Metrics

Release 202001

https://chaoss.community/metrics

CHAOSS Contributors include:

Ahmed Zerouali, Akshita Gupta, Amanda Brindle, Alberto Martín, Alberto Pérez García-Plaza, Alexander Serebrenik, Alexandre Courouble, Alolita Sharma, Alvaro del Castillo, Ahmed Zerouali, 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, 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, Dylan Marcy, Eleni Constantinou, Emma Irwin, Fil Maj, Gabe Heim, Georg J.P. Link, Gil Yehuda, Harish Pillay, Harshal Mittal, Henri Yandell, Henrik Mitsch, Ildiko Vancsa, Jacob Green, Jaice Singer Du Mars, Jason Clark, Javier Luis Cánovas Izquierdo, Jeff McAffer, Jeremiah Foster, Jessica Wilkerson, Jesus M. Gonzalez-Barahona, Jocelyn Matthews, Johan, Johan Linåker, John Mertic, Jon Lawrence, Jonathan Lipps, Jono Bacon, Jordi Cabot, Jose Manrique Lopez de la Fuente, Joshua R. Simmons, Josianne Marsan, Kate Stewart, Keanu Nichols, Kevin Lumbard, Kristof Van Tomme, Lars, Laura Gaetano, Lawrence Hecht, Leslie Hawthorne, Luis Cañas-Díaz, Luis Villa, Lukasz Gryglicki, Mark Matyas, Martin Coulombe, Matthew Broberg, Matt Germonprez, Matt Snell, Michael Downey, Miguel Ángel Fernández, Mike Wu, Neil Chue Hong, Nick Vidal, Nicole Huesman, Nishchith K Shetty, Nithya Ruff, Parth Sharma, Patrick Masson, Peter Monks, Pranjal Aswani, Prodromos Polychroniadis, Quan Zhou, Ray Paik, Remy DeCausemaker, Robert Lincoln Truesdale III, Robert Sanchez, Rupa Dachere, Saloni Garg, Saleh Motaal, Samantha Logan, Santiago Dueñas, Sarvesh Mehta, Sarah Conway, Sean P. Goggins, Shane Curcuru, Sharan Foga, Shreyas, Stefano Zacchiroli, Thom DeCarlo, Tobie Langel, Tom Mens, UTpH, Valerio Cosentino, Venu Vardhan Reddy Tekula, Vicky Janicki, Victor Coisne, Vinod Ahuja, Vipul Gupta, Will Norris, Xavier Bol, Zibby Keaton

The CHAOSS Governing Board at time of release:

- Andrea Gallo, Linaro
- Ben Lloyd Pearson, Nylas
- Brian Proffitt, Red Hat
- Daniel Izquierdo, Bitergia
- Daniel M. German, University of Victoria
- Dawn Foster, Pivotal
- Don Marti, Mozilla
- 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

CHAOSS Metrics

Release 202001

Released metrics are only a subset of metric ideas that are being developed. If you would like to learn more and discuss different metrics please visit the working group repositories. The metrics are sorted into Focus Areas. CHAOSS uses a Goal-Question-Metric format to present metrics. Individual metrics are released based on identified goals and questions. The metrics include a detail page with definitions, objectives, and examples.

Focus Areas by Working Group

Common Metrics WG

- When: Time
- Who: People

Diversity and Inclusion WG

- Event Diversity
- Governance
- Leadership

Evolution WG

- Code Development Activity
- Code Development Efficiency
- Code Development Process Quality
- Issue Resolution
- Community Growth

Risk WG

- Business Risk
- Code Quality
- Licensing

- Security

Value WG

- Ecosystem Value
- Labor Investment
- Living Wage

Important Dates for Release 202001

Release Freeze: January 1st, 2020
Candidate Release: January 24th, 2020
Public Comment Period: January 1st, 2020 to January 24th, 2020
Metrics Release Date: January 31st, 2020

Common Metrics

Common Metrics Repository: https://github.com/chaoss/wg-common

Focus Area - When: Time

Goal:
Understand when contributions from organizations and people are happening.

Metric/Details	Question
Activity Dates and Times	What are the dates and timestamps of when contributor activities occur?
Time to First Response	How much time passes between when an activity requiring attention is created and the first response?

Focus Area - Who: People

Goal:
Understand organizational and personal engagement with open source projects.

Metric/Details	Question
Contributors	Who are the contributors to a project?
Organizational Diversity	What is the organizational diversity of contributions?

Diversity and Inclusion

D&I Repository: https://github.com/chaoss/wg-diversity-inclusion

Focus Area - Event Diversity

Goal:
Identify the diversity and inclusion at events.

Metric/Details	Question
Attendee Demographics	How diverse and inclusive are the attendees?
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?
Family Friendliness	How does enabling families to attend together support diversity and inclusion of the event?
Speaker Demographics	How well does the speaker lineup for the event represent a diverse set of demographics and can be improved in the future?

Focus Area - Governance

Goal:
Identify how diverse and inclusive project governance is.

Metric/Details	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?

Focus Area - Leadership

Goal:
Identify how healthy community leadership is.

Metric/Details	Question
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?

Evolution

Evolution Repository: https://github.com/chaoss/wg-evolution

Scope: Aspects related to how the source code changes over time, and the mechanisms that the project has to perform and control those changes.

Focus Area - Code Development Activity

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

Metric/Details	Question
Code Changes	What changes were made to the source code during a specified period?

Metric/Details	Question
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?

Focus Area - Code Development Efficiency

Goal:
Learn how efficiently activities around code development get resolved.

Metric/Details	Question
Reviews Accepted	How many accepted reviews are present in a code change?
Reviews Declined	What reviews of code changes ended up declining the change during a certain period?
Reviews Duration	What is the duration of time between the moment a code review starts and moment it is accepted?

Focus Area - Code Development Process Quality

Goal:
Learn 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/Details	Question
Reviews	What new review requests for changes to the source code occurred during a certain period?

Focus Area - Issue Resolution

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

Metric/Details	Question
Issues New	What are the number of new issues created during a certain period?
Issues Active	What is the count of issues that showed activity during a certain period?
Issues Closed	What is the count of issues that were closed during a certain period?
Issue Age	What is the average time that open issues have been 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?

Focus Area - Community Growth

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

Metric/Details	Question
New Contributors Closing Issues	How many contributors are closing issues for the first time?

Risk

Risk Repository: https://github.com/chaoss/wg-risk

Focus Area - Business Risk

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

Metric/Details	Question
Committers	How robust and diverse are the contributors to a community?
Elephant Factor	What is the distribution of work in the community?

Focus Area - Code Quality

Goal:
Understand the quality of a given software package.

Metric/Details	Question
Test Coverage	How well is the code tested?

Focus Area - Licensing

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

Metric/Details	Question
License Count	How many different licenses are there?
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?

Focus Area - Security

Goal:
Understand how transparent a given software package is with respect to dependencies, licensing (?), security processes, etc.

Metric/Details	Question
CII Best Practices badge	What is the current CII Best Practices status for the project?

Value

Value Repository: https://github.com/chaoss/wg-value

Focus Area - Ecosystem Value

Goal:
Estimate the value of an open source project's ecosystem.

Metric/Details	Question
Social Currency Metric System (SCMS)	How does one measure the value of community interactions and accurately gauge "trust" within a community as evident from qualitative sentiment?

Focus Area - Labor Investment

Goal:
Estimate the labor investment in open source projects.

Metric/Details	Question
Labor Investment	What was the cost of an organization for its employees to create the counted contributions (e.g., commits, issues, and pull requests)?
Project Velocity	What is the development speed for an organization?

Focus Area - Living Wage

Goal:
Expanding opportunities for people to make a living wage in open source.

Metric/Details	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?

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

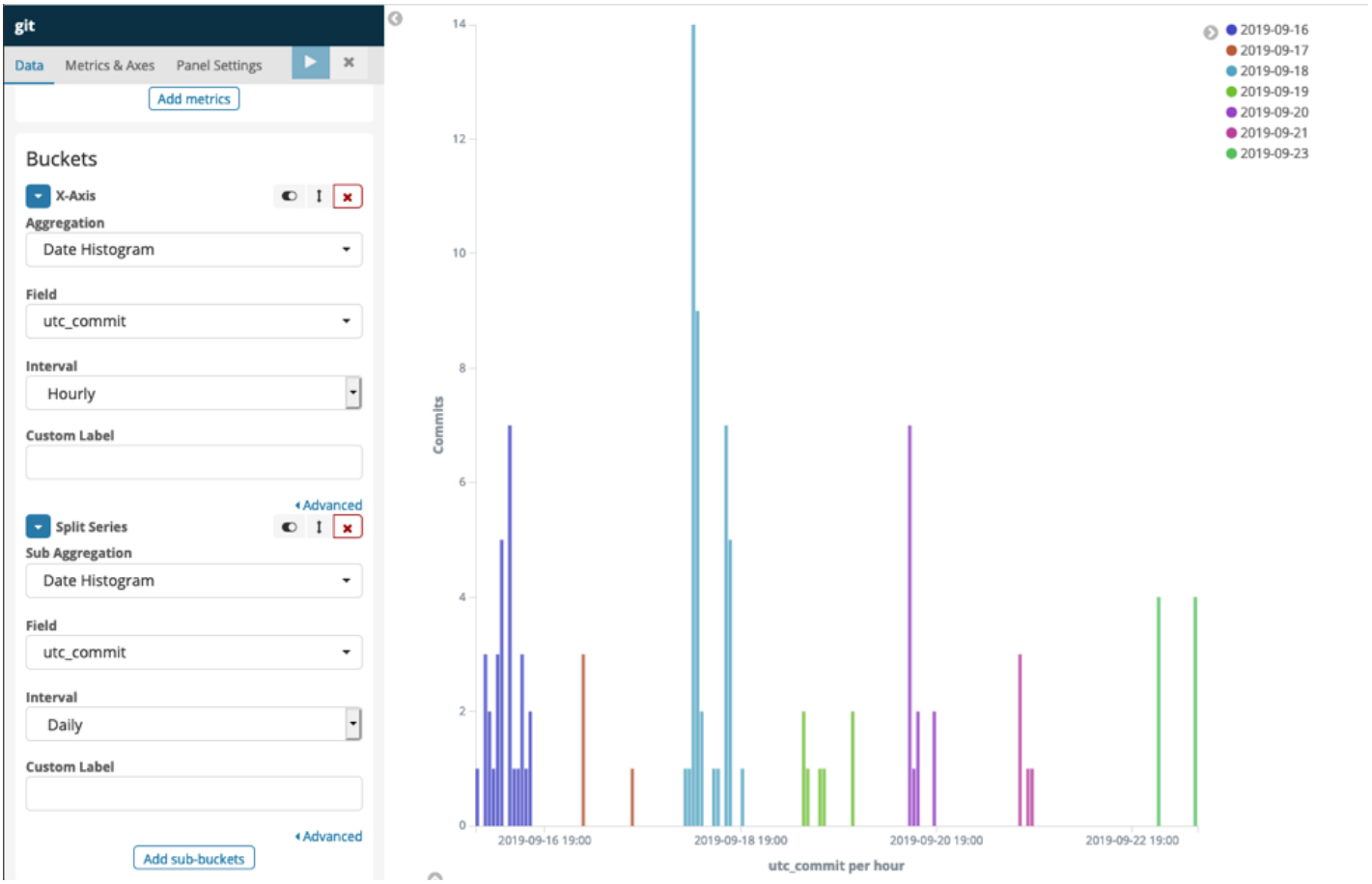
- Improve transparency for employers about when organizational employees are engaging with open source projects
- Improve transparency for open source project and community managers as to when activity is occurring

Implementation

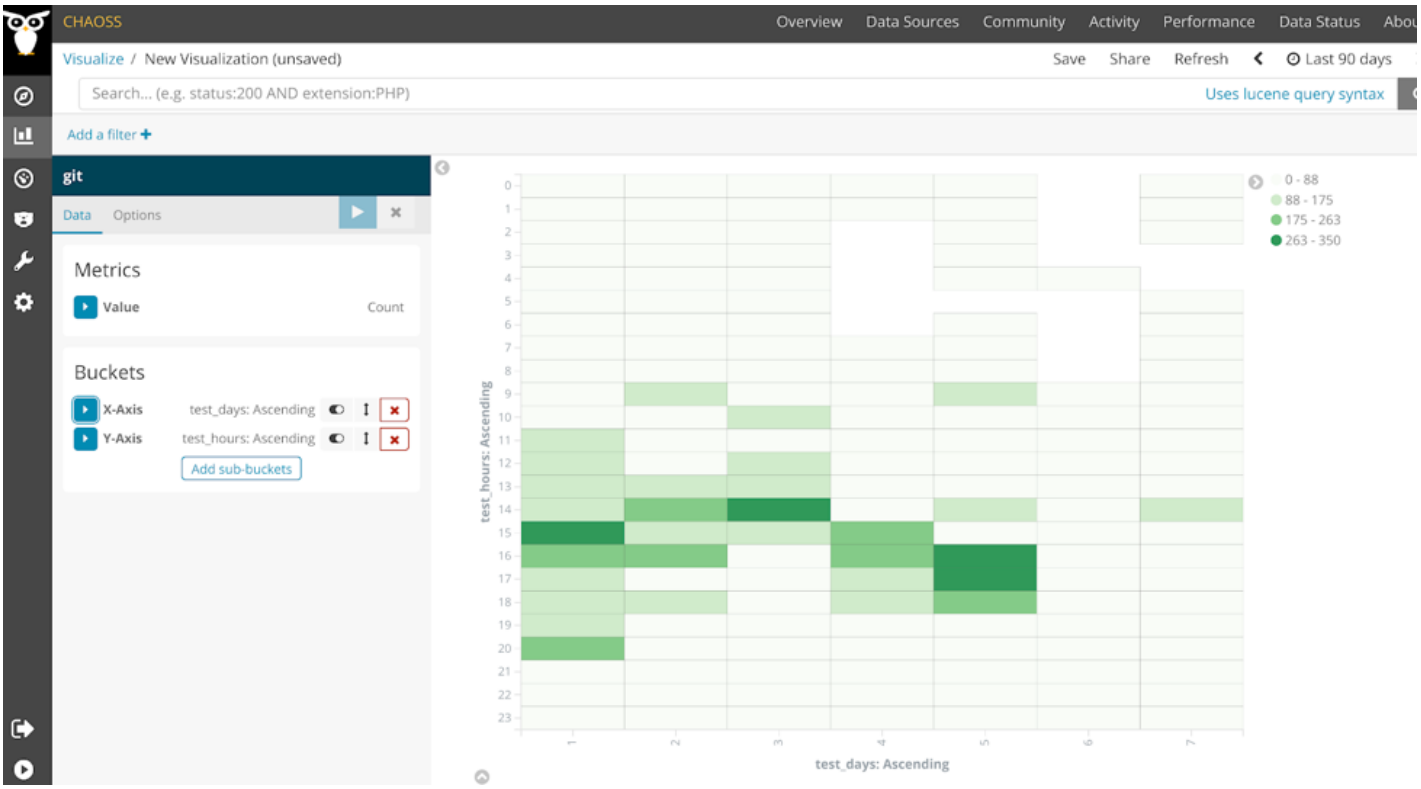
Filters

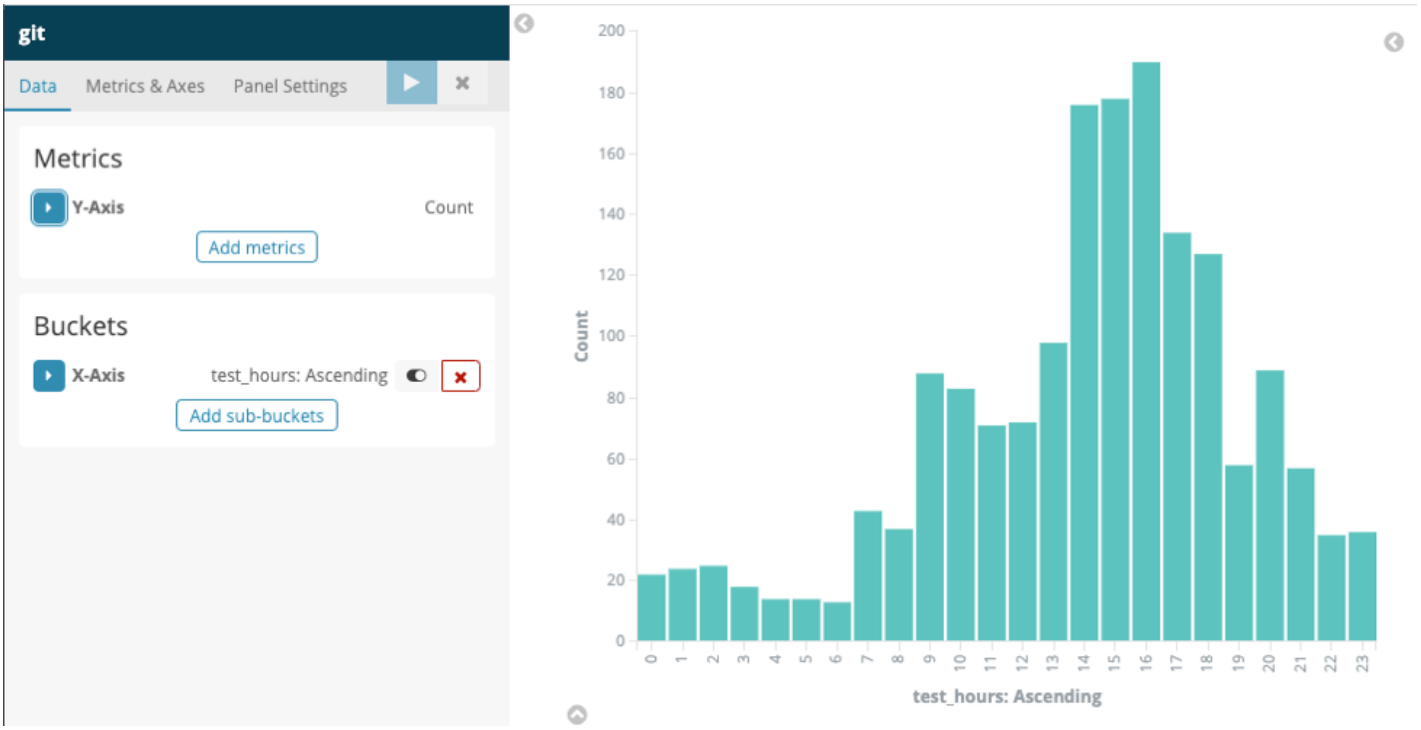
- Individual by Organization
- Aggregation of time by UTC time
  - Can show what times across the globe contributions are made; when the project is most active.
- Aggregation of time by local time
  - Can show what times of day in their local times they contribute. Conclusions about the If contributions are more during working hours, or if contributions are more during evening hours.
- Repository ID

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

1) Activity Dates and Times

References

Coordinated Universal Time

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.

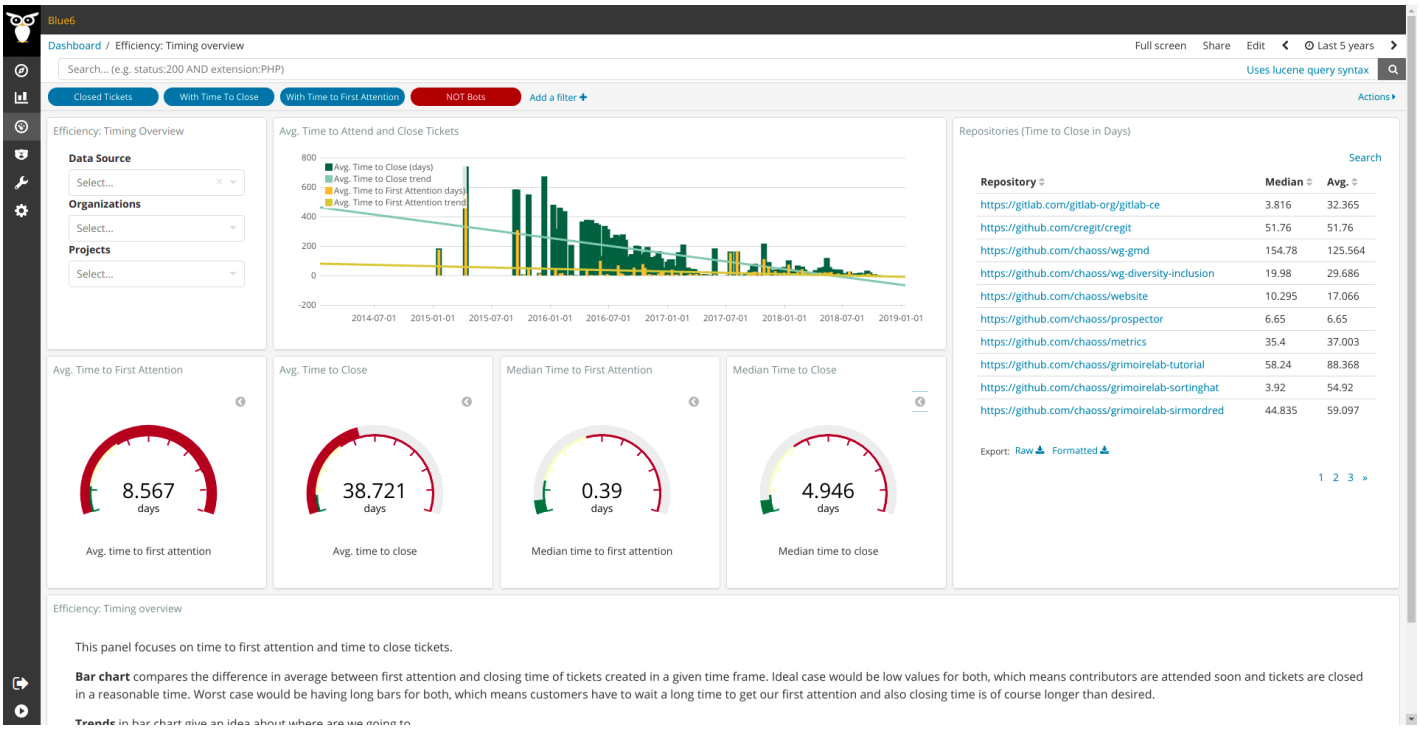
Implementation

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

Visualizations



Tools Providing the Metric

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

References

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:

- 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
- Programming languages of the project
- Role or function in project

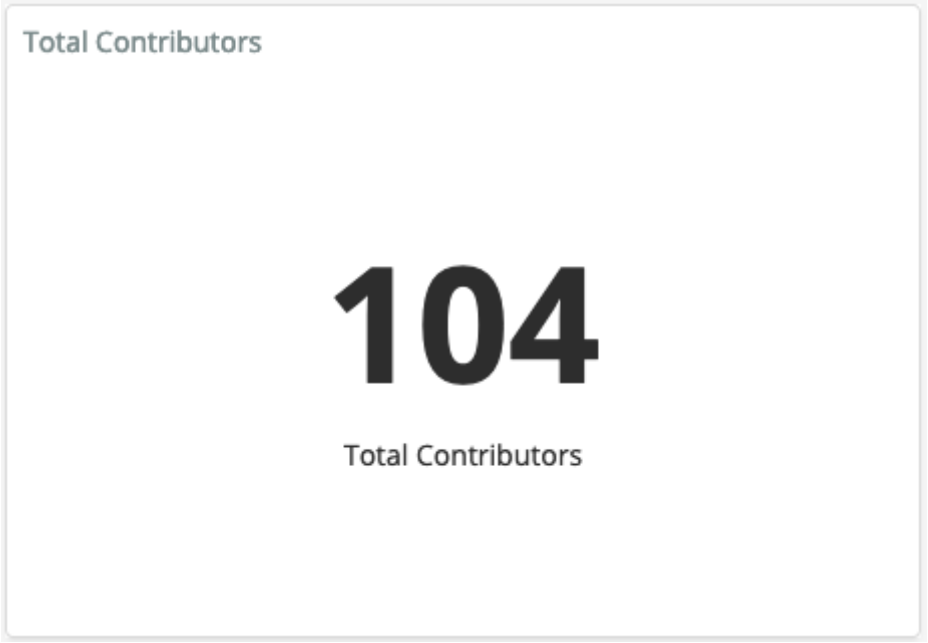
Visualizations

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

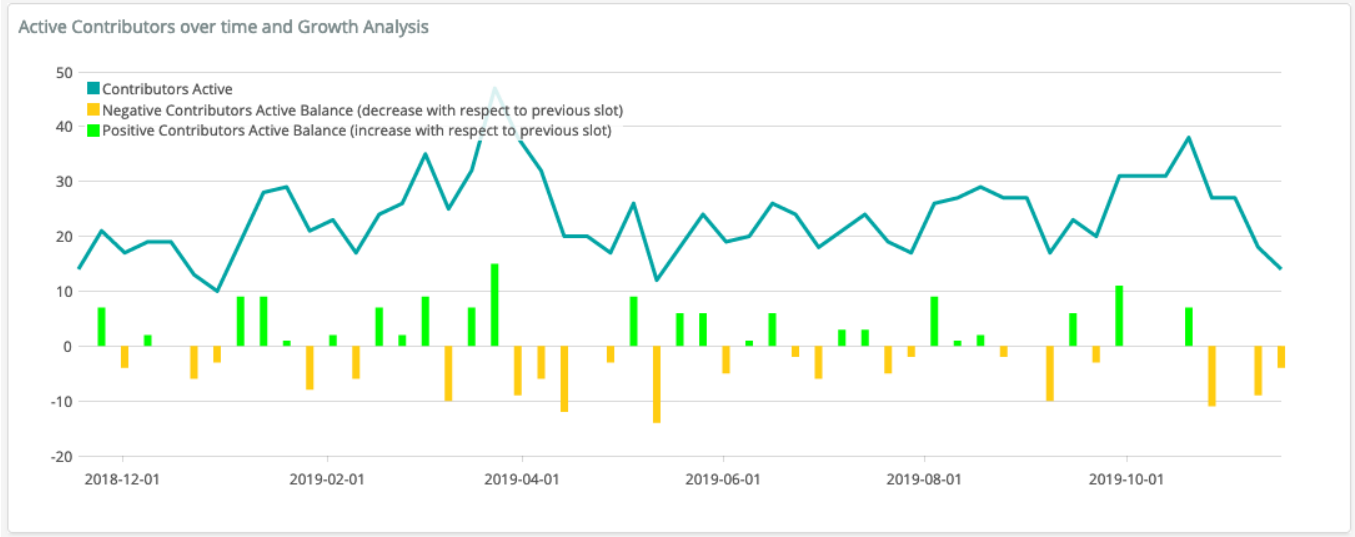
Lines of code added by the top 10 authors

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

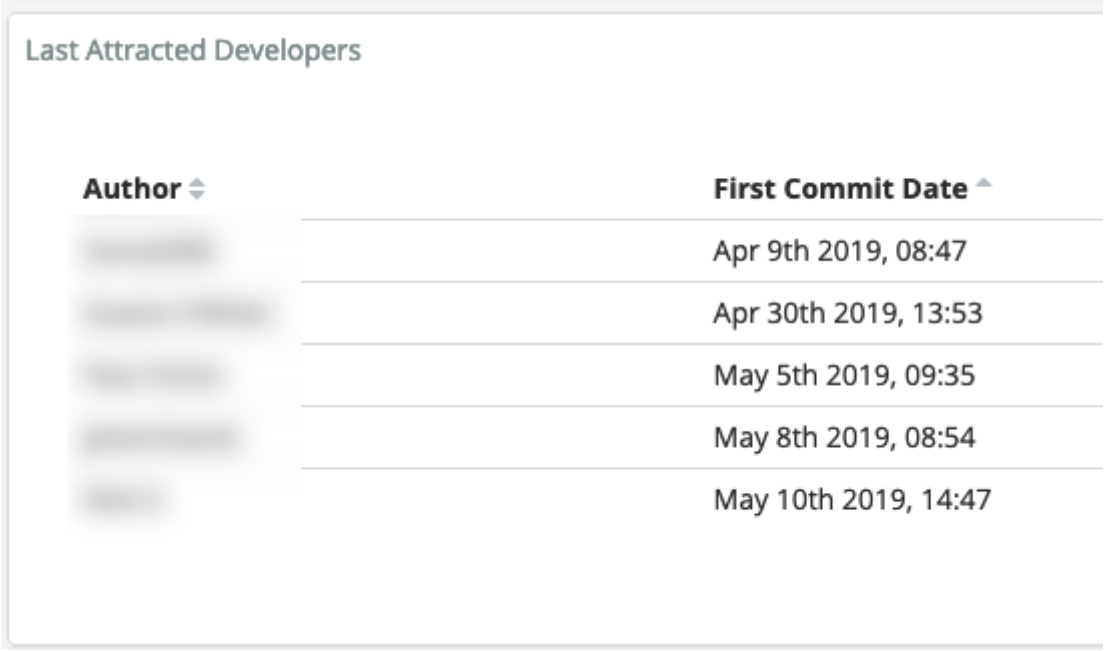
2. Summary number of contributors



3. Change in the number of active contributors over time



4. 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

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

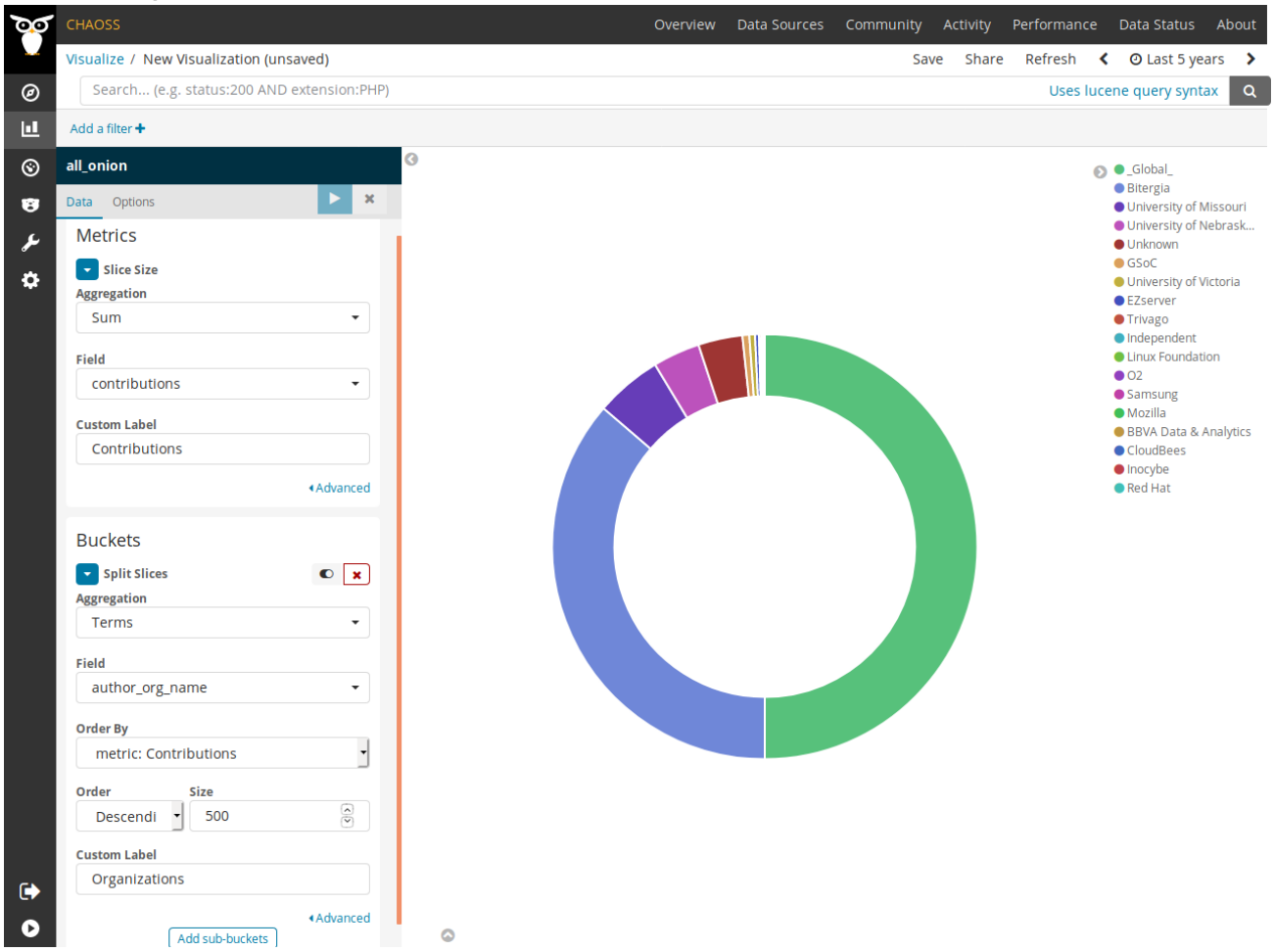
        - Metrics Slice Size: `Sum` Aggregation, `contributions` Field, `Contributions`

          Custom Label
        - Buckets Split Slices: `Terms` Aggregation, `author_or_name` Field,

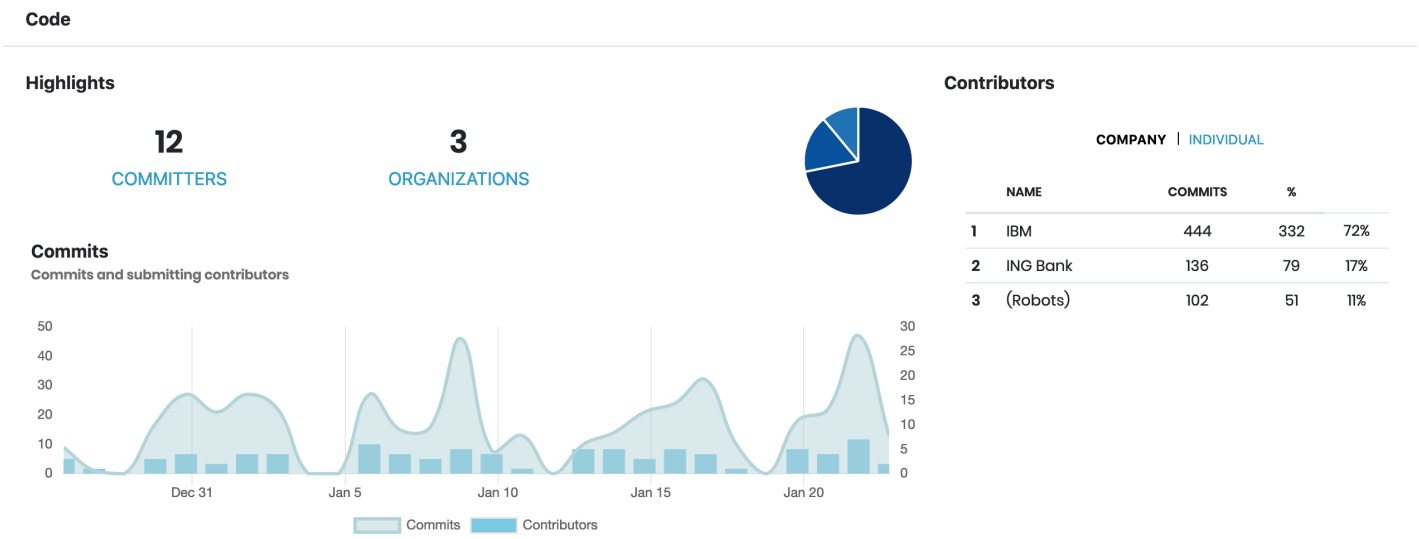
          `metric: Contributions` Order By, `Descending` Order, `500` Size,

          `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

Attendees Demographics

Question: How diverse and inclusive are the attendees?

Description

Attendee demographics help indicate the potential for different viewpoints and broader perspectives at an event.

Objectives

Determine if attendees are from diverse backgrounds. Determine if the diversity is shared across different event spaces like sessions and tracks. Help retain attendees from diverse backgrounds for future events.

Implementation

Filters

- demographics: Some subjective questions should be analyzed in light of the responder's demographics. Everyone has a different perspective. Investigating the responses for each group of demographics can indicate whether some demographics feel less included than the average.
- Keynotes, sessions, and tracks

Data Collection Strategies

- Interview attendees to understand more about why the event did or did not meet their diversity and inclusion expectations.
    - Interview question: What can this event do to improve the diversity and inclusion at this event?
    - Interview question: What are some examples of how this event met, exceeded, or fell short of your diversity and inclusion expectations?
- Survey speakers and attendees to learn to what extent the event met their diversity and inclusion expectations. Sample questions include:

    - Likert scale [1-x] item (ask all attendees): How well did the event meet your diversity and inclusion expectations?
    - Likert scale [1-x] item (ask all attendees): How was your diversity and inclusion experience at this event?
- Quantify the demographics of attendees.

    - Use registration data for attendee demographics (if available).
    - Use a survey to gather attendee demographics. (For example, using the Open Demographics questions)

References

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.

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

keynotes

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.

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_
- 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

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

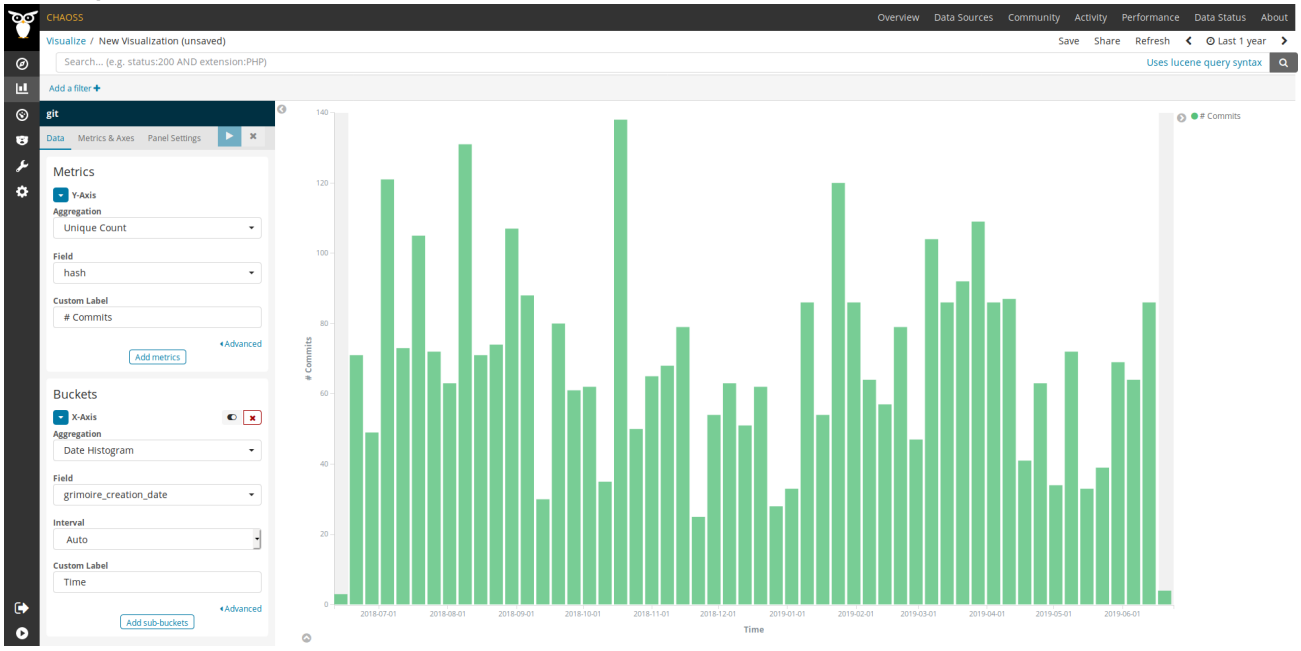
These could be represented as bar charts, with time running in the X axis. Each bar would represent a code changes during a certain period (eg, a month).

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

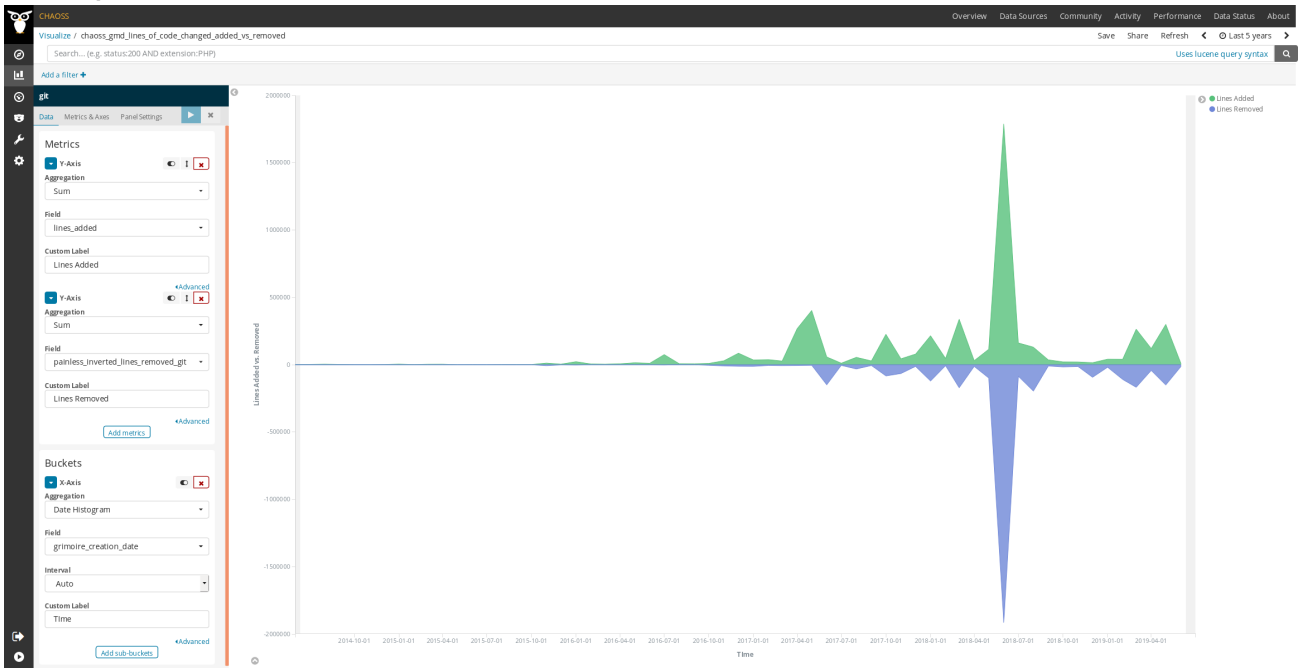
These could be represented as bar charts, with time running in the X axis. Each bar would represent a code changes during a certain period (eg, a month).

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

Reviews Accepted

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

Description

Reviews are defined as in Reviews. Accepted reviews are those that end with the corresponding changes finally merged into the code base of the project. Accepted reviews 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 code review usually corresponds to a single commit.

Objectives

- Volume of coding activity.
  Accepted code reviews are a proxy for the activity in a project. By counting accepted code reviews 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 reviews during the period.
- Ratio. Ratio of accepted reviews over total number of reviews during that period.

Parameters:

- Period of time. Start and finish date of the period during which accepted reviews 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 reviews 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.
  - Add a sample visualization for GitHub Pull requests to any GrimoreLab Kibiter dashboard following these instructions:
    - Create a new `Timelion` visualization.
    - Select `Auto` as Interval.
    - Paste the following Timelion Expression:

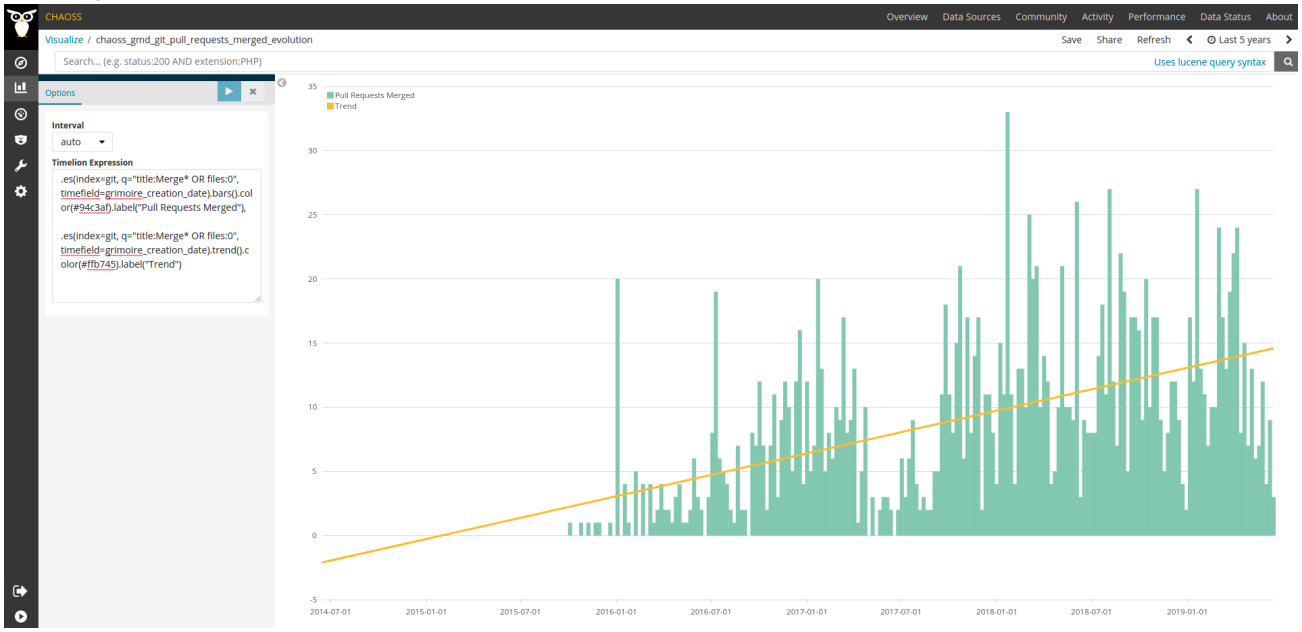
```
.es(index=git, q="title:Merge* OR files:0", timefield=grimoire_creation_da
```

- The expression, step by step:
  - `.es()` : used to define an ElasticSearch query.
    - `index=git` : use git index.
    - `q="title:Merge* OR files:0"` : heuristic to filter in merges.
    - `timefield=grimoire_creation_date` : time will be based on commit creation date (as our query looks for merge commits, it should be the date in which the merge was effectively done).
  - `.bars()` : draw bars instead of lines.
  - `.color()` and `.label()` : some formatting options.
- If you wish to get also the trend, use this instead (i.e. repeating the same expression twice and calling `trend()` the second time):

```
.es(index=git, q="title:Merge* OR files:0", timefield=grimoire_creation_da
.es(index=git, q="title:Merge* OR files:0", timefield=grimoire_creation_da
```

- As discussed above for GitHub case, sometimes is not easy to identify merges. As you probably noticed, in this example we based our expression on GrimoireLab Git index. Besides, it could be applied to any other similar environment using Git repositories, not only to GitHub.

- Example screenshot:



Data Collection Strategies

Specific description: GitHub

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

But reviews 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 pull request.

In a more difficult scenario, reviews 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 they were accepted, the problem is that if they are included in the git repository manually, the only way of knowing that the pull request was

accepted is finding the corresponding commits in the git repository.

In some cases, projects have policies of mentioning the commits when the pull 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 pull requests not accepted via the web interface. Default: None.

Specific description: GitLab

In the case of GitLab, accepted reviews 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 pull requests not accepted via the web interface. Default: None.

Specific description: Gerrit

In the case of Gerrit, accepted reviews 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

Reviews Declined

Question: What reviews of code changes ended up declining the change during a certain period?

Description

Reviews are defined as in Reviews. Declined reviews 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 reviews in this system.

Objectives

- Volume of coding activity. Declined code reviews are a proxy for the activity in a project. By counting declined code reviews 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 reviews during the period.
- Ratio. Ratio of declined reviews over the total number of reviews during that pariod.

Parameters:

- Period of time. Start and finish date of the period during which declined reviews 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 reviews during a certain period (eg, a month).

Data Collection Strategies

Specific description: GitHub

In the case of GitHub, accepted reviews 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 Reviews Accepted, since it applies here as well.

Mandatory parameters (for GitHub):

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

Specific description: GitLab

In the case of GitLab, accepted 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 merge 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 reviews are defined as "changesets abandoned", as long as they propose changes to source code files.

Mandatory parameters (for Gerrit): None.

References

Reviews Duration

Question: What is the duration of time between the moment a code review starts and the moment it is accepted?

Description

Reviews are defined as in Reviews. Accepted reviews are defined in Reviews Accepted.

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

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

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

Objectives

- Duration of acceptance of contributions processes. Review duration for accepted reviews 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 review duration for all accepted reviews in the considered period of time.

Parameters:

- Period of time. Start and finish date of the period. Default: forever.
  Period during which accepted reviews are considered. An accepted review 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 reviews to change the code during a certain period (eg, 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 cod 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

Reviews

Question: What new review requests for changes to the source code occurred during a certain period?

Description

When a project uses code review 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 proposals are intended to be reviewed by other developers, who may suggest improvements that will lead to the original proposers sending new versions of their proposals, until reviews are positive, and the code is accepted, or until it is decided that the proposal is declined.

For example, "reviews" 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. Reviews are a proxy for the activity in a project. By counting reviews to code changes in the set of repositories corresponding to a project, you can have an idea of the overall activity in reviewing 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 reviews during the period.

Parameters:

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

Visualizations

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

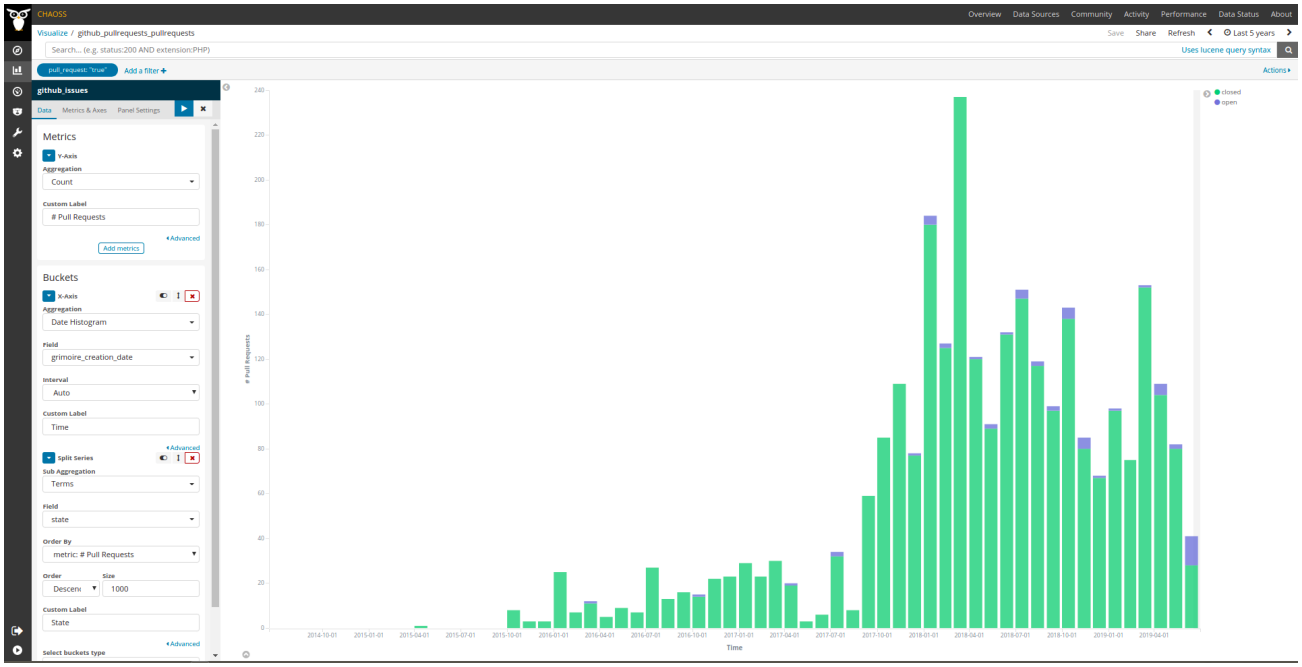
These could be represented as bar charts, with time running in the X axis. Each bar would represent reviews to change the code during a certain period (eg, a month).

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.
  - Add a sample visualization for GitHub Pull requests to any GrimoreLab Kibiter dashboard following these instructions:
    - Create a new `Vertical Bar` chart.
    - Select the `github_issues` index.
    - Filter: `pull_request` is `true` .
    - Metrics Y-axis: `Count` Aggregation, `# Pull Requests` Custom Label.

- X-axis: `Date Histogram` Aggregation, `grimoire_creation_date` Field, `Auto` Interval, `Time` Custom Label.
- Buckets Split Series: `Terms` Sub Aggregation, `state` Field, `metric: # Pull Requests` Order By, `Descending` Order, `1000` Size, `State` Custom Label. Notice this visualization is based on Pull Requests creation date, so items are counted at the date they were created and its state, as set here, would be their current state at the moment of visualizing the data, e.g. `n` Pull Requests created at a give time range are currently `open` or `closed`.

- Example screenshot:



Data Collection Strategies

Specific description: GitHub

In the case of GitHub, a review 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 review is defined as a "merge 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 merge request was submitted.

Specific description: Gerrit

In the case of Gerrit, a review 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 review can be defined (for considering it in a period or not) as the date in which the code review was started by submitting a patchset for review.

References

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

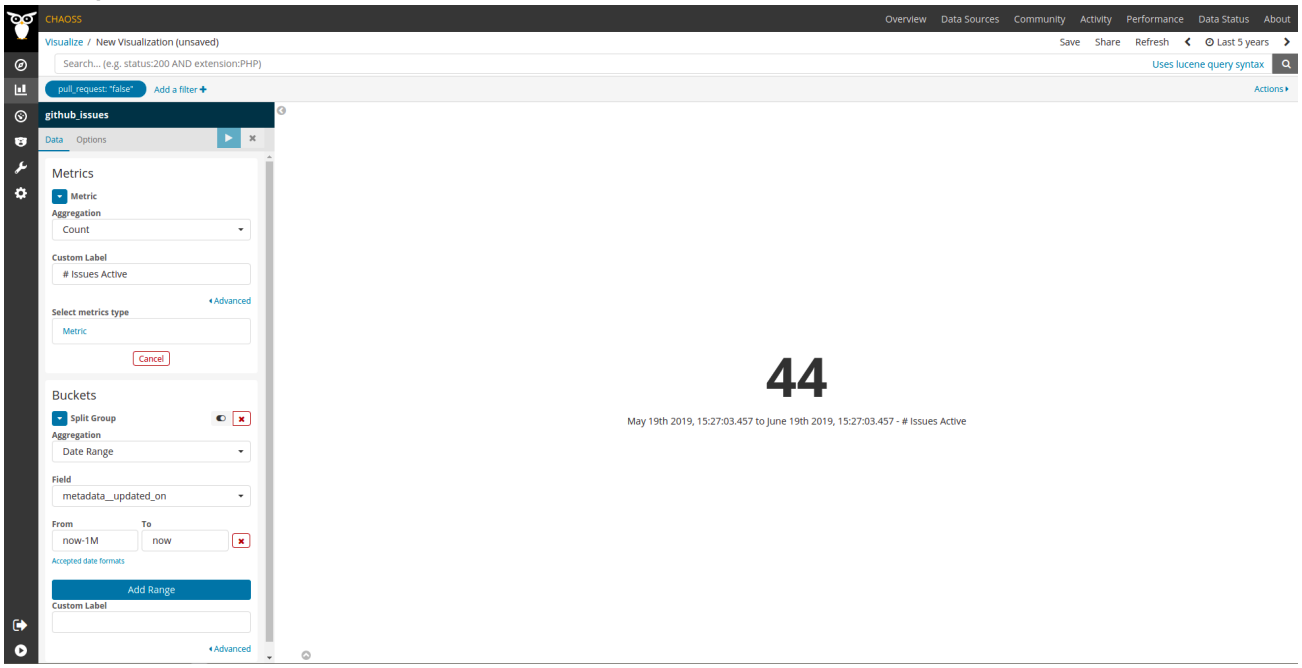
These could be grouped by applying the previously defined filters. These could be represented as bar charts, with time running in the X axis. Each bar would represent proposals to change the code during a certain period (eg, a month).

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 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. 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.

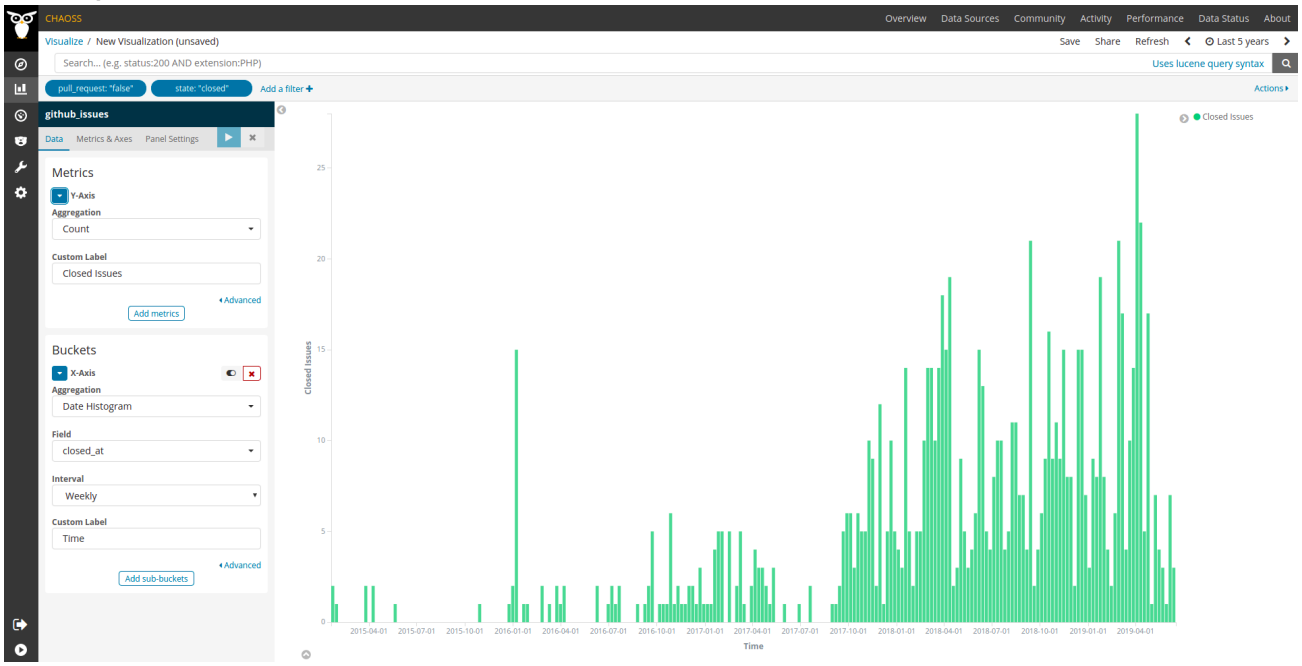
Tools Providing the Metric

- 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, active issues are defined as "issues that are closed".

Specific description: Jira

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

Specific description: Bugzilla

In the case of Bugzilla, active 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:

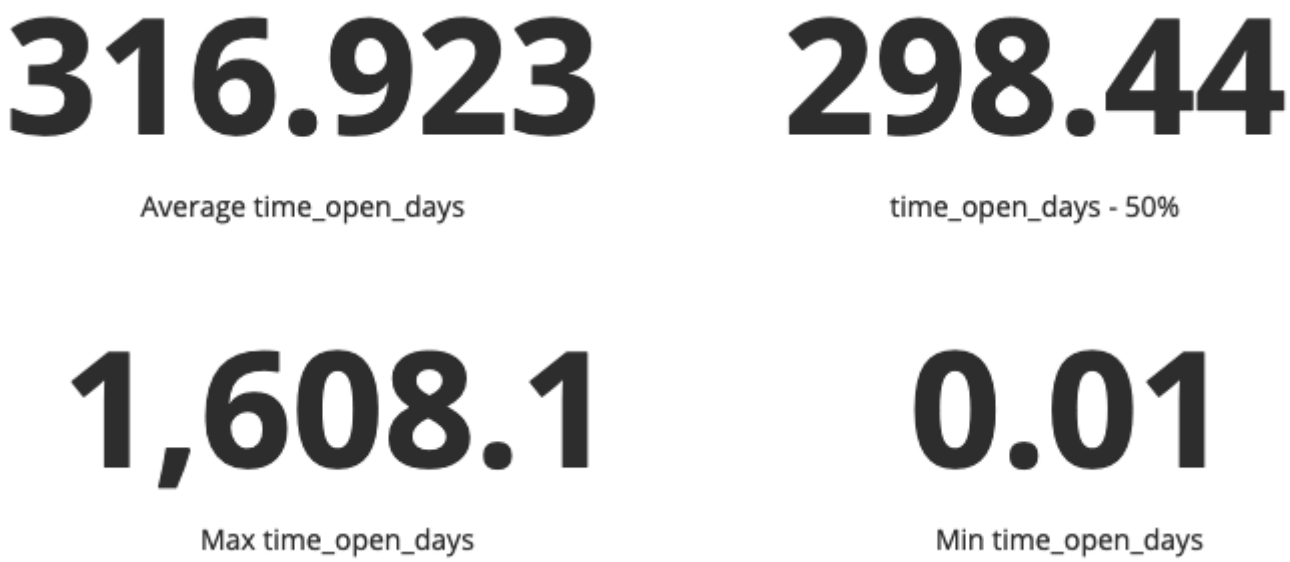
- Period of time. Start and finish date of the period during which open issues are considered. Default: forever (i.e., the entire observable period of the project's issue activity).

Filters

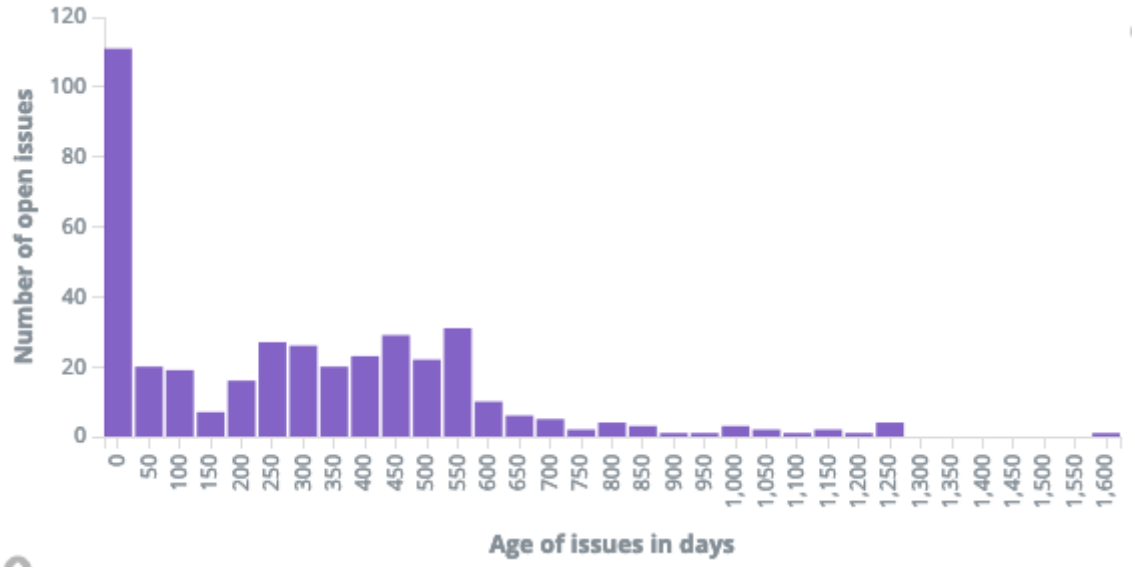
- Module or working group
- Tags/labels on issue

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:

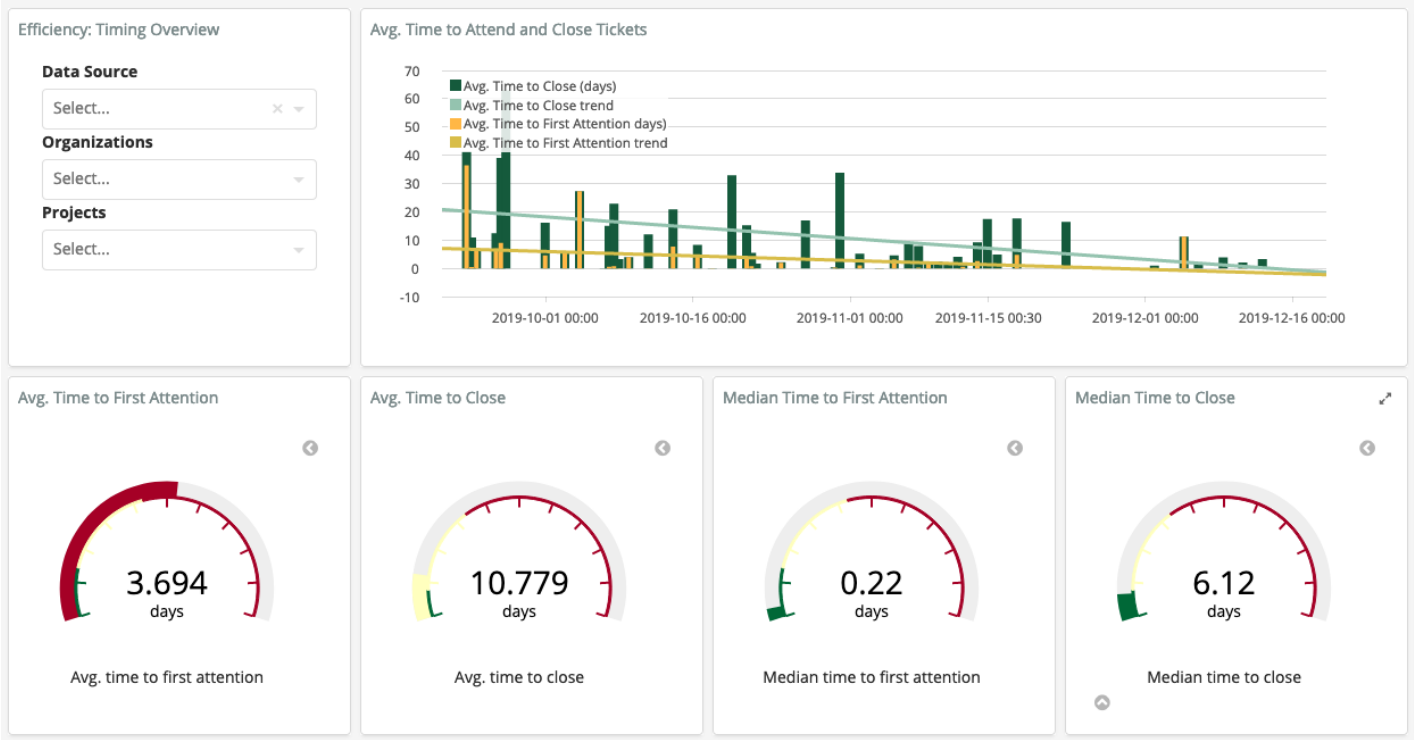
- Period of time. Start and finish date of the period. Default: forever.
- Period during which responses are counted.

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
- 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.
    - 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.
- 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

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

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 beg
```

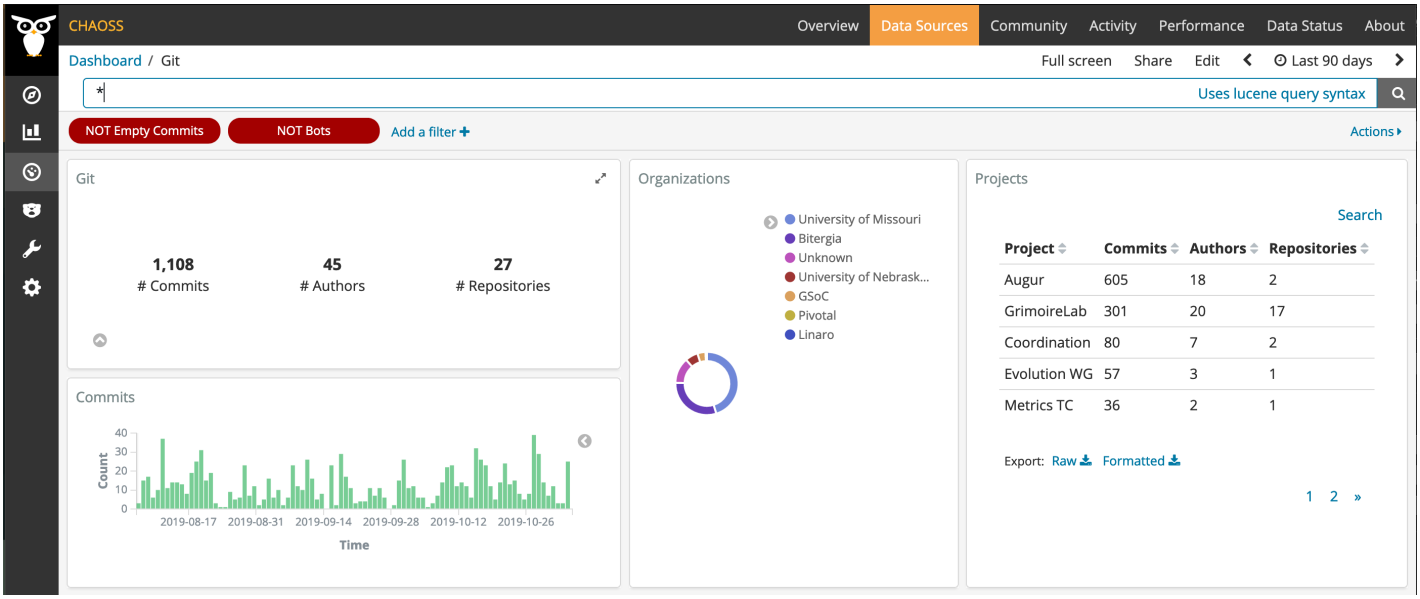
. For example, I may want to know how many distinct people have committed code to a project in the past 18 months. `Committers` reveals the answer.

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

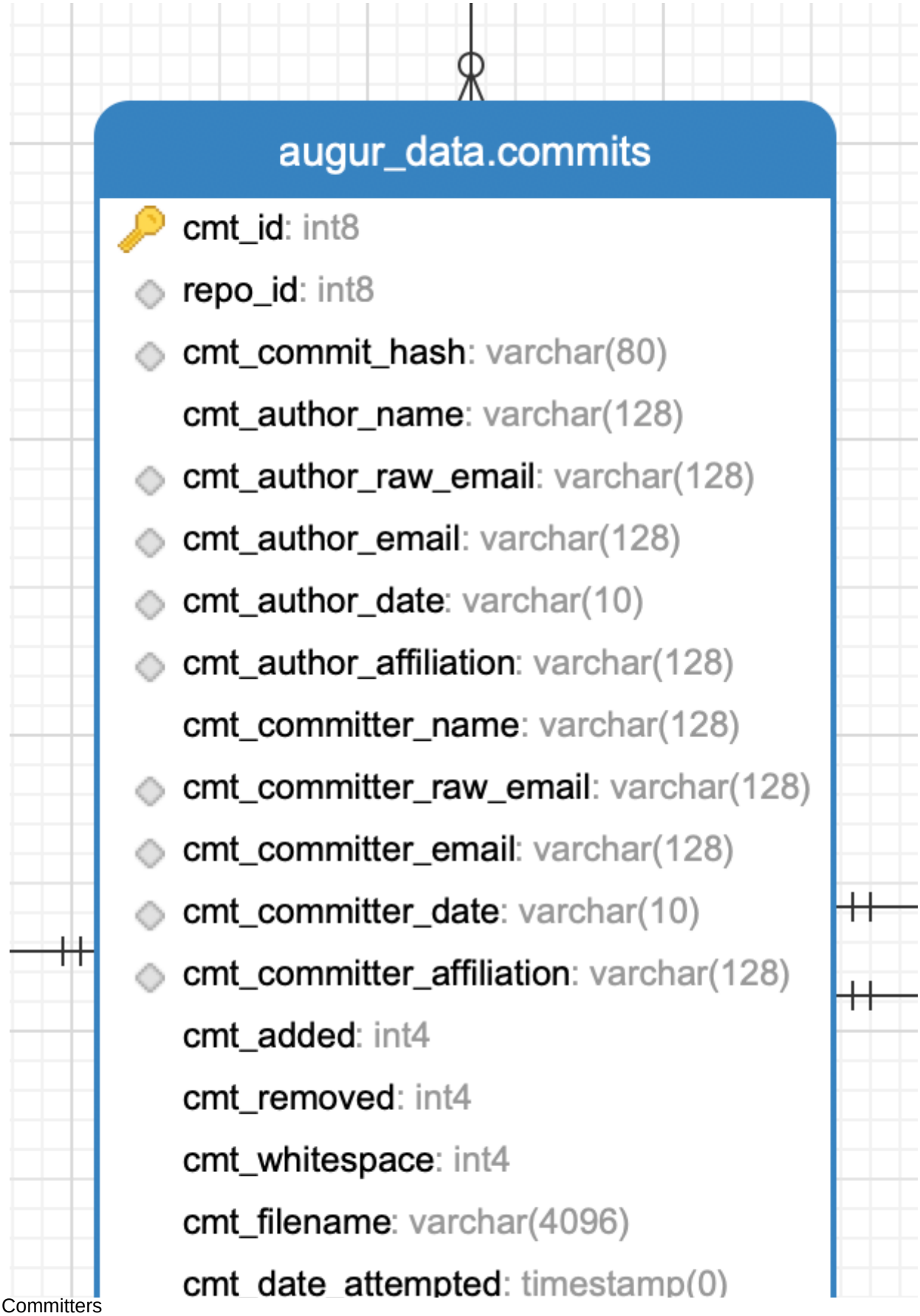


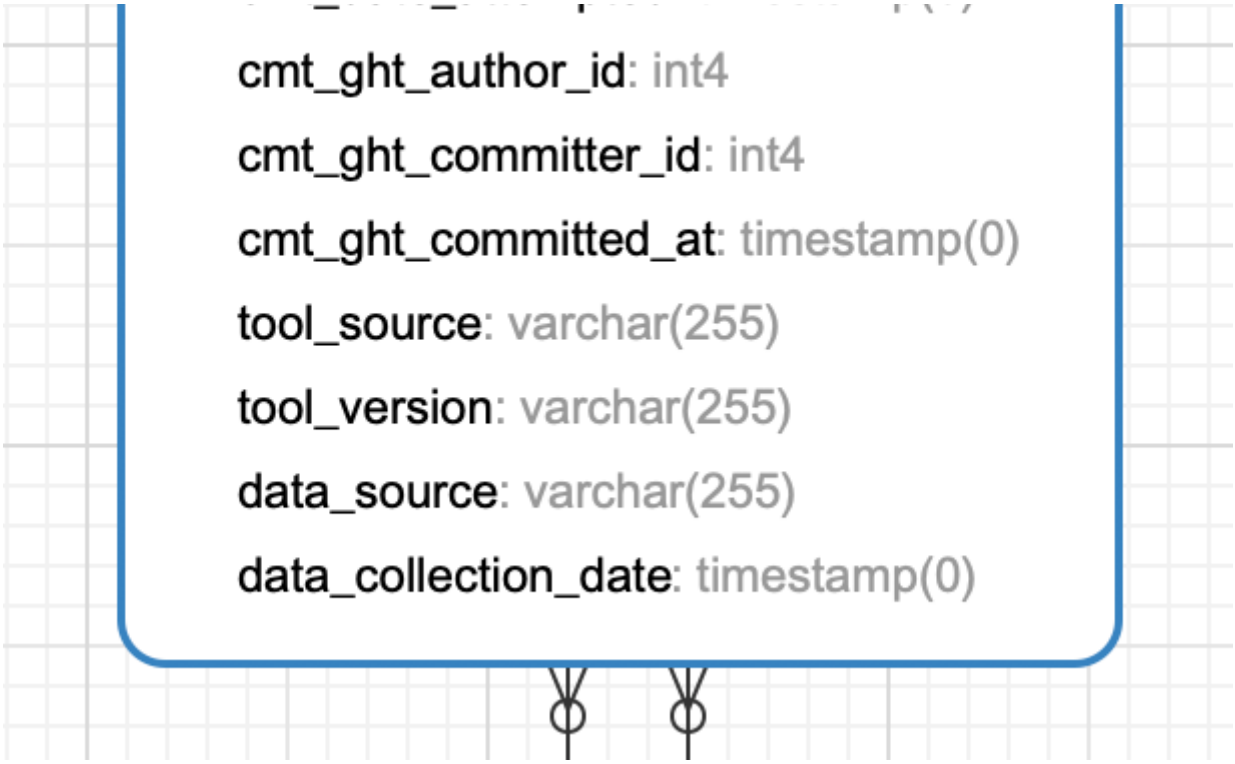
Filters

- 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:

```
SELECT
    cmt_author_name,
    COUNT ( * ) AS counter
FROM
    commits
WHERE
    repo_id = 22159
GROUP BY
    cmt_author_name
ORDER BY
    counter DESC
```

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 percentile calculation. If we have 8 organizations who each contribute the following number of commits to a project: `1000,202,90,33,332,343,42,433`, then we can determine the elephant factor by first

identifying the 50th percentile of total commits for all the companies.

Summary: 50th percentile = 267, so the elephant factor is 4.

Full Solution:

1. Arrange the data in ascending order: 33, 42, 90, 202, 332, 343, 433, 1000
2. Compute the position of the pth percentile (index i):
    1. `i = (p / 100) * n), where p = 50 and n = 8`
    2. `i = (50 / 100) * 8 = 4`
3. The index i is an integer ⇒ the 50th percentile is the average of the values in the 3th and 4th positions (202 and 332 respectively)
4. Answer: the 50th percentile is (202 + 332) / 2 = 267, therefore the `elephant factor = 4.`

Filters

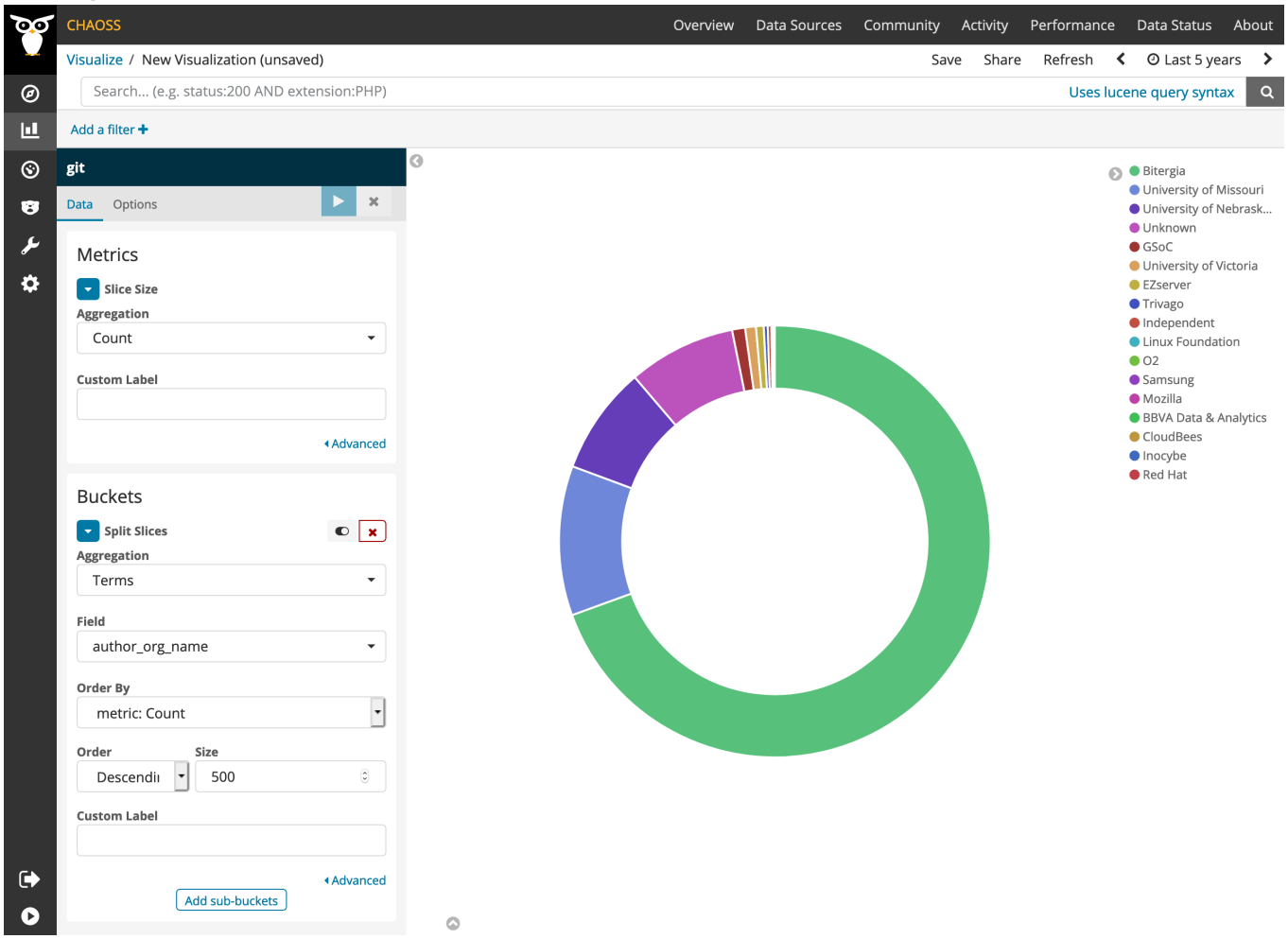
- Time: Reasonably the Elephant Factor will change if one takes a snapshot of any prior time period, so the elephant factor over the life of a product may misrepresent the current level of organizational diversity supported by the project.
- Repository Group: Many open source projects include multiple 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/

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 onFoundations 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

License Count

Question: How many different licenses are there?

Description

The total count of identified licenses in a software package where the same license can be counted more than once if it is found in multiple source files. This can include both software and document related licenses. This metric also provides a binary indicator of whether or not the repository contains files without license declarations. This metric is a "complexity of licensing case" flag for open source managers. For example, the ideal case would look like the table below:

Number of Licenses	Files Without Licenses
1	FALSE

A more common case would require references to Licenses Declared and License Coverage metrics, in the situation reflected in this table:

Number of Licenses	Files Without Licenses
11	TRUE

Objectives

The most simple case for an IT Manager overseeing the acquisition and management of open source software or an Open Source Program Office or community manager delivering open source software to the marketplace is to have a single license type declared across all files. This metric will illustrate quickly and visibly if there is one license or more than one; and the larger the number, the more complex the considerations grow for decision makers.

The second aspect of this metric is the binary indicator of whether or not the repository (package) includes files that do not have license declarations.

Implementation

Tools Providing the Metric

1. Augur

License count can be found on any Augur risk page under the section "Licenses Declared". The number of rows in the table is the number of licenses.

References

1. https://spdx.org/
2. https://www.fossology.org

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.

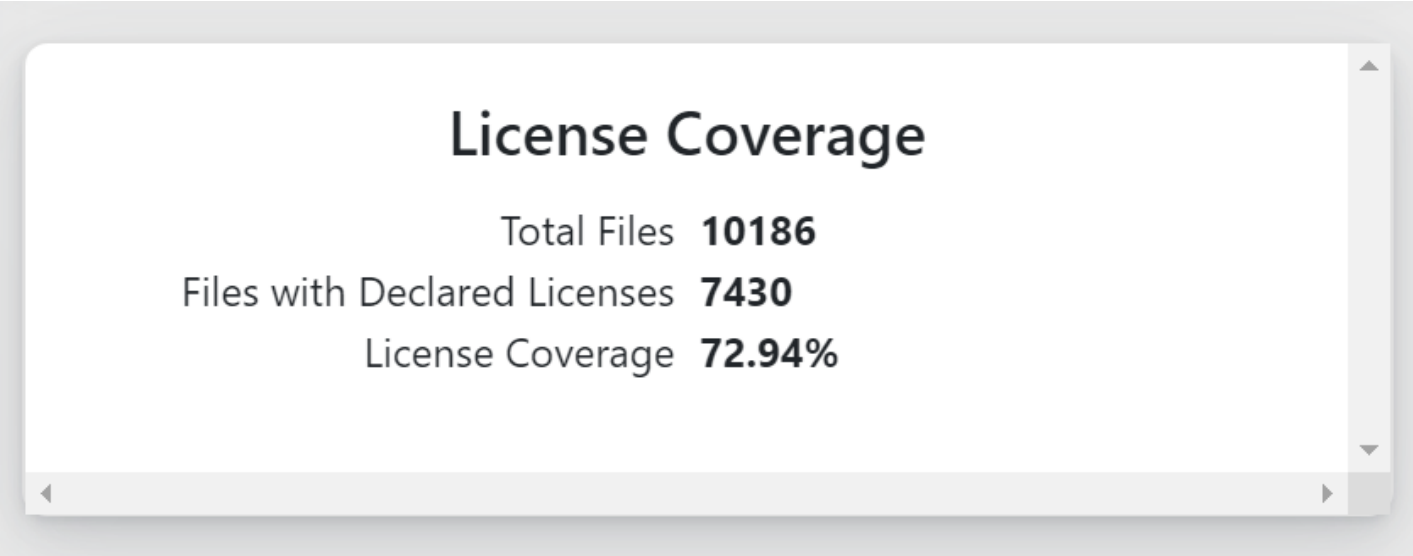
Implementation

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:


```
▼ 2:
    Total Files:                "5168"
    License-Declared Files:     "926"
    Percent Total Coverage:     "17.92%"
```


Tools Providng 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".

2. 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 declared 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."

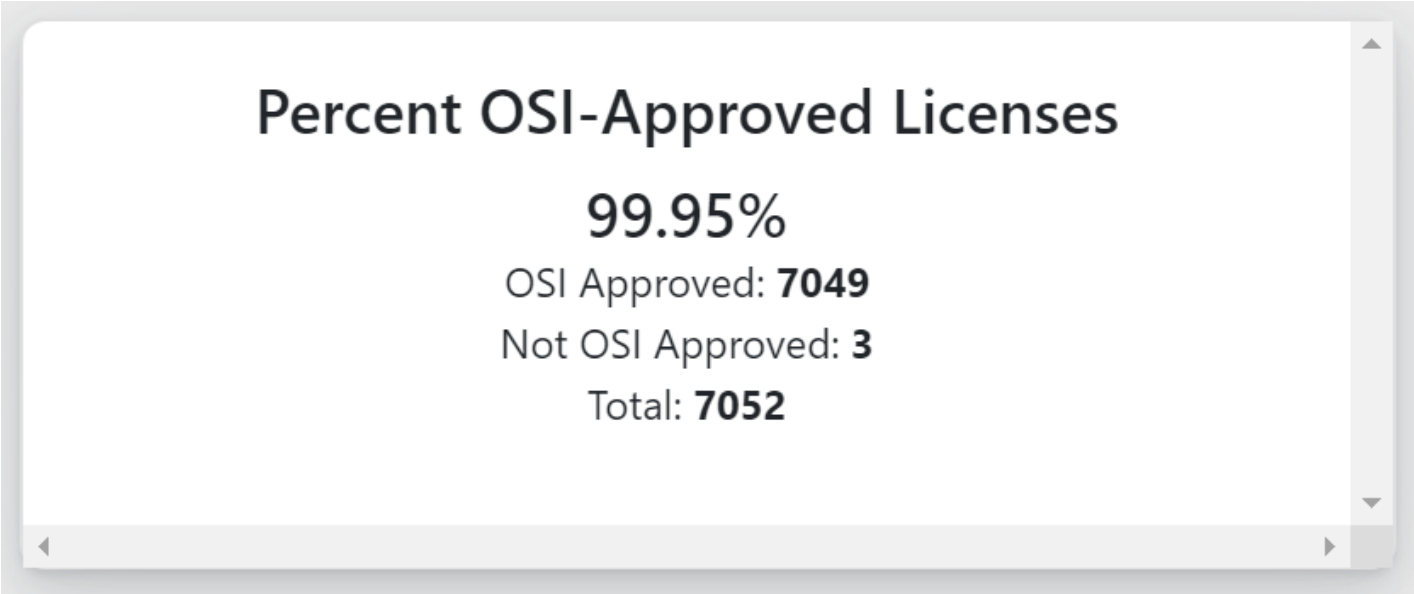
Objectives

Identify whether a project has licenses present which do not conform to the 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 on the OSI license list.

Resources

- OSI license page
- SPDX License List

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

Social Currency Metric System (SCMS)

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

Description

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 Currency Metric System 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.

The Social Currency Metrics System 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 Currency Metrics System continually measures the sentiment from those interactions. It illustrates the reputation and level of trust between community members and leaders.

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

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 SCMS 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.
3. **Analyze the Communication Traces** -- Community sentiment is analyzed in the SCMS 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.
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

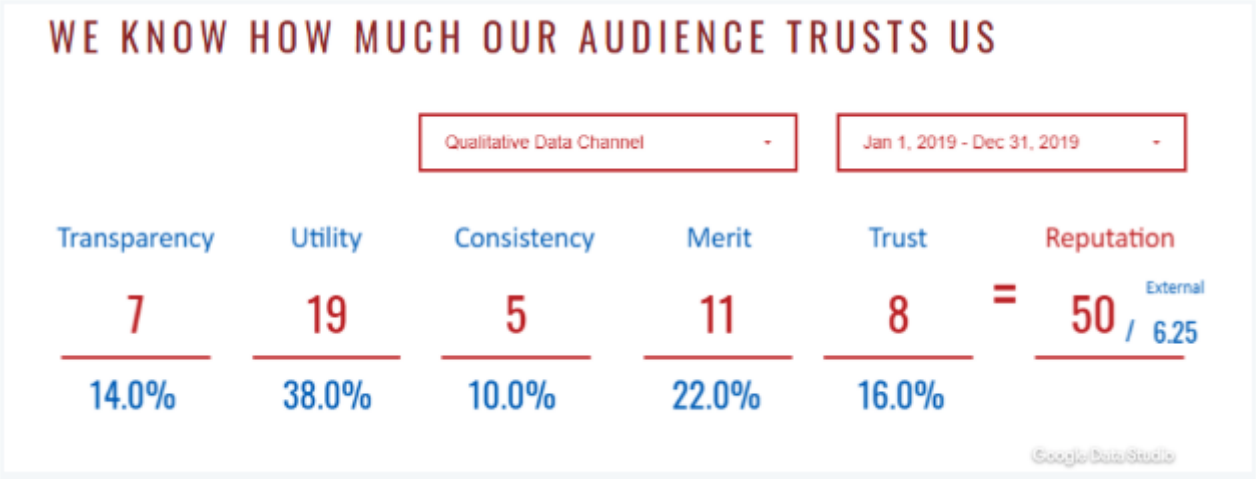
results produce an easy to observe dashboard of trends. Share analysis with team members.
5. **Benchmark, Set Goals & Predict Future Growth** -- After getting enough data to form a benchmark, take stock of where your community stands. What are its strengths and weaknesses? What actions can be taken to 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. **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.

Filters

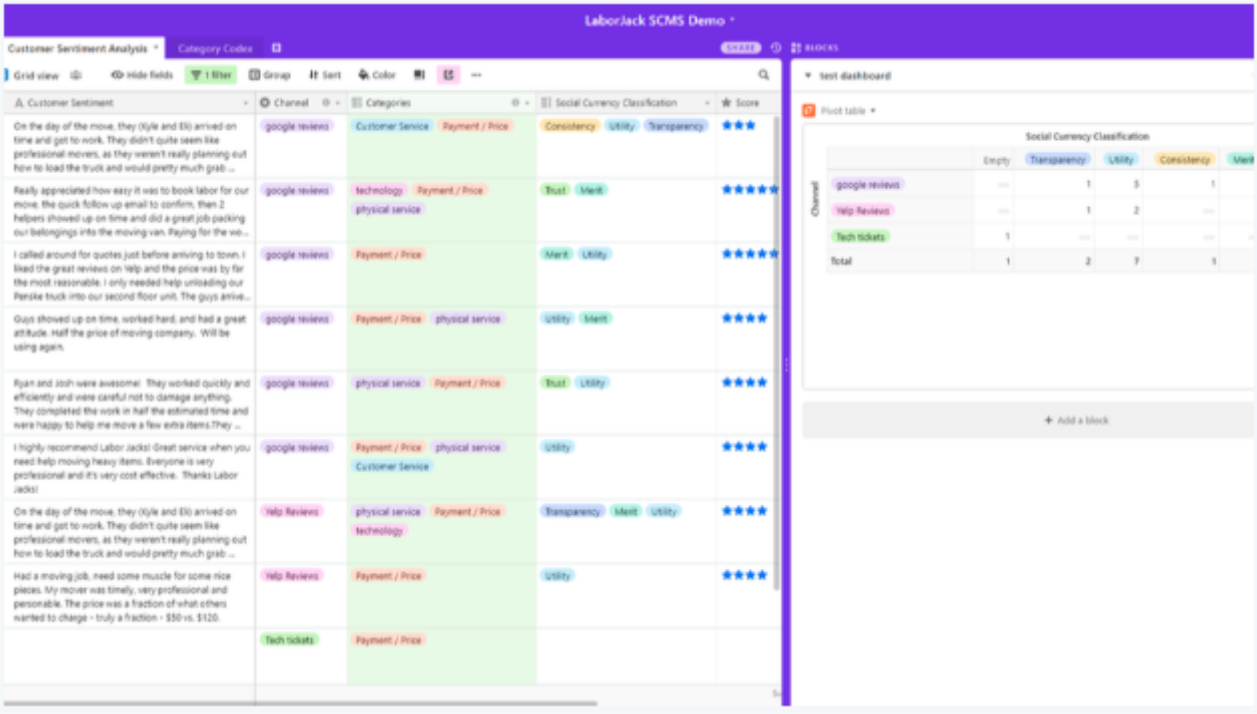
1. **Channel**: Sort by where the data was collected from.
2. **Tag**: Show data based on what codex tags were used to identify sentiment in comments.
3. **Time**: Show trends in the data over time and pull specific data-sets.
4. **Most impactful comments**: Sort and filter by flags that can be placed 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

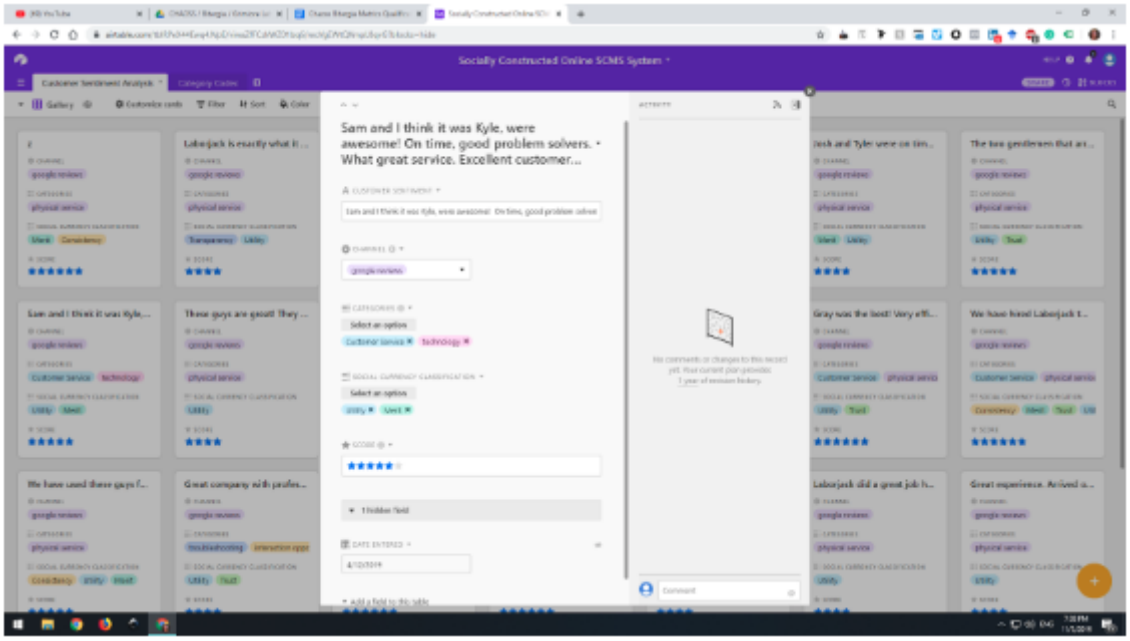
Dashboard visualizing the aggregate metrics:



Example SCMS 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 SCMS 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:

Data Points	Description
Date of entry	Date data was imported to SCMS 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

Data Points	Description
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:

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 personalble.	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.

Category Term	Definition	When to use	When not to use
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	Do people believe these things strongly enough to warrant conversation or action?		
EXTERNAL REPUTATION	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:

"A concept applies to B population ONLY SO FAR AS C limitation."

Data Collection Strategies

Community member comments are available from trace data. The SCMS 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, instanct 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)

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 increasly 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 visualizatoin 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

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:

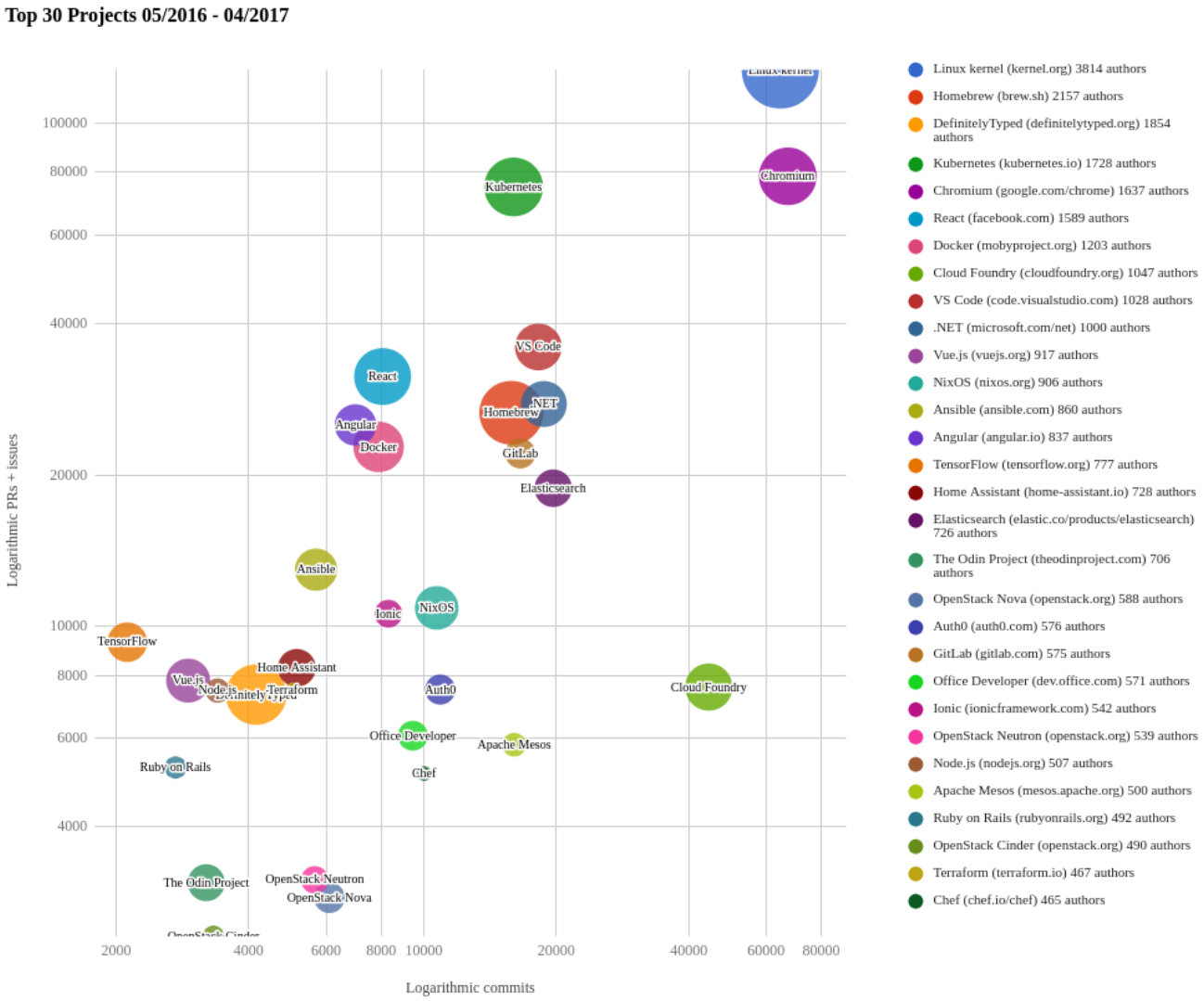
- issues closed
- number of reviews
- # of code changes
- # of committers

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 Providng 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

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 for their

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.

The MIT License