and compare this to how many participants with those tickets sign-in at the event.

References

- https://diversitytickets.org/
- https://internetfreedomfestival.org/internet-freedom-festival-diversity-inclusion-fund/

Event Diversity - Family Friendliness

Question: How does enabling families to attend together support diversity and inclusion of the event?

Description

Family friendliness at events can lower the barrier of entry for some attendees by allowing them to bring their family. This could include childcare, activities for children, or tracks at the event targeted at youths and families.

Objectives

- An open source project wants to know whether an event is inclusive for attendees with families.
- An event organizer wants to know whether inviting people who are caregivers know about the availability of these family-oriented activities.
- A parent, guardian, or caregiver with children under the age of 18, want to know if they can bring their children.
- A parent, guardian, or caregiver, with children under the age of 18, with no option, but to bring their children, evaluate their ability to attend as a family.

Implementation

Data Collection Strategies

- Interview conference staff
    - Question: What services does the conference have for attendees who have children to take care of?
    - Question: Do you have a mother's room? If yes, describe.
    - Question: Do you offer child care during the event? If yes, describe.
    - Question, if childcare is offered, for what ages?
    - Question: Are there activities and care that support tweens/teens (youth) and not only young children.
    - Question: Do you have special sessions for children? If yes, describe.
- Survey conference participants
    - Likert scale [1-x] item: How family friendly is the event?
    - Likert scale [1-x] item: Anyone can bring their children to the event and know that they have things to do.
    - Likert scale [1-x] item: Children have a place at the conference to play without disturbing other attendees.
- Analyze conference website [check list]
    - Does the conference promote having a mother's room?
    - Does the conference promote activities for children and youth?
    - Does the conference promote family-oriented activities?

    - Does the conference explicitly invite attendees to bring their children?
    - Does the conference offer childcare, including youth space?

References

- Childcare at Conferences Toolkit by Adacare
- Improving Childcare at Conferences

Event Demographics

Question: To what extent does an event consider and pay attention to attendee, speaker, and volunteer demographics?

Description

Attendee and Speaker demographics help indicate the potential for different viewpoints and broader perspectives and representation at an event. Because everyone has a different perspective, investigating the responses for each group of demographics can indicate whether people from various demographics feel less included than the average.

Objectives

- Determine the [diversity] (https://github.com/drnikki/open-demographics) of attendees, speakers, and volunteers. This can include gender identity, age, first language, and dis/ability.
- Help retain attendees, speakers, and volunteers from diverse backgrounds for future events.
- Determine if the groups of speakers are diverse when grouped by keynotes, sessions, and tracks.
- Determine if new attendees, speakers, and volunteers are from diverse backgrounds.
- Determine if keynote, track, and session selection committees are diverse.
- Share attendee, speaker, and volunteer demographic information with others
- Create a statement of privacy management of demographic information
- Discover less-visible stories with regard to intersectionality of the demographic data

Implementation

Filters

- Demographics: Responses to subjective questions should be analyzed to gather diverse perspectives. Investigating the responses for each group of demographics can indicate whether some demographics feel less included than the average.
- Keynotes, sessions, and tracks.
- Diversity throughout the conference that is not restricted to a single track or series.
- Attendees
- Speakers
- Volunteers
- Conference Committee

Data Collection Strategies

- Request attendee and speaker demographics during conference registration
- Use a survey before, during, or after an event to gather attendee and speaker demographics. (For example, using the Open Demographics questions)

- For virtual events: If a virtual event platform supports real-time polling, use polls to better understand the audience and background of people who attend your event.

References

Thanks to Nikki Stevens and Marie Nordin for sharing experiences and resources that informed the creation of this metric.

- http://nikkistevens.com/open-demographics/
- https://github.com/chaoss/wg-diversity-inclusion/tree/master/demographic-data

Contributors

- Matt Germonprez
- Justin W. Flory
- Elizabeth Barron
- Matt Cantu
- Georg Link
- Nikki Stevens
- Kevin Lumbard
- Lauren Phipps
- Yehui Wang

Inclusive Experience at Event

Question: To what extent does an event organizing team commit to an inclusive experience at an event?

Description

If a tree falls in the forest and no one hears it, did the tree really fall?

Similarly, we need to judge the inclusiveness of an event not only by the effort of the organizers but also by the response of event attendees.

Inclusiveness of an event is most impactful when attendees feel included and welcome. All efforts that organizers invest into making an event inclusive are in vain if attendees do not experience the event as inclusive and welcoming. This metric focuses on the work by event organizers and the outcomes from that work on the inclusive experiences of event attendees.

Objectives

This metric is designed to:

- Enable event organizers to ensure all attendees are welcomed and included at events
- Encourage event organizers to build inclusivity into the event to provide a sense of belonging
- Provide methods to measure inclusion in experience at an event

Implementation

Provide attendees with a venue to express concerns regarding inclusivity at the event

- Ensure that anyone who voices a concern is heard
- Ensure violations are acted upon
- Survey attendees about perceptions of inclusivity

Data Collection Strategies

Ask attendees and speakers:

Survey with Likert Scale [1-5] (Or Emoji Scale):

- I feel included at the event
- I feel uniquely valued by the event staff
- I had chances for my voice to be heard in sessions
- I plan to come back to this event

Interview attendees and speakers:

- Have you experienced anything during the event that made you feel unwelcome or unincluded?
- Did you observe something that could make someone else feel unwelcome and unincluded?
- Could you provide specific examples to help event organizers mitigate this in the future?
- What is something that organizers of this event could improve on in the future regarding inclusivity at the event?

References

- Linux Foundation Inclusivity Training
- Fedora Community Badge

Contributors

- Trisha Rajaram
- Matt Cantu

Time Inclusion for Virtual Events

Question: How can organizers of virtual events be mindful of attendees and speakers in other time zones?

Description

Global accessibility is vital due to the role it brings in choosing when an event should take place. Time Inclusion for Virtual Events ensures support for participation from a global community during an event. Through this metric, we are able to be more inclusive to attendees and speakers in all regions of the world.

Objectives

Global accessibility should be understood when creating a virtual event in order to allow attendees and speakers in all regions to benefit from the event. The goal of this metric is to help organizers of virtual events develop a reasonable method to ensure those in different time zones feel included.

Implementation

Examples of Time Inclusion which can be implemented include:

- Having the speaker do a pre-recording where the speaker is able to answer questions live in the chat
- Doing a presentation within a certain time block for certain time zones
- Preparing and sending a recording to be live within an hour
- Network bandwidth options by platform for both live participation and video streaming at a later date

Data Collection Strategies

Survey attendees with Likert Scale [1-5] (Or Emoji Scale):

- The event keeps time zone differences in mind
- The event caters to my needs related to differences in time
- The event provides adequate recordings of the talks I care about
- I was provided with adequate low-bandwidth options at the event

Survey speakers with Likert Scale [1-5] (Or Emoji Scale):

- The time block selected for my presentation at the event is fair to me
- I have, or will receive, a recording of my presentation for later viewing

References

Medium Article

Contributors

- Ruth Ikegah
- Trisha Rajaram
- Justin W. Flory
- Matt Germonprez

- Matt Cantu
- Lauren Phipps
- Yehui Wang

Focus Area - Governance

Goal: Identify how diverse and inclusive our governance is.

Metric	Question
Board/Council Diversity	What is the diversity within our governing board or council?
Code of Conduct for Project	How does the Code of Conduct for the project support diversity and inclusion?

Board/Council Diversity

Question: What is the diversity within our governing board or council?

Description

A governance board or council comprises a group of people who steer an open source project. Diversity in this important group is beneficial to signal that the project embraces diversity and that leadership roles are available to everyone. A diverse board or council also increases the chance that issues of members of a diverse background are heard and understood.

Objectives

A project member needs to know that someone on the board or council is representing their interests. A board or council wants to make sure that it is not living in an echo chamber. Having a diverse board or council helps others to see themselves reflected and advance in their careers. This metric helps the board to be aware of the current situation of the diversity of the board if compared to other areas such as technical leaders or the community itself.

Implementation

Data Collection Strategies

- Observe whether open elections are held to see if candidates are representative of the member base. (Note: elections typically favor candidates of majorities and do not ensure that minorities are represented after the election)
- Observe the diversity of board or council members from the project webpage (limitation: demographics are not always self-evident).
- Ask the board for a report of the diversity of the board or council.
- Survey project members about their perceived diversity of the board or council.
    - Likert scale [1-x] item: I feel represented in the board or council.
    - Likert scale [1-x] item: The board or council attentive to minorities within the project.
- Likert scale [1-x] item: The board or council represents the diversity of the project community.

References

- https://www.openstack.org/foundation/2018-openstack-foundation-annual-report
- https://www.linuxfoundation.org/about/diversity-inclusiveness/

Code of Conduct for Project

Question: How does the Code of Conduct for the project support diversity and inclusion?

Description

A code of conduct signals to project members what behavior is acceptable. A code of conduct also provides enforcement mechanisms to deal with people who misbehave.

Objectives

A code of conduct in a project provides the following:

- Knowing that a community takes diversity and inclusion seriously.
- Evaluating a community for diverse and inclusive design, before investing any time in that project.
- Identifying whether or not their demographic is protected prior to participating in a project.
- Ensuring that project partnerships, and allies take diversity and inclusion seriously as it can impact their own reputation and community health.
- Understanding how a violation is reported and handled.
- Observing that the code of conduct is being enforced; and not just the potential for enforcement.

Implementation

Tools Providing the Metric

- Mozilla Code of Conduct Assessment Tool

Data Collection Strategies

- Identify the location of the code of conduct as it pertains to primary areas of interaction and participation in projects and events (i.e., repo root, event entrance, communication channels).
- Survey project members about their perception of how a code of conduct influences their participation and sense of safety.
- Follow-up survey for reporting, around discoverability, clarity, and relevance.

Qualitative

- Code of Conduct passes Mozilla's Code of Conduct Assessment Tool
- Interview and/or survey community members to understand more about why the code of conduct does or does not meet their expectations.
    - What can the project do to improve the code of conduct?
    - What are some examples of how this community met or exceeded your code of conduct expectations?

Quantitative

- Browse the project website. If code of conduct is posted and there is a clear avenue for reporting violations, this criteria is fulfilled. (Note: ideally, the code of conduct would be discoverable)
- Survey participants about the code of conduct:

- Likert scale [1-x] item: How well did the project meet your code of conduct expectations?
- Likert scale [1-x] item: How clear are you on the rights and responsibilites of community members as described in the code of conduct?
- Were you made aware of the code of conduct and how to report violations? [i]
- Did the existence of the code of conduct make you feel safer, and more empowered to fully participate in this project? [i]
- If you reported a violation of the code of conduct, was it resolved to your satisfaction? [i]

References

- CHAOSS metric: Code of Conduct at Events

[i] Some sample questions re-used from the Mozilla project.

Focus Area - Leadership

Goal: Identify how healthy our community leadership is.

Metric	Question
Inclusive Leadership	How well is a project and community setup for diverse leadership?
Mentorship	How effective are our mentorship programs at supporting diversity and inclusion in our project?
Sponsorship	How effective are long-time members who sponsor people in supporting diversity and inclusion in a community?

Inclusive Leadership

Question: How well is a project and community setup for diverse leadership?

Description

Leadership is central to a project/community's culture and how the leadership is determined requires intentional design and accountability for the inclusion of others. Note the term leadership may vary depending on the project/community. Leadership roles have high visibility and the diversity of this group is critically important to fostering an inclusive community.

Objectives

- Enable communities to reflect on their own leadership practices.
- Signal to newcomers that everyone is welcome, has an opportunity to take on leadership roles, and can be successful within the project.

Implementation

Filters

Examples of open source leadership (e.g., influential community roles):

- Individuals serving as project maintainers (sometimes referred to as owners)
- Individuals with repository merge access
- Individuals who are organization members for the repository
- Individuals who have defined community roles (community representatives, community speakers)
- Individuals listed in project documents as contacts for issues with builds, documentation, or other community concerns
- Individuals serving as community event organizers
- Individuals serving as community mentors for available mentorship programs

Data Collection Strategies

Step 1: Understand

The first step is to understand the principles of inclusive leadership by asking some questions that can include:

Review & Renewal

- Are all leadership roles limited by time and require term renewal?
- Do the term renewals include opportunities for review and feedback from communities involved?
- Is there a graceful way to move people out of leadership roles and into emeritus or advisory roles?

Distribution of Leaders

- Are leaders limited in the number of leadership roles one person can hold?
- Are new leadership opportunities and pathways decided and consulted on by the community involved in that area?

- Are the criteria for the role requirements validated by the community involved?
- Does the project create clear definitions for roles?
- Does the project publicly document their definitions for roles?
- Is leadership responsibility held by groups and not individuals, where possible?

Accountability

- Do the project's leaders agree to a standard by which they can be held accountable?
- Do the staff & community know and understand how they can hold leaders accountable?
- Are leaders aware that they represent the organization/project in their actions?
- Are leaders following and continually evolving a shared framework for decision making?
- Does the leadership openly communicate periods of inactivity or unavailability with the project and community?

Diverse Participation

- Do the project leaders strive to include diverse voices and groups whenever possible?
- Does the project enforce the Code of Conduct consistently and strictly?
- Do the leadership pathway explicitly consider inclusion dimensions (e.g., time zone, language, bandwidth, and cultural norms)?

Roles & Consistency

- Are the leadership roles valued similarly across the project (e.g., recognition, access to resources, and opportunities)?
- Do all leaders have clarity in their roles and expectations?
- Do all leaders have a shared foundational knowledge base & skills (e.g., Community Participation Guidelines).
- Does project leadership follow a shared framework for decision making?
- Are project roles clearly defined?
- Are the related tools consistent and coherent, where possible?

Equal Value

- Does the project value and recognize different types of experts equally (e.g., technical and non-technical)?

Acknowledgement of Limitations

- Have project leaders identified technical, financial, or other limitations out of their control that may have an adverse impact on inclusivity?
- Has leadership clearly and explicitly acknowledged these limitations publicly?
- Are leaders open to discussing and implementing suggestions that reduce the impact of these limitations?

Step 2: Evaluate The second step is to evaluate your project/community according to existing goals for leadership.

Example Goal: Increase % in contribution typically done by project maintainers (pull request review, responding to, and trying to replicate issues).

Step 3: Take Action The third step is to act on what you've learned. What those steps are depends on your specific project community and giving recommendations is outside the scope of this document.

References

- Related blog post
- Create an Inclusive Leadership team page
- Run an 'Open Source Maintainer training program

Background

The principles and practices above were created and agreed upon by a cross-functional set of diverse group of Mozilla staff and volunteers who are most closely connected to the primary contribution areas SUMO, Mozilla Reps, L10N).

Read more here.

They have been applied to the following projects/products:

- Mozilla Open Source Support Program (MOSS)
- Mozilla Reps Program
- Firefox Developer Tools / Debugger Program
- 24 Pull Requests ('File an Inclusion Bug!')
- Add your project.

Mentorship

Question: How effective are our mentorship programs at supporting diversity and inclusion in our project?

Description

Mentorship programs are a vital component in growing and sustaining a community by inviting newcomers to join a community and helping existing members grow within a community, thereby ensuring the health of the overall community. Through mentorship programs, we can invite diverse contributors and create inclusive environments for contributors to grow and ideally give back to the community.

Objectives

To increase the number and diversity of new contributors as well as increase the level of contributions from each contributor. To increase the number of and diversity of contributors, encouraging them to grow into new roles with increasing responsibility within a community. To increase the number of advocates/evangelists for a project as well as the number of paid mentors and mentees. To cultivate a culture of inclusivity through identifying and supporting mentors and mentees.

Implementation

Data Collection Strategies

- Interview mentors and mentees
    - Ask community members about existing formal or informal mentorship programs
- Observe people informally mentoring new contributors
- Observe contributions that mentees make during and after the mentorship program
- Observe the trajectory of mentees within a project during and after the mentorship program
    - Mentee became a subject matter expert
    - Mentee is integrated into the community
    - Mentee is comfortable asking for help from anyone in the community
    - Mentee grew their professional network
    - Mentee has taken on responsibilities within the community
    - Mentee contributes to the community after the end of the mentorship project
    - Observe the retention rate of mentees vs the usual retention rate in the community
- Observe the nature of the mentorship program
    - Variety of projects participating in a mentorship program (e.g., in the OpenStack Gender Report a few of them were participating)
    - Variety of mentorship programs across a project ecosystem targeting different contribution levels (e.g., contribution ladder, onion layers)
    - Number of official mentors in the mentorship programs
        - Mentors experience (rounds or years mentored)
        - How many mentors repeat the position in the following years

- Survey mentors and mentees
    - Survey Likert item (1-x): I have found the mentoring experience personally rewarding.
    - Survey Likert item (1-x): I would recommend mentoring for this community.
    - Mentor survey feedback:
        - What training did you receive that helped your mentoring activity?
        - What community support was helpful for your mentoring activity?
        - What communication channels did you use for mentoring?
        - What are the first things you do with a new mentee?
- Collect demographic information from mentors and mentees
    - Number of mentees who finished a mentorship program (assumes a time/project bound mentorship program like GSoC, Outreachy, or CommunityBridge)
    - Number of mentees who started a mentorship
    - Number of mentors in the community
    - Number of diverse mentees
    - Geographic distribution of mentees

References

- GSoC Mentor Guide
- GSoC Student Guide
- Esther Schindler, 2009. Mentoring in Open Source Communities: What Works? What Doesn't?
- OpenStack Gender Report: Mentorship focused

Sponsorship

Question: How effective are long-time members who sponsor people in supporting diversity and inclusion in a community?

Description

Sponsoring a community member is different from mentoring (4). A mentor provides guidance, shares experience, and challenges a mentee to grow. A mentor can be from outside a community. In contrast, a sponsor is someone within the community who puts their reputation on the line to create opportunities for the sponsee to move up in the community.

Sponsoring is only recently adopted in industry. Sponsorship is an effective leadership tactic for increasing diversity. Sponsored people are 90 percent less likely to perceive bias in a community (1). A sponsee knows that their sponsor has their back and champions for them to get opportunities in the community. A sponsee is likely to go into a community or experience with a peace of mind that they know that someone wishes the best for them and wants to recognize them. This is effective to overcome protégés previous negative experiences possibly tied to their different or diverse background. A sponsor puts their reputation on the line to advance the protégés trajectory within the community.

Objectives

- Retain new members longer.
- Grow base of contributors, and convert newer (or less active) members into more active members, and ultimately, leaders in the community
- Foster stronger community bonds between members.
- Reduce perceived bias within the community.
- Give new members from different backgrounds a chance.
- Guide and train new members for new responsibilities and increased leadership.
- Demonstrate dedication toward increasing diversity and inclusion and promote sponsorship activity through blogs, speaking, and press interviews.
- Recognize potential subject matter experts with diverse backgrounds to sponsor.
- Convert sponsees into sponsors, continue the cycle.

Implementation

Data Collection Strategies

- Interview members:
    - For protégés: In what ways has the sponsorship program helped you become more successful?
    - In what ways has the sponsorship program improved diversity and inclusion within the project?
    - In what ways could the sponsorship program be improved?
    - Did the sponsorship program help you gain more responsibility and/or more leadership within the project?
    - Capture information about potential diverse proteges in events/meetups

        - Have you sponsored someone who identifies as a different gender than you?
        - Exchange gender with any dimension of interest from the Dimensions of Demographics.
- Survey members:
    - Survey members: “Do you consider yourself to be sponsoring other members?”
    - Survey protégés: “Do you have a sponsor helping you?”
    - Likert scale [1-x] item: I am sponsoring other members.
    - Likert scale [1-x] item: I am sponsoring members who are different from me.
    - Likert scale [1-x] item: I have a sponsor within the community who puts their reputation on the line to advocate for me.
    - Likert scale [1-x] item: How effective is the sponsorship program?

References

- https://fortune.com/2017/07/13/implicit-bias-perception-costs-companies/
- Sponsor Effect 2.0: Road Maps for Sponsors and Protégés
    - By Center for Talent Innovation (Pay Link)
    - http://www.talentinnovation.org/publication.cfm?publication=1330
    - The Key Role of Sponsorship:
    - https://inclusion.slac.stanford.edu/sites/inclusion.slac.stanford.edu/files/The_Key_Role_of_a_Sponsorship_for_Diverse_Talent.pdf
- Perceived unfairness is a major reason to leave a community
    - https://www.kaporcenter.org/tech-leavers/
- Sponsors Need to Stop Acting Like Mentors
    - By Julia Taylor Kennedy and Pooja Jain-Link
    - https://hbr.org/2019/02/sponsors-need-to-stop-acting-like-mentors

Focus Area - Project and Communities

Goal: Identify how diverse and inclusive our project places, where community engagement occurs, are.

Metric	Question
Chat Platform Inclusivity	How do you review Chat Platform inclusivity for your community?
Documentation Accessibility	How does the documentation accommodate different users?
Documentation Discoverability	How easily can users and contributors find the information they are looking for in a project's documentation?
Documentation Usability	What is the usability of documentation from content and structure perspectives?
Issue Label Inclusivity	How well are project issues labeled to invite new contributors, skilled contributors, non-code contributors, and other types of contributors?
Project Burnout	How is project burnout identified and managed within an open source project?
Psychological Safety	To what extent do community members feel safe within a community, including adding contributions, influencing change, bringing their authentic selves, and generally participating in the project?

Chat Platform Inclusivity

Question: How do you review Chat Platform inclusivity for your community?

Description

Open Source communities need places for contributors to interact and engage with each other. Much of this communication is likely to be project-related, but there may also be spaces for general social connection with other contributors. This metric describes synchronous communication in Chat Platforms, which includes platforms for public shared chat rooms, real-time conversations, and video conferences.

Chat Platforms are essential to a functioning open source community. Chat Platforms support a healthy project by enabling cross-team collaboration in different skill areas, helping peer collaborators find and interact with each other faster, increasing project transparency which allows team members to notice potential problems before they occur, and saving time by cutting across organizational hierarchies. Persistent Chat Platforms also become a reference for newcomers to familiarize themselves with project history, practice, and culture.

Chat Platforms should be chosen, moderated, and managed transparently, using inclusive language in order to accelerate innovative problem-solving and sustain contributor participation.

Objectives

- Support decision-makers when choosing a Chat Platform.
- Provide ideas that contributors and potential contributors may use to evaluate the specific context of any given project.
- Identify characteristics of the Chat Platforms, such as those identified under filters, that suggest greater inclusivity.

Implementation

Filters

How do these filters measure inclusivity? Think of it like an office space. How many teams (Chat Platforms) are there? How active are they? For example, very active chats can be difficult to follow and that difficulty will vary from person to person.

- General:
  - Number of active Chat rooms / groups
  - Number of messages per day, week, month
  - Number of active participants per day, week, month
  - Number of passive participants (i.e. reading but not participating)
  - Number of external participants (i.e. people who are not associated with a leading company working on an Open Source Project, such as Red Hat with the Fedora Project)
- Identities:
  - Number of (independent) volunteers
  - Number of corporate volunteers (people being paid by another entity to to volunteer work)
  - Number of workers (employees or contractors)

- Platforms without persistent connections (i.e., IRC):
  - Number of daily connections to room
  - Number of repeat logins

Visualizations

- Most active hours in community-specific Chat Platforms
- Number of active new chat members vs. older chat members
- Word clouds (for public Chat Platforms, e.g., public Mattermost rooms)
- “On-topic” vs. “off-topic” conversation
  - How often certain keywords or phrases show up, etc.

Tools Providing the Metric

GrimoireLab Perceval currently supports data collection from various Chat Platforms.

Data Collection Strategies

- **Trace data from Chat Platforms**:
  - See supported platforms for data collection in Perceval.
- **Open Source status of Chat Platform**:
  - See the CHAOSS blog post on open source platform status.
  - Does the chat platform offer Open Source client applications?
  - Does the chat platform offer Open Source server implementations?
  - Does the chat platform offer an Open API for integrating with data collection or bridging?
- **Features for low-bandwidth connections**:
  - Does it require a persistent Internet connection to use (e.g., IRC)?
  - Does the Chat Platform function with minimal bandwidth?
- **Features for community management and moderation**:
  - Does the Chat Platform offer robust moderation tools for moderators?
  - Does the Chat Platform have bot functionality or does it require fully manual intervention for moderation?
- **Features for internationalization**:
  - Is the Chat Platform available in multiple countries, or are there geographic limitations?
  - Does the Chat Platform provide tools for translation?
- **Features for access**:
  - Does the Chat Platform support tools for accessibility (e.g., friendly for screen readers)?
  - Is the Chat Platform easy to use for newcomers?
  - Is the Chat Platform available for a variety of devices?

- **Features for privacy**:
    - Does the Chat Platform require personal information to be verified or submitted for users to sign up?

References

1. Love or hate chat? 4 best practices for remote teams — *Opensource.com*
2. Open source status of chat platforms — *Chaoss.community*
3. **Moderation advice** (can inform choice of platform):
    (a) Moderation guidelines for inclusive community — *rhea.dev*
    (b) Participation & Moderation Guidelines — *drupaldiversity.com*

Documentation Accessibility

Question: How does the documentation accommodate different users?

Description

Documentation accessibility is critical due to the role documentation plays in open source projects. Users of documentation have different abilities, requiring the documentation to be offered in different formats to be equally empowering for these different users. The goal is to foster understanding for the widest audience of contributors to a project.

Objectives

These objectives help measure whether your documentation is accessible to a broad audience without disproportionately creating or perpetuating artificial debts on certain segments of its intended audiences.

- **Accessibility Screen Reader** — Documentation is accessible according to a standard for screen readers.
- **Learning Flexibility** — Documentation is accessible to people with various cognitive approaches, sensory differences, and neurodiversity (1).
- **Blind or Visually Impaired** — Documentation is accessible for people who primarily read text. Charts and images are examples of non-accessible types of documentation.

Implementation

Note: Be attentive to the target audiences for documentation. Due to the broad spectrum of projects and people who may contribute, documentation should address the different requirements of all its audiences.

Data Collection Strategies

- Use a **tool** that evaluates screen-reader friendliness (5) (6) to determine if the documentation is screen-reader friendly?
- **Interview** newcomers to figure out how documentation helped with, (a) understanding the contribution process, and/or, (b) helping to complete tasks in a project.
  Sample interview questions:
  - What is your experience with using the documentation to understand the contribution process?
  - What is your experience with consulting the documentation when you have a question about doing work in the project?
  - Were you comfortable with the technical terms present in the documentation?
- **Survey** project members
  - Matrix item: When you need to locate information about this project's processes, policies, or guidelines, which of the following describes your experience? (2)
    - Matrix rows can be:
      - Mailing list communication
      - Chat communication
      - Performing code reviews

- Process of getting code accepted
- Code of conduct
- Onboarding newcomers
- Licensing
- Trademark
- Add new Committers/Maintainers
- Project releases
- Voting process
- Installation procedures
- Feature development
- API integration
- Test suites
- Architecture overview
- User guide

■ Matrix columns can be:

- Always easy to find
  - Easy to find
  - Difficult to find
  - Very difficult to find

◦ Multiple choice: Did you face any challenges related to the accessibility of documentation when you started to participate in the project (e.g., language barriers, discoverability of documentation, structure of documentation)? (2)

■ Answer options:

- No challenges
- A Few challenges
- Several challenges
- Many challenges

■ Open-ended follow-up question if the answer is not “No challenges”: Describe an example of when you experienced the challenge, how the challenge affected you, and how, if at all, you overcame the challenge.

◦ Open-ended question: What suggestions do you have for improving the project’s policies, processes, or guidelines available to new contributors? (2)

- Look for organizing constructs regarding readability and scannability such as:
  - Headings
  - Text and Code Blocks

    - Bullets or Paragraphs
    - Anchors
- Evaluate searchability by considering:
    - How easily can this documentation be found by a user?
    - How easily can a user find what they require within the documentation?
    - Is the document easy to navigate with a keyboard?
- Provide a quick micro-survey with only one question to readers of the documentation (i.e., bottom-page or popup when leaving documentation page):
    - Yes/No question: Was this documentation page accessible to you?
    - Likert Scale [1-x]: How accessible was this documentation to you?
    - Short Answer: How do you feel about the accessibility of the documentation?
- **Walkthrough** with intended users of the documentation. Observe how they interact and use the documentation and where they get stuck. This can be a video conference session where the user of the documentation shares their screen.
- Ask users of documentation to write a **friction log** and describe what issues they had with documentation. This gives concrete use cases for documentation editors to understand how to improve the documentation for the specific user.
- Consider if **different versions of documentation** are available for different audiences? For example, a light-weight version and a very detailed version.

Resources

1. Breaking Down Barriers to Kubernetes Contribution for Neurodivergent Individuals
2. Apache Software Foundation Community Survey 2020
3. GNOME Accessibility Team
4. W3C Web Content Accessibility Guidelines
5. W3C Web Accessibility Evaluation Tools List
6. Some quick tests to evaluate web accessibility
7. Knowability - a group that specializes in Accessibility
8. Thoughts on Accessibility metrics
9. Paypal's list of Guidelines for Accessibility
10. Universal Design

Documentation Discoverability

Question: How easily can users and contributors find the information they are looking for in a project's documentation?

Description

Documentation discoverability is critical due to the role it plays in open source project's inclusivity. If people cannot find and then easily navigate through the body of documentation, or find the answers they seek, they are less likely to use or contribute to the project thus decreasing the project ability to attract diverse participants. Documentation discoverability has the goal to ensure searchability and visibility as well as improve inclusivity across a diverse audience.

Objectives

Visibility — Documentation is easy to find and reference. **Searchability** — Documentation can easily be navigated by users based on their ability, and they can find the relevant information they need through search. **Method of Discovery** — Identification of the ways in which people find the documentation **Inclusivity** - Knowledge is shared more equitably among project contributors, and new contributors can more easily participate..

Implementation

Data Collection Strategies

Survey project members: Likert scale [easy to find - impossible to find] regarding the discoverability of documentation such as:

- Mailing list archived communication
- Mailing list membership management
- Chat channel archived communication
- Performing code reviews
- Process of getting code accepted
- Code of conduct
- Onboarding newcomers
- Licensing, trademark
- Project leadership
- Project releases
- Voting process

Multiple-choice checkbox: How do you discover project documentation?

- Website
- Repositories
- Documentation within software
- Search engine
- Other

Multiple choice: Did you experience any challenges related to the discoverability of documentation when you started to participate in the project (e.g., language barriers, or structure of documentation)? Answer options:

- No challenges
- A few challenges
- Several challenges
- Many challenges

Open-ended follow-up question if the answer is anything other than “No challenges”:

- Describe an example of what challenge you experienced and when, how the challenge affected you, and how, if at all, you overcame the challenge.
- Open-ended follow-up question:
- What suggestions do you have for improving the project’s documentation regarding policies, processes, or guidelines available to new contributors?

Walkthrough with intended users of the documentation.

Observe how they interact and use the documentation and where they get stuck. For example, this can be a video conference session where the user of the documentation shares their screen.

Ask users of documentation to write a friction log to describe what issues they had with documentation. This gives concrete use cases for documentation editors to understand how to improve the documentation for the specific user.

Resources

- Curse of Knowledge
- Knowledge Base 101
- 5 Tips for Making Documentation a Priority in Open Source Projects
- Building navigation for your documentation site: 5 best practices in design

Documentation Usability

Question: What is the usability of documentation from content and structure perspectives?

Description

Documentation usability addresses the critical role of an open source project's documentation by making sure that various users can understand it. Documentation usability contains issues of structure, content, jargon, and readability with the goal to foster understanding for the widest audience of contributors to the project.

Objectives

- **Technical Jargon** — Documentation uses an appropriate level of technical jargon and provides an explanation for terminology as necessary to ensure the documentation is understandable for an entry-level contributor to the given project.
- **Structural Clarity** — Documentation is easy to follow and understand.
- **Readability** — Documentation uses language that is clear and concise, using common meaning words and short sentences, to ensure the documentation is understandable to people for whom the language is not native or those who may not follow similar shorthand conventions or inference patterns.
- **Language Inclusion** — Documentation avoids non-inclusive language (for example, 'brogrammer' language or exclusionary/derogatory language).
- **Language Diversity** — Documentation is available in a common vernacular for the intended audience and different languages.
- **Time/attention Diversity** — Documentation is inclusive of people that have differences in the time they can spend in documentation either reading or setting up/running commands. It is worthwile to consider people with childcare or other caring roles that may pull their attention from their screens at any given point.

Implementation

Data Collection Strategies

- Interview newcomers to determine how documentation helped the contributor to, (a) understand the contribution process, and/or, (b) complete the task. Sample interview questions:
    - Describe your experience with using the documentation to understand the contribution process.
    - Describe your experience with using the documentation when you have a question about doing work in the community.
    - Describe your experience with using the documentation to understand how to help outreach efforts.
    - How comfortable were you with the amount of technical terms present here? (adapt to survey using Likert scale [1-5])
    - Were there any terms or language you didn't understand?
    - What suggestions do you have for improving the project's policies, processes, or guidelines available to new contributors?

    - After interviewing, the community can track responses to each prompt as `Positive experience`, `Negative experience`, or `Neutral experience` and report these month-over-month to see improvement over time.

- Ask questions regarding readability and scannability such as: Does the documentation use organizing constructs, such as:
    - Headings
    - Text and Code Blocks
    - Bullets versus Paragraphs
    - Anchors
- Walkthrough with intended users of the documentation. Observe how they interact and use the documentation and where they get stuck. This can be a video conference session where the user of the documentation shares their screen.
- Ask users of documentation to write a friction log and describe what issues they had with documentation. This gives concrete use cases for documentation editors to understand how to improve the documentation for the specific user.
- Consider if different versions of documentation are available for different audiences. For example, a light-weight version and a very detailed version of the documentation.

Resources

- W3C Web Content Accessibility Guidelines
- W3C Web Accessibility Evaluation Tools List
- Some quick tests to evaluate web accessibility
- A group that specializes in Accessibility
- Thoughts on Accessibility metrics
- GNOME on Accessibility
- Paypal's list of Guidelines for Accessibility
- The Core Infrastructure Initiative: Best Practices Badge
- Breaking Down Barriers to Kubernetes Contribution for Neurodivergent Individuals
- `documentation_basics`
- `documentation_interface`
- Friction Log
- Stanford: Screen Reader Testing

Issue Label Inclusivity

Question: How well are project issues labeled to invite new contributors, skilled contributors, non-code contributors, and other types of contributors?

Description

Issue label inclusivity helps gauge the affordability of issues to different contributors at different levels of familiarity (e.g., newcomer, occasional contributor, and core contributors), with different skills (e.g., languages, frameworks, APIs, documentation, frontend, and backend), and with different goals (e.g., code and non-code contributions).

Objectives

Newcomer Friendly: Issues include appropriately labeled issues (e.g, “good first issue”, “newcomer-friendly”) for a new contributor to start with

Mentor Availability: Issues identify a mentor who can provide guidance and help with the review process of a particular issue (e.g., “Mentor Available” tag)

Issue List Diversity: Issues provide a diverse list of labels pertaining to different types of contributions (code and/or non-code) (e.g., “Documentation” tag)

Usable Title and Description: Issue titles and descriptions follow the Documentation Usability metric objectives

Consistent Use of Labels: Issues use a curated list of labels with distinct colors in a consistent way. For example, families of tags with a distinct color for each:

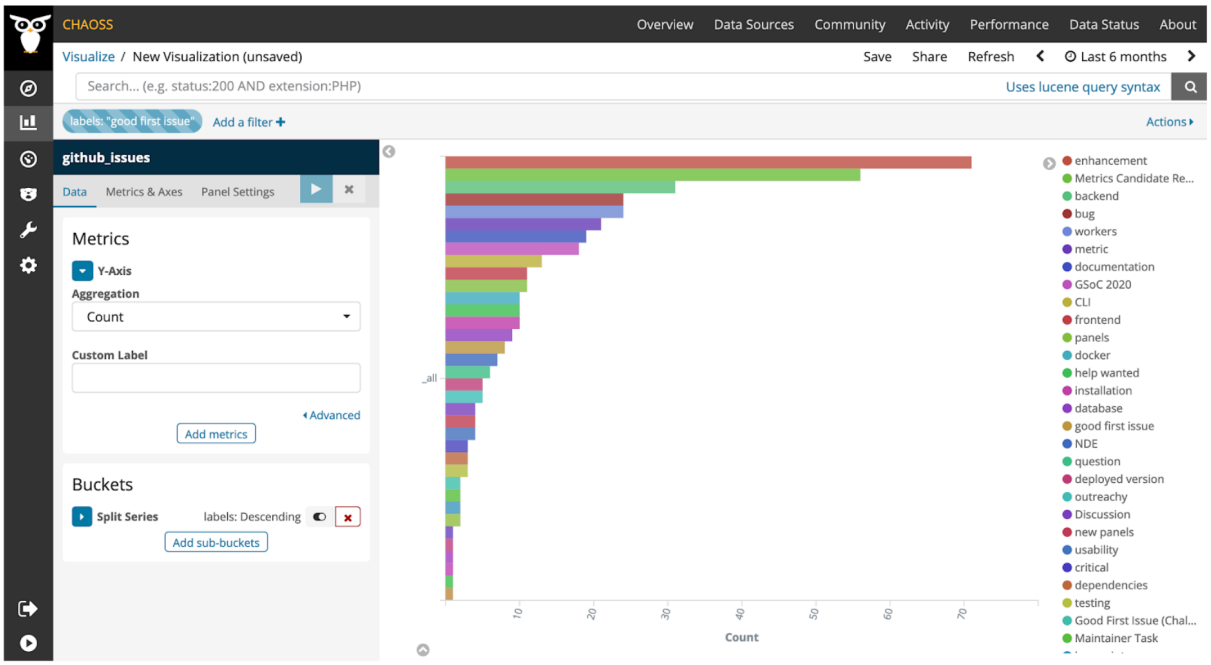
- Issue Type: “Feature” vs “Bug” vs “Documentation”...
- Issue Skills: the skills needed to resolve the issue (e.g, js, html, css)
- Issue Level of Familiarity: mentions the lowest level of familiarity needed (“good for newcomer” or “occasional contributor”or “core contributor”...)

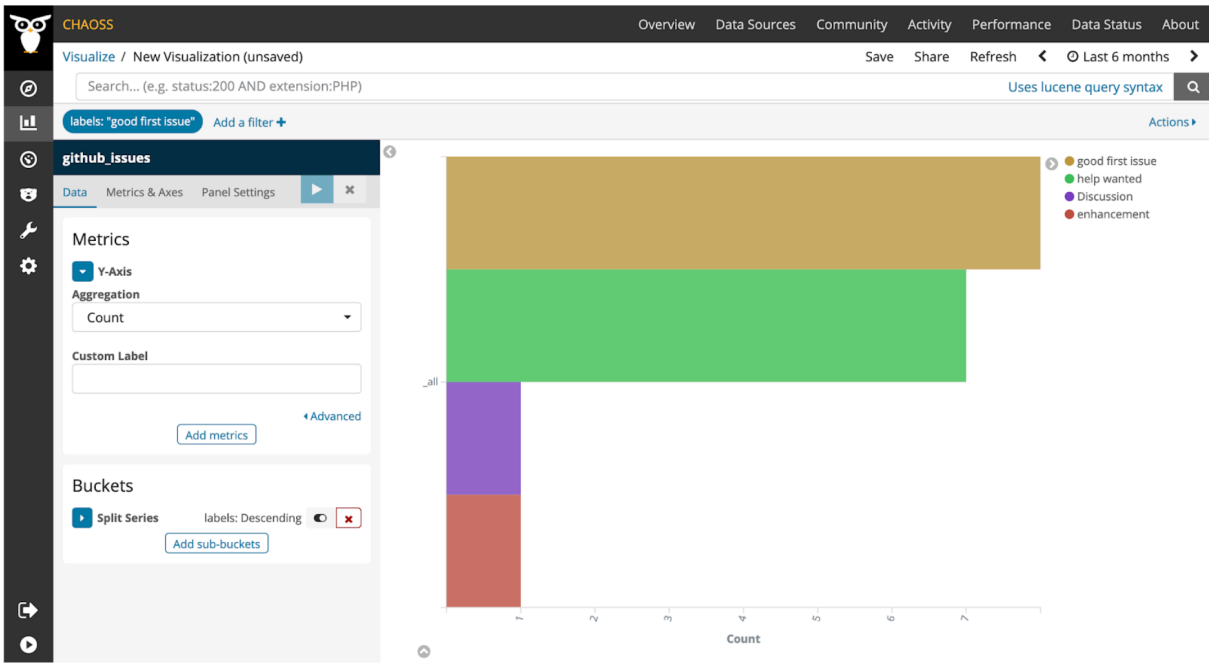
Implementation

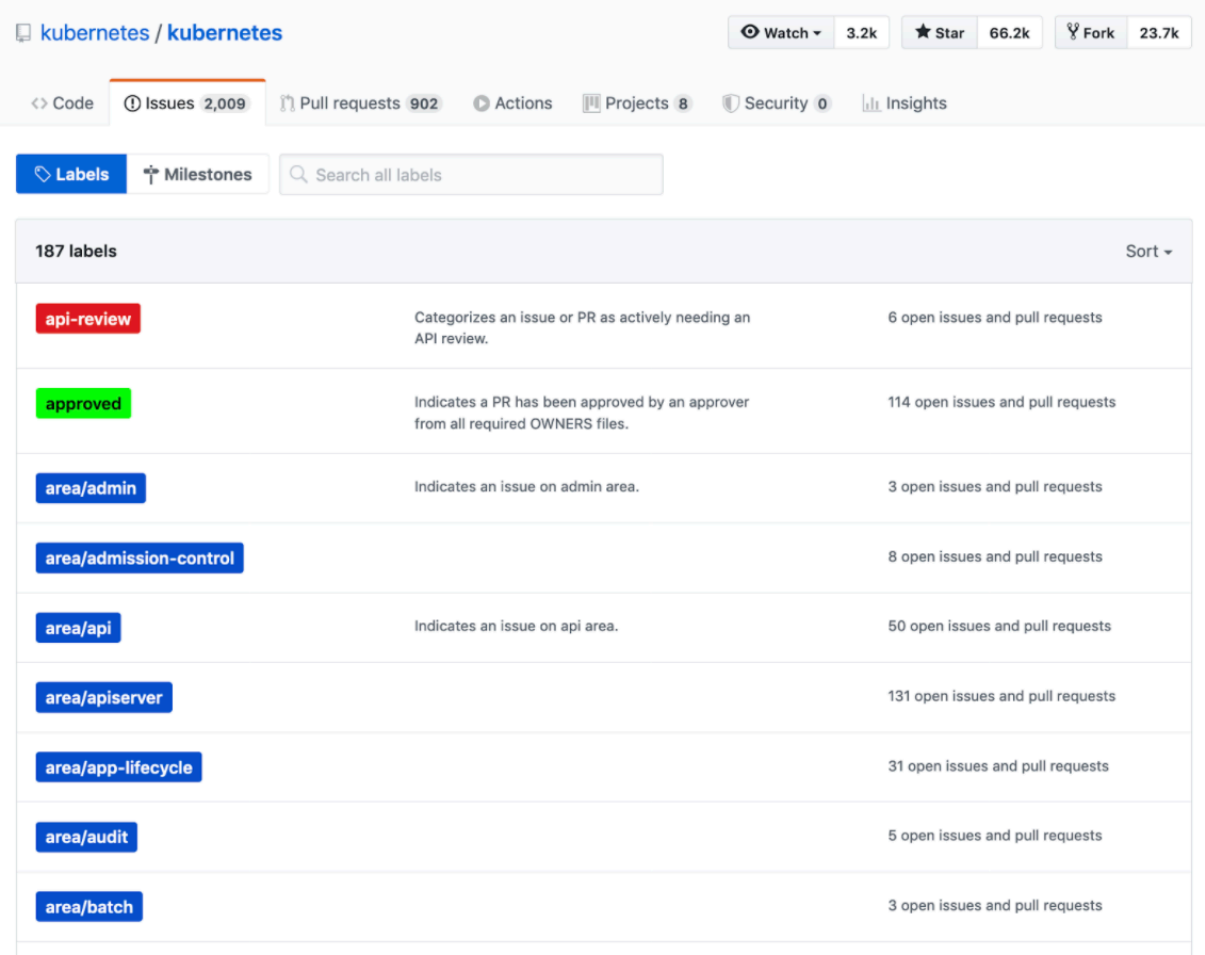
Filters

- Type of label
- Type of repository
- Age of open issue
- Number of open issues
- Date an issue was opened
- Code-related issues vs. documentation-related issues

Visualization


CHAOSS
Overview
Data Sources
Community
Activity
Performance
Data Status
About
Visualize / New Visualization (unsaved)
Save
Share
Refresh
Last 6 months
Search... (e.g. status:200 AND extension:PHP)
Uses lucene query syntax
labels: "good first issue"
Add a filter
Actions
github_issues
Data
Metrics & Axes
Panel Settings
Metrics
Y-Axis
Aggregation
Count
Custom Label
Advanced
Add metrics
Buckets
Split Series
labels: Descending
Add sub-buckets
_all
10
20
30
40
50
60
70
Count
enhancement
Metrics Candidate Re...
backend
bug
workers
metric
documentation
GSoC 2020
CLI
frontend
panels
docker
help wanted
installation
database
good first issue
NDE
question
deployed version
outreachy
Discussion
new panels
usability
critical
dependencies
testing
Good First Issue (Chal...
Maintainer Task


CHAOSS
Overview
Data Sources
Community
Activity
Performance
Data Status
About
Visualize / New Visualization (unsaved)
Save
Share
Refresh
Last 6 months
Search... (e.g. status:200 AND extension:PHP)
Uses lucene query syntax
labels: "good first issue"
Add a filter
Actions
github_issues
Data
Metrics & Axes
Panel Settings
Metrics
Y-Axis
Aggregation
Count
Custom Label
Advanced
Add metrics
Buckets
Split Series
labels: Descending
Add sub-buckets
_all
1
2
3
4
5
6
7
Count
good first issue
help wanted
Discussion
enhancement



From: https://github.com/kubernetes/kubernetes/labels

Tools Providing the Metric:

- Grimoire Lab
- Augur

Data Collection Strategies

- Identify the published list of issue labels used for each project
    - *General labels*:
        - Presence of labels to identify general needs of “Feature”, “Bug”, and “Documentation” label”, a “Front End”, and a “Back End” labels in the project’s list of labels and in the issue list (labels in use)
    - *Inclusive labels*:
        - Newcomer friendly ones look for (*newcomer*, *first*) in the project’s list of labels and in the issue list (labels in use)
        - Mentor friendly ones look for (*mentor*) in the project’s list of labels and in the issue list (labels in use)
    - *Skill labels*:

    - Presence of labels to identify skills needed (e.g, Java, Python, HTML, machine learning) in the project's list of labels and in the issue list (labels in use)

- Observe the frequency of each label used across issues in a project

References

- GitHub Satellite

Project Burnout

Question: How is project burnout identified and managed within an open source project?

Description

Burnout is a condition of mental, emotional, and physical exhaustion resulting from chronic stress. The three dimensions characterising burnout are feelings of energy depletion or exhaustion, increased cynicism or psychological distance from one's work and reduced professional efficacy (WHO, 2020). Project Burnout can occur:

- When project members become overwhelmed with the amount of work expected with project participation
- As a result of any community and contributor activities, including technical work, community management, and organizational work.

Objectives

Project Burnout should be understood within projects and managed effectively to improve the well-being of all. This metric is intended to:

- Develop strategies with communities to help project-related burnout
- Identify activities as leading indicators to identify project burnout
- Help people manage burnout by identifying signs before it happens
- Help people make decisions to maintain a healthy life in relation to project activities

Implementation

Data Collection Strategies

- The following is a way to better assess the well-being of open source project contributors and maintainers through a useful set of questions that can be asked regarding the well-being of community members
    - https://cdn.ymaws.com/www.palibraries.org/resource/collection/9E7F69CE-5257-4353-B71B-905854B5FA6B/Self-CareBurnoutSelf-Test.pdf
    - Following the questions, aggregate the results for the community and use individual scores, if they are shared, too.
- Surveys: Ask about the well-being of individuals in the project
    - Survey Likert item (1-x): I feel energized working on this open source project
    - Survey Likert item (1-x): I feel emotionally drained from my work on this project
    - Survey Likert item (1-x): I have felt tired when working on this open source project
    - Survey Likert item (1-x): I take time for self-care, self-initiated practices that enhance health and positive well-being, [Bickley, 1998] when working on this open source project.
    - Survey Likert item (1-x): In the past six months, I have thought about leaving this project.
    - Survey Likert item (1-x): I have thought about or have taken a break from the project because of project-related stress.

    - Survey Likert item (1-x): I have thought about or have taken a break from the project to balance with other parts of my life.
    - Survey Likert item (1-x): I never seem to have enough time to get everything done on this project.
    - Survey Likert item (1-x): I have to neglect some tasks related to this project because I have too much to do.
    - Survey Likert item (1-x): I feel that my contributions in the project are valued and rewarding to me.
    - Survey Likert item (1-x): I feel that my voice is heard in the project.
    - Survey Likert item (1-x): We can openly talk in the project about how we are doing and feeling and check-in on each other.
- Trace Data: Explore online data to get a better understanding of the well-being of individuals in the project
    - Analyze activity metrics around the number of contributions over time per individual to see if there is an abrupt drop off after an extended contribution time.
    - Analyze if there are continuous contributions over a long period that abruptly end
    - Analyze if there are a large number of contributions by a very small group of people (see Bus Factor or Elephant Factor metrics)
- Interviews: Talk with open source project contributors and maintainers with their own interpretation of terms
- Contributor Questions:
    - How do you feel about working on this project?
    - Has a poor state of well-being affected your engagement with this open source project? How?
- Maintainer Questions:
    - How should we be monitoring the well-being of individuals?
    - How do you measure the well-being of your community members in your open source project?
    - How do you determine the well-being of contributors to your project?

References

1. What you need to know about burnout in open source communities
2. What does an open source maintainer do after burnout?
3. Raman, N., Cao, M., Tsvetkov, Y., Kästner, C., & Vasilescu, B. (2020). Stress and Burnout in Open Source: Toward Finding, Understanding, and Mitigating Unhealthy Interactions. In International Conference on Software Engineering, New Ideas and Emerging Results (ICSE-NIER).
4. Emotional Resilience In Leadership Report 2020 (*"but not for OSS"*)
5. Help people identify what "stage" of burnout they are in, or if they are on a path to a more severe burnout stage: Practical guide for avoiding Burnout and living a happier life
6. Burned out
7. How I learned about burnout the hard way

Psychological Safety

Question: To what extent do community members feel safe within a community, including adding contributions, influencing change, bringing their authentic selves, and generally participating in the project?

Description

A key component that is integral to the goal of community inclusion, is psychological safety. Psychological safety can be defined in several different stages, according to the Center for Creative Leadership:

- Stage 1 – Inclusion Safety: Inclusion safety satisfies the basic human need to connect and belong. In this stage, you feel safe to be yourself and are accepted for who you are, including your unique attributes and defining characteristics.
- Stage 2 – Learner Safety: Learner safety satisfies the need to learn and grow. In this stage, you feel safe to exchange in the learning process, by asking questions, giving and receiving feedback, experimenting, and making mistakes.
- Stage 3 – Contributor Safety: Contributor safety satisfies the need to make a difference. You feel safe to use your skills and abilities to make a meaningful contribution.
- Stage 4 – Challenger Safety: Challenger safety satisfies the need to make things better. You feel safe to speak up and challenge the status quo when you think there's an opportunity to change or improve.

In communities with a strong level of psychological safety, contributors are more likely to be empowered to propose changes and take on leadership and/or decision-making roles. Psychological safety in communities impacts diversity, equity, and inclusion, and is signalled by indicators such as:

- Diversity: Contributors from under-represented groups are more likely to be active participants and encourage others to be so as well
- Equity: Distribution of decision making and leadership roles is evenly distributed to include traditionally marginalized groups
- Inclusion: Those from marginalized or under-represented backgrounds feel welcome and as if they can bring their authentic selves to the project in a safe environment
- Attraction: Creating a safe environment increases the likelihood of attracting new contributors
- Retention: Contributors are more likely to remain connected to the project if they feel safe in doing so

There will be a positive effect on contributor diversity, sense of inclusion and belonging, and ultimately contributor retention if:

- community members feel safe participating in the community through code contributions, comments, discussions, and other forms of participation
- there is a level of trust that policies have been put in place to mitigate harassment and intolerance
- there is a mechanism in place to effectively address problems if they occur
- contributors have no serious privacy concerns related to the project
- there is a high degree of transparency between project leadership and the community
- any potential vectors for abuse in the code itself have been prioritized as problematic by project leadership

Objectives

This metric is designed to:

- Enable project maintainers to understand the level of psychological safety around project policies, such as code of conduct, and code of conduct enforcement
- Allow community members to voice concerns around psychological safety
- Identify potential areas for improving psychological safety within the community

Implementation

Data Collection Strategies

The following is an example of one way to assess the level of psychological safety of open source project contributors and maintainers through a useful set of questions that can be asked regarding the well-being of community members.

Candidate questions include:

1. Have you ever observed any of the following in the context of an open source project? Answer options: Yes/No. If yes [select all that apply]:

- (1) Lack of response to contributions or questions
- (2) Rejection of contributions without explanation
- (3) Dismissive responses to contributions or questions
- (4) Documentation that is incomplete or difficult to understand
- (5) Conflict or interpersonal tension between contributors
- (6) Language or other content that made you feel unwelcome (e.g. profanity, racist jokes, sexual imagery, etc.)

1. Have you ever **witnessed** any of the following behaviors directed at another person in the context of an open source project? (not including something directed at you) Answer options: Yes/No. If yes [select all that apply]:

- (1) Hostility or rudeness
- (2) Name calling
- (3) Threats of violence
- (4) Impersonation
- (5) Harassment over a sustained period
- (6) Harassment across multiple platforms
- (7) Stalking
- (8) Unsolicited sexual advances or comments
- (9) Stereotyping based on perceived demographic characteristics
- (10) Malicious publication of personal information (doxxing)
- (11) Other (please describe)

1. Have you ever **experienced** any of the following behaviors directed at you in the context of an open source project? Answer options: Yes/No. If yes [select all that apply]:

- (1) Hostility or rudeness
- (2) Name calling
- (3) Threats of violence
- (4) Impersonation
- (5) Harassment over a sustained period
- (6) Harassment across multiple platforms
- (7) Stalking
- (8) Unsolicited sexual advances or comments
- (9) Stereotyping based on perceived demographic characteristics
- (10) Malicious publication of personal information (doxxing)
- (11) Other (please describe)

3a. If yes to the above, when thinking of the last time you **experienced** harassment, how did you respond? Choose all that apply.

- (1) Asked the user(s) to stop the harassing behavior
- (2) Solicited support from other community members
- (3) Blocked the user(s) harassing me
- (4) Reported the incident to project maintainers
- (5) Reported the incident to the hosting service or ISP
- (6) Consulted legal counsel/ an attorney
- (7) Contacted law enforcement
- (8) Other (please describe)
- (9) I did not react / ignored the incident

3b. On a scale of 1-5, how effective were your responses? (Use Likert scale with the following options):

- 1: Not at all effective
- 2: A little effective
- 3: Somewhat effective
- 4: Mostly effective
- 5: Completely effective

1. As a result of experiencing or witnessing harassment, which, if any, of the following have you done?

- (1) Stopped contributing to a project
- (2) Started contributing under a pseudonym
- (3) Worked, asked questions, or collaborated in private channels more often

- (4) Changed or deleted a username
- (5) Removed or changed content on my public online presence
- (6) Suggested the creation or modification of a Code of Conduct
- (7) Engaged in public discussion with community members about the issue
- (8) Engaged in private discussion with community members about the issue
- (9) Made changes in my life offline (e.g. stopped attending meetups or conferences, etc.)
- (10) Other (please describe)
- (11) None of the above

Additional survey questions may include:

- Do you feel any private details shared with other project contributors or leadership will be kept in the strictest confidence?
- Do you feel as if project leadership values the safety of its participants?
- Do you feel as if project leadership values transparency in its communications and interactions?
- Do you feel as if project leadership is open to critical feedback regarding trust and safety issues?
- Does project leadership acknowledge potential areas in the code that could be used for abuse?
- Did project leadership prioritize these issues as areas to be fixed?
- Did the community push back on fixing these issues?

Filters

- Demographic segments
- Role of contributor (code, community management and advocacy, documentation, etc.)
- Length of time in the community

References

- Psychological Safety, Trust, and Learning in Organizations: A Group-level Lens
- Adding Community and Safety Checks to New Features (GitHub Blog)
- Open Source Survey 2017
- What is Psychological Safety at Work

Contributors

- Elizabeth Barron
- Matt Germonprez
- Kevin Lumbard
- Lauren Phipps
- Dawn Foster
- Matt Cantu

- Lucas Gonze
- Justin W. Flory
- Emily Brown
- Amy Marrich
- Trisha Rajaram
- Ruth Ikegah
- Emily Brown
- Sean Goggins
- Georg Link

Evolution WG

Focus Area	Goal
Code Development Activity	Learn about the types and frequency of activities involved in developing code.
Code Development Efficiency	Learning how efficiently activities around code development get involved.
Code Development Process Quality	Learning about the processes to improve/review quality that are used (for example: testing, code review, tagging issues, tagging a release, time to response, CII Badging).
Issue Resolution	Identify how effective the community is at addressing issues identified by community participants.
Community Growth	Identify the size of the project community and whether it is growing, shrinking, or staying the same.

Focus Area - Code Development Activity

Goal: Learn about the types and frequency of activities involved in developing code.

Metric	Question
Branch Lifecycle	How do projects manage the lifecycle of their version control branches?
Code Changes	How many changes were made to the source code during a specified period?
Code Changes Lines	What is the sum of the number of lines touched (lines added plus lines removed) in all changes to the source code during a certain period?

Branch Lifecycle

Question: How do projects manage the lifecycle of their version control branches?

Description

This metric makes the lifecycle of version control branches visible. A branch lifecycle includes acts of branch creation and deletion, as well as persistence of version control branches. When writing code, a development team may create multiple branches, focused around specific features. Subsequently, those branches may be merged into more persistent branches, such as the main branch of a repository. Some branches persist for the life of the repository, while others are deleted after code is merged into a more persistent branch. By understanding these patterns, we can learn how branch creation, destruction, and merging reflect the development practices of a particular project. A repository's typical branch lifecycle and management approach can be used to help identify a project's repository management style.

Objectives

This metric's objective is to increase the visibility of a repository's volume and frequency of branch creation and deletion, as well as persistence of version control branches, in the context of other project characteristics. In turn, this helps potential contributors understand how to initiate and sustain participation in a project. Questions that can be addressed through this metric include:

- How many branches have been merged but not deleted?
- How long has a branch been merged and not deleted?
- How many branches have existed longer than a certain number of days, months, or years?
- How often do projects or repositories create branches?
- In the aggregate, how long do branches usually live?
- How can we distinguish between branch "death" (i.e., never intended to be used again; deletion) or branch "dormancy" (i.e., inactive for long periods of time, but may be used again) in cases where branches are infrequently deleted in a repository?

Implementation

The stated advice regarding management of the branch lifecycle for a project may be visible in a CONTRIBUTING.md document, and these documents can be compared across repositories using linguistic analysis, and contrasted with data derived from actual project practices to draw insights within, and across repositories. In most cases, however, the data we focus on in this metric is quantitative in nature.

Aggregators:

- Count of branches created.
- Count of branches deleted.
- Count of branches merged.
- Count of branches abandoned (had unique commits, but never got merged before it was deleted)
- Total count of branches.
- Average age of open branches.
- Ratio at which branches are created vs. deleted.

Parameters:

- Period of time. Start and finish date of the period. Default: forever.
- Period during which change requests are considered.

Filters (optional)

- Collections of repositories
- Default branch name versus descriptive name with regard to branch persistence

Data Collection Strategies

Specific description: Git

Git branching data exists at several different levels in a version control ecosystem, usually both locally on a developer's machine as well on a hosted platform like GitHub or BitBucket. This branch data on the hosted platform may also differ from each developer's machine, and additionally, different developers may have different branch data on their machine even for the same repository.

In the specific case of Git, a significant amount of additional complexity is introduced due to Git's design as a distributed version control system, which means that Git allows multiple remotes for a single repository (for example, a user's fork at github.com/user/project and the upstream version at github.com/chaoss/project). More often than not, many individual contributors may work in the same branch locally, and push changes to the remote repository. The local copies, therefore, will sometimes be different than the remote, hosted internally or on platforms like GitHub, GitLab, and BitBucket, since they're likely either being managed by different people (likely with different branching styles) or they are both being used by one person to "silo" the work they are doing.

Data about Git branches can be derived from a Git log directly, or through a Git platform's API.

Is picking a "canonical" version of the repo for branching data is useful? This gives a meaningful base for comparing between instances of the repo with different data. Could also prove to

`git branch -a` will list all existing branches.

References

Adopting a Git Branching Strategy: https://docs.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops

Choose the Right Git Branching Strategy: https://www.creativebloq.com/web-design/choose-right-git-branching-strategy-121518344

The Effect of Branching Strategies on Software Quality by Emad Shihab, Christian Bird, and Thomas Zimmermann https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/shihab-esem-2012.pdf

OpenStack: https://docs.openstack.org/project-team-guide/other-branches.html

Code Changes

Question: How many changes were made to the source code during a specified period?

Description

These are changes to the source code during a certain period. For ”change” we consider what developers consider an atomic change to their code. In other words, a change is some change to the source code which usually is accepted and merged as a whole, and if needed, reverted as a whole too. For example, in the case of git, each ”change” corresponds to a ”commit”, or, to be more precise, ”code change” corresponds to the part of a commit which touches files considered as source code.

Objectives

- Volume of coding activity. Code changes are a proxy for the activity in a project. By counting the code changes in the set of repositories corresponding to a project, you can have an idea of the overall coding activity in that project. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of changes during the period.

Parameters:

- Period of time. Start and finish date of the period. Default: forever. Period during which changes are considered.
- Criteria for source code. Algorithm. Default: all files are source code. If focused on source code, criteria for deciding whether a file is a part of the source code or not.

Filters

- By actors (author, committer). Requires actor merging (merging ids corresponding to the same author).
- By groups of actors (employer, gender...). Requires actor grouping, and likely, actor merging.
- By tags (used in the message of the commits). Requires a structure for the message of commits. This tag can be used in an open-source project to communicate to every contributors if the commit is, for example, a fix for a bug or an improvement of a feature.

Visualizations

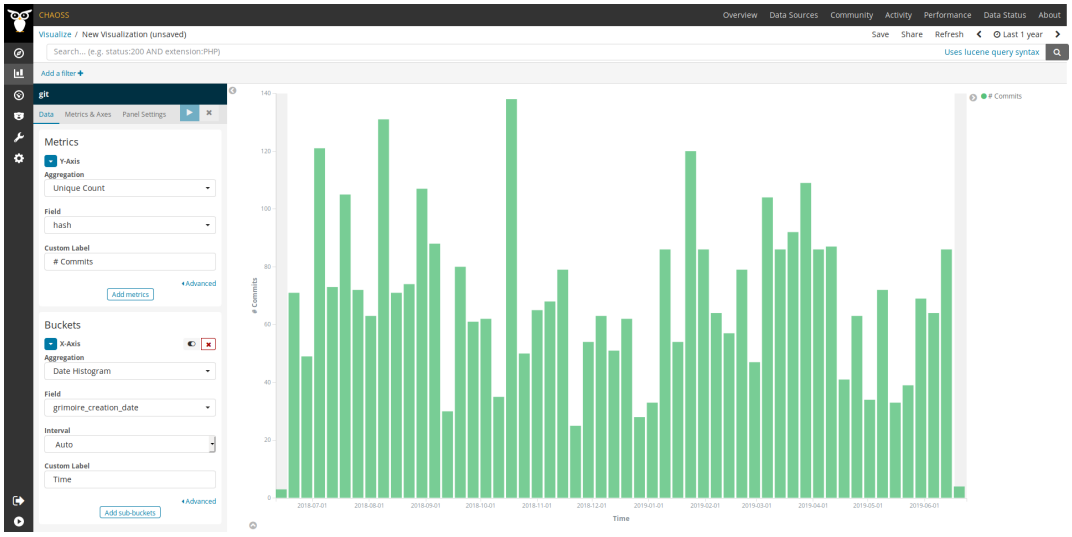
- Count per month over time
- Count per group over time

These could be represented as bar charts, with time running in the X axis. Each bar would represent a code changes during a certain period (eg, a month).

Tools Providing the Metric

- GrimoireLab provides this metric out of the box.
    - View an example on the CHAOSS instance of Bitergia Analytics.

- Download and import a ready-to-go dashboard containing examples for this metric visualization from the GrimoireLab Sigils panel collection.
- Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
  - Create a new `Vertical Bar` chart
  - Select the `git` index
  - Y-axis: `Unique Count` Aggregation, `hash` Field, `# Commits` Custom Label
  - X-axis: `Date Histogram` Aggregation, `grimoire_creation_date` Field, `Auto` Interval, `Time` Custom Label
- Example screenshot:



- Augur provides this metric both as Code Changes and as Code Changes Lines. Both metrics are available in both the `repo` and the `repo_group` metric forms - more on that in the Augur documentation.
- Gitdm

Data Collection Strategies

Specific description: Git

See reference implementation for git

Mandatory parameters (for Git):

- Date type. Either author date or committer date. Default: author date.
  For each git commit, two dates are kept: when the commit was authored, and when it was committed to the repository. For deciding on the period, one of them has to be selected.
- Include merge commits. Boolean. Default: True.
  Merge commits are those which merge a branch, and in some cases are not considered as reflecting a coding activity.
- Include empty commits. Boolean. Default: True.
  Empty commits are those which do not touch files, and in some cases are not considered as reflecting a coding activity.

References

- https://www.odoo.com/documentation/13.0/reference/guidelines.html#tag-and-module-name

Code Changes Lines

Question: What is the sum of the number of lines touched (lines added plus lines removed) in all changes to the source code during a certain period?

Description

When introducing changes to the source code, developers touch (edit, add, remove) lines of the source code files. This metric considers the aggregated number of lines touched by changes to the source code performed during a certain period. This means that if a certain line in a certain file is touched in three different changes, it will count as three lines. Since in most source code management systems it is difficult or impossible to tell accurately if a lines was removed and then added, or just edited, we will consider editing a line as removing it and later adding it back with a new content. Each of those (removing and adding) will be considered as ”touching”. Therefore, if a certain line in a certain file is edited three times, it will count as six different changes (three removals, and three additions).

For this matter, we consider changes to the source code as defined in Code Changes. Lines of code will be any line of a source code file, including comments and blank lines.

Objectives

- Volume of coding activity:
  Although code changes can be a proxy to the coding activity of a project, not all changes are the same. Considering the aggregated number of lines touched in all changes gives a complementary idea of how large the changes are, and in general, how large is the volume of coding activity.

Implementation

Aggregators:

- Count. Total number of lines changes (touched) during the period.

Parameters:

- **Period of time:** Start and finish date of the period. Default: forever.
  Period during which changes are considered.
- **Criteria for source code; Algorithm Default:** all files are source code.
  If we are focused on source code, we need a criterion for deciding whether a file is a part of the source code or not.
- **Type of source code change:**
  - Lines added
  - Lines removed
  - Whitespace

Filters

- By actors (author, committer). Requires actor merging (merging ids corresponding to the same author).
- By groups of actors (employer, gender...). Requires actor grouping, and likely, actor merging.
- By tags (used in the message of the commits). Requires a structure for the message of commits. This tag can be used in an open-source project to communicate to every contributors if the commit is, for example, a fix for a bug or an improvement of a feature.

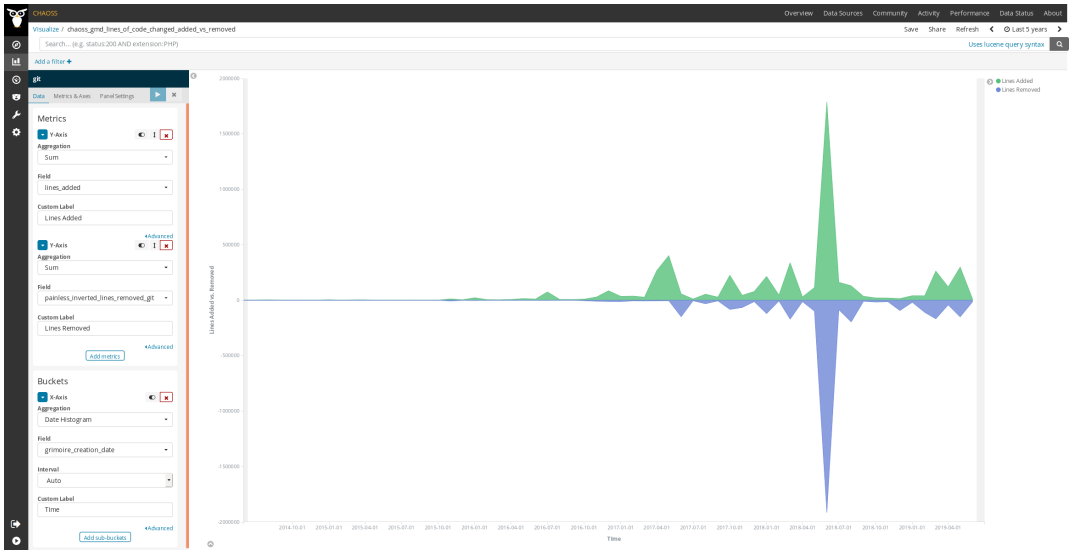
Visualizations

- Count per month over time
- Count per group over time

These could be represented as bar charts, with time running in the X axis. Each bar would represent a code changes during a certain period (eg, a month).

Tools Providing the Metric

- GrimoireLab provides this metric out of the box.
  - View an example on the CHAOSS instance of Bitergia Analytics.
  - Download and import a ready-to-go dashboard containing examples for this metric visualization from the GrimoireLab Sigils panel collection.
  - Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
    - Create a new `Area` chart
    - Select the `git` index
    - Y-axis 1: `Sum` Aggregation, `lines_added` Field, `Lines Added` Custom Label
    - Y-axis 2: `Sum` Aggregation, `painless_inverted_lines_removed_git` Field, `Lines Removed` Custom Label
    - X-axis: `Date Histogram` Aggregation, `grimoire_creation_date` Field, `Auto` Interval, `Time` Custom Label
  - Example screenshot:



Data Collection Strategies

Specific description: Git

In the cases of git, we define ”code change” and ”date of a change” as we detail in Code Changes. The date of a change can be defined (for considering it in a period or not) as the author date or the committer date of the corresponding git commit.

Since git provides changes as diff patches (list of lines added and removed), each of those lines mentioned as a line added or a line removed in the diff will be considered as a line changed (touched). If a line is removed and added, it will be considered as two ”changes to a line”.

Mandatory parameters:

- Kind of date. Either author date or committer date. Default: author date.
  For each git commit, two dates are kept: when the commit was authored, and when it was committed to the repository. For deciding on the period, one of them has to be selected.
- Include merge commits. Boolean. Default: True.
  Merge commits are those which merge a branch, and in some cases are not considered as reflecting a coding activity

References

- https://www.odoo.com/documentation/13.0/reference/guidelines.html#tag-and-module-name

Focus Area - Code Development Efficiency

Goal: Learning how efficiently activities around code development get involved.

Metric	Question
Change Requests Accepted	How many accepted change requests are present in a code change?
Change Requests Declined	What change requestsended up being declined during a certain period?
Change Requests Duration	What is the duration of time between the moment a change request starts and the moment it is accepted or closed?
Change Request Acceptance Ratio	What is the ratio of change requests accepted to change requests closed without being merged?

Change Requests Accepted

Question: How many accepted change requests are present in a code change?

Description

Change requests are defined as in Change Requests. Accepted change requests are those that end with the corresponding changes finally merged into the code base of the project. Accepted change requests can be linked to one or more changes to the source code, those corresponding to the changes proposed and finally merged.

For example, in GitHub when a pull request is accepted, all the commits included in it are merged (maybe squashed, maybe rebased) in the corresponding git repository. The same can be said of GitLab merge requests. In the case of Gerrit, a change request usually corresponds to a single commit.

Objectives

- Volume of coding activity.
  Accepted change requests are a proxy for the activity in a project. By counting accepted change requests in the set of repositories corresponding to a project, you can have an idea of the overall coding activity in that project that leads to actual changes. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of accepted change requests during the period.
- Ratio. Ratio of accepted change requests over total number of change requests during that period.

Parameters:

- Period of time. Start and finish date of the period during which accepted change requests are considered. Default: forever.
- Criteria for source code. Algorithm. Default: all files are source code.
  If we focus on source code, we need a criterion for deciding whether a file belongs to the source code or not.

Filters

- By actor type (submitter, reviewer, merger). Requires merging identities corresponding to the same actor.
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

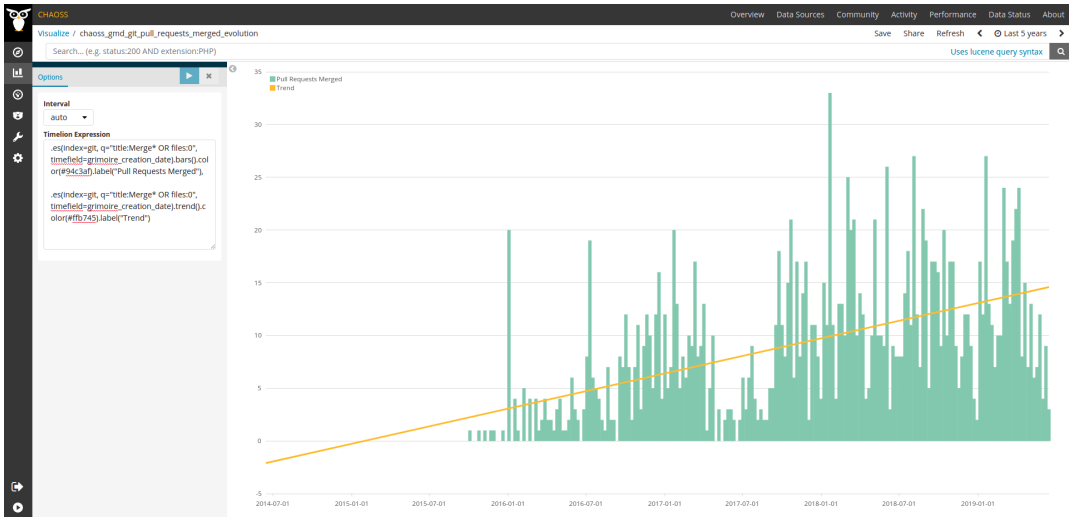
Visualizations

- Count per time period over time
- Ratio per time period over time

These could be grouped by actor type or actor group by applying the filters defined above. These could be represented as bar charts, with time running in the X axis. Each bar would represent accepted change requests to change the code during a certain period (eg, a month).

Tools Providing the Metric

- Grimoirelab provides this metric out of the box for GitHub Pull Requests and also provides data to build similar visualizations for GitLab Merge Requests and Gerrit Changesets.
    - View an example on the CHAOSS instance of Bitergia Analytics.
    - Download and import a ready-to-go dashboard containing examples for this metric visualization based on GitHub Pull Requests data from the GrimoireLab Sigils panel collection.
    - Example screenshot:



Data Collection Strategies

Specific description: GitHub

In the case of GitHub, accepted change requests are defined as ”pull requests whose changes are included in the git repository”, as long as it proposes changes to source code files.

Unfortunately, there are several ways of accepting change requests, not all of them making it easy to identify that they were accepted. The easiest situation is when the change request is accepted and merged (or rebased, or squashed and merged). In that case, the change request can easily be identified as accepted, and the corresponding commits can be found via queries to the GitHub API.

But change requests can also be closed, and commits merged manually in the git repository. In this case, commits may still be found in the git repository, since their hash is the same found in the GitHub API for those in the change request.

In a more difficult scenario, change requests can also be closed, and commits rebased, or maybe squashed and then merged, manually. In these cases, hashes are different, and only an approximate matching via dates and authors, and/or comparison of diffs, can be used to track commits in the git repository.

From the point of view of knowing if change requests were accepted, the problem is that if they are included in the git repository manually, the only way of knowing that the change request was accepted is finding the corresponding commits in the git repository.

In some cases, projects have policies of mentioning the commits when the change request is closed (such as ”closing by accepting commits xxx and yyyy”), which may help to track commits in the git repository.

Mandatory parameters (for GitHub):

- Heuristic for detecting accepted change requests not accepted via the web interface. Default: None.

Specific description: GitLab

In the case of GitLab, accepted change requests are defined as ”merge requests whose changes are included in the git repository”, as long as it proposes changes to source code files.

Mandatory parameters (for GitLab):

- Heuristic for detecting accepted change requests not accepted via the web interface. Default: None.

Specific description: Gerrit

In the case of Gerrit, accepted change requests are defined as ”changesets whose changes are included in the git repository”, as long as they proposes changes to source code files.

Mandatory parameters (for Gerrit): None.

References

Change Requests Declined

Question: What change requestsended up being declined during a certain period?

Description

Change requests are defined as in Change Requests. Declined change requests are those that are finally closed without being merged into the code base of the project.

For example, in GitHub when a pull request is closed without merging, and the commits referenced in it cannot be found in the git repository, it can be considered to be declined (but see detailed discussion below). The same can be said of GitLab merge requests. In the case of Gerrit, code reviews can be formally ”abandoned”, which is the way of detecting declined change requests in this system.

Objectives

- Volume of coding activity. Declined change requests are a proxy for the activity in a project. By counting declined change requests in the set of repositories corresponding to a project, you can have an idea of the overall coding activity in that project that did not lead to actual changes. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of declined change requests during the period.
- Ratio. Ratio of declined change requests over the total number of change requests during that period.

Parameters:

- Period of time. Start and finish date of the period during which declined change requests are considered. Default: forever.
- Criteria for source code. Algorithm. Default: all files are source code. If we focus on source code, we need a criterion to decide whether a file belongs to the source code or not.

Filters

- By actors (submitter, reviewer, merger). Requires merging identities corresponding to the same actor.
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

Visualizations

- Count per period over time
- Ratio per period over time

These could be grouped (per actor type, or per group of actors) by applying the filters, and could be represented as bar charts, with time running in the X axis. Each bar would represent declined change requests during a certain period (e.g., a month).

Data Collection Strategies

Specific description: GitHub

In the case of GitHub, declined change requests are defined as ”pull requests that are closed with their changes not being included in the git repository”, as long as it proposes changes to source code files.

See the discussion in the specific description for GitHub in Change Requests Accepted, since it applies here as well.

Mandatory parameters (for GitHub):

- Heuristic for detecting declined change requests, telling apart those cases where the change request was closed, but the changes were included in the git repository manually. Default: None.

Specific description: GitLab

In the case of GitLab, declined reviews are defined as ”merge requests that are closed with their changes not being included in the git repository”, as long as it proposes changes to source code files.

Mandatory parameters (for GitLab):

- Heuristic for detecting declined change requests, telling apart those cases where the merge request was closed, but the changes were included in the git repository manually. Default: None.

Specific description: Gerrit

In the case of Gerrit, declined change requests are defined as ”changesets abandoned”, as long as they propose changes to source code files.

Mandatory parameters (for Gerrit): None.

References

Change Requests Duration

Question: What is the duration of time between the moment a change request starts and the moment it is accepted or closed?

Description

Change requests are defined as in Change Requests. Accepted change requests are defined in Change Requests Accepted.

The change request duration is the duration of the period since the change request started, to the moment it ended (by being accepted and being merged in the code base). This only applies to accepted change requests.

For example, in GitLab a change request starts when a developer uploads a proposal for a change in code, opening a change request. It finishes when the change request is finally accepted and merged in the code base, closing the change request.

In case there are comments or other events after the code is merged, they are not considered for measuring the duration of the change request.

Objectives

- Duration of acceptance of change requests. Review duration for accepted change requests is one of the indicators showing how long does a project take before accepting a contribution of code. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Median. Median (50% percentile) of change request duration for all accepted change requests in the considered period of time.

Parameters:

- Period of time. Start and finish date of the period. Default: forever.
  Period during which accepted change requests are considered. An accepted change request is considered to be in the period if its creation event is in the period.
- Criteria for source code. Algorithm. Default: all files are source code.
  If we are focused on source code, we need a criteria for deciding whether a file is a part of the source code or not.

Filters

- By actors (submitter, reviewer, merger). Requires actor merging (merging ids corresponding to the same author).
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

Visualizations

- Median per month over time
- Median per group over time

These could be represented as bar charts, with time running in the X axis. Each bar would represent accepted change requests to change the code during a certain period (e.g., a month).

- Distribution of durations for a certain period

These could be represented with the usual statistical distribution curves, or with bar charts, with buckets for duration in the X axis, and number of reviews in the Y axis.

Data Collection Strategies

Specific description: GitHub

In the case of GitHub, duration is considered for pull requests that are accepted and merged in the code base. For an individual pull request, duration starts when it is opened, and finishes when the commits it includes are merged into the code base.

Mandatory parameters (for GitHub): None.

Specific description: GitLab

In the case of GitLab, duration is considered for merge requests that are accepted and merged in the code base. For an individual merge request, duration starts when it is opened, and finishes when the commits it includes are merged into the code base.

Mandatory parameters (for GitLab): None.

Specific description: Gerrit

In the case of Gerrit, duration is considered for code reviews that are accepted and merged in the code base. For an individual code review, duration starts when it is opened, and finishes when the commits it includes are merged into the code base.

Mandatory parameters (for Gerrit): None.

References

Change Request Acceptance Ratio

Question: What is the ratio of change requests accepted to change requests closed without being merged?

Description

Each change request can be in one of three states: open, merged (accepted), and closed without merge (declined). This metric measures the ratio of change requests merged (accepted) vs change requests closed without being merged.

Objectives

The ratio of change requests merged to change requests closed without merging provides insight into several repository characteristics, including openness to outside contributions, growth of the contributor community, the efficiency of code review processes, and, when measured over time, the trajectory of a project in its evolution. Different ratios should be interpreted in the context of each repository or project.

Implementation

Parameters

- Time Period Granularity (Weekly, Monthly, Annually).
- Change in ratio over the period of time.
- Show contributor count
- Origin of change request: branch or fork? Change requests from repository forks are more commonly from outside contributors, while branch originating change requests come from people with repository commit rights.

Aggregators

- Total change requests merged (accepted)
- Total change requests closed without merge
- Total change requests in an open state

Visualizations

CHAOSS tools provide a number of visualizations for this metric. The first visualization shows the accepted and declined change requests organized annually, from which ratios can be derived.

Figure One:

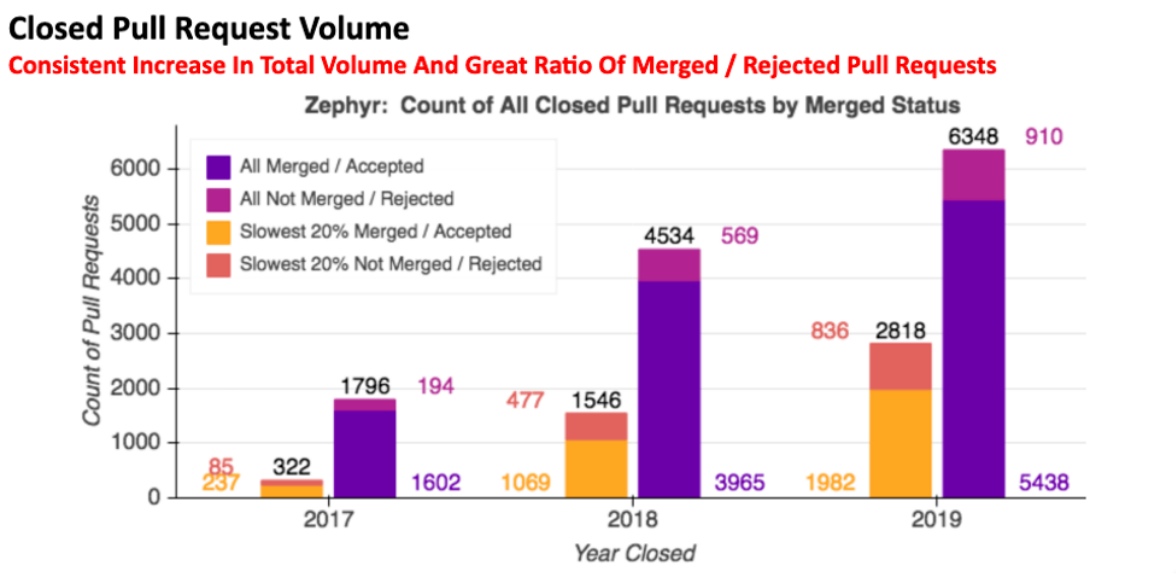


Figure Two:

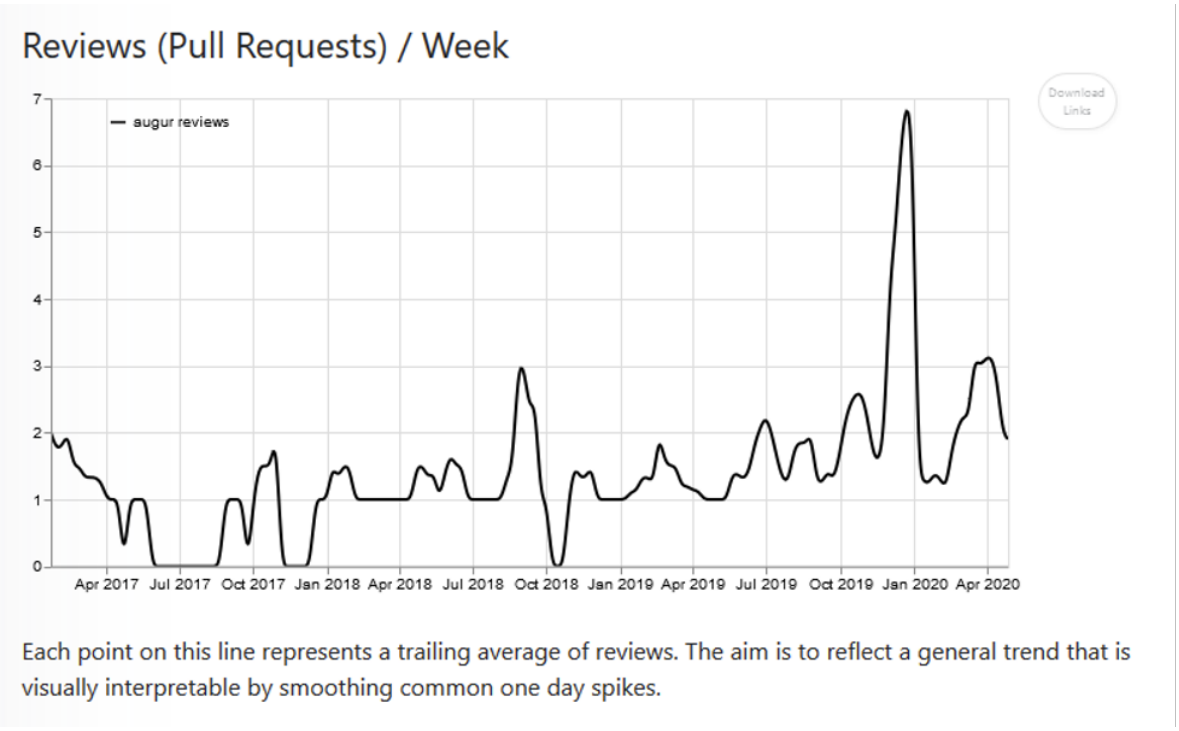
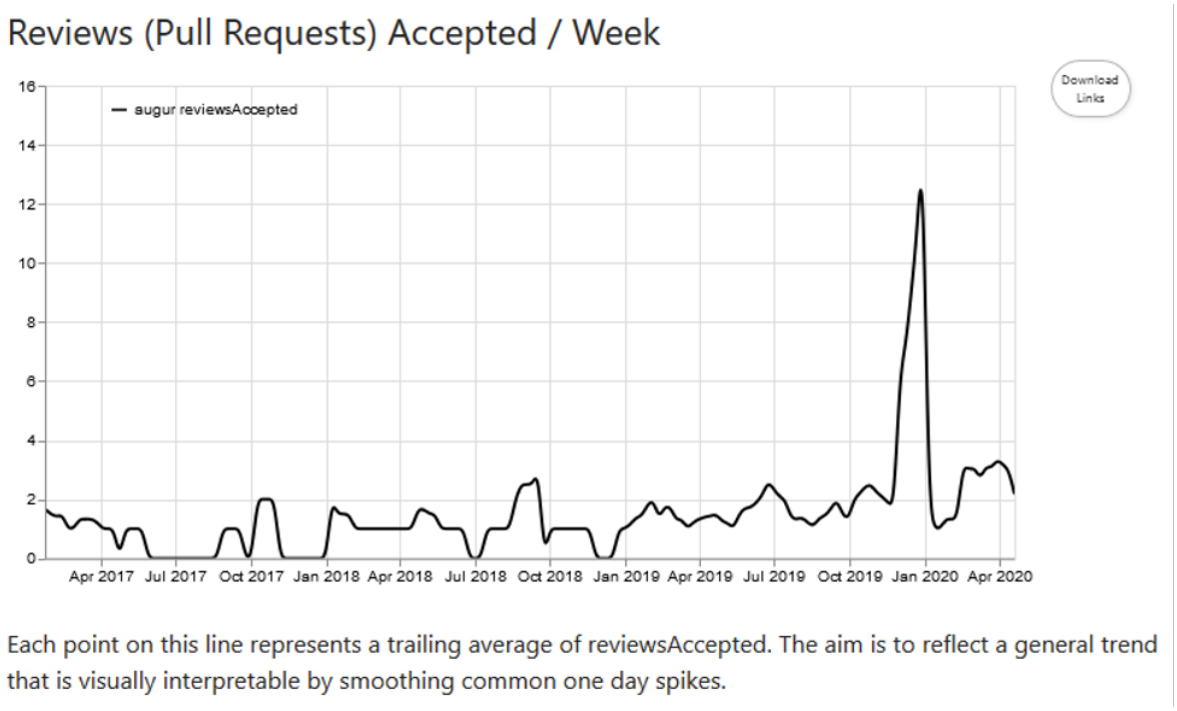


Figure Three:



Tools Providing the Metric

- https://github.com/chaoss/augur
- https://github.com/chaoss/augur-community-reports

Data Collection Strategies

Accepted change requests are defined as in the Change Requests Accepted metric, and Declined change requests are defined as in the Change Requests Declined metric.

References

Augur Zephyr report on pull requests: https://docs.google.com/presentation/d/11b48Zm5Fwsmd1OIHg4bse5ibaVJUWkUIZb edit#slide=id.g7ec7768776_1_56

Focus Area - Code Development Process Quality

Goal: Learning about the processes to improve/review quality that are used (for example: testing, code review, tagging issues, tagging a release, time to response, CII Badging).

Metric	Question
Change Requests	What new change requests to the source code occurred during a certain period?

Change Requests

Question: What new change requests to the source code occurred during a certain period?

Description

When a project uses change request processes, changes are not directly submitted to the code base, but are first proposed for discussion as ”proposals for change to the source code”. Each of these change requests are intended to be reviewed by other developers, who may suggest improvements that will lead to the original proposers sending new versions of their change requests, until reviews are positive, and the code is accepted, or until it is decided that the proposal is declined.

For example, ”change requests” correspond to ”pull requests” in the case of GitHub, to ”merge requests” in the case of GitLab, and to ”code reviews” or in some contexts ”changesets” in the case of Gerrit.

Objectives

- Volume of changes proposed to a project. Change requests are a proxy for the activity in a project. By counting change requests in the set of repositories corresponding to a project, you can have an idea of the overall activity in changes to that project. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of change requests during the period.

Parameters:

- Period of time. Start and finish date of the period. Default: forever.
  Period during which change requests are considered.
- Criteria for source code. Algorithm. Default: all files are source code.
  If we are focused on source code, we need a criteria for deciding whether a file is a part of the source code or not.

Filters

- By actors (submitter, reviewer, merger). Requires actor merging (merging ids corresponding to the same author).
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.
- Status (open, closed)

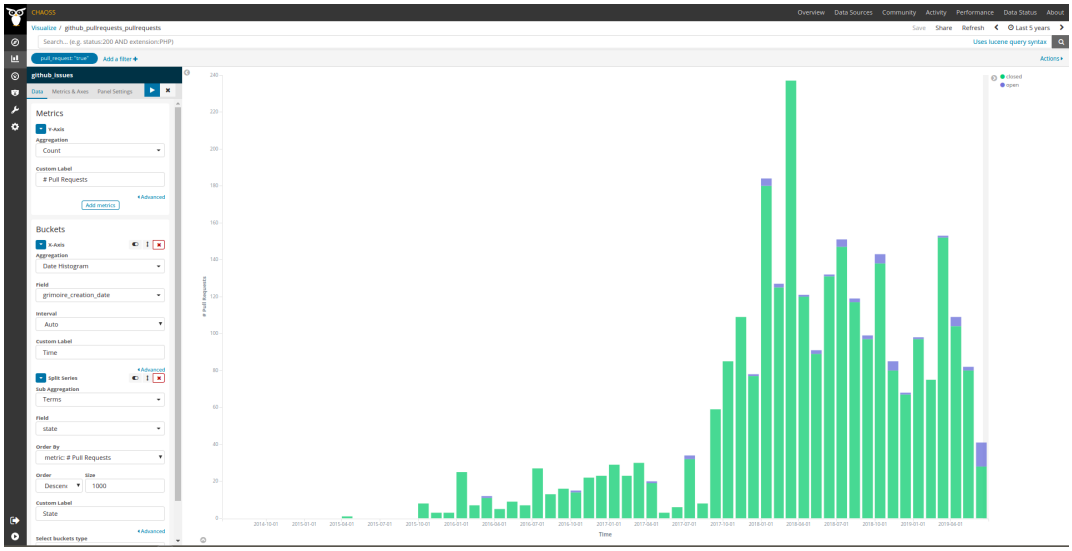
Visualizations

- Count per month over time
- Count per group over time

These could be represented as bar charts, with time running in the X axis. Each bar would represent change requests to change the code during a certain period (eg, a month).

Tools Providing the Metric

- Grimoirelab provides this metric out of the box for GitHub Pull Requests, GitLab Merge Requests, and Gerrit Changesets.
    - View an example on the CHAOSS instance of Bitergia Analytics.
    - Download and import a ready-to-go dashboard containing examples for this metric visualization based on GitHub Pull Requests data from the GrimoireLab Sigils panel collection.
    - Example screenshot:



Data Collection Strategies

Specific description: GitHub

In the case of GitHub, a change request is defined as a ”pull request”, as long as it proposes changes to source code files.

The date of the review can be defined (for considering it in a period or not) as the date in which the pull request was submitted.

Specific description: GitLab

In the case of GitLab, a change request is defined as a ”merge request”, as long as it proposes changes to source code files.

The date of the change request can be defined (for considering it in a period or not) as the date in which the change request was submitted.

Specific description: Gerrit

In the case of Gerrit, a change request is defined as a ”code review”, or in some contexts, a ”changeset”, as long as it proposes changes to source code files.

The date of the change request can be defined (for considering it in a period or not) as the date in which the change request was started by submitting a patchset for review.

References

Focus Area - Issue Resolution

Goal: Identify how effective the community is at addressing issues identified by community participants.

Metric	Question
New Issues	How many new issues are created during a certain period?
Issues Active	How many issues were active during a certain period?
Issues Closed	How many issues were closed during a certain period?
Issue Age	How long have open issues been left open?
Issue Response Time	How much time passes between the opening of an issue and a response in the issue thread from another contributor?
Issue Resolution Duration	How long does it take for an issue to be closed?

New Issues

Question: How many new issues are created during a certain period?

Description

Projects discuss how they are fixing bugs, or adding new features, in tickets in the issue tracking system. Each of these tickets (issues) are opened (submitted) by a certain person, and are later commented and annotated by many others.

Depending on the issue system considered, an issue can go through several states (for example, ”triaged”, ”working”, ”fixed”, ”won’t fix”), or being tagged with one or more tags, or be assigned to one or more persons. But in any issue tracking system, an issue is usually a collection of comments and state changes, maybe with other annotations. Issues can also be, in some systems, associated to milestones, branches, epics or stories. In some cases, some of these are also issues themselves.

At least two ”high level” states can usually be identified: open and closed. ”Open” usually means that the issues is not yet resolved, and ”closed” that the issue was already resolved, and no further work will be done with it. However, what can be used to identify an issue as ”open” or ”closed” is to some extent dependent on the issue tracking system, and on how a given project uses it. In real projects, filtering the issues that are directly related to source code is difficult, since the issue tracking system may be used for many kinds of information, from fixing bugs and discussing implementation of new features, to organizing a project event or to ask questions about how to use the results of the project.

In most issue trackers, issues can be reopened after being closed. Reopening an issue can be considered as opening a new issue (see parameters, below).

For example, ”issues” correspond to ”issues” in the case of GitHub, GitLab or Jira, to ”bug reports” in the case of Bugzilla, and to ”issues” or ”tickets” in other systems.

Objectives

Volume of issues discussed in a project. Issues are a proxy for the activity in a project. By counting issues discussing code in the set of repositories corresponding to a project, you can have an idea of the overall activity in discussing issues in that project. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of new issues during the period.
- Ratio. Ratio of new issues over total number of issues during that period.

Parameters:

- Period of time. Start and finish date of the period during which issues are considered. Default: forever.
- Criterion for source code. Algorithm. Default: all issues are related to source code.
  If we focus on source code, we need a criterion for deciding whether an issue is related to the source code or not.
- Reopen as new. Boolean. Default: False.
  Criterion for defining whether reopened issues are considered as new issues.

Filters

- By actors (submitter, commenter, closer). Requires merging identities corresponding to the same author.
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

Visualizations

- Count per time period over time
- Ratio per time period over time

These could be grouped by applying the filters defined above. These could be represented as bar charts, with time running in the X axis. Each bar would represent proposals to change the code during a certain period (eg, a month).

Data Collection Strategies

Specific description: GitHub

In the case of GitHub, an issue is defined as an ”issue”.

The date of the issue can be defined (for considering it in a period or not) as the date in which the issue was opened (submitted).

Specific description: GitLab

In the case of GitHub, an issue is defined as an ”issue”.

The date of the issue can be defined (for considering it in a period or not) as the date in which the issue was opened (submitted).

Specific description: Jira

In the case of Jira, an issue is defined as an ”issue”.

The date of the issue can be defined (for considering it in a period or not) as the date in which the issue was opened (submitted).

Specific description: Bugzilla

In the case of Bugzilla, an issue is defined as a ”bug report”, as long as it is related to source code files.

The date of the issue can be defined (for considering it in a period or not) as the date in which the bug report was opened (submitted).

References

Issues Active

Question: How many issues were active during a certain period?

Description

Issues are defined as in Issues New. Issues showing some activity are those that had some comment, or some change in state (including closing the issue), during a certain period.

For example, in GitHub Issues, a comment, a new tag, or the action of closing an issue, is considered as a sign of activity.

Objectives

- Volume of active issues in a project. Active issues are a proxy for the activity in a project. By counting active issues related to code in the set of repositories corresponding to a project, you can have an idea of the overall activity in working with issues in that project. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of active issues during the period.
- Ratio. Ratio of active issues over total number of issues during that period.

Parameters:

- Period of time. Start and finish date of the period during which issues are considered. Default: forever.
- Criteria for source code. Algorithm. Default: all issues are related to source code.
  If we focus on source code, we need a criterion for deciding whether an issue is related to the source code or not.

Filters

- By actor (submitter, commenter, closer). Requires merging identities corresponding to the same author.
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

Visualizations

- Count per period over time
- Ratio per period over time

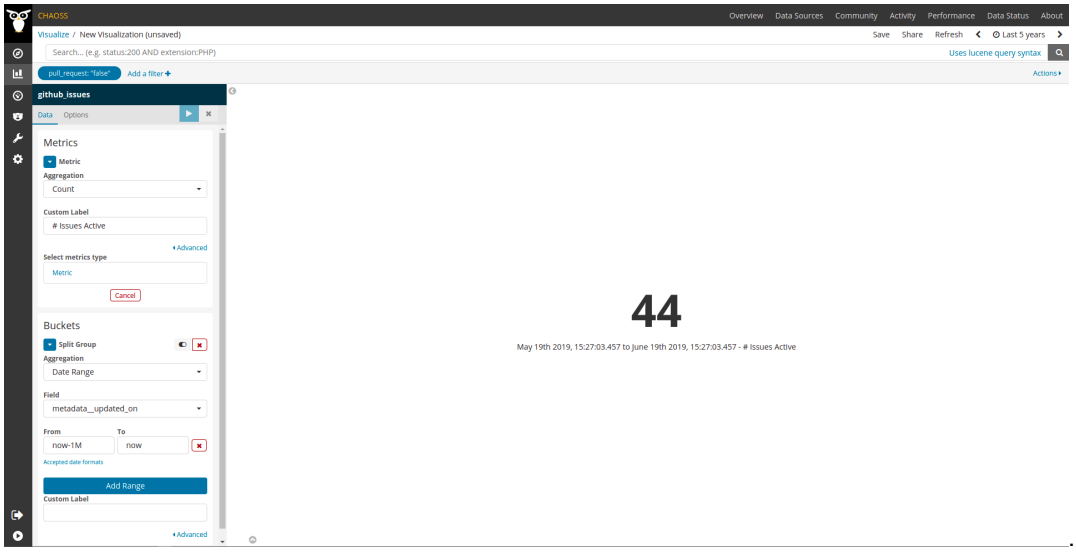
These could be grouped by applying the previously defined filters. These could be represented as bar charts, with time running in the X axis. Each bar would represent proposals to change the code during a certain period (eg, a month).

Tools Providing the Metric

- GrimoireLab provides data for computing a metric close to the one described in this page for GitHub Issues, GitLab issues, Jira, Bugzilla and Redmine. In terms of the metric, **GrimoireLab data have only**

the date of the last update of each item, which limits computing this metric to time ranges ending on the current date.

- Depending on the source API, the definition of what is considered an update on the issue could vary. GrimoireLab uses `metadata__updated_on` to store latest issue update, please check Perceval documentation to look for the specific API field being used in each case and understand its limitations, if any.
- Currently, there is no dashboard showing this in action. Nevertheless, it is easy to build a visualization that shows the number uses which last activity occurred at some point between a date and current date (we'll do it for GitHub Issues here).
- Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
  - Create a new `Metric` visualization.
  - Select the `github_issues` index.
  - Filter: `pull_request` is `false` .
  - Metric: `Count` Aggregation, `# Issues Active` Custom Label.
  - Buckets: `Date Range` Aggregation, `metadata__updated_on` Field, `now-1M` From (or whatever interval may fit your needs), `now` To, leave Custom Label empty to see the specific dates in the legend.
  - Have a look at the time picker on the top right corner to make sure it is set to include the whole story of the data so we are not excluding any item based on its creation date.
- Example screenshot:



Data Collection Strategies

Specific description: GitHub

In the case of GitHub, active issues are defined as "issues which get a comment, a change in tags, a change in assigned person, or are closed".

Specific description: GitLab

In the case of GitLab, active issues are defined as ”issues which get a comment, a change in tags, a change in assigned person, or are closed”.

Specific description: Jira

In the case of Jira, active issues are defined as ”issues which get a comment, a change in state, a change in assigned person, or are closed”.

Specific description: Bugzilla

In the case of Bugzilla, active issues are defined as ”bug reports which get a comment, a change in state, a change in assigned person, or are closed”.

References

Issues Closed

Question: How many issues were closed during a certain period?

Description

Issues are defined as in Issues New. Issues closed are those that changed to state closed during a certain period.

In some cases or some projects, there are other states or tags that could be considered as ”closed”. For example, in some projects they use the state or the tag ”fixed” for stating that the issue is closed, even when it needs some action for formally closing it.

In most issue trackers, closed issues can be reopened after they are closed. Reopening an issue can be considered as opening a new issue, or making void the previous close (see parameters, below).

For example, in GitHub Issues or GitLab Issues, issues closed are issues that were closed during a certain period.

Objectives

Volume of issues that are dealt with in a project. Closed issues are a proxy for the activity in a project. By counting closed issues related to code in the set of repositories corresponding to a project, you can have an idea of the overall activity in finishing work with issues in that project. Of course, this metric is not the only one that should be used to track volume of coding activity.

Implementation

Aggregators:

- Count. Total number of closed issues during the period.
- Ratio. Ratio of closed issues over total number of issues during that period.
- Reactions. Number of ”thumb-ups” or other reactions on issues.

Parameters:

- Period of time. Start and finish date of the period during which issues are considered. Default: forever.
- Criteria for source code. Algorithm. Default: all issues are related to source code.
  If we focus on source code, we need a criterion for deciding whether an issue is related to the source code or not. All issues could be included in the metric by altering the default.
- Reopen as new. Boolean, defining whether reopened issues are considered as new issues. If false, it means the closing event previous to a reopen event should be considered as void. Note: if this parameter is false, the number of closed issues for any period could change in the future, if some of them are reopened.
- Criteria for closed. Algorithm. Default: having a closing event during the period of interest.

Filters

- By actors (submitter, commenter, closer). Requires merging identities corresponding to the same author.
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

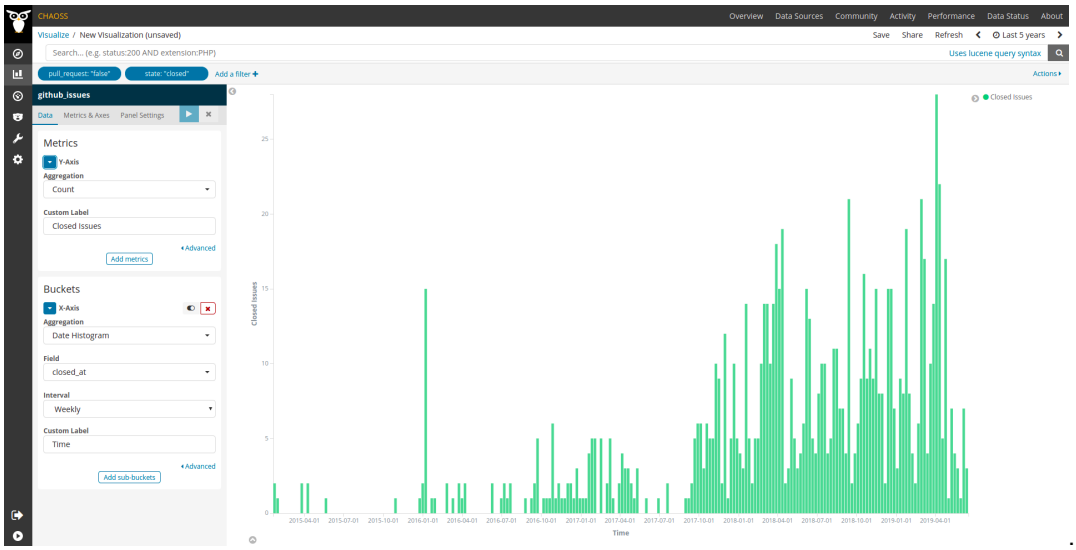
Visualizations

- Count per time period over time
- Ratio per time period over time

These could be grouped by applying the filters defined aboce. These could be represented as bar charts, with time running in the X axis.

Tools Providing the Metric

- GrimoireLab provides data for computing this metric for GitHub Issues, GitLab issues, Jira, Bugzilla and Redmine. Current dashboards show information based on creation date, that means they show current status of the issues that were created during a time period (e.g. GitHub Issues dashboard, you can see it in action). Nevertheless, it is easy to build a visualization that shows issues based on closing date by following the next steps:
  - Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
    - Create a new `Vertical Bar` chart.
    - Select the `github_issues` index.
    - Filter: `pull_request` is `false` .
    - Filter: `state` is `closed` .
    - Metrics Y-axis: `Count` Aggregation, `# Closed Issues` Custom Label.
    - Buckets X-axis: `Date Histogram` Aggregation, `closed_at` Field, `Weekly` Interval (or whatever interval may fit your needs, depending on the whole time range you wish to visualize in the chart), `Time` Custom Label.
  - Example screenshot:



Data Collection Strategies

Specific description: GitHub

In the case of GitHub, closed issues are defined as ”issues which are closed”.

Specific description: GitLab

In the case of GitLab, closed issues are defined as ”issues that are closed”.

Specific description: Jira

In the case of Jira, closed issues are defined as ”issues that change to the closed state”.

Specific description: Bugzilla

In the case of Bugzilla, closed issues are defined as ”bug reports that change to the closed state”.

References

Issue Age

Question: How long have open issues been left open?

Description

This metric is an indication of how long issues have been left open in the considered time period. If an issue has been closed but re-opened again within that period if will be considered as having remained open since its initial opening date.

Objectives

When the issue age is increasing, identify the oldest open issues in a project to gain insight as to why they have been open for an extended period of time. Additionally, to understand how well maintainers are resolving issues and how quickly issues are resolved.

Implementation

For all open issues, get the date the issue was opened and calculate the number of days to current date.

Aggregators:

- Average. Average age of all open issues.
- Median. Median age of all open issues.

Parameters:

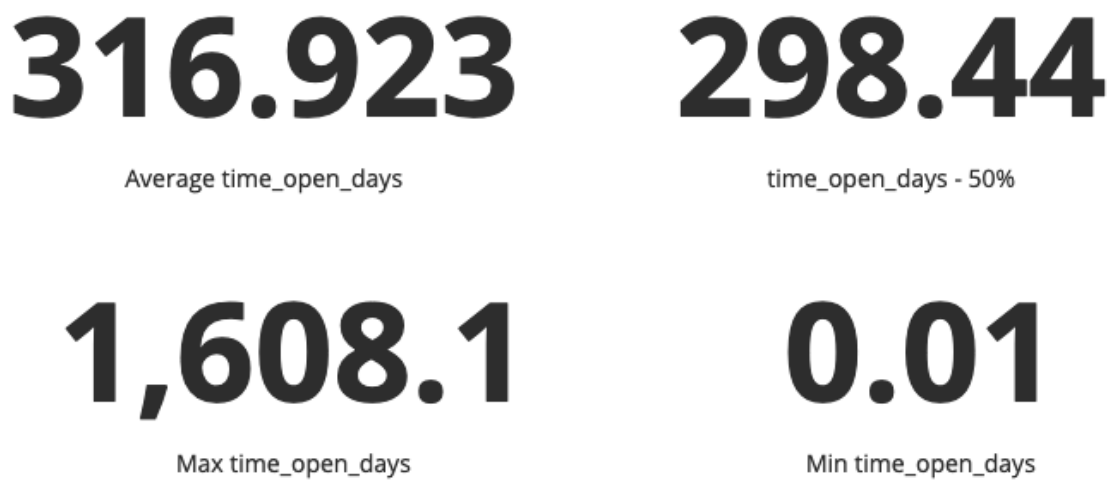
- Period of time. Start and finish date of the period during which open issues are considered. Default: forever (i.e., the entire observable period of the project's issue activity).

Filters

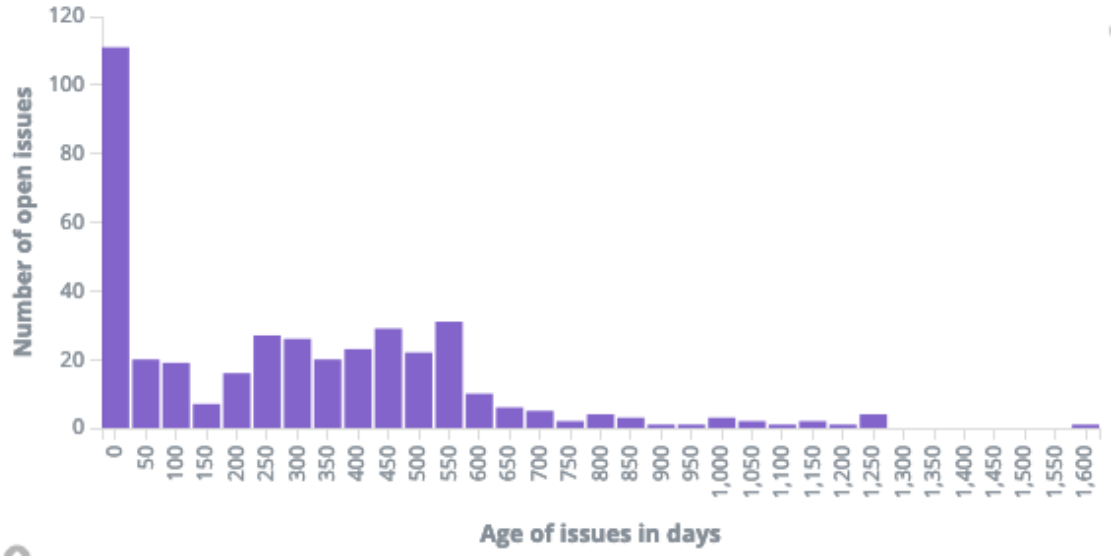
- Module or working group
- Tags/labels on issue

Visualizations

1. Summary data for open issues



2. Count of open issues per day



Tools Providing the Metric

- GrimoireLab
- Augur

Data Collection Strategies

For specific descriptions of collecting data about closed issues, please refer to the corresponding section of Issues New.

References

Issue Response Time

Question: How much time passes between the opening of an issue and a response in the issue thread from another contributor?

Description

This metric is an indication of how much time passes between the opening of an issues and a response from other contributors.

This metric is a specific case of the Time to First Reponse metric in the Common working group.

Objectives

Learn about the responsiveness of an open source community.

Implementation

Aggregators:

- Average. Average response time in days.
- Median. Median response time in days.

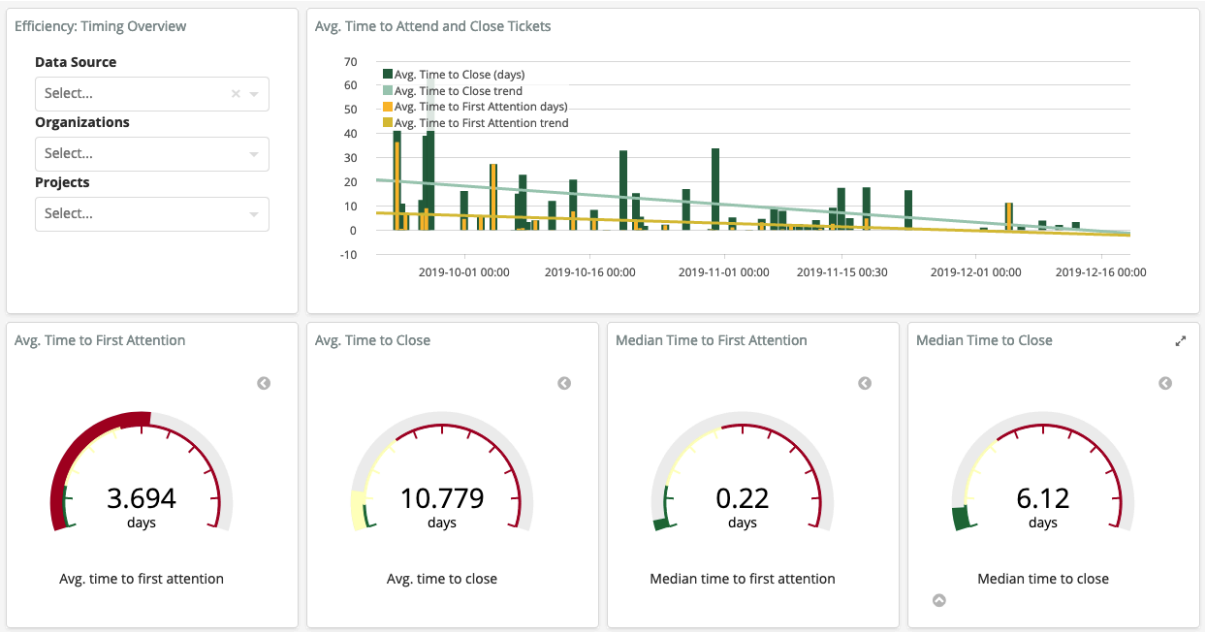
Parameters:

- Period of time. Start and finish date of the period. Default: forever.
- Period during which responses are counted.

Filters

- response from role in project (e.g., first maintainer response)
- bot versus human (e.g., filter out automated “welcome first contributor messages”)
- opened by role in project (e.g., responsiveness to new contributors versus long-timers)
- date issue was opened
- status of issue (e.g., only look at currently open issues)

Visualizations



Tools Providing the Metric

- GrimoireLab: General for any ticketing system, GitHub Issues, GitLab Issues
- Augur

Data Collection Strategies

Look at the Issues New metric for a definiton of “issues.” Subtract the issue opened timestamp from the first response timestamp. Do not count responses if created by the author of the issue.

References

Issue Resolution Duration

Question: How long does it take for an issue to be closed?

Description

This metric is an indication of how long an issue remains open, on average, before it is closed. Issues are defined as in Issues Closed.

For issues that were reopened and closed again, only the last close date is relevant for this metric.

Objectives

This metric can be used to evaluate the effort and time needed to conclude and resolve discussions. This metric can also provide insights to the level of responsiveness in a project.

Implementation

For each closed issue:

- Issue Resolution Duration = Timestamp of issue closed - Timestamp of issue opened

Aggregators:

- Average. Average amount of time (in days, by default) for an issue in the repository to be closed.

Parameters:

- Period of time. Start and finish date of the period. Default: forever.
  Period during which issues are considered.

Filters

- By time. Provides average issue resolution duration time starting from the provided beginning date to the provided end date.
  - By open time. Provides information for how long issues created from the provided beginning date to the provided end date took to be resolved.(The issue may be resolved in time later than the specified time period)
  - By closed time. Provides information for how long old issues were that were closed from the provided beginning date to the provided end date took to be resolved.(The issue may be created in time earlier than the specified time period)
- By actors (submitter, commenter, closer). Requires actor merging (merging ids corresponding to the same author).
- By groups of actors (employer, gender... for each of the actors). Requires actor grouping, and likely, actor merging.

Visualizations

- Average over time (e.g. average for January, average for February, average for March, etc.)
- Average for a given time period (e.g. average for all of 2019, or average for January to March)

Tools Providing the Metric

- Augur provides this metric as Closed Issue Resolution Duration. This metrics are available in both the `repo` and the `repo_group` metric forms - more on that in the Augur documentation.

Data Collection Strategies

For specific descriptions of collecting data about closed issues, please refer to the corresponding section of Issues Closed.

References

Focus Area - Community Growth

Goal: Identify the size of the project community and whether it is growing, shrinking, or staying the same.

Metric	Question
Contribution Attribution	Who has contributed to an open source project and what attribution information about people and organizations is assigned for contributions?
Inactive Contributors	How many Contributors have gone inactive over a specific period of time?
New Contributors	How many contributors are making their first contribution to a given project and who are they?
New Contributors Closing Issues	How many contributors are closing issues for the first time in a given project?

Contribution Attribution

Question: Who has contributed to an open source project and what attribution information about people and organizations is assigned for contributions?

Description

This metric evaluates who has worked on the project and specific project tasks and provides the attribution to project contributors and affiliated organizations. The aim is to understand, through insights into the paid contribution dynamics of a community, “how the work gets done.”

Objectives

1. Who is working on the project?
2. What is the ratio of volunteer work, sponsored work, and blended work?
3. How many contributions are sponsored?
4. Who is sponsoring the contributions?
5. What types of contributions are sponsored?
6. How diverse is the community of contributors working on a project?

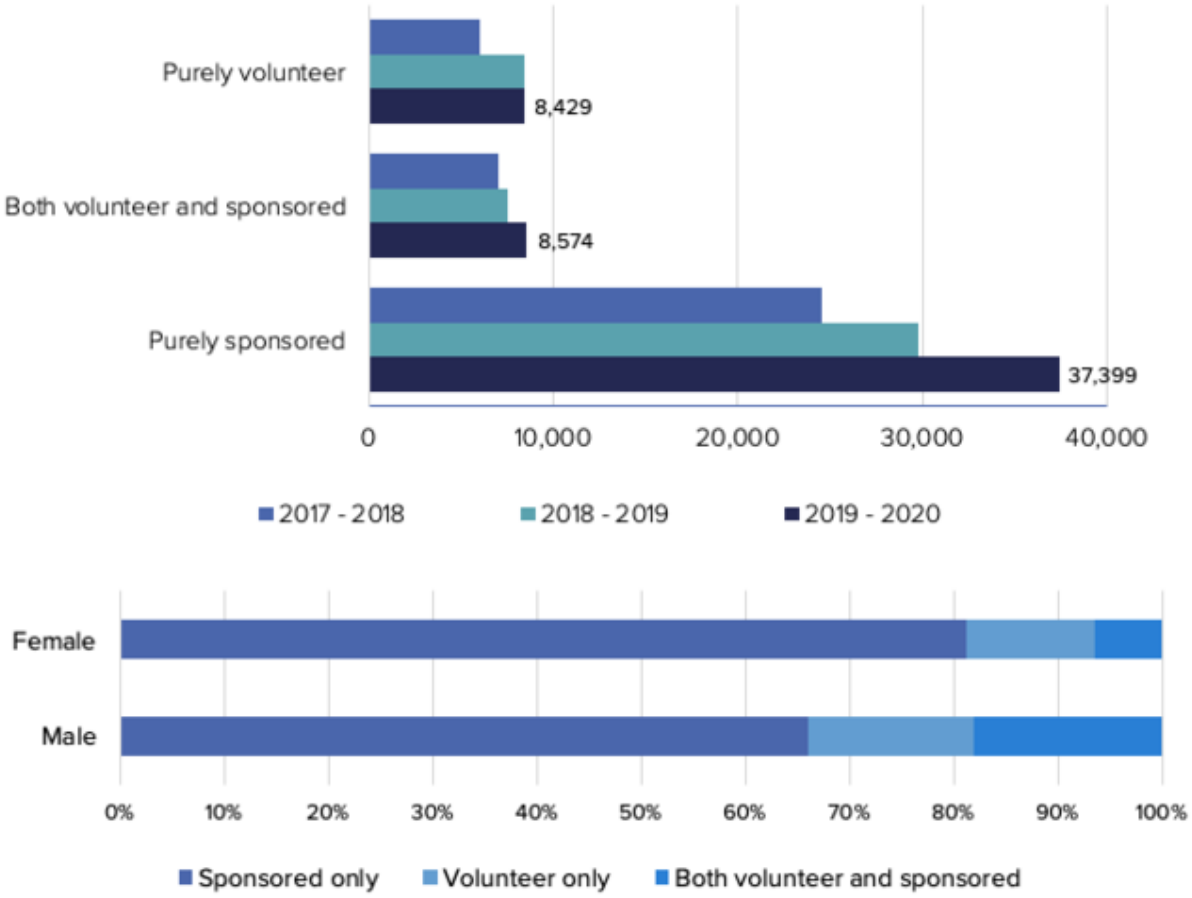
Implementation

Most contributions can be implicitly attributed using trace data, and these attributions are reflected in other metrics. However, this metric relies heavily on data that is volunteered by contributors and interpreted by project leadership. The implementation of this metric demands that the human in the loop determine what organizations, and what individual contributors a contribution is attributed to. Each individual contributor should be offered the opportunity to indicate what firm, foundation, project, and/or client paid for a particular change.

Filters

- Type of Contributor (individual, organization, gender, race, global status, work location)
    - Volunteer
    - Sponsored by a firm and/or client
    - Role that a contributor plays on a project (i.e., maintainer, board member, etc.)
- Type of Contribution
    - Links to contribution artifacts, like merge requests, issues, and the like, where relevant.
    - Indication of contribution types not managed in Git platforms like GitHub, GitLab, and BitBucket.
    - See also: https://chaoss.community/metric-types-of-contributions/
- Volunteer versus sponsored - Related to Organizational DiversityandLabor Investment

Visualizations



Tools Providing the Metric

1. The Drupal community built this tool, began using it in 2015, and has been reporting their results annually
2. There is an issue open with GitLab to implement this functionality

Data Collection Strategies

The Drupal Community implemented an example of how to gather information necessary for this metric to be calculated. It associates individuals, and organizations those individuals indicate as warranting attribution, for each discrete contribution.

Data Ethics Considerations

Although this metric requires the capture of a relationship between individuals and the contributions they make, the intent of this metric is NOT to measure individuals. The aim is to enable a wider understanding of how contributions to this project are motivated. Explicitly, it is not the intent of this metric to contribute to gamification of individual contributor work.

References

- https://dri.es/a-method-for-giving-credit-to-organizations-that-contribute-code-to-open-source
- https://www.drupal.org/blog/who-sponsors-drupal-development
- https://dri.es/who-sponsors-drupal-development-2020

- https://gitlab.com/gitlab-org/gitlab/-/issues/327138

Contributors

- Matthew Tift
- Sean Goggins
- Elizabeth Barron
- Vinod Ahuja
- Armstrong Foundjem
- Kevin Lumbard

Inactive Contributors

Question: How many Contributors have gone inactive over a specific period of time?

Description

A metric that shows how many contributors have stopped contributing over a specific period of time. The period of time required for a contributor to be counted as inactive can be decided by a variable and this metric will display the number of contributors that have been labeled as inactive over a given time frame.

Objectives

The objective is to determine how many people have stopped contributing actively. This could be useful for community managers to determine if key members are losing interest, or are taking a break.

Implementation

The metric will take in two variables - a time period and a interval. The time period will be the period over which the number of inactive members will be displayed. For example if time period=year then it will display the number of contributors that have gone inactive each year. The interval will determine how long it takes for a contributor to be labeled as inactive. If a contributor has not made a contribution for a length of time longer than the interval, they will be counted as inactive.

The metric will work by:

1. getting a list of all contributors
2. checking the last contribution date
3. if the last contribution date is before the cutoff then add them to the inactivity count of the period they last contributed in.
4. create list of inactive contributors

Aggregators:

- inactive: number of inactive contributors

Filters

- Minimum contributions required to be considered active
- Period of time to determine inactivity
- Start date/End date
- Period of graph

Data Collection Strategies

The list of contributors can be collected using the existing contributors metric. To determine the last contribution date new code may be needed.

New Contributors

Question: How many contributors are making their first contribution to a given project and who are they?

Description

An increase or decline in new contributors can be an early indicator of project health. Understanding the behavior and barriers of new community members requires knowing who they are -- helping to welcome new contributors and thank them for their efforts.

This is a specific implementation of the Contributors metric.

Objectives

An increase or decline in new contributors can be an early indicator of project health. Understanding the behavior and barriers of new community members requires knowing who they are. Welcome new contributors and thank them for their first contribution.

Implementation

For each Contributor, only consider their first contribution.

Filters

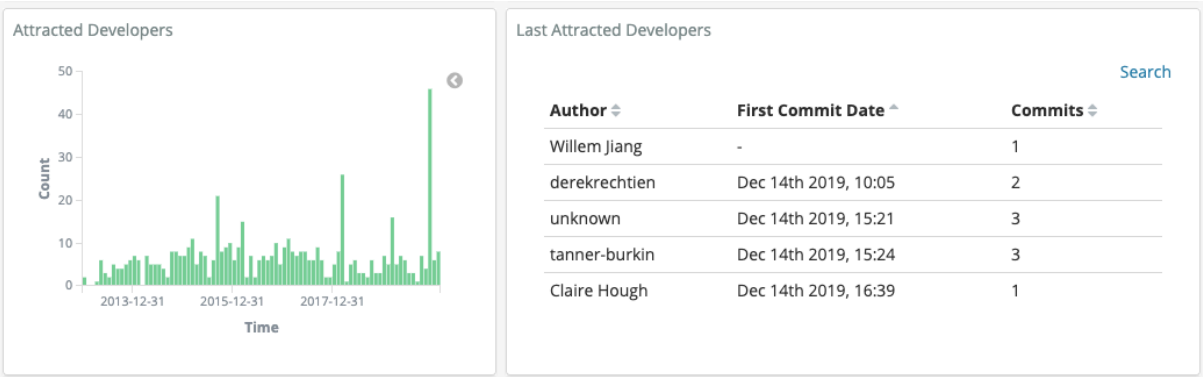
Period of Time: When was the first contribution made.

By location of engagement. For example:

- Repository authors
- Issue authors
- Code review participants
- Mailing list authors
- Event participants
- IRC authors
- Blog authors
- By release cycle
- Timeframe of activity in the project, e.g, find new contributors
- Programming languages of the project
- Role or function in project

Stage of engagement (e.g., opening pull request vs. getting it accepted).

Visualizations



Tools Providing the Metric

Augur GrimoireLab

References

https://opensource.com/article/17/4/encourage-new-contributors

New Contributors Closing Issues

Question: How many contributors are closing issues for the first time in a given project?

Description

This metric is an indication of the volume of contributors who are closing issues for their first time within a given project. When a contributor closes an issue for the first time it is some indication of ”stickiness” of the individual within that project, especially for contributors who are not also committers.

Objectives

To know how contributors are moving through the contributor funnel by identifying “closing issues” as a milestone in the contributor journey.

Implementation

Aggregators:

- Count. Total number of contributors closing issues on this project for the first time during a given time period.
- Percentage. Proportion of contributors closing issues on this project *for the first time* during a given time period, computed against *all* contributors having closed issues on this project during the same time period.

Parameters:

- Period of time. Start and finish date of the period during which new issue closers are counted. Default: forever (i.e., the entire observable project lifetime)

Filters

- Exclude reopened issues (optionally, filter if reopened in less than 1 hour)

Visualizations

- Table with names of contributors who closed an issue for the first time and when that was.
- Timeline showing the time on the x-axis, and the aggregated metric value on the y-axis for fixed consecutive periods of time (e.g. on a monthly basis). This visualisation allows to show how the metric is evolving over time for the considered project.

Data Collection Strategies

Based on the Issues Closed and Contributor definitions, enrich contributors with the date of their first time closing an issue.

References

- Contributor Funnel by Mike McQuaid

Risk WG

Focus Area	Goal
Business Risk	Understand how active a community exists around/to support a given software package.
Code Quality	Understand the quality of a given software package.
Dependency Risk Assessment	Understand software dependency risk.
Licensing	Understand the potential IP issues associated with a given software package's use.
Security	Understand security processes and procedures associated with the software's development.

Focus Area - Business Risk

Goal: Understand how active a community exists around/to support a given software package.

Metric	Question
Bus Factor	How high is the risk to a project should the most active people leave?
Committers	How robust are the contributors to a community?
Elephant Factor	What is the distribution of work in the community?

Bus Factor

Question: How high is the risk to a project should the most active people leave?

Description

The Bus Factor is a compelling metric because it visualizes the question ”how many contributors can we lose before a project stalls?” by hypothetically having these people get run over by a bus (more pleasantly, how many would have to win in a lottery and decide to move on).

The Bus Factor is the smallest number of people that make 50% of contributions.

Objectives

- Identify how widely the work in a project is distributed across contributors.
- Identify the key people in a project that are doing the majority of the work.

Implementation

The formula for Bus Factor is a percentage calculation -50% will be our threshold- followed by adding up each contributor’s contributions sorted in decreasing order until we reach the threshold.

If we have 8 contributors who each contribute the following number of contributions to a project: `1000, 202, 90, 33, 332, 343, 42, 433` , then we can determine the Bus Factor by first identifying the 50% of total contributions for all the contributors.

Summary: 50% of total contributions = `1,237.5` , so the Bus Factor is `2` .

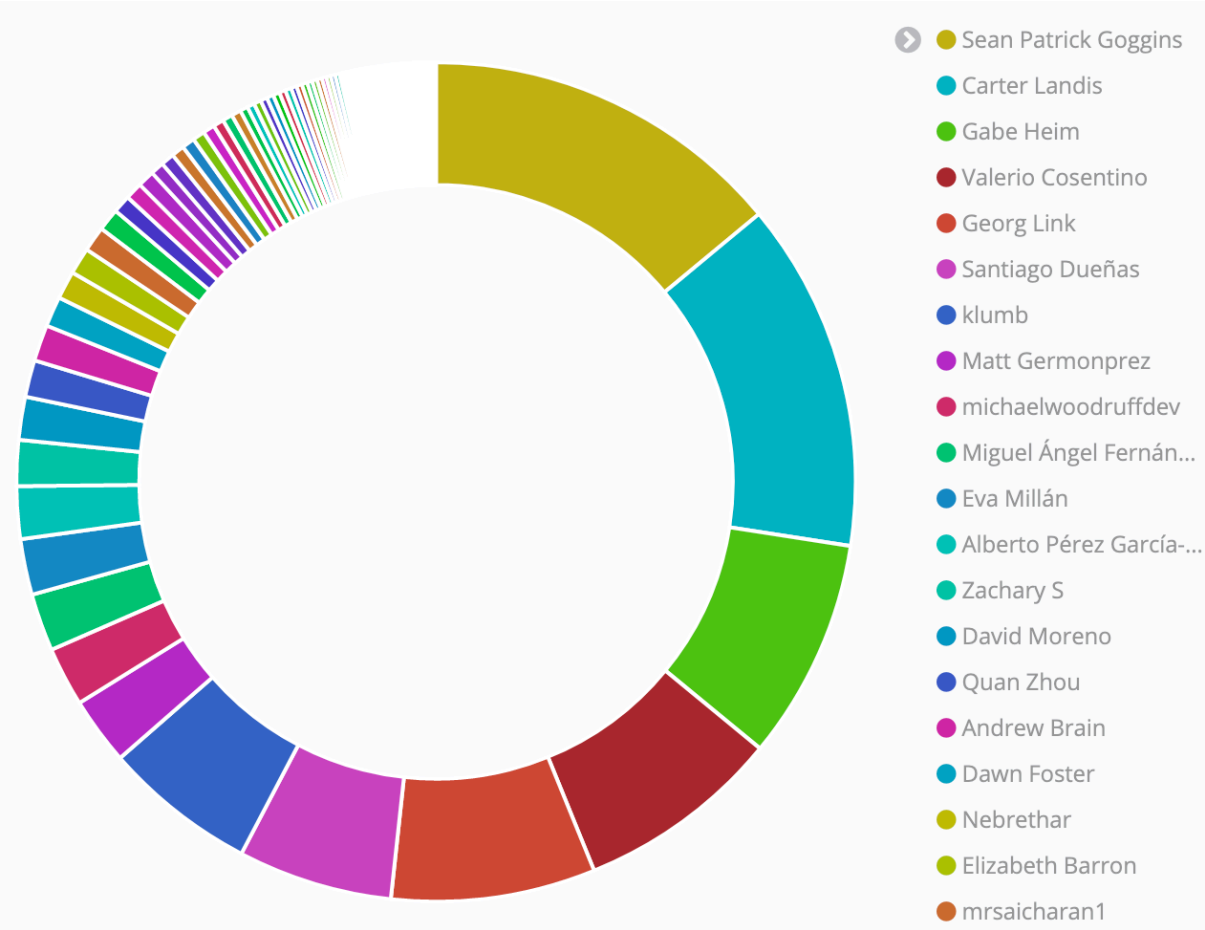
Full Solution:

1. Arrange the data in descending order: `1000, 433, 343, 332, 202, 90, 42, 33`
2. Compute the 50% of the total:
   - `(1,000 + 433 + 343 + 332 + 202 + 90 + 42 + 33)  0.5 = 1,237.5`
3. Adding up the first two contributors in our ranking we get `1,433` .
4. **Answer**: as `1,433 > 1,237.5` , more than the 50% of contributions is performed by only `2` contributors, thus the `Bus Factor = 2` .

Filters

- Time: The Bus Factor may be vary for different time periods. The Bus Factor over the life of a project may misrepresent the current level of contributor engagement in the project.
- Repository Group: Many open source projects include multiple repositories, and in some cases examining all of the repositories associated with any given project provides a more complete picture of the Bus Factor.

Visualizations (optional)



The Bus Factor for the CHAOSS Project in 2020, when considering only git commits, was 5.

Tools Providing the Metric

1. Augur
2. GrimoireLab provides this metric out of the box, not as a single number but as a visualization.

Data Collection Strategies

The data collection strategy depends on the Types of Contributions used for calculating the Bus Factor. Bus Factor is often calculated for commits but other types of contributions can be used for this as well, separately or combined.

References

Committers

Question: How robust are the contributors to a community?

Description

The Committers metric is the number of individuals who have committed code to a project. This is distinct from the more broadly construed ”Contributors” CHAOSS metric, speaking directly to the one specific concern that arises in the evaluation of risk by managers deciding which open source project to use. While not necessarily true in all cases, it is generally accepted that the more contributors there are to a project, the more likely that project is to continue to receive updates, support, and necessary resources. The metric therefore allows organizations to make an informed decision as to whether the number of committers to a given project potentially poses a current or future risk that the project may be abandoned or under-supported.

Objectives

From the point of view of managers deciding among open source projects to incorporate into their organizations, the number of committers sometimes is important. Code contributions, specifically, can vary significantly from larger scale contributor metrics (which include documentation authors, individuals who open issues, and other types of contributions), depending on the management style employed by an open source project. McDonald et al (2014) drew attention to how different open source projects are led using an open, distributed model, while others are led following highly centralized models. Fewer code contributors may indicate projects less open to outside contribution, or simply projects that have a small number of individuals who understand and contribute to the code base.

Implementation

In an open source project every individual email address that has a commit merged into the project is a ”committer” (see ”known issues” in the next section). Identifying the number of unique committers during a specific time period is helpful, and the formula for doing so is simple:

`Number_of_committers = distinct_contributor_ids (during some period of time with a begin date a`

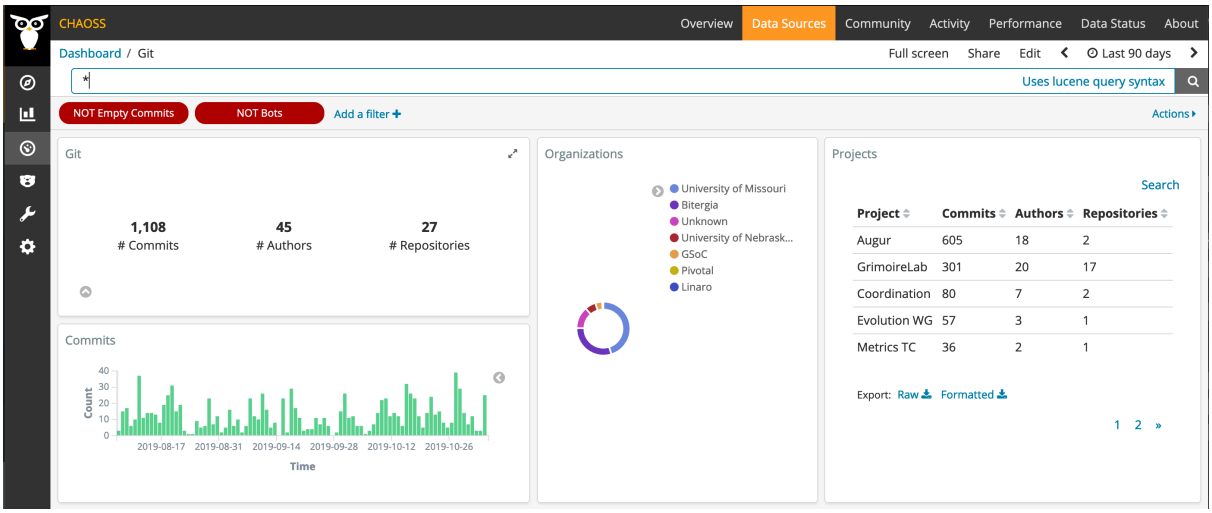
. For example, I may want to know how many distinct people have committed code to a project in the past 18 months. `Committers` reveals the answer.

Known Issues with Data Quality

- Many contributors use more than one email, which may artificially elevate the number of total committers if these shared identities are not reconciled.
- Several committers making small, ”drive by” contributions may artificially elevate this number as well.

Visualizations

From Grimoire Lab showing committers

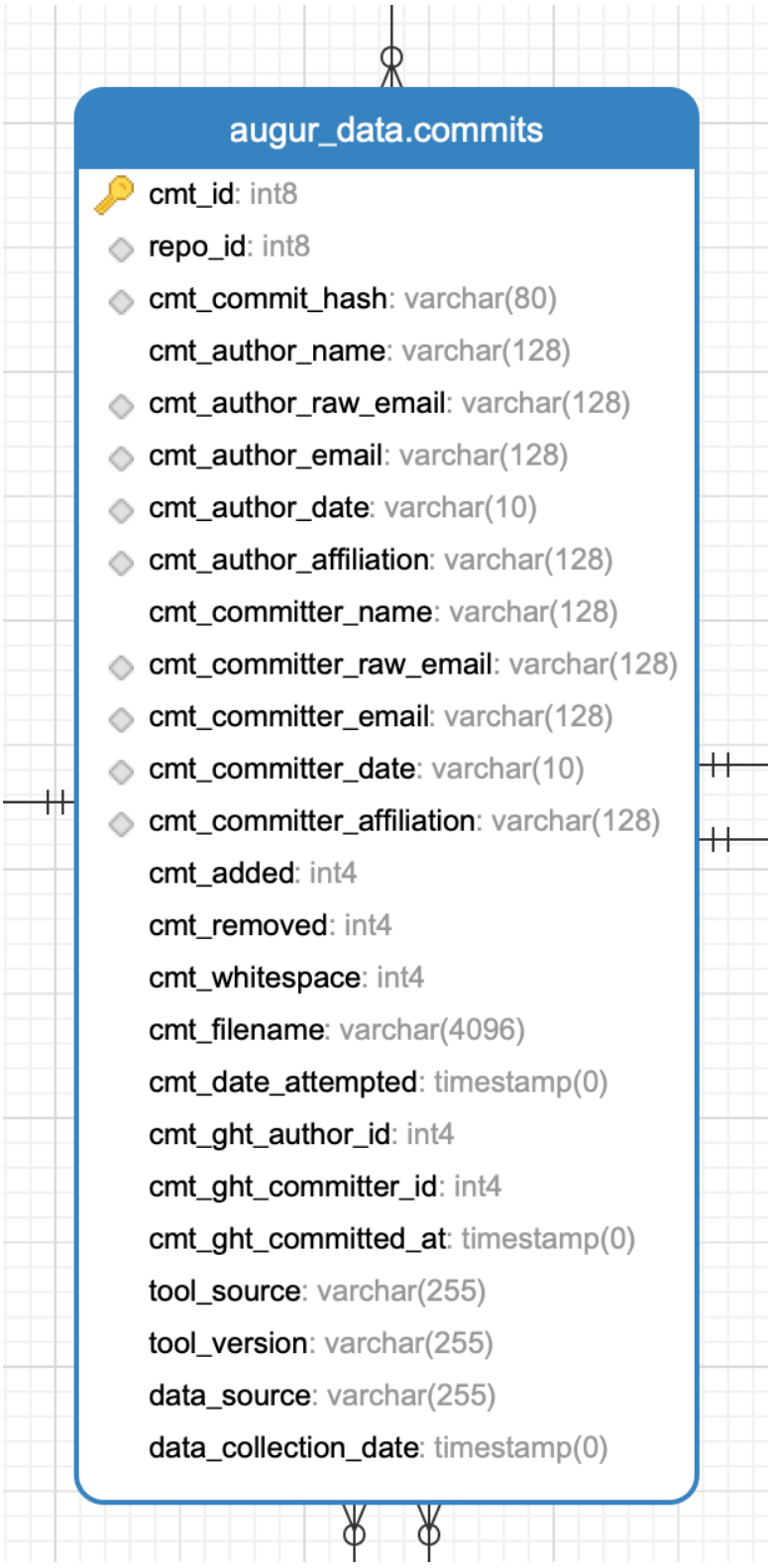


Filters

- Time: Knowing the more recent number of distinct committers may more clearly indicate the number of people engaged in a project than examining the number over a project's (repository's) lifetime.
- Commit Size: Small commits, as measured by lines of code, could be excluded to avoid a known issue
- Commit Count: Contributors with fewer than some minimum threshold of commits in a time period could be excluded from this number.

Tools Providing the Metric

Augur maintains a table for each commit record in a repository.



To evaluate distinct committers for a repository, the following SQL, or documented API endpoints can be used:

This expression allows an end user to filter by commit count thresholds easily, and the number of rows returned is the ”Total_Committers” for the repository.

Grimoire Lab additionally provides insight into committers.

References

1. Nora McDonald, Kelly Blincoe, Eva Petakovic, and Sean Goggins. 2014. Modeling Distributed Collaboration on GitHub. Advances in Complex Systems 17(7 & 8).

Elephant Factor

Question: What is the distribution of work in the community?

Description

The minimum number of companies whose employees perform a parameterizable definition of the total percentage of commits in a software repository is a project's 'elephant factor'. For example, one common filter is to say 50% of the commits are performed by `n companies` and that is the elephant factor. One would begin adding up to the parameterized percentage using the largest organizations making contributions, and then the next largest and so on. So, for example, a project with 8 contributing organizations who each contributed 12.5% of the commits in a project would, if the elephant factor is parameterized at 50%, have an elephant factor of "4". If one of those organizations was responsible for 50% of commits in the same scenario, then the elephant factor would be "1".

Elephant Factor provides an easy-to-consume indication of the minimum number of companies performing a certain percentage (i.e. 50%) of the work. The origin of the term "elephant factor" is not clearly delineated in the literature, though it may arise out of the general identification of software sustainability as a critical non-functional software requirements by Venters et al (2014).

Objectives

A company evaluating open source software products might use elephant factor to compare how dependent a project is on a small set of corporate contributors. Projects with low elephant factors are intuitively more vulnerable to decisions by one enterprise cascading through any enterprise consuming that tool. The parameterized filter should reasonably be different for a project to which 1,000 organizations contribute than one to which, perhaps 10 contribute. At some volume of organizational contribution, probably something less than 1,000 organizations, elephant factor is likely not a central consideration for software acquisition because reasonable managers will judge the project not vulnerable to the decisions of a small number of actors. Such thresholds are highly contextual.

Implementation

The formula for elephant factor is a percentage calculation -it will be our threshold- followed by adding up each company's contributions sorted in decreasing order until we reach the threshold.

If we have 8 organizations who each contribute the following number of commits to a project: `1000, 202, 90, 33, 332, , then we can determine the elephant factor by first identifying the 50% of total commits for all the companies.

Summary: 50% of total contributions = `1,237.5` , so the elephant factor is `2` .

Full Solution:

1. Arrange the data in descending order: `1000, 433, 343, 332, 202, 90, 42, 33`
2. Compute the 50% of the total:
   - `(1,000 + 433 + 343 + 332 + 202 + 90 + 42 + 33)  0.5 = 1,237.5`
3. Adding up the first two companies in our ranking we get `1,433` .
4. **Answer**: as `1,433 > 1,237.5` , more than the 50% of contributions is performed by only `2` companies, thus we can say the `elephant factor = 2` .

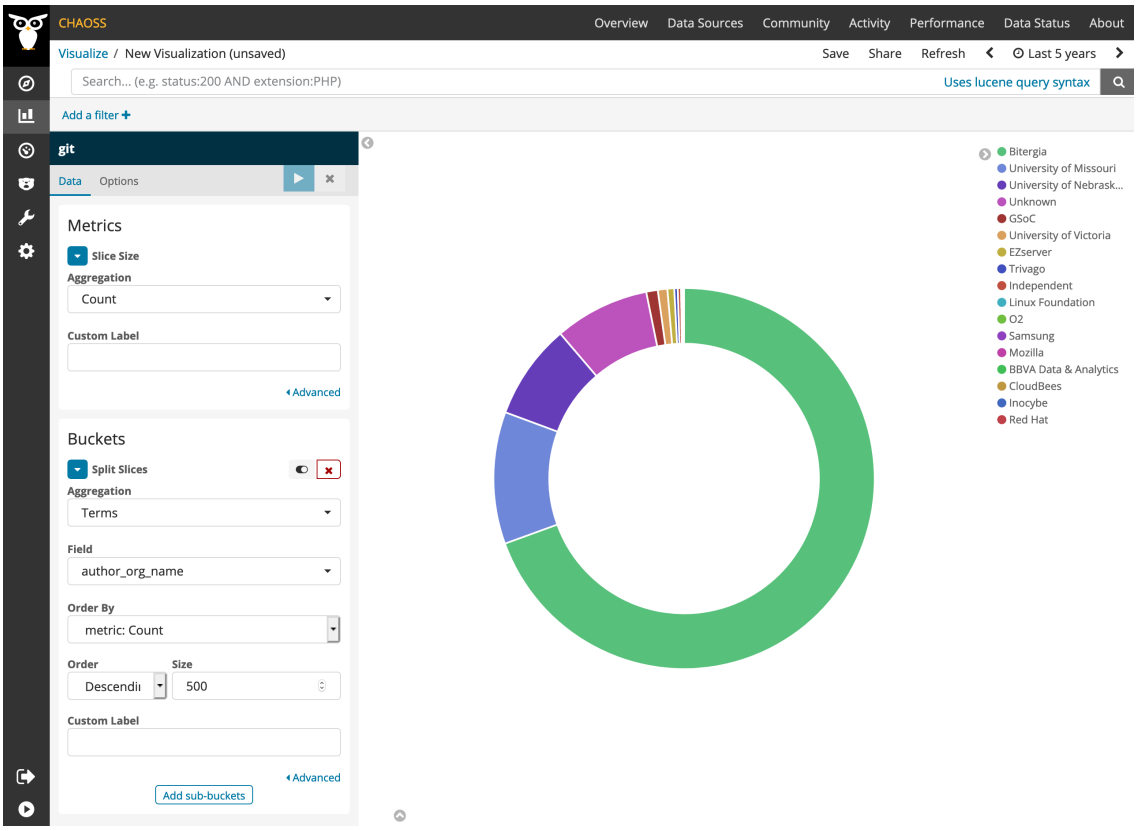
Filters

- Time: Reasonably the Elephant Factor will change if one takes a snapshot of any prior time period, so the elephant factor over the life of a product may misrepresent the current level of organizational diversity supported by the project.
- Repository Group: Many open source projects include multiple repositories, and in some cases examining all of the repositories associated with any given project provides a more complete picture of elephant factor.

Tools Providing the Metric

1. Augur
2. GrimoireLab provides this metric out of the box, not as a single number but as a visualization.

- View an example on the CHAOSS instance of Bitergia Analytics.
- Download and import a ready-to-go dashboard containing examples for this metric visualization from the GrimoireLab Sigils panel collection.
- Add a sample visualization to any GrimoreLab Kibiter dashboard following these instructions:
  - Create a new `Pie` chart
  - Select the `git` index
  - Metrics Slice Size: `Count` Aggregation
  - Buckets Splice Slices: `Terms` Aggregation, `author_org_name` Field, `metric: Count` Order By, `Descending` Order, `500` Size
- Example screenshot:



References

1. Colin C. Venters, Lydia Lau, Michael K. Griffiths, Violeta Holmes, Rupert R. Ward, Caroline Jay, Charlie E. Dibsdale, and Jie Xu. 2014. The Blind Men and the Elephant: Towards an Empirical Evaluation Framework for Software Sustainability. Journal of Open Research Software 2, 1. https://doi.org/10.5334/jors.ao
2. http://philslade.blogspot.com/2015/07/what-is-elephant-factor.html
3. https://blog.bitergia.com/2016/06/07/landing-the-new-eclipse-open-analytics-dashboard/
4. https://www.stackalytics.com/

Focus Area - Code Quality

Goal: Understand the quality of a given software package.

Metric	Question
Test Coverage	How well is the code tested?

Test Coverage

Question: How well is the code tested?

Description

Test coverage describes how much of a given code base is covered by at least one test suite. There are two principle measures of test coverage. One is the percentage of **subroutines** covered in a test suite run against a repository. The other principle expression of test coverage is the percentage of **statements** covered during the execution of a test suite. The CHAOSS metric definition for "Test Coverage" includes both of these discrete measures.

Programming languages refer to **subroutines** specifically as "functions", "methods", "routines" or, in some cases, "subprograms." The percentage of coverage for a particular repository is constrained in this definition to methods defined within a specific repository, and does not include coverage of libraries or other software upon which the repository is dependent.

Objectives

Understanding the level of test coverage is a signal of software quality. Code with little test coverage signals a likelihood of less rigorous software engineering practice and a corresponding increased probability that defects will be detected upon deployment and use.

Implementation

Statements include variable assignments, loop declarations, calls to system functions, "go to" statements, and the common `return` statement at the completion of a function or method, which may or may not include the return of a `value` or `array of values` .

Subroutine Coverage

$$subroutine{-}coverage{-}percentage = \frac{subroutines - tested}{total - subroutines - in - repository} * 100$$

Statement Coverage

$$statement{-}coverage{-}percentage = \frac{statements - excecuted - in - testing}{total - statements - in - repository} * 100$$

Filters

- Time: Changes in test coverage over time provide evidence of project attention to maximizing overall test coverage. Specific parameters include `start date` and `end date` for the time period.
- Code_File: Each repository contains a number of files containing code. Filtering coverage by specific file provides a more granular view of test coverage. Some functions or statements may lead to more severe software failures than others. For example, untested code in the `fail safe` functions of a safety critical system are more important to test than `font color` function testing.
- Programming_Language: Most contemporary open source software repositories contain several different programming languages. The coverage percentage of each `Code_File`

Tools Providing the Metric

- Providing Test Coverage Information
  - Python's primary testing framework is PyTest
  - The Flask web framework for python enables coverage testing
  - Open source code coverage tools for common languages like Java, C, and C++ are available from my sites, including this one.
- Storing Test Coverage Information
  - Augur has test coverage implemented as a table that is a child of the main repository table in its repository. Each time test coverage is tested, a record is made for each file tested, the testing tool used for testing and the number of statements/subroutines in the file, as well as the number of statements and subroutines tested. By recording test data at this level of granularity, Augur enables `Code_File` and `Programming_Language` summary level statistics and filters.

References

1. J.H. Andrews, L.C. Briand, Y. Labiche, and A.S. Namin. 2006. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. IEEE Transactions on Software Engineering 32, 8: 608–624. https://doi.org/10.1109/TSE.2006.83
2. Phyllis G Frankl and Oleg Iakounenko. 1998. Further Empirical Studies of Test Effectiveness. In Proceedings of the 6thACM SIGSOFT International Symposium on Foundations of Software Engineering, 153–162.
3. Phyllis G Frankl and Stewart N Weiss. 1993. An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing. EEE Transactions on SoftwareEngineering 19, 8: 774–787.
4. Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, 435–445. https://doi.org/10.1145/2568225.2568271
5. Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In Proceedings of the eighteenth international symposium on Software testing and analysis - ISSTA '09, 57. https://doi.org/10.1145/1572272.1572280

Focus Area - Dependency Risk Assessment

Goal: Understand software dependency risk.

Metric	Question
Libyears	What is the age of the project's dependencies compared to current stable releases?
Upstream Code Dependencies	What projects and libraries does my project depend on?

Libyears

Question: What is the age of the project's dependencies compared to current stable releases?

Description

Libyears explain the age of dependencies upon which a project (repository, or repositories of buildable software) relies, compared to the current stable releases of those dependencies. A dependency's "current stable" release is a release intended for general use, and does not include candidate / draft releases. Age is cumulative across all dependencies for a project, and can be ascertained in multiple ways like reporting total age, average age, and possibly median age (note that median age can hide issues if there is a long tail of ages). In general, a lower Libyear number is better. Tools should also provide a list of dependencies sorted by age to provide more insight. The concept of measuring this was proposed in "Measuring Dependency Freshness in Software Systems" by Cox et al [Cox 2015].

Note: In some cases, using an older version instead of a current version may be the wiser choice; that said, this metric provides insight to help others identify where those older versions are in use.

Objectives

The objective of the Libyears metric is to assist in the identification of dependencies that have a higher probability of posing stability, security, and vulnerability risks to a project. A long-obsolete component is more likely to have publicly-known vulnerabilities, and is also less likely to be well-supported. It is a useful initial filter for sorting dependencies to help identify the dependencies most warranting closer examination on a project.

Implementation

Parameters

By default this information will be within an ecosystem (e.g., JavaScript or Maven), as that is easier to compute. This metric can be computed across multiple ecosystems, but that tends to require more information, e.g., a cross-ecosystem Software Bill of Materials (SBOM).

This information could only consider direct dependencies, or it could also include all transitive dependencies. It is more likely to reveal potential problems if transitive dependencies are included, but this is not supported by all such tools.

If a project has multiple stable/supported branches, is it acceptable to consider "current" in any branch? By default, only the most recent stable branch is considered, as typically over time the older branches receive less maintenance. An alternative (though not the default) is to also accept older versions as "current" if that older version is actively supported; reports must clearly note when this alternative is used. Should there be a grace period when a new dependency is still considered "current"? By default, the answer is no; any particular grace period is arbitrary, and the point is to try to stay current

Filters (optional)

Dependency level (direct only, includes transitive dependencies, etc. as defined in the [Upstream Code Dependencies]((https://github.com/chaoss/wg-risk/blob/master/focus-areas/dependency-risk-assessment/upstream-code-dependencies.md) metric.

- Cumulative Libyears as a total age.
- Average age of dependencies
- Median age of dependencies

- Sorted list of dependencies (sorted oldest-first) so the “most risky due to age” dependencies are identified first

Visualizations (optional)

This is an example of Libyear as a cumulative measure of Libyears for direct dependencies, in this case with a cumulative value of 103.78 cumulative libyears.

```
libyear -r requirements.txt
+--------------------------+-----------------+----------------+-----------------+
|         Library          | Current Version | Latest Version | Libyears behind |
+--------------------------+-----------------+----------------+-----------------+
|           pytz           |      2015.2     |     2019.3     |       4.54      |
|         urllib3          |      1.15.1     |     1.25.7     |       3.58      |
|         astroid          |      1.5.3      |     2.3.3      |       2.43      |
|          django          |     1.11.23     |      3.0       |       0.34      |
|      django-celery       |      3.2.1      |     3.3.1      |       2.54      |
|        httpretty         |      0.8.3      |     0.9.7      |       5.31      |
|         Pygments         |       1.6       |     2.5.2      |       6.81      |
|          flake8          |      3.6.0      |     3.7.9      |       1.01      |
|      django-waffle       |      0.14.0     |     0.18.0     |       1.66      |
|    requests_oauthlib     |      0.8.0      |     1.3.0      |       2.72      |
|   django-debug-toolbar   |       1.8       |      2.1       |       2.52      |
|         libsass          |      0.13.3     |     0.19.4     |       2.06      |
|     django-storages      |      1.6.6      |      1.8       |       1.65      |
|      edx-i18n-tools      |      0.4.2      |     0.5.0      |       2.02      |
|           six            |      1.10.0     |     1.13.0     |       4.08      |
|   djangorestframework    |      3.6.3      |     3.11.0     |       2.58      |
|          isort           |      4.2.15     |     4.3.21     |       2.05      |
|         futures          |      2.1.6      |     3.3.0      |       5.5       |
|          Pillow          |      2.7.0      |     6.2.1      |       4.8       |
| edx-django-release-util  |      0.3.1      |     0.3.2      |       2.44      |
|      beautifulsoup4      |      4.6.0      |     4.8.1      |       2.42      |
|       mysqlclient        |   1.4.2.post1   |     1.4.6      |       0.77      |
|         newrelic         |    4.14.0.115   |   5.4.0.132    |       0.78      |
|          redis           |      2.10.6     |     3.3.11     |       2.16      |
|         oauthlib         |      2.1.0      |     3.1.0      |       1.21      |
|        django-ses        |      0.7.1      |     0.8.13     |       3.65      |
|           mock           |      1.3.0      |     3.0.5      |       3.79      |
|      django-hamlpy       |      1.1.1      |      1.2       |       1.52      |
|          bottle          |      0.12.9     |    0.12.18     |       4.1       |
|      pylint-django       |      0.7.2      |     2.0.13     |       3.44      |
|       user-agents        |      1.1.0      |      2.0       |       2.13      |
|          jsmin           |      2.2.1      |     2.2.2      |       1.15      |
|         Markdown         |       2.4       |     3.1.1      |       5.26      |
|         gunicorn         |      0.17.4     |     20.0.4     |       6.59      |
|         requests         |      2.18.4     |     2.22.0     |       1.75      |
|          pylint          |      1.7.2      |     2.4.4      |       2.39      |
+--------------------------+-----------------+----------------+-----------------+
Your system is 103.78 libyears behind
```

This image is source from https://github.com/nasirhjafri/libyear

Tools Providing the Metric (optional)

Note that some tools can also compute differences between version ids (e.g., 1.1.1 vs. 1.2.3); this can be informative, but not all dependencies use the same version numbering approaches, so for simplicity we are focusing on measuring time.

Here is an example of some tools that implement libyersars:

- https://github.com/nasirhjafri/libyear - for Python where a requirements.txt file is used.
- https://github.com/sesh/piprot
- https://github.com/chaoss/augur - deps_libyear_worker - For Python, and Javascript right now.
- https://github.com/jaredbeck/libyear-bundler - for Ruby where Gemfile is used

References

- [Cox 2015] “Measuring Dependency Freshness in Software Systems” by Joel Cox, Eric Bouwers, Marko van Eekelen, and Joost Visser, https://ericbouwers.github.io/papers/icse15.pdf
- https://libyear.com/

Contributors

- Sophia Vargas (Google)
- David A. Wheeler (Linux Foundation)
- Vinod Ahuja (University of Nebraska Omaha)
- Kate Stewart (Linux Foundation)
- Duane O’Brien
- Sean Goggins (University of Missouri / CHAOSS Project)

Upstream Code Dependencies

Question: What projects and libraries does my project depend on?

Description

The aim of this metric is to understand the number of code based dependencies embedded within a piece of open source software. This metric explicitly excludes infrastructure focused dependencies like databases and operating systems, which will be developed as a distinct metric. By extension, awareness of Upstream Code Dependencies enables a project to evaluate the health and sustainability of each dependency, using other CHAOSS metrics.

Objectives

The Upstream Code Dependency metric is aimed at understanding the code based dependencies which are required to build, test, or run a piece of software. The Upstream Code Dependency metric can help identify what projects, libraries, or versions my project directly or transitively depend on.

Implementation

Upstream Code Dependency metric can be implemented by analyzing project's dependency file or by using existing tools that examine package manager data for the languages in use (e.g., package.json for JavaScript npm, pyproject.toml / requirements.txt for Python, Gemfile / Gemfile.lock for Ruby, etc.). Note: C/C++ generally use system package managers. Things get more complex with multiple languages, insofar as several language specific dependency files will need to be scanned.

Parameters

All enumerated dependencies should include the specific version(s) that are used for each dependency. Note that some systems do not support, or do not use, "version pinning" and thus do not enforce a specific version.

- Depth of Dependency Tree
    - Direct Dependency - first order dependencies, as declared in the source code and/or package manager configuration (e.g., requirements.txt, Gemfile, etc.)
    - Transitive Dependency - indirect dependencies, that is, dependencies beyond first order dependencies also referred to as nested or second order dependencies. For example project A under evaluation is dependent on project B and project B is dependent on Project C. For project A, project C is a transitive dependency.
    - Circular Dependency - dependencies where if traced eventually lead back to themselves. In systems that allow circular dependencies, we assume that a given dependency is only counted once in this case.
- Dependency State
    - Static Dependency - Dependency is present in all the cases.
    - Dynamic Dependency - Dependency changes in usage and in other contexts
- Dependency on external service like use of API
- Execution Dependency - dependencies required to execute the software. Note that certain kinds of dependencies are typically excluded from counts, as described below. These may be one or more of the following:

    - Build Dependency - Code require to build a piece of software
    - Test Dependency - Code require to test a piece of software
    - Runtime Dependency - Code require to run a piece of software
- Language runtime dependency detail (i.e., Python's runtime environment)? (default no). These details are provided because of the importance of runtime dependencies for quality assurance in safety critical systems.
    - Often which language runtime will be used is controlled by virtual environments , e.g., venv in Python ; in Ruby you'd often use rbenv or rvm to implement (& typically included in "Gemfile" or "Gemfile.lock" and .ruby-version)
    - PyPi is steadily increasing its "refusal to compile incompatible libraries/dependencies logic. It's starting to "break builds".
    - Unfortunately not all packaging systems have a convention for recording version information of all transitive dependencies, even within their ecosystem (it should in the long run)
    - In some systems there are many possible runtimes that might be hard to distinguish. (E.g., there are many implementations of Common Lisp & often any of them would work.)
- Language's built-in libraries in count (e.g., "re" in Python)? (default no)
    - Typically many built-in libraries are executable dependencies. However, they are typically installed "in mass" by selecting the language implementation, and are often excluded from counts to simplify analysis.
    - Example: By default, `pip freeze` does not include these types of "included with the language" libraries/dependencies.
- Multiple versions of the same dependency are counted independently. Some systems support multiple versions of the same dependency within a system; in such cases, they are counted separately.

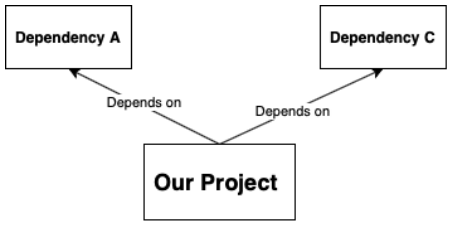
Note: It is often important to provide information on the language implementation major and minor release version at runtime.

- Some counts and analysis needs this information. Often language runtimes and built-in libraries are omitted (see above), and this information serves as a shorthand to provide this additional information.
- Example: The Ruby ecosystem supports the specification of the language runtime version in Gemfiles & a .ruby-version file.
- Example: Python releases from PyPi and Anaconda often curate different versions of libraries in different ways.

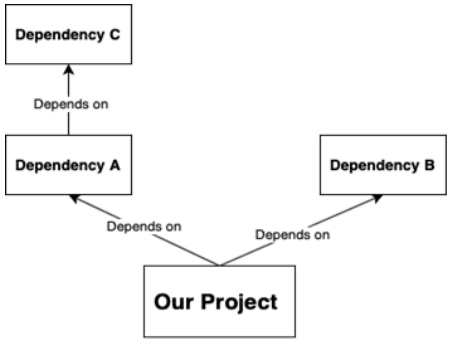
Filters

- Trends over time (e.g., am I depending on more or fewer projects than last year)
- Number of versions for each dependency
- Number of references to the same dependency

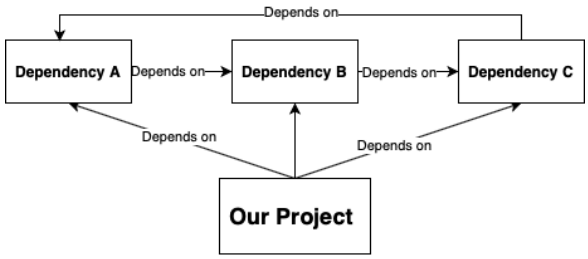
Visualizations



Direct Dependencies



Transitive Dependencies



Circular Dependencies

Tools Providing the Metric

- deps.dev
- Deps.cloud
- OWASP
- Libraries.io
- BackYourStack.com
- Software bill of materials
- PackagePhobia
- Augur

Data Collection Strategies (optional)

- Augur has an implementation of code scanning, and package manager scanning dependency identification.
- Libraries.io provides a package manager focused dependency scanner (also available through Tidelift).

Contributors

- Georg Link
- Matt Germonprez
- Sean Goggins
- Sophia Vargas
- Kate Stewart
- Vinod Ahuja
- David A. Wheeler
- Arfon Smith
- Elizabeth Barron
- Ritik Malik
- Dhruv Sachdev
- Daune O’Brien
- Michael Scovetta

Focus Area - Licensing

Goal: Understand the potential IP issues associated with a given software package's use.

Metric	Question
License Coverage	How much of the code base has declared licenses?
License Declared	What are the declared software package licenses?
OSI Approved Licenses	What percentage of a project's licenses are OSI approved open source licenses?
SPDX Document	Does the software package have an associated SPDX document as a standard expression of dependencies, licensing, and security-related issues?

License Coverage

Question: How much of the code base has declared licenses?

Description

How much of the code base has declared licenses that scanners can recognize which may not be just OSI-approved. This includes both software and documentation source files and is represented as a percentage of total coverage.

Objectives

License Coverage provides insight into the percentage of files in a software package that have a declared license, leading to two use cases:

1. A software package is sourced for internal organizational use and declared license coverage can highlight points of interest or concern when using that software package.
2. Further, a software package is provided to external, downstream projects and declared license coverage can make transparent license information needed for downstream integration, deployment, and use.

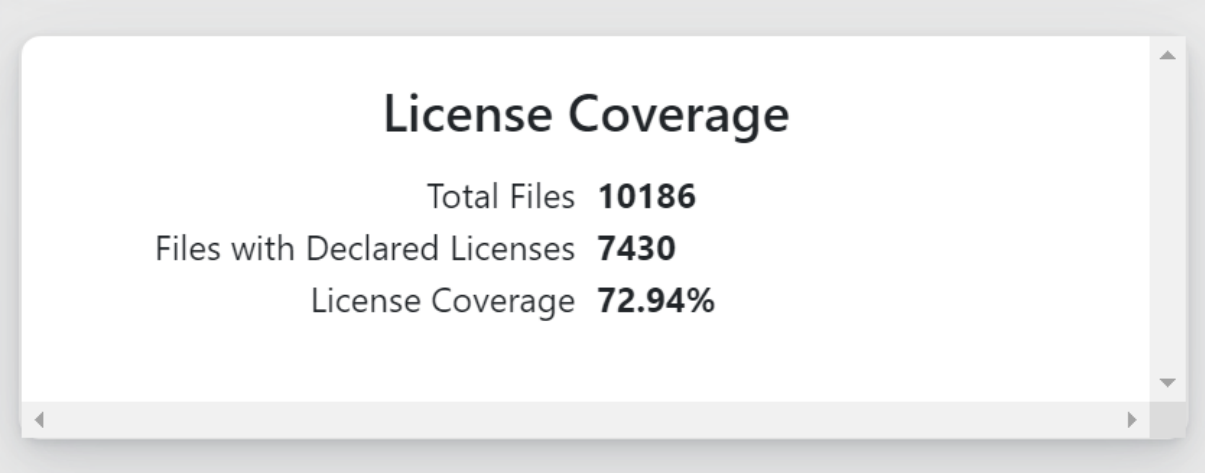
Implementation

Filters

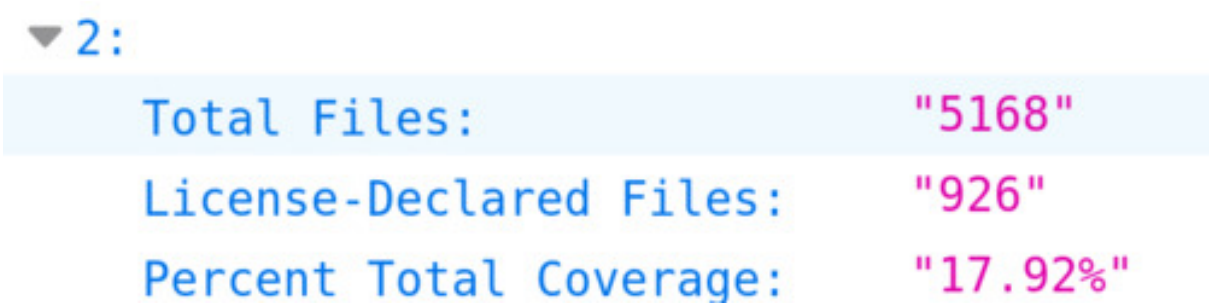
Time: Licenses declared in a repository can change over time as the dependencies of the repository change. One of the principle motivations for tracking license presence, aside from basic awareness, is to draw attention to any unexpected new license introduction.

Visualizations

Web Presentation of Augur Output:



JSON Presentation of Augur Output:



Tools providing the Metric

1. Augur

Data can be pulled and filtered to get the desired information. License Coverage data can be found on any Augur risk page

References

- https://spdx.org/
- https://www.fossology.org

License Declared

Question: What are the declared software package licenses?

Description

The total number and specific licenses declared in a software package. This can include both software and documentation source files. This metric is an enumeration of licenses, and the number of files with that particular license declaration. For Example:

SPDX License Type	Number of Files with License
MIT	44
AGPL	2
Apache	88

Objectives

The total number and specific licenses declared is critical in several cases:

1. A software package invariability carries for multiple software licenses and it is critical in the acquisition of software packages to be aware of the declared licenses for compliance reasons. Licenses Declared can provide transparency for license compliance efforts.
2. Licenses can create conflicts such that not all obligations can be fulfilled across all licenses in a software package. Licenses Declared can provide transparency on potential license conflicts present in software packages.

Implementation

Filters

- Time: Licenses declared in a repository can change over time as the dependencies of the repository change. One of the principle motivations for tracking license presence, aside from basic awareness, is to draw attention to any unexpected new license introduction.
- Declared and Undeclared: Separate enumeration of files that have license declarations and files that do not.

Tools Providing the Metric

1. Augur

Licenses Declared can be found on any Augur risk page under the section ”License Declared”.

1. Augur-SPDX

The Augur-SPDX package is implemented as an Augur Plugin, and uses this data model for storing file level license information. Specifically:

- Each `package` (repository) can have a declared and an undeclared license, as determined by the scan of all the files in the repository.

- Each `package` can also have a number of different non-code `documents`, which are SPDX license declarations.
- Each `file` can be associated with one or more `packages_files`. Through the relationship between `files` and `packages_files`, Augur-SPDX allows for the possibility that one file in a large collection of repositories could be part of more than one package, although in practice this seems unlikely.
- `packages` and `packages_files` have a one to many relationship in both directions. Essentially, this is a reinforcement of the possibility that each `file` can be part of more than one `package`, though it is, again, typical that each `package` will contain many `package_files`.
- `licenses` are associated with `files` and `packages_files`. Each `file` could possibly have more than one `licenses` reference, which is possible under the condition that the `license` declaration changed between `Augur-SPDX` scans of the repository. Each `package` is stored in its most recent form, and each `packages_file` can have one `license` declaration.

References

- https://spdx.org/
- https://www.fossology.org

OSI Approved Licenses

Question: What percentage of a project's licenses are OSI approved open source licenses?

Description

This metric provides transparency on the open source licenses used within a project. The reason for the needed transparency is that a number of licenses are being propagated that are not, in fact, open source friendly. Open source projects may not want to include any licenses that are not OSI-approved.

As from OSI: "Open source licenses are licenses that comply with the Open Source Definition — in brief, they allow the software to be freely used, modified, and shared. To be approved by the Open Source Initiative (also known as the OSI), a license must go through the Open Source Initiative's license review process."

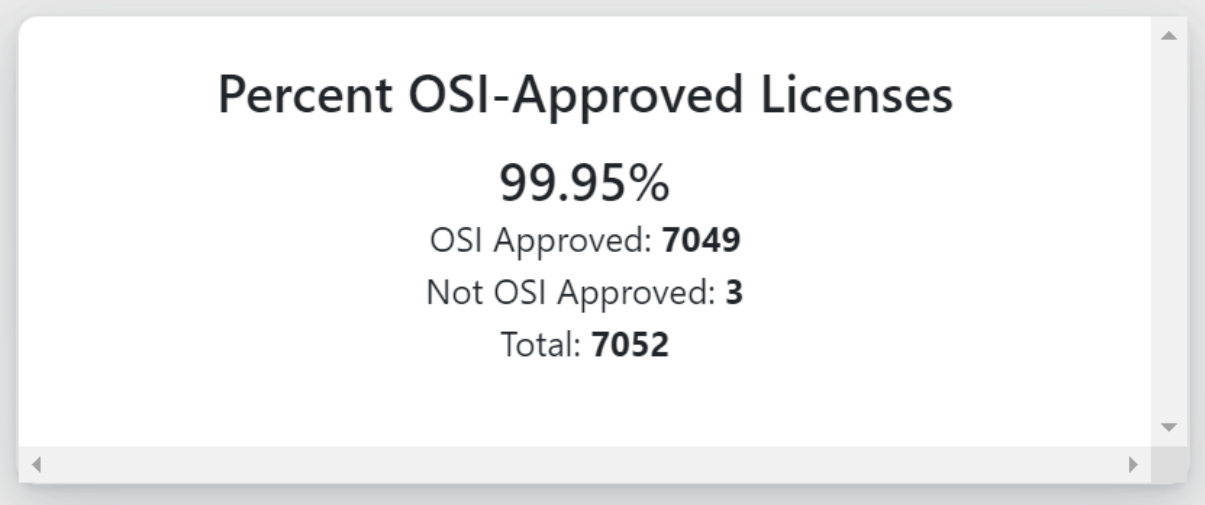
Objectives

Identify whether a project has licenses present which do not conform to the OSI definition of open source. This transparency helps projects make conscious decisions about whether or not to include licenses that are not approved by OSI.

Implementation

The OSI-approved licenses can be found in the SPDX-provided Licenses.json.

Visualizations



Tools Providing Metric

Augur provides this metric under the Risk page for a project.

Example: http://augur.osshealth.io/repo/Zephyr-RTOS/zephyr/risk

Data Collection Strategies

- Extract list of licenses from a code base, same as in License Coverage metric.
- Compare the list of licenses to Licenses.json and take note of how many licenses are approved by the OSI.
- Calculate the percentage of files with an OSI approved license.

Resources

- OSI license page
- SPDX License List

SPDX Document

Question: Does the software package have an associated SPDX document as a standard expression of dependencies, licensing, and security-related issues?

Description

A software package has an associated SPDX document as a standard expression of dependencies, licensing, and security-related issues. More information on the SPDX specification can be found at: https://spdx.org/

Objectives

For managers acquiring open source software as part of an IT or Open Source Program Office portfolio, an SPDX document provides an increasingly essential core piece of management information. This arises because, as software packages exist in complex software supply chains, it is important to clearly express, in a standardized way, the associated dependencies, licenses, and security-related issues with that software package. An SPDX document provides a single source of information both for internal use and downstream distribution of software packages. An SPDX document assists in how organizations routinize open source work to better integrate with their own open source risk management routines.

Implementation

Filters

augur-SPDX was used to scan the GitHub repository Zephyr. Here are the licenses identified from the scan in JSON format:

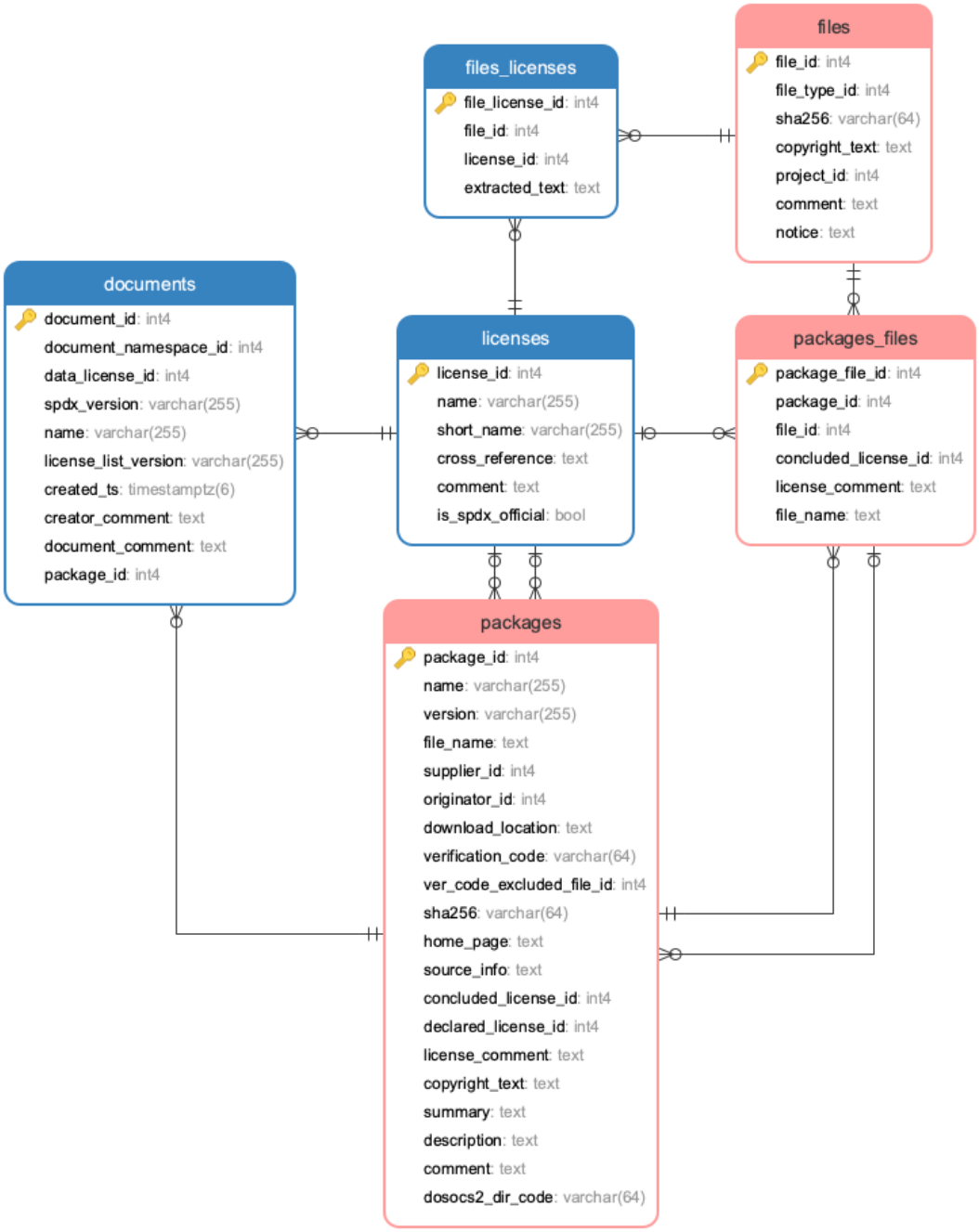

```
{
  "0": "Apache-2.0",
  "1": "BSD-2-Clause",
  "2": "BSD-3-Clause",
  "3": "GPL-2.0",
  "4": "GPL-2.0+",
  "5": "GPL-3.0+",
  "6": "ISC",
  "7": "MIT"
  "8": "BSD-4-Clause-UC",
  "9": "CC0-1.0"
}
```


This document was generated by Augur.

Tools Providing the Metric

- DoSOCSv2 embedded as an Augur Service. A file by file SPDX document is available with Augur configured using the DoSOCSv2 plugin. The relevant parts of the database schema are illustrated below.
- Augur-SPDX embedded as an Augur Service. A file by file SPDX document is available with Augur configured using the augur-spdx plugin, which is derived from DOSOCS. The relevant parts of the database schema are illustrated below. This implementation is a fork of DoSOCSv2.
- Packages

- Package_Files
- Files (which may be, but are unlikely to be also included in other packages). License information is included as part of an SBOM, but the complexity of license identification is clarified in the License_Count, License_Coverage, and License_Declared metrics.



References

- https://spdx.org
- https://www.ntia.doc.gov/SoftwareTransparency

Focus Area - Security

Goal: Understand security processes and procedures associated with the software's development.

Metric	Question
Core Infrastructure Initiative Best Practices Badge	What is the current CII Best Practices status for the project?

Core Infrastructure Initiative Best Practices Badge

Question: What is the current CII Best Practices status for the project?

Description

As from the CII Best Practices Badging Page: The Linux Foundation Core Infrastructure Initiative (CII) Best Practices badge is a way for open source projects to show that they follow best practices. Projects can voluntarily self-certify, at no cost, by using this web application to explain how they follow each best practice. Projects receive the passing badge if they meet all of the required criteria.

Objectives

As from the CII Best Practices Badging Page: CII badging indicates a project's level of compliance with "open source project best practices" as defined by the Linux Foundation's core infrastructure initiative, which focuses on CyberSecurity in open source software. The goal of the program is to encourage projects to produce better more secure software and allow others to determine if a project is following best practices.

Consumers of the badge can quickly assess which open source projects are following best practices and as a result are more likely to produce higher-quality secure software.

Implementation

See CII's API documentation for relevant information.

Visualizations

Tools Providing Metric

Augur provides an example implementation for the CII Best Practices metric. An example of CII metrics in use can be found at http://augur.osshealth.io/repo/Zephyr-RTOS/zephyr/risk

Data Collection Strategies

See CII's API documentation for relevant information.

As from the CII Best Practices Badging Page: Projects receive the passing badge if they meet all of the required criteria. The status of each criterion, for a given project, can be 'Met', 'Unmet', 'N/A' or 'unknown'. Each criterion is in one of four categories: 'MUST', 'SHOULD', 'SUGGESTED', or 'FUTURE'. To obtain a badge, all the MUST and MUST NOT criteria must be met, all SHOULD criteria must be met OR the rationale for not implementing the criterion must be documented, and all SUGGESTED criteria have to be rated as met or unmet. Advanced badge levels of silver and gold are available if the project satisfies the additional criterion.

References

- CII Badging Website: https://bestpractices.coreinfrastructure.org/en
- Augur: https://github.com/chaoss/augur

Value WG

Focus Area	Goal
Academic Value	Identify the degree to which a project is valuable to researchers and academic institutions.
Communal Value	Identify if a project is valuable to its community of users (including downstream projects) or contributors.
Individual Value	Identify if a project is valuable to me as an individual user or contributor.
Organizational Value	Identify if a project is monetarily valuable from an organization’s perspective.

Focus Area - Academic Value

Goal: Identify the degree to which a project is valuable to researchers and academic institutions.

Metric	Question
Academic Open Source Project Impact	What is the impact of open source projects that an academician or a team of academicians creates as an important part of a university reappointment, tenure, and promotion process?

Academic Open Source Project Impact

Question: What is the impact of open source projects that an academician or a team of academicians creates as an important part of a university reappointment, tenure, and promotion process?

Description

Academics need to show evidence of their scholarly output in tenure and promotion cases. Creating an open source project may be an impactful contribution and this metric helps to show this. This metric is for a new project that was created as part of an academic job and released as open source. This metric is not related to open source contributions made to an existing open source project.

Objectives

The goal is to support RPT (Reappointment, Tenure, and Promotion) by drawing forward key open source impact measures, such as:

- Understanding the scope of the community around the created open source project
- Understanding the growth, maturity, or decline of the open source project
- Identifying other projects that depend on your project
- Identifying journal articles that reference your project

Implementation

Some ideas for how to measure Academic Open Source Project Impact:

- Impact measured through publication in Journal of Open Source Software
- Number of downstream-dependencies of software in consideration
- Something similar to H-Index (doesn't currently exist)
- CiteAs API provides standardized citations for open source software
- Number of downloads
- Number of contributors
- Number of software/library citations
- Number of stars (GitHub)
- Number of published articles that cite the software or project
- Downloads of pre-prints that cite the software or project
- Regularity of updates
- Lines of Code
- Number of community contributions, not from the research team
- Downstream dependencies

Filters

Visualizations

Tools Providing the Metric

- GrimoireLab
- Augur
- CiteAS
- Journal of Open Source Software (JOSS)
- GitHub Citation
- arXiv.org code
- Center for Open Science OSF (Open Science Framework)
- ACM Artifact Review and Badging

Data Collection Strategies

References

- GitHub Citation Guidelines for Software
- arXiv.org code
- ResearchStory
- ACM Artifact Review and Badging
- Altmetric
- Related Metric: Project Popularity
- CiteAs
- Zhao, R., & Wei, M. (2017). Impact evaluation of open source software: An Altmetrics perspective. Scientometrics, 110(2), 1017–1033. https://doi.org/10.1007/s11192-016-2204-y
- Moral-Muñoz, J. A., Herrera-Viedma, E., Santisteban-Espejo, A., & Cobo, M. J. (2020). Software tools for conducting bibliometric analysis in science: An up-to-date review. El Profesional de La Información, 29(1). https://doi.org/10.3145/epi.2020.ene.03
- Searles, A., Doran, C., Attia, J., Knight, D., Wiggers, J., Deeming, S., Mattes, J., Webb, B., Hannan, S., Ling, R., Edmunds, K., Reeves, P., & Nilsson, M. (2016). An approach to measuring and encouraging research translation and research impact. Health Research Policy and Systems, 14(1), 60. https://doi.org/10.1186/s12961-016-0131-2

Contributors

- Stephen Jacobs
- Vinod Ahuja
- Elizabeth Barron
- Matt Germonprez
- Kevin Lumbard
- Georg Link

- Sean P Goggins
- Johan Linaker

Focus Area - Communal Value

Goal: Identify if a project is valuable to its community of users (including downstream projects) or contributors.

Metric	Question
Project Popularity	How popular is an open source project?
Project Velocity	What is the development speed for an organization?
Social Listening	How does one measure the value of community interactions and accurately gauge “reputation” of a community as evident from qualitative sentiment?

Project Popularity

Question: How popular is an open source project?

Description

Project popularity can be measured by how much activity is visible around a project. Popularity has a positive feedback loop in which more popular projects get more attention, attract more users or developers, and see increases in popularity, spinning the popularity wheel.

Project popularity may be used as a proxy for understanding project value because open source project economic value is hard to measure, due to a lack of available usage or sales information for open source projects.

Objectives

In a quest to earn a living wage, and to maximize future employment opportunities, workers may be interested in knowing which projects are growing and are underserved. Similarly, from an organizational perspective, knowing which projects are highly used can be helpful in knowing which projects might be worth investing in. The Project Popularity metric can be used to identify the trajectory of a project's development.

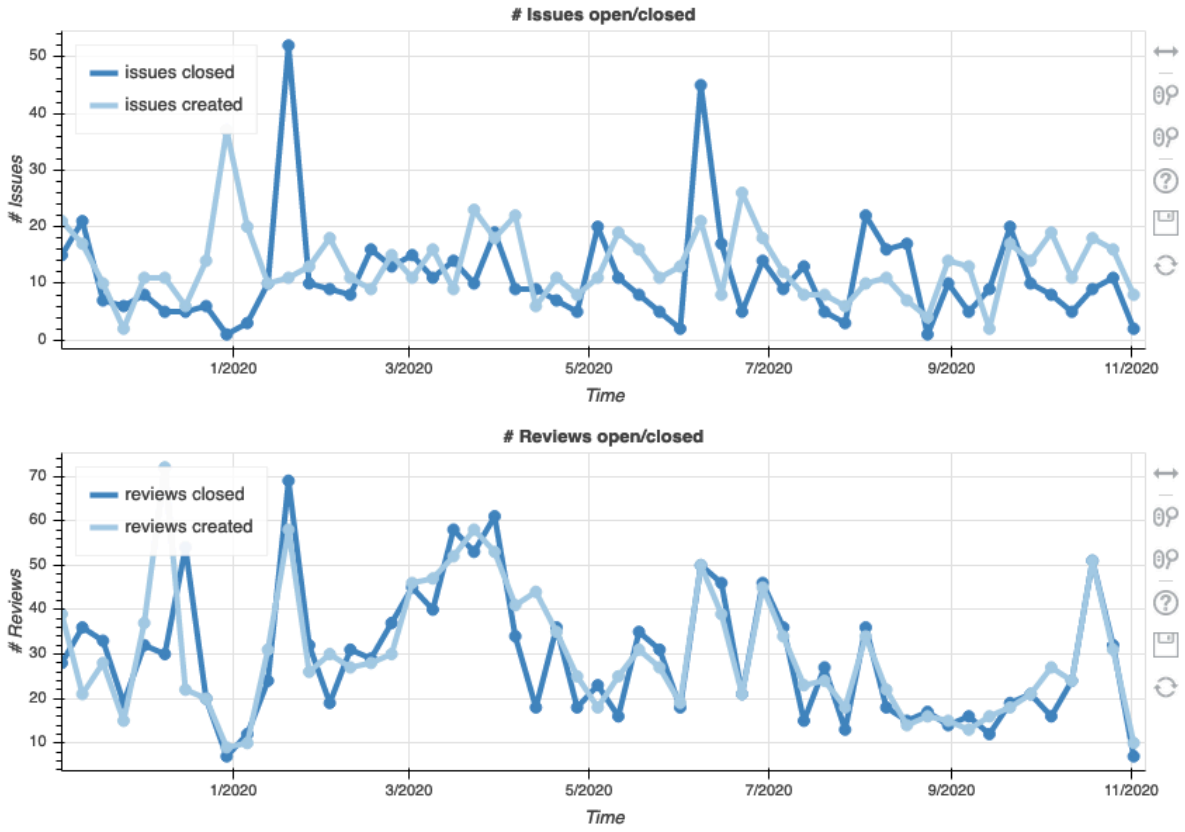
Implementation

The project popularity metric is often considered with changes over time. There are numerous example vectors to consider when measuring project popularity based on the number of:

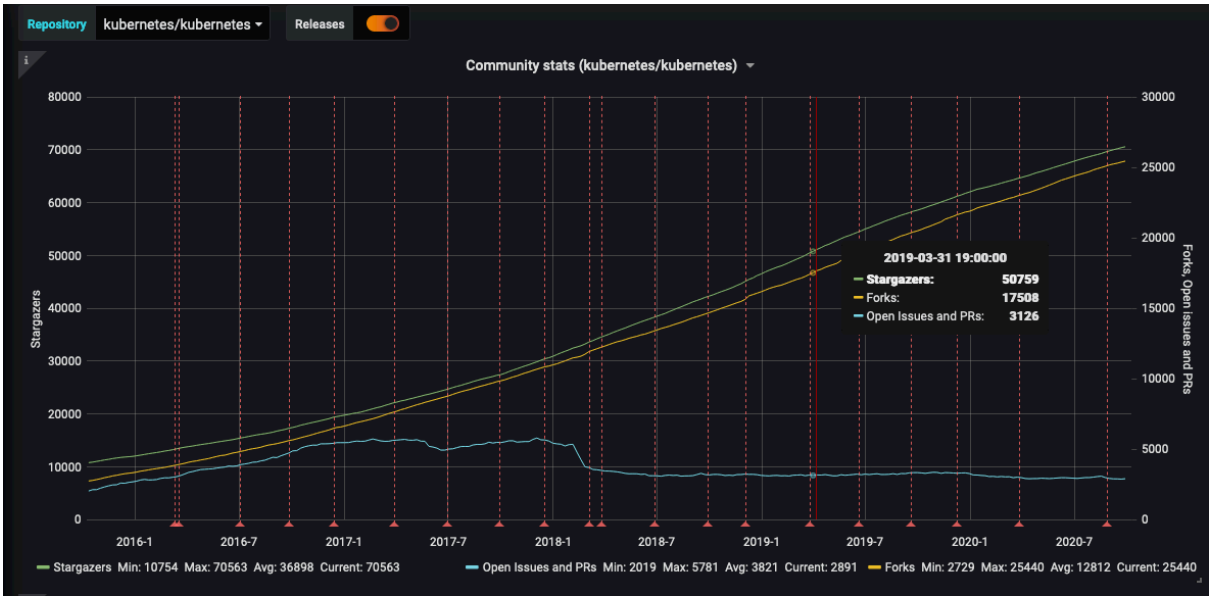
1. Social media mentions
2. Forks
3. Change requests
4. New Issues
5. Stars, badges, likes
6. New contributors
7. Organizational Diversity
8. Job postings requesting skills in project
9. Conversations within and outside of project
10. Clones
11. Followers
12. Downstream dependencies
13. People attending events that focus on a project

Visualizations

Issues and reviews (change requests) visualization from Cauldron (GrimoireLab):



Kubernetes project popularity statistics from DevStats:



Tools Providing the Metric

- Augur
- GrimoireLab
- Cauldron

References

- Popular OpenSource Projects
- Is It Maintained?
- Open Source Project Trends
- Kubernetes Salary

Project Velocity

Question: What is the development speed for an organization?

Description

Project velocity is the number of issues, the number of pull requests, volume of commits, and number of contributors as an indicator of 'innovation'.

Objectives

Gives an Open Source Program Office (OSPO) manager a way to compare the project velocity across a portfolio of projects.

The OSPO manager can use the Project Velocity metric to:

- Report project velocity of open source projects vs in-house projects
- Compare project velocity across a portfolio of projects
- Identify which projects grow beyond internal contributors (when filtering internal vs. external contributors)
- Identify promising areas in which to get involved
- Highlight areas likely to be the successful platforms over the next several years

See Example

Implementation

Base metrics include:

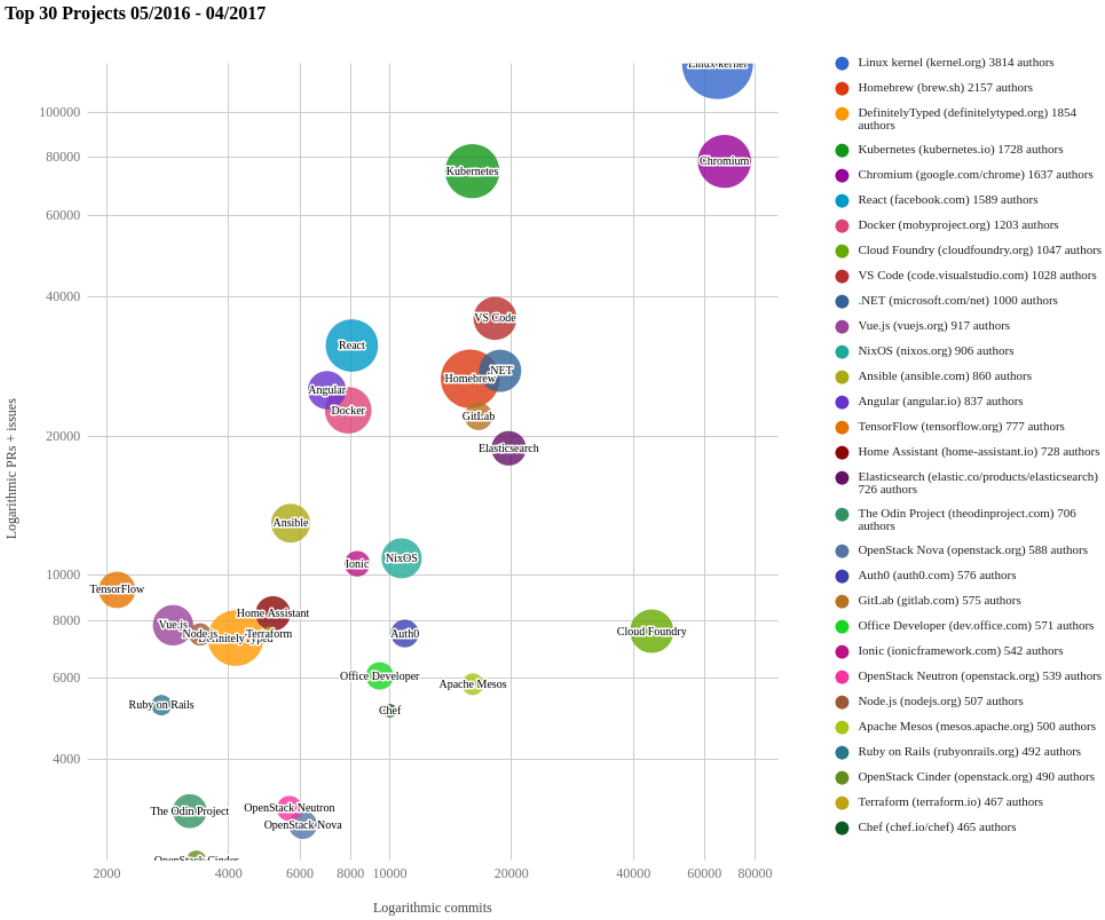
- issues closed
- number of reviews
- # of code changes
- # of committers

Filters

- Internal vs external contributors
- Project sources (e.g., internal repositories, open-source repositories, and competitor open-source repositories)
- Time

Visualizations

- X-Axis: Logarithmic scale for Code Changes
- Y-Axis: Logarithmic scale of Sum of Number of Issues and Number of Reviews
- Dot-size: Committers
- Dots are projects



From CNCF

Tools providing the Metric

- CNCF - https://github.com/cncf/velocity

References

- Can Open Source Innovation work in the Enterprise?
- Open Innovation for a High Performance Culture
- Open Source for the Digital Enterprise
- Highest Velocity Open Source Projects

Social Listening

Question: How does one measure the value of community interactions and accurately gauge “reputation” of a community as evident from qualitative sentiment?

Note: This metric synthesizes several other metrics that can be derived from trace data, and several process-oriented metrics. Embedded footnotes annotate areas planned for later clarification, and questions for later resolution.

Description

Social Listening is a combination of social listening practices across multiple channels along with a meaningful set of categorizations. The combination of these tactics can lead to systematic community analysis and can inform a community strategy that leads to measurable business value. [1]

Theory and Origin

Social currency or social capital is a social scientific theory. It broadly considers how human interactions build relationships and trust in a community. The Social Listening metric represents the reputation of a community as measured via community trust, transparency, utility, consistency, and merit.

Interpersonal relationships are the social fabric of communities. This is shown in the Levinger’s Relationship Model and Social Penetration Theory. Community members’ sense of personal and group identity grows as they interact. Members build shared values, accumulate a sense of trust, encourage cooperation, and garner reciprocity through acts of self-disclosure. These interactions build an increased and measurable sense of connection. The measure of these characteristics is called social currency. [2]

Results

The Social Listening metric is a way to sort through a fire hose of qualitative data from community interactions. A central premise of this approach is that community members’ interactions have an impact on the community. The Social Listening metric continually measures the sentiment [3] from those interactions. It illustrates the reputation and level of trust between community members and leaders. [4]

Objectives

Analyze the qualitative comments in community interactions. Gain an overview of sentiment in a community. Get metrics that show at a glance how a community is and was doing. Use lead metrics from continuous measurements for proactive community strategy development. Instill trust in community members that their thoughts and opinions are valued.

Implementation

The Social Listening requires the collection of community comments (communication traces), the definition of a codex, and the on-going review of the communication traces. [5]

Set up a Data Collection Platform of your choice as described in the “Tools” section below. Ensure it has a minimum of 4 dimensions and 3 communication channels. Once it is set up, the following method is used to collect, analyze, and interpret results:

1. **Collect Communication Traces** -- Identify online platforms that your community is communicating on. Set up data funnels from the primary platform to your Social Listening tool. The critical data for the system is user generated content.
2. **Standardize How Communication Traces Should Be Assessed** -- Use a codex to define important concepts as a "tracking keyword" or "category" in the focal community. This unified codex of terms ensures consistent analysis as different people read and tag community sentiment. Formalizing the revision and addition structure to this codex on a regular basis is a must. [5]
3. **Analyze the Communication Traces** -- Community sentiment is analyzed in the Social Listening tool by tagging data with codex terms. If the tagging is done by a team of people, it is recommended that everyone gets together regularly to discuss trends and ensure consistent tag use. If the tagging is done by an artificial intelligence algorithm, then a human team should supervise and retrain the AI as necessary. [5]
4. **Share and Visualize the Aggregated Analysis** -- Visualize the quantitative count of codex terms over time, e.g., in a dashboard. This is where the qualitative analysis results produce an easy to observe dashboard of trends. Share analysis with team members. [6]
5. **Benchmark, Set Goals & Predict Future Growth** -- After getting enough data to form a benchmark, take stock of where your community stands. What are its strengths and weaknesses? What actions can be taken to make the community healthier and more robust? Then form community initiatives with well-defined goals and execute on these projects to affect the social currency metrics for next week. [6]
6. **Repeat the Process** -- In regular evaluation meetings, discuss the shortcomings of the dataset or collection methods. Come up with methods to address these shortcomings in the future. Work solutions into the system and move forward. Truth is in the trend, power is in the pattern. [7]

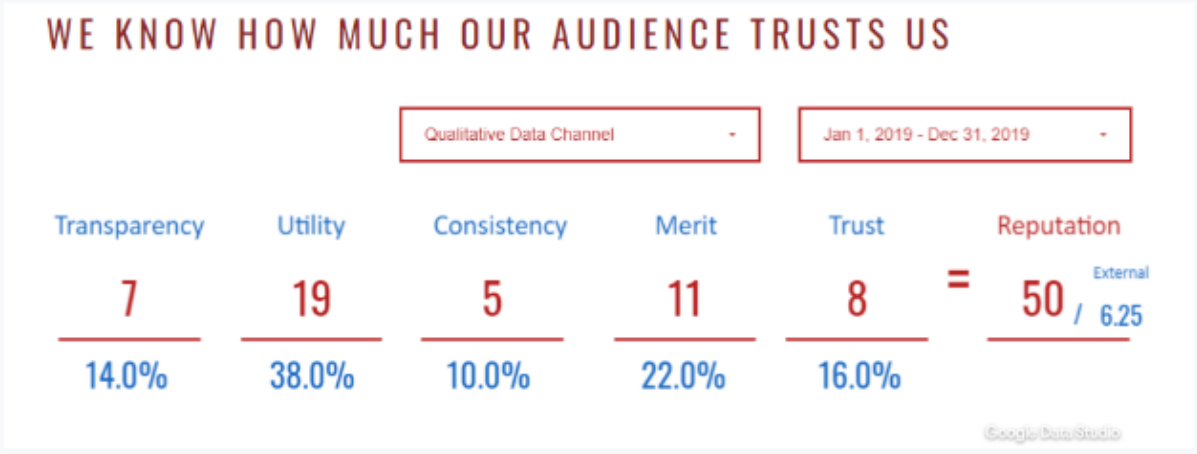
Filters

1. **Channel**: Sort by where the data was collected from.
2. **Tag**: Show data based on what codex tags were used to identify sentiment in comments.
3. **Time**: Show trends in the data over time and pull specific data-sets.
4. **Most impactful comments**: Sort and filter by flags that can be placed in the data to highlight specific data points and explain their importance.
5. **AI vs. Human tagged**: Filter by whether tags were applied programmatically or by a person.

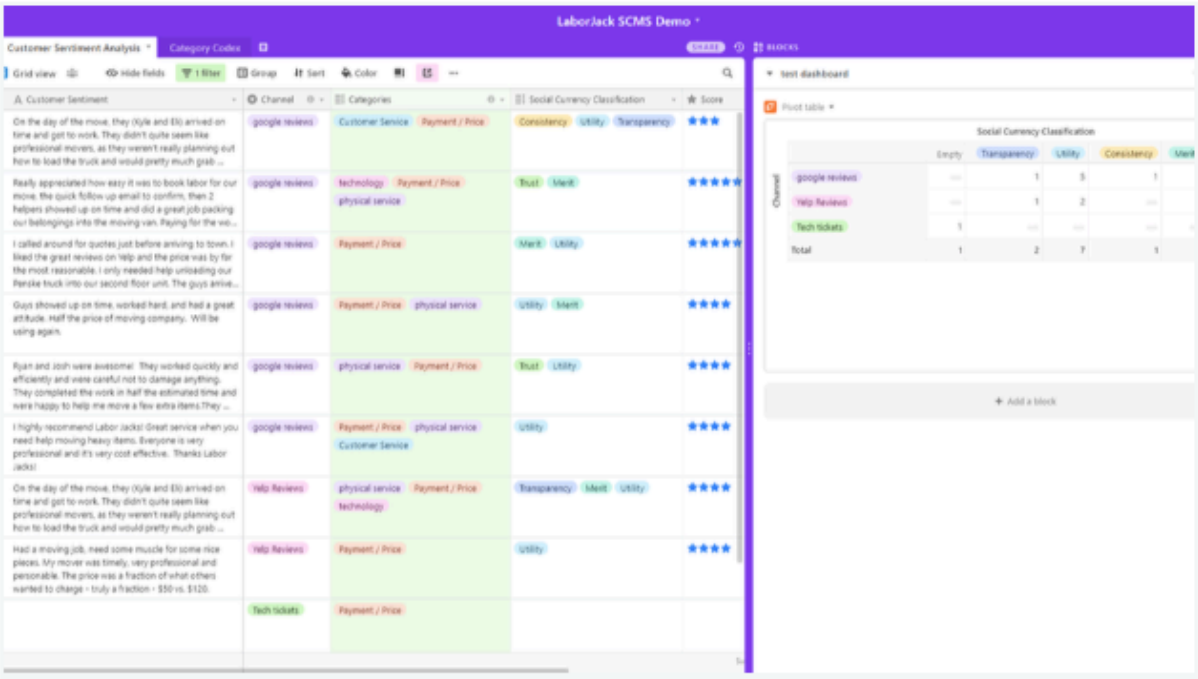
6. **Weighted currency:** Weight the “importance” of certain comments based on any one individually selected criteria. A resulting weighted view is simply a re-order of information based on weight.

Visualizations

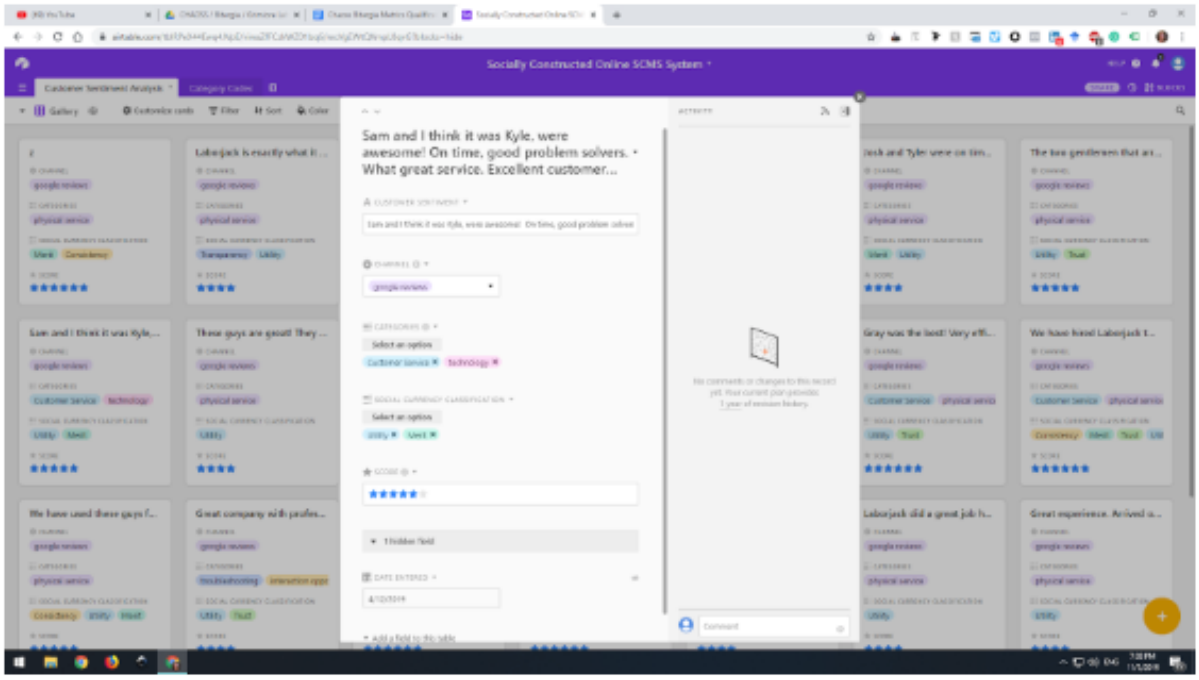
Dashboard visualizing the aggregate metrics:



Example Social Listening tool: On the left, raw community comments are shown and tags are added in columns immediately to the right. On the right, a pivot table shows in numbers how often tags occurred in combination with other tags.



Expanded comments view: remove the “quantitative” from the fields and provide the best possible way to read the different comments.



Tools Providing the Metric

To implement the metric any MySQL, smart-sheet, excel, or airtable-like excel datasheet program works fine. This data should be simplified enough to interact with other data interfaces to ensure that data migration is simple, straightforward, and can be automated (such as google data studio). This requires that systems used to implement the Social Listening metric work with CSV and other spreadsheet files, and we heavily recommend open source programs for its implementation.

Once you have this, create a data set with the following data points: [8]

Data Points	Description
Date of entry	Date data was imported to Social Listening tool
Date of comment	Date comment was made on original platform
Comment Text	Qualitative data brought in. Decide on how large you want these chunks ported. Some may port an entire email while others will be broken into one row per sentence. It should only have one “sentiment”
Data channel	Originating data channel the comment came from
Tags (created on codex document below)	Based on the unified codex of terms, decide what tags to track. There can be two kinds of tags. On the one hand, tags can be based on “themes” or recurring sentiment that people voice (e.g., gamer gate, flamewar, or thank you notes). On the other hand, tags based on “categories” can describe different aspects of a community that members comment on (e.g., events, release, or governance).
Social Currency Metric	The social currency being awarded or demerited in the system. This will directly affect numbers.
Weighted Score	Once you’ve decided what your “weight” will be, you can assign a system of -3 to +3 to provide a weighted view of human-tagged metrics (AI will not assign a weight for several reasons). This enables the “most impactful comment” filter.

Create a second sheet for the Unified Codex of Terms which will define terms. It should look like this: [8]

Category Term	Definition	When to use	When not to use
[Custom Tags - themes and categories]			
[Community specific jargon]			
Social Currency Dimensions:			
TRANSPARENCY	Do people recognize and feel a connection to your community?	When they have the "words" to pinpoint why they feel you are authentic or personable.	This is not about good customer service, or doing well. That is utility. This is about whether they understand who you are as a business and show they are onboard with it.
UTILITY	Is your community doing something useful or is it contributing value?	Provide parameters that exclude when the term is used so that people know when the category tag should not be implemented.	This is not about good customer service, or doing well. That is utility. This is about whether they understand who you are as a business and show they are onboard with it.
CONSISTENCY	Do you have a history of being reliable and dependable?	When they suggest they have used your brand, or interacted with you several times	If they've only provided their comment to suggest you were useful once, use utility instead.
MERIT	Does your community merit respect and attention for your accomplishments?	When the social currency garnered from customers seems it will continue for a while, and will impact other people's opinions.	When they suggest they will use you again in the future use trust instead as that is a personal trust in the brand. Merit is external.
TRUST	Can people trust that your community will continue to provide value and grow in the future?	When they suggest they trust you well enough to continue conversations with you in the future	When there is not substantial enough evidence to suggest they will continue to work with and trust you as a loyal customer or community member.
INTERNAL REPUTATION [9]	Do people believe these things strongly enough to warrant conversation or action?		
EXTERNAL REPUTATION [9]	What amount of your reputation in your community is transferable to strangers outside of your community (cold audiences)?		

The codex is filled in by stakeholders on a regular basis by specific communities and forms the basis for analysis of the data. This is the MOST IMPORTANT part. Without this the subjectivity of qualitative data does not follow the rule of generalization: [9]

> “A concept applies to B population ONLY SO FAR AS C limitation.”

Data Collection Strategies

Community member comments are available from trace data. The Social Listening metric ideally imports the comment text automatically into a tool for tagging. Trace data can be collected from a communities’ collaboration platforms where members write comments, including ticketing systems, code reviews, email lists, instant messaging, social media, and fora.

Legal and Regulatory Considerations

Points of destruction: Detailed data is destroyed after *xx* months has passed. Quantitative calculations are kept for up to 250 weeks per GDPR compliance. Data older than 250 weeks becomes archived data you cannot

manipulate but can see. Users can negotiate the primary statistic.

References

- An example implementation on airtable
- An example implementation in data studio (report)
- An example implementation in data studio (data source)
- An example implementation in google sheets
- Implementation documentation (starts on page 33)

Annotated Footnotes

[1] CHAOSS metrics historically is to create standard definitions that can be used reliably across projects to support comparisons. This metric may evolve into a project in the future.

[2] What metrics emerge from this description? Likely included are: 1. community trust, 2. transparency, 3. utility, 4. consistency, and 5. merit

[3] Analysis of sentiment suggests that metric (6) is likely ”Communications Sentiment”, and the definition may need to include references to common sentiment analysis tools, sometimes called ”bags of words”.

[4] Measuring how trust trust is instilled in community members, such that their thoughts and opinions are valued is likely metric (7) that will define a process, and perhaps is not measurable via trace data.

[5] A substantial portion of any codex for open source software will be common across projects, and each project is likely to have a set of particular interests that are a subset of that codex. In some cases, their main interests may not be present in an established codex component. In general, the codex, like the CHAOSS project itself, is open sourced as shared metadata to ensure shared understanding across open source communities.

[6] This describes the evolution of a standard codex, and its elements through the process of CHAOSS working groups and projects, characterized in the previous footnote. Likely this will be a process metric (8).

[7] Candidate process oriented metric (9).

[8] Examples of data coded using the open sourced codex, as it evolves, will be essential components for advancing open source software through Social Listening. Implementations will require these examples, and their provision as open source assets of the CHAOSS project will return value as shared data.

[9] Internal and external reputation are likely metrics (10), and (11) arising from the Social Listening metric.

Focus Area - Individual Value

Goal: Identify if a project is valuable to me as an individual user or contributor.

Metric	Question
Organizational Project Skill Demand	How many organizations are using this project and could hire me if I become proficient?
Job Opportunities	How many job postings request skills with technologies from a project?

Organizational Project Skill Demand

Question: How many organizations are using this project and could hire me if I become proficient?

Description

Organizations engage with open source projects through use and dependencies. This metric is aimed at determining downstream demand of skills related to an open source project. This metric looks at organizations that deploy a project as part of an IT infrastructure, other open source projects with declared dependencies, and references to the project through social media, conference mentions, blog posts, and similar activities.

Objectives

As a developer, I'd like to invest my skills and time in a project that has a likelihood of getting me a decent paying job in the future. People can use the Downstream Organizational Impact of a Project Software metric to discover which projects are used by organizations, and they may, therefore, be able to pursue job opportunities with, possibly requiring IT support services.

Implementation

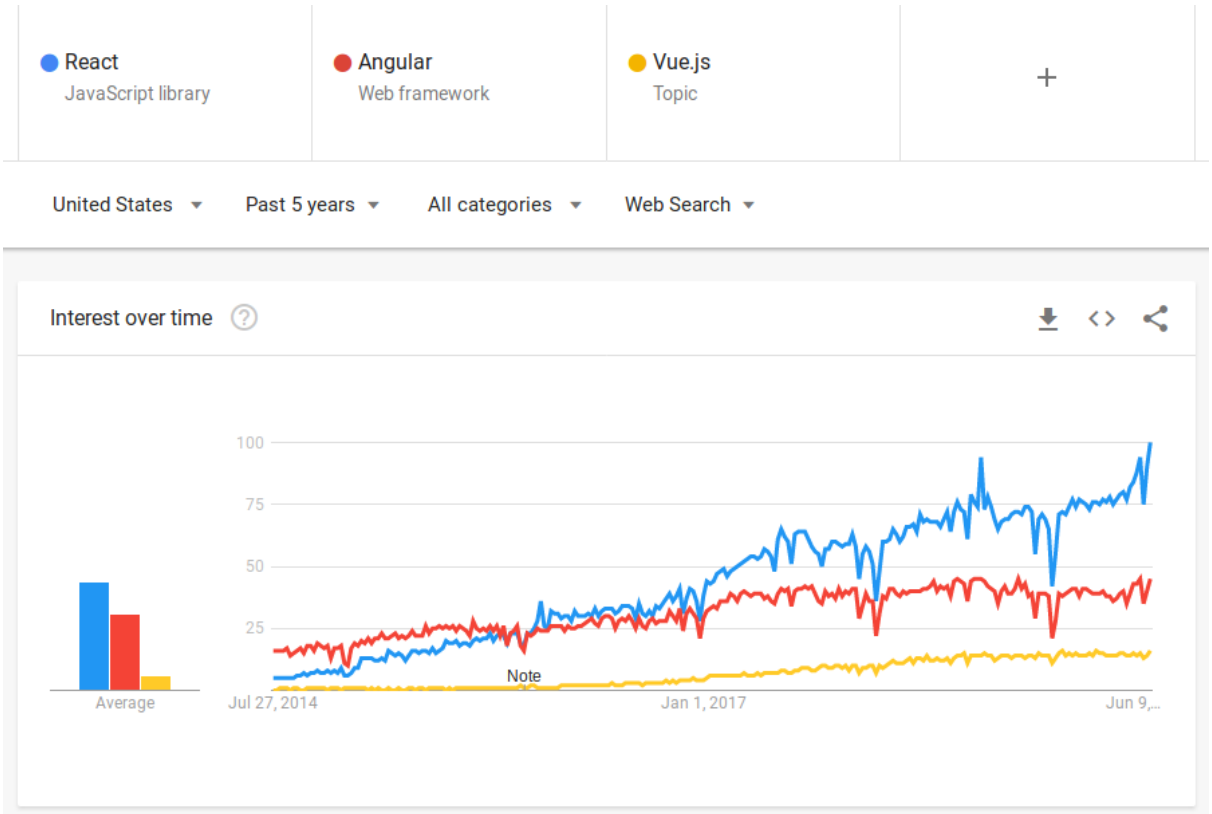
Base metrics include:

- Number of organizations that created issues for a project
- Number of organizations that created pull requests for a project
- Number of organizations that blog or tweet about a project
- Number of organizations that mention a project in open hiring requests
- Number of organizations that are represented at meetups about this project
- Number of other projects that are dependent on a project
- Number of books about a project
- Google search trends for a project

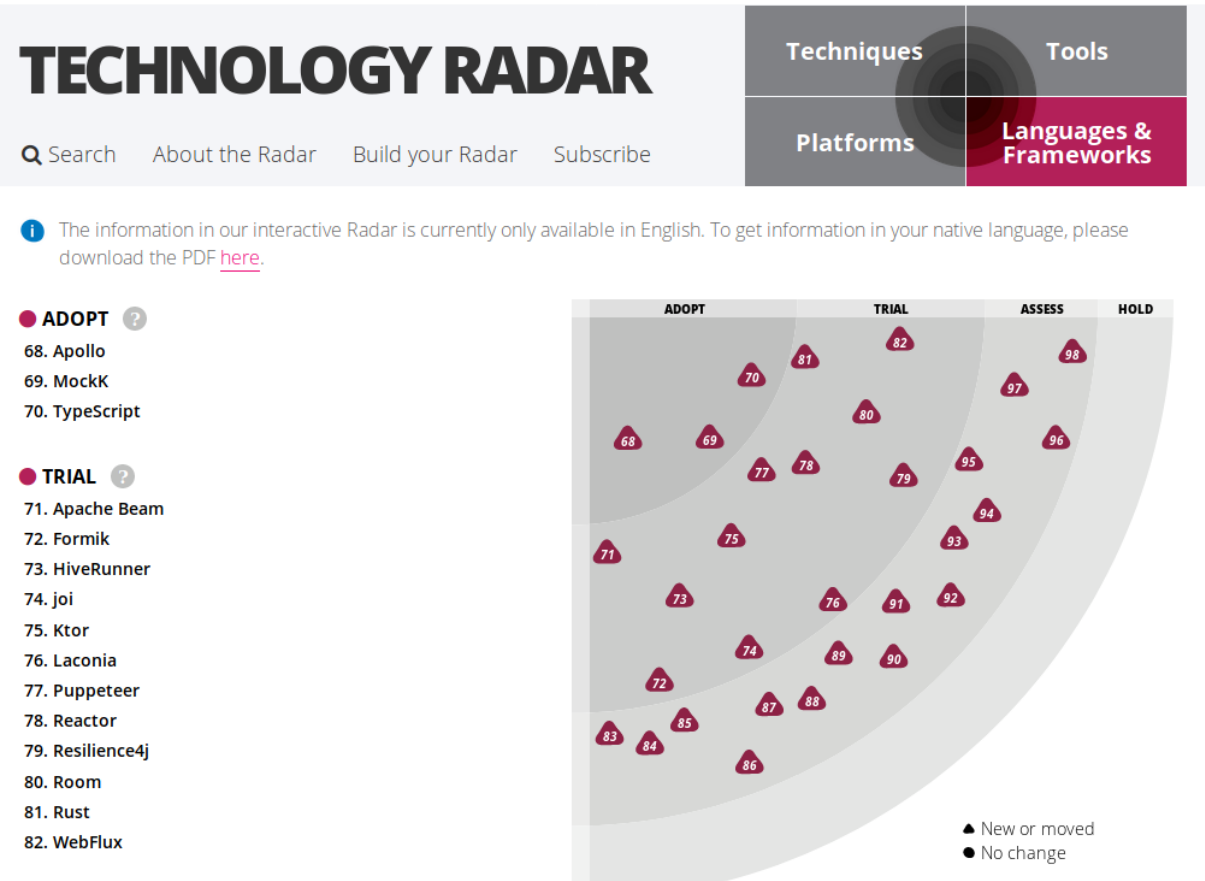
Visualizations

The following visualization demonstrates the number of downstream projects dependendent on the project in question. While this visualization does not capture the entirety of the Downstream Organizational Impact of a Project Software metric, it provides a visual for a portion.

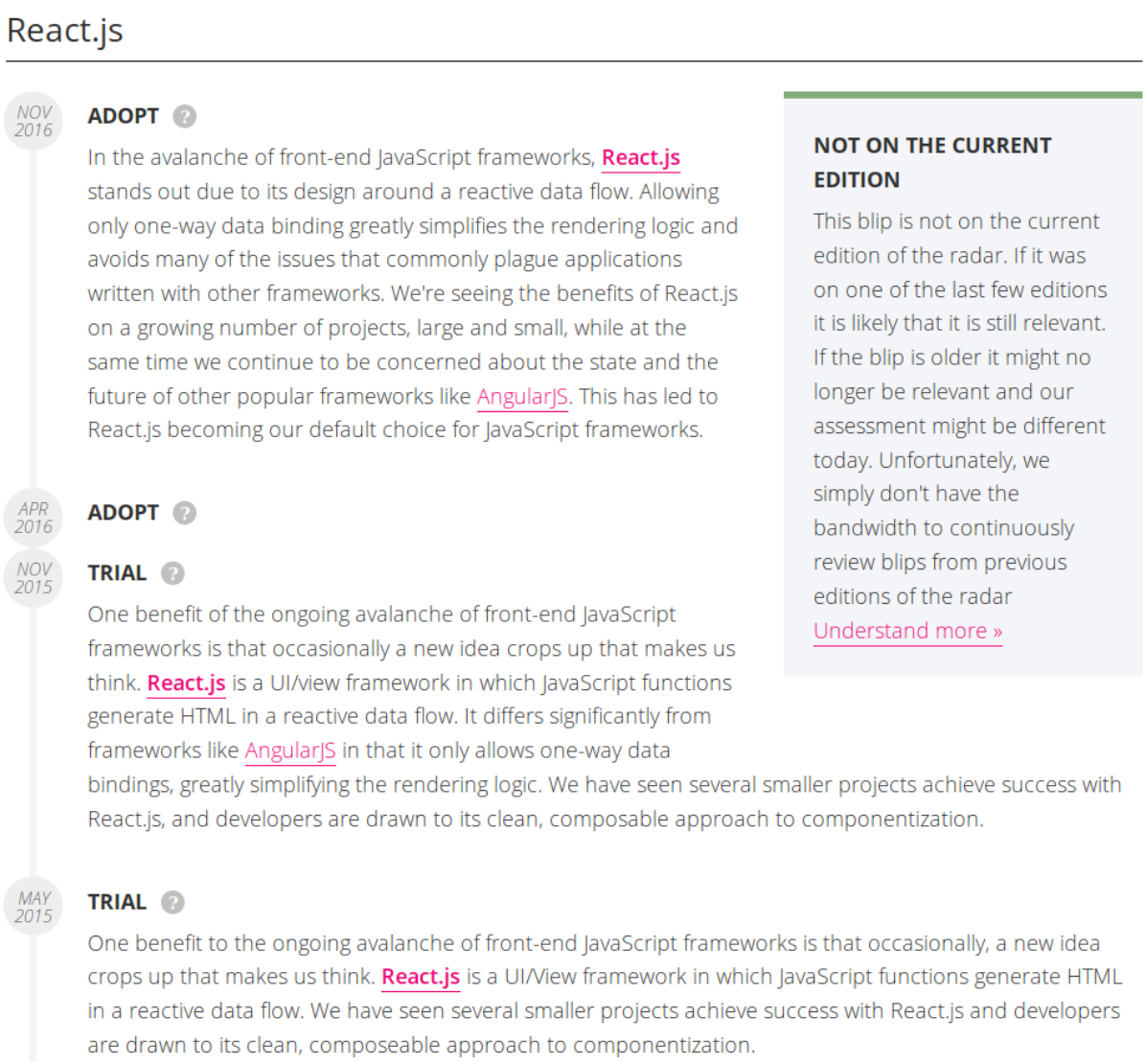
Other visualizations could include Google search trends (React vs. Angular vs. Vue.js)



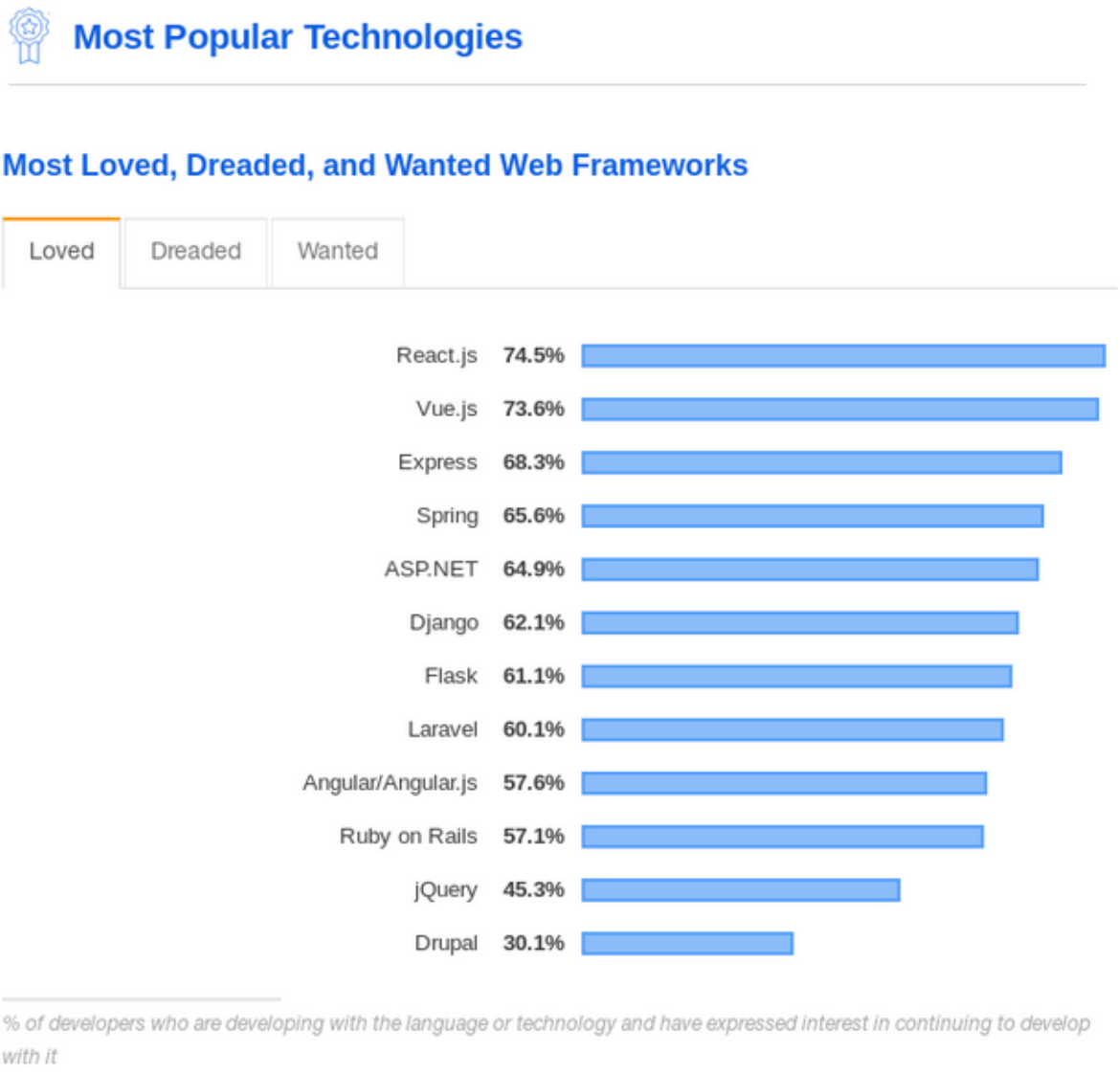
ThoughtWorks publishes a series called 'Tech Radar' that shows the popularity of technologies.



Tech Radar allows you to drill down on projects to see how the assessment has changed over time.



StackOverview publishes an annual developer's survey



Tools Providing the Metric

- Google Trends - for showing search interest over time
- ThoughtWorks TechRadar - project assessments from a tech consultancy
- StackOverflow Developer's Survey - annual project rankings
- Augur; Examples are available for multiple repositories:
  - Rails
  - Zephyr
  - CloudStack

References

- Open Source Sponsors
- Fiscal Sponsors and Open Source
- Large Corporate OpenSource Sponsors
- Google Trends API
- Measuring Open Source Software Impact
- ThoughtWorks Tech Radar
- Stack Overflow Developer's Survey

Job Opportunities

Question: How many job postings request skills with technologies from a project?

Description

A common way for open source contributors to earn a living wage is to be employed by a company or be a self-employed or freelance developer. Skills in a specific project may improve a job applicant's prospects of getting a job. The most obvious indicator for demand related to a skill learned in a specific open source project is when that project or its technology is included in job postings.

Objectives

The metric gives contributors a sense of how much skills learned in a specific open source project are valued by companies.

Implementation

To obtain this metric on a job search platform (e.g., LinkedIn, Indeed, or Dice), go to the job search and type in the name of the open source project. The number of returned job postings is the metric. Periodically collecting the metric through an API of a job search platform and storing the results allows to see trends.

Filters

- Age of job posting; postings get stale and may not be removed when filled

Visualizations

The metric can be extended by looking at:

- Salary ranges for jobs returned
- Level of seniority for jobs returned
- Availability of jobs like on-site or off-site
- Location of job
- Geography

References

- LinkedIn Job Search API: https://developer.linkedin.com/docs/v1/jobs/job-search-api#
- Indeed Job Search API: https://opensource.indeedeng.io/api-documentation/docs/job-search/
- Dice.com Job Search API: http://www.dice.com/external/content/documentation/api.html
- Monster Job Search API: https://partner.monster.com/job-search
- Ziprecruiter API (Requires Partnership): https://www.ziprecruiter.com/zipsearch

Note: This metric is limited to individual projects but engagement in open source can be beneficial for other reasons. This metric could be tweaked to look beyond a single project and instead use related skills such as programming languages, processes, open source experience, or frameworks as search parameters for jobs.

Focus Area - Organizational Value

Goal: Identify if a project is monetarily valuable from an organization's perspective.

Metric	Question
Organizational Influence	How much influence does an organization have on an open source community?
Labor Investment	What was the cost of an organization for its employees to create the counted contributions (e.g., commits, issues, and pull requests)?

Organizational Influence

Question: How much influence does an organization have on an open source community?

Description

Organizational Influence is a measure of the influence that an organization has on an open source community. An organizations may influence the direction of a project, signaling some level of control within a community. This equates to the level of influence an organization has over the development trajectory of an open source project that it contributes to and has a vested interest in.

Objectives

An organization can have influence in open source projects. The measure of an organization's influence can be enlightening on its own or serve as an opportunity to show growth in an organization's influence over time. This metric can help an open source advocate in an organization justify continued funding and support for engagement. This metric can also help open source maintainers track and measure organizational influence to add legitimacy to their project, and monitor the level of control by individual organizations. An organization's influence may also serve as a signal of how easy or difficult it may be for new members to contribute to an open source project.

Implementation

Some specific examples to consider when measuring organizational influence include:

- How many organizational members are contributing to a project - contributors. It can be understood as a ratio of (number of contributors from each organization with more than one contributor)/(number of contributors)
- Level and [types of contributions] (https://chaoss.community/metric-types-of-contributions/) by organizational members
- Organizations that are contributing change requests at a high rate in proportion to the community's level of activity. For example, one standard deviation or less of other contributing organizations. Related to organizational diversity
- Organizational members on the technical steering committee
- Organizational members on governing board
- Organizational members in project maintainer roles
- Organizations sponsoring an open source project as a proportion of the total

Tools Providing the Metric

- Augur
- GrimoireLab

Contributors

- Sean Goggins
- Matt Germonprez
- Vinod Ahuja

- Kevin Lumbard
- Lawrence Hecht
- Matt Snell
- Dhruv Sachdev
- Elizabeth Barron
- Matt Broberg
- Stephen Jacobs

Labor Investment

Question: What was the cost of an organization for its employees to create the counted contributions (e.g., commits, issues, and pull requests)?

Description

Open source projects are often supported by organizations through labor investment. This metric tracks the monetary investment of organizations (as evident in labor costs) to individual projects.

Objectives

As organizational engagement with open source projects becomes increasingly important, it is important for organization to clearly understand their labor investment. The objective of this metric is to improve transparency in labor costs for organizations engaged with open source projects. This metric gives an Open Source Program Office (OSPO) manager a way to compare contributed labor costs across a portfolio of projects. For example, the Labor Investment metric can be used to prioritize investment or determine return on investment such as:

- Labor Investment as a means of evaluating OSPO priorities and justifying budgets
- Labor Investment as a way to explain product/program management priority
- Labor Investment as an argument for the value of continued investing in OSPOs
- Labor Investment to report and compare labor costs of contributed vs in-house work
- Labor Investment to compare project effectiveness across a portfolio of projects

Implementation

Base metrics include:

- Number of contributions
- Number of contributions broken out by contributor types (internal / external)
- Number of contributions broken out by contribution types (e.g., commits, issues, pull requests)

Parameters include:

- Hourly labor rate
- Average labor hours to create contribution (by contribution type)

Labor Investment = For each contribution type, sum (Number of contributions * Average labor hours to create contribution * Average hourly rate)

Filters

- Internal vs external contributors
- Issue tags
- Project sources (e.g., internal, open-source repos, competitor open-source repos)

Visualizations

```
IssueID, Severity, Title          , Status, Contributor, Tag
34234  , High    , Add CSV Graphic, Open  , andyl      , metrics
23421  , Med     , Fix typos      , Closed, mattg      , metrics
56743  , High    , Reword section , Open  , georg      , augur
85879  , Low     , Add CNCF PNG   , Open  , seang      , metrics
34183  , High    , Remove button  , Closed, vinod      , implementation
76790  , Low     , Use large font , Open  , kevin      , metrics
57432  , Med     , Sync with web  , Closed, carol      , implementation
```

Our first visualization of parameterized metrics rely on CSV exports that can be made available from Augur. Spreadsheets are used for metric parameters and calculation formulas. Future implementations may add features for parameter manipulation directly in the webapp.

References

- Starting an Open Source Program Office
- Creating an Open Source Program Office
- Open Source in the Enterprise

Release Notes

CHAOSS metrics are released continuously. The regular release is when we update the version number, update the full release notes, and make a big announcement. These releases occur one to two times a year and may correspond with the dates of a CHAOSScon event. Prior to regular release, continuous released metrics go through a comment period of at least 30 days.

Release 2021-10 Notes:

- PDF of released CHAOSS Metrics (v.2021-10)
- **Common WG**
  - Focus Area Name Revisions:
    - People (was ”Who”)
    - Contribution (was ”What”)
    - Time (was ”When”)
    - Place (was ”Where”)
  - New metrics include:
    - Bot Activity
    - Clones
    - Collaboration Platform Activity
    - Event Locations
    - Language Distribution
  - Metric Revision:
    - Technical Forks
- **Diversity & Inclusion WG**
  - New metrics include:
    - Documentation Discoverability
    - Inclusive Experience at Event
    - Psychological Safety
  - Name Change/Revision:
    - Attendee Demographics and Speaker Demographics merged into Event Demographics
- **Evolution WG**
  - New metrics include:
    - Contribution Attribution
- **Risk WG**
  - New Focus Area:
    - Dependency Risk Assessment
  - New metrics include:
    - Libyears
    - Upstream Code Dependencies
- **Value WG**
  - New Focus Area:
    - Academic Value
  - New metrics include:
    - Academic Open Source Project impact
    - Organizational Influence

CHAOSS Contributors

Aastha Bist, Abhinav Bajpai, Ahmed Zerouali, Akshara P, Akshita Gupta, Amanda Brindle, Anita Ihuman, Alberto Martín, Alberto Pérez García-Plaza, Alexander Serebrenik, Alexandre Courouble, Alolita Sharma, Alvaro del Castillo, Ahmed Zerouali, Amanda Casari, Amy Marrich, Ana Jimenez Santamaria, Andre Klapper, Andrea Gallo, Andy Grunwald, Andy Leak, Aniruddha Karajgi, Anita Sarma, Ankit Lohani, Ankur Sonawane, Anna Buhman, Armstrong Foundjem, Atharva Sharma, Ben Lloyd Pearson, Benjamin Copeland, Beth Hancock, Bingwen Ma, Boris Baldassari, Bram Adams, Brian Proffitt, Camilo Velazquez Rodriguez, Carol Chen, Carter Landis, Chris Clark, Christian Cmehil-Warn, Damien Legay, Dani Gellis, Daniel German, Daniel Izquierdo Cortazar, David A. Wheeler, David Moreno, David Pose, Dawn Foster, Derek Howard, Don Marti, Drashti, Duane O'Brien, Dylan Marcy, Eleni Constantinou, Elizabeth Barron, Emily Brown, Emma Irwin, Eriol Fox, Fil Maj, Gabe Heim, Georg J.P. Link, Gil Yehuda, Harish Pillay, Harshal Mittal, Henri Yandell, Henrik Mitsch, Igor Steinmacher, Ildiko Vancsa, Jacob Green, Jaice Singer Du Mars, Jaskirat Singh, Jason Clark, Javier Luis Cánovas Izquierdo, Jeff McAffer, Jeremiah Foster, Jessica Wilkerson, Jesus M. Gonzalez-Barahona, Jilayne Lovejoy, Jocelyn Matthews, Johan Linåker, John Coghlan, John Mertic, Jon Lawrence, Jonathan Lipps, Jono Bacon, Jordi Cabot, Jose Manrique Lopez de la Fuente, Joshua Hickman, Joshua R. Simmons, Josianne Marsan, Justin W. Flory, Kate Stewart, Katie Schueths, Keanu Nichols, Kevin Lumbard, King Gao, Kristof Van Tomme, Lars, Laura Dabbish, Laura Gaetano, Lawrence Hecht, Leslie Hawthorne, Luis Cañas-Díaz, Luis Villa, Lukasz Gryglicki, Mariam Guizani, Mark Matyas, Martin Coulombe, Matthew Broberg, Matt Germonprez, Matt Snell, Michael Downey, Miguel Ángel Fernández, Mike Wu, Neil Chue Hong, Neofytos Kolokotronis, Nick Vidal, Nicole Huesman, Nishchith K Shetty, Nithya Ruff, Nuritzi Sanchez, Parth Sharma, Patrick Masson, Peter Monks, Pranjal Aswani, Pratik Mishra, Prodromos Polychroniadis, Quan Zhou, Ray Paik, Remy DeCausemaker, Ria Gupta, Richard Littauer, Robert Lincoln Truesdale III, Robert Sanchez, Rupa Dachere, Ruth Ikegah, Saicharan Reddy, Saloni Garg, Saleh Abdel Motaal, Samantha Lee, Samantha Venia Logan, Samson Goddy, Santiago Dueñas, Sarit Adhikari, Sarvesh Mehta, Sarah Conway, Sean P. Goggins, Shane Curcuru, Sharan Foga, Shaun McCance, Shreyas, Silona Bonewald, Sophia Vargas, Sri Ramkrishna, Stefano Zacchiroli, Stefka Dimitrova, Stephen Jacobs, Tharun Ravuri, Thom DeCarlo, Tianyi Zhou, Tobie Langel, Saleh Abdel Motaal, Tom Mens, UTpH, Valerio Cosentino, Venu Vardhan Reddy Tekula, Vicky Janicki, Victor Coisne, Vinod Ahuja, Vipul Gupta, Will Norris, Xavier Bol, Xiaoya, Zibby Keaton

Are you eligible to be on this list? You are if you helped in any capacity, for example: Filed an issue. Created a Pull Request. Gave feedback on our work. Please open an issue or post on the mailing list if we've missed anyone.

CHAOSS Governing Board Members

- Amy Marrich, Red Hat
- Andrea Gallo, Linaro
- Armstrong Foundjem, MCIS Laboratory at Queen’s University
- Daniel Izquierdo, Bitergia
- Dawn Foster, VMware
- Don Marti, CafeMedia
- Georg Link, Bitergia
- Ildikó Vancsa, OpenStack
- Kate Stewart, Linux Foundation
- Matt Germonprez, University of Nebraska at Omaha
- Nicole Huesman, Intel
- Ray Paik, GitLab
- Sean Goggins, University of Missouri
- Wayne Beaton, Eclipse Foundation

LICENSE