Metrics

Release 201908

https://chaoss.community/metrics

CHAOSS Metrics (Release 201908)

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

- Organizational Affiliation

Diversity and Inclusion WG

- Event Diversity
- Governance
- Leadership

Evolution WG

- Code Development

Risk WG

- Business Risk
- Code Quality
- Licensing
- Transparency

Value WG

- Labor Investment
- Living Wage

Important Dates for Release 201908

Release Freeze: June 21st, 2019
Candidate Release: June 24th, 2019
Comments Close: July 24th, 2019
Release Date: August 5th, 2019

Common Metrics

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

Focus Area - Organizational Affiliation

Goal:
Understand organizational engagement with open source projects.

Metric	Question
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	Question
Diversity Access Tickets	How are Diversity Access Tickets used to support diversity and inclusion for an event?
Code of Conduct at Event	How does the Code of Conduct for events support diversity and inclusion?
Family Friendliness	How does enabling families to attend together support diversity and inclusion of the event?

Focus Area - Governance

Goal:
Identify how diverse and inclusive our governance is.

Metric	Question
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 our community leadership is.

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

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

Goal:
Activity - Learn about activity involved in changing (or adding) code.
Question: How many **changes** are happening to the source code, during a certain time period?

Metric	Question
Code Changes	What changes were made to the source code during a specified period?
Code Changes Lines	What is the sum of the number of lines touched (lines added plus lines removed) in all changes to the source code during a certain period?

Question: How many **reviews** to proposed changes to the source code are happening during a certain time period?

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

Question: How many **issues** related to the source code are happening during a certain time period?

Metric	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?

Goal:
Efficiency - Learn how effective new code is merged into the code base.
Question: How efficient is the project in **reviewing** proposed changes to the code, during a certain time period?

Metric	Question
Reviews Accepted	How many accepted reviews are present in a code change?
Review Duration	What is the duration of time between the moment a code review starts and moment it is accepted?

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

Focus Area - Code Quality

Goal:
Understand the quality of a given software package.

Metric	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	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?

Focus Area - Transparency

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

Metric	Question
Bill of Materials	Does the software package have a standard expression of dependencies, licensing, and security-related issues?

Value

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

Focus Area - Labor Investment

Goal:
Estimate the labor investment in open source projects.

Metric	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	Question

Metric	Question
Organizational Project Skill Demand	How many organizations are using this project and could hire me if I become proficient?

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.

Strategies

- 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.

Success Metrics

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.

Known Implementations

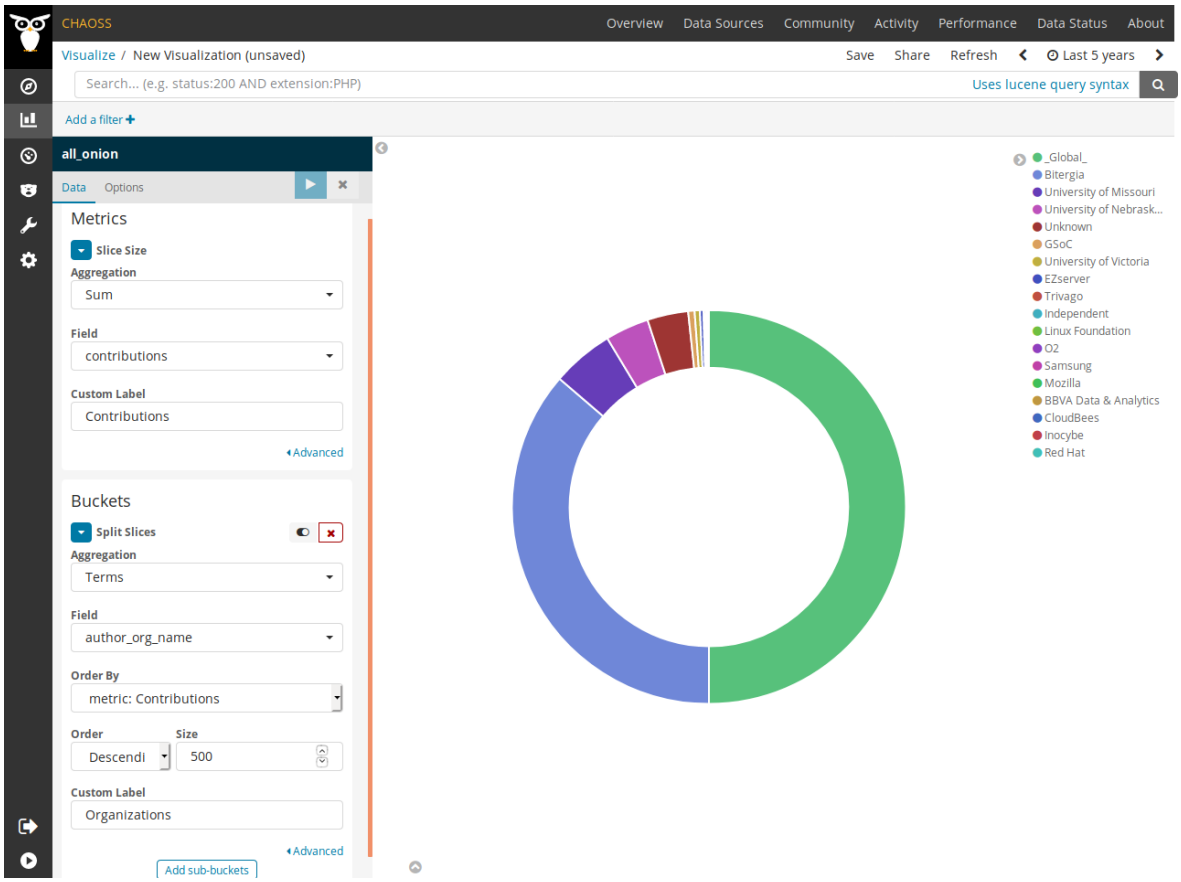
- GrimoireLab supports organizational diversity metrics out of the box. The GrimoireLab SortingHat manages identities. The GrimoireLab Hatstall user interface allows correcting organizational affiliation of people and even recording affiliation changes.
  - View an example visualization on the CHAOSS 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



Resources

- 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

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.

Strategies

- Observe website for availability and pricing of diversity access tickets.
- Interview organizers to understand where and how the diversity access tickets were allocated (some organizers offer tickets directly to individuals or local groups focused on underrepresented segments, which cannot always be found by looking at the website).

- Survey participants about perception of diversity access tickets.
- Track attendance numbers based on diversity access tickets.

Success Metrics

Qualitative

- Interview organizers:
  - How were the diversity access tickets allocated?

Quantitative

- 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 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.

Known Implementations

Resources

- https://diversitytickets.org/

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.

Strategies

- Observe event website for a code of conduct.
- 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?
- Observe whether a code of conduct is posted at an event.
- Survey participants about the code of conduct.

Success Metrics

Qualitative

- 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?

Quantitative

- Observe event website for a code of conduct.
- 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)
- Observe whether a code of conduct is posted at an event.
- 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]

Known Implementations Resources

- https://github.com/python/pycon-code-of-conduct/blob/master/Attendee%20Procedure%20for%20incident%20handling.md
- https://pycon.blogspot.com/2018/04/code-of-conduct-updates-for-pycon-2018.html
- https://geekfeminism.wikia.org/wiki/Conference_anti-harassment

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

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.

Strategies

- Analyze conference website.
- Interview conference staff.
- Survey conference participants

Success Metrics

Qualitative

- 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.

Quantitative

- 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?

Known Implementations

Resources

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.

Sample Objectives

- A project member wants to know that a community takes diversity and inclusion seriously.
- A new contributor wants to evaluate a community for diverse and inclusive design, before investing any time in that project.
- A person wants to identify whether or not their demographic is protected prior to participating in a project.
- A grant, or award program wants to ensure that a project/org takes diversity and inclusion seriously by requiring the existence of a Code of Conduct.
- An open source project, or other technology company/org wants to ensure their partnerships, and allies take diversity and inclusion seriously as it can impact their own reputation and community health.
- A project member who experiences a violation needs to understand how to report the behavior.
- A project member, or event participant wants to see that the code of conduct is being enforced; and not just the potential for enforcement.

Sample 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.

Sample Success Metrics

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]

Known Implementations

- Mozilla Code of Conduct Assessment Tool

Resources

- 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

- Increase the number and diversity of new contributors
- Increase the level of contributions from each contributor, from a quantitative or qualitative perspective
- Increase the number of and diversity of contributors, encouraging them to grow into new roles with increasing responsibility within a community, e.g. from newcomers to contributing more extensively and becoming core reviewers, and ultimately, growing into maintainer roles within a community, or those who are more seasoned and moving into roles within new communities
- Increase the number of advocates/evangelists for a project
- Increase the number of paid mentors and mentees
- Cultivate a culture of inclusivity through identifying and supporting mentors and mentees

Strategies

- Ask Foundation members about existing formal or informal mentorship

programs
- Look for people informally mentoring new contributors (e.g., those reviewing newcomers in the code review process)
- Survey mentors and mentees
- Interview mentors and mentees
- Observe contributions that mentees make during and after the mentorship program
- Observe trajectory of mentees within project during and after mentorship program
- Collect demographic information from mentors and mentees

Success Metrics

Qualitative

- Mentee became a subject matter expert
- Mentee is integrated in community
- Mentee is comfortable asking for help from anyone in the community
- Mentee grew their professional network
- Mentee has taken on responsibilities within community
- Mentee contributes to community after end of mentorship project
- 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.
- 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?

Quantitative

- 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 mentorship

- Number of mentors in community
- Number of diverse mentees
- Number of contributions from mentees
- Mentors experience (rounds or years mentored)
- Geographic distribution of mentees
- Retention rate of mentees vs the usual retention rate in the community
- 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
    - How many mentors repeat the position in the following years

Known Implementations

Resources

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

Code Changes

Question: What 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.

Parameters

Mandatory:

- 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.

Aggregators

- Count. Total number of changes during the period.

Specific description: git

In the cases of git, a code change is defined as the part of a git commit that "touches" a source file. That is, for each git commit, only the part related to

deletions or additions of lines to files that are considered source code will be considered as the code change.

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.

In a set of repositories, the same commit may be present in more than one of them. Therefore, for counting unique changes, repeated commits should be counted only once.

Git parameters

Mandatory:

- 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.

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.

Sample Filter and Visualization

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.

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).

Reference Implementation

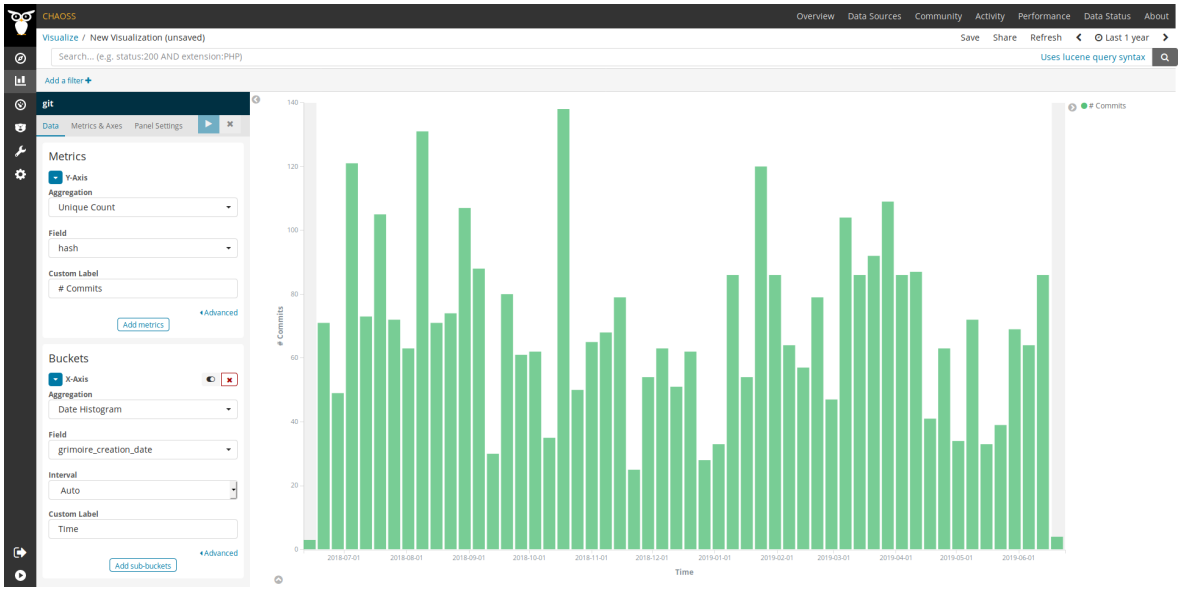
See reference implementation for git

Known Implementations

- 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
- Gitdm

Resources

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.

Parameters

Mandatory:

- 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 criteria for deciding whether a file is a part of the source code or not.

Optional:

- Type of source code change.
  - Lines added
  - Lines removed
  - Whitespace

Aggregators

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

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".

Git parameters

Mandatory:

- 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

Use Cases

- 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.

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.

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).

Reference Implementation

Known Implementations

- 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:

Resources

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.

Parameters

Mandatory:

- 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.

Aggregators

- Count. Total number of reviews during the period.

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.

GitHub parameters

None.

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.

GitLab parameters

None.

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.

Gerrit parameters

None.

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.

Filters and Visualizations

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).

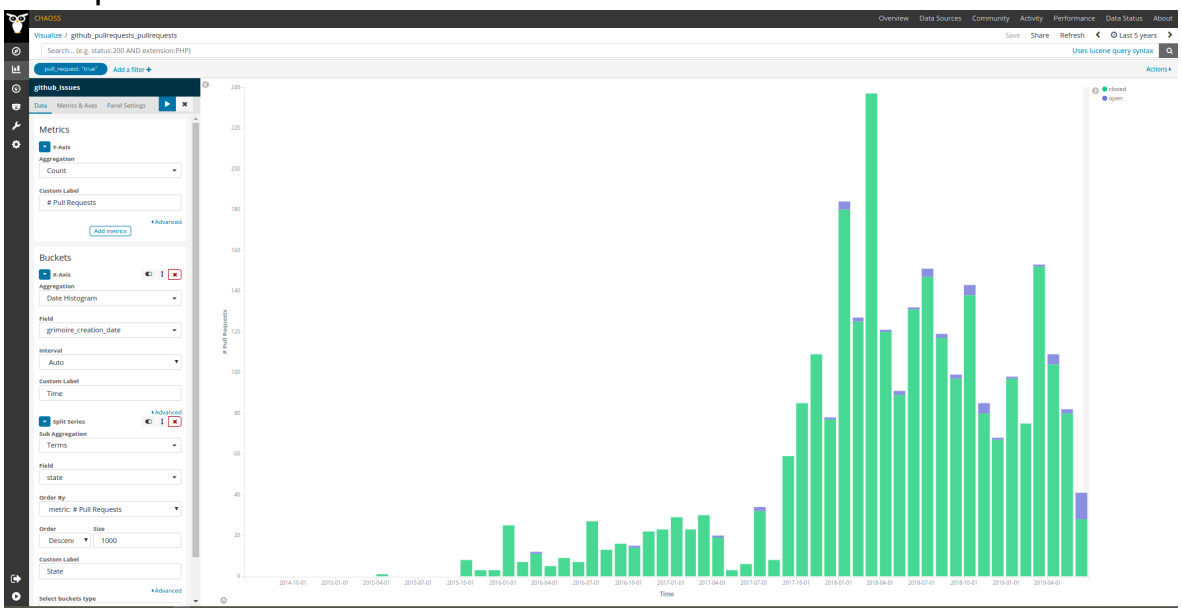
Reference Implementation

Known Implementations

- 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:



Resources

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.

Parameters

Mandatory:

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

Aggregators

- Count. Total number of accepted reviews during the period.

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.

GitHub parameters

Mandatory:

- 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.

GitLab parameters

Mandatory:

- 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.

Gerrit parameters

None.

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.

Filters and Visualizations

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 accepted reviews to change the code during a certain period (eg, a month).

Reference Implementation

Known Implementations

- 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_date).bars()
```

    - 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.

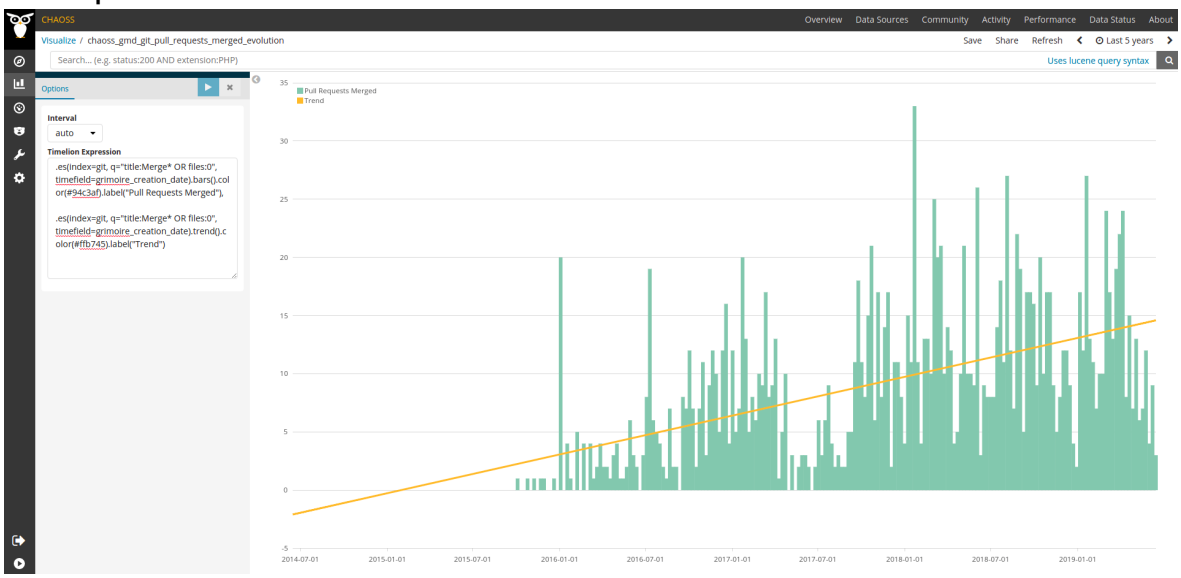
- `.color()` and `.label()` : some formatting options.

- 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_date).bars()
.es(index=git, q="title:Merge* OR files:0", timefield=grimoire_creation_date).trend(
```

- 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:



Resources

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.

Parameters

Mandatory:

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

Aggregators

- Count. Total number of declined reviews during the period.

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.

GitHub parameters

Mandatory:

- 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.

GitLab parameter

Mandatory:

- 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.

Gerrit parameters

None.

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.

Filters and Visualizations

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 declined reviews during a certain period (eg, a month).

Reference Implementation

Known Implementations

Resources

Issues New

Question: What are the number of 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.

Parameters

Mandatory:

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

Aggregators

- Count. Total number of issues during the period.

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).

GitHub parameters

None.

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).

GitLab parameters

None.

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).

Jira parameters

None.

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).

Bugzilla parameters

None.

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.

Filters and Visualizations

Filters

- 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

- 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 proposals to change the code during a certain period (eg, a month).

Reference Implementation

Known Implementations

Resources

Issues Active

Question: What is the count of issues that showed activity 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.

Parameters

Mandatory:

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

Aggregators

- Count. Total number of active issues during the period.

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".

GitHub parameters

None.

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".

GitLab parameters

None.

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".

Jira parameters

None.

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".

Bugzilla parameters

None.

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.

Filters and Visualizations

Filters

- 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

- 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 proposals to change the code during a certain period (eg, a month).

Reference Implementation

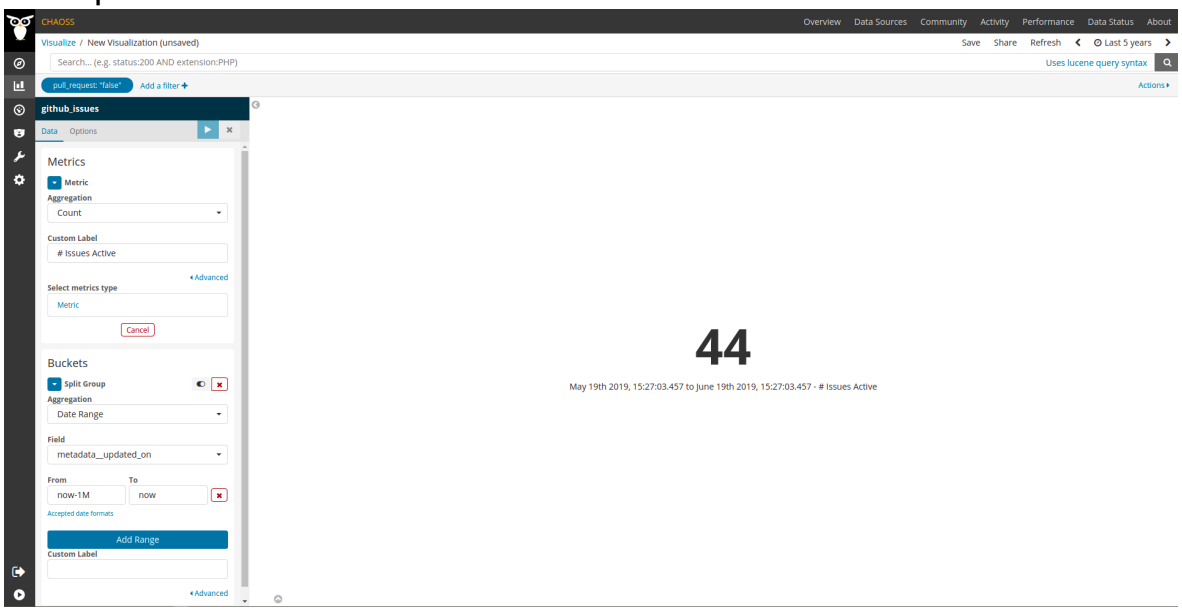
Known Implementations

- 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:



.

Resources

Issues Closed

Question: What is the count of issues that 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.

Parameters

Mandatory:

- Period of time. Start and finish date of the period. Default: forever.
  Period during which issues are considered.
- Criteria for source code. Algorithm. Default: all issues are related to source code.
  If we are focused on source code, we need a criteria for deciding whether an issues 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.

Aggregators

- Count. Total number of active issues during the period.

Specific description: GitHub

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

GitHub parameters

None.

Specific description: GitLab

- In the case of GitLab, active issues are defined as "issues that are closed".

GitLab parameters

- None.

Specific description: Jira

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

Jira parameters

None.

Specific description: Bugzilla

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

Bugzilla parameters

None.

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.

Filters and Visualizations

Filters

- 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

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

These could be represented as bar charts, with time running in the X axis.

Reference Implementation

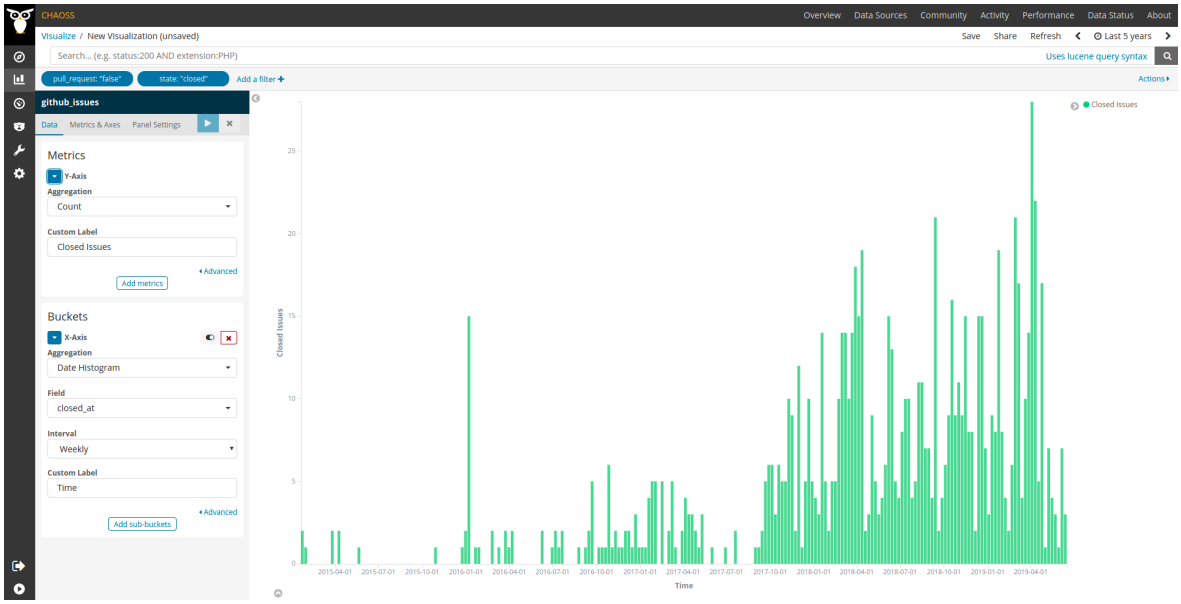
Known Implementations

- 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:



.

Resources

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.

Parameters

Mandatory:

- 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.

Aggregators

- Median. Median (50% percentile) of review duration for all accepted reviews in the considered period of time.

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.

GitHub parameters

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.

GitLab parameters

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.

Gerrit parameters

None.

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.

Filters and Visualizations

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.

Reference Implementation

Known Implementations

Resources

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 parameterized filter (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).

Formula

Essentially, 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 .`

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.

Sample Filter and Visualization

- 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.

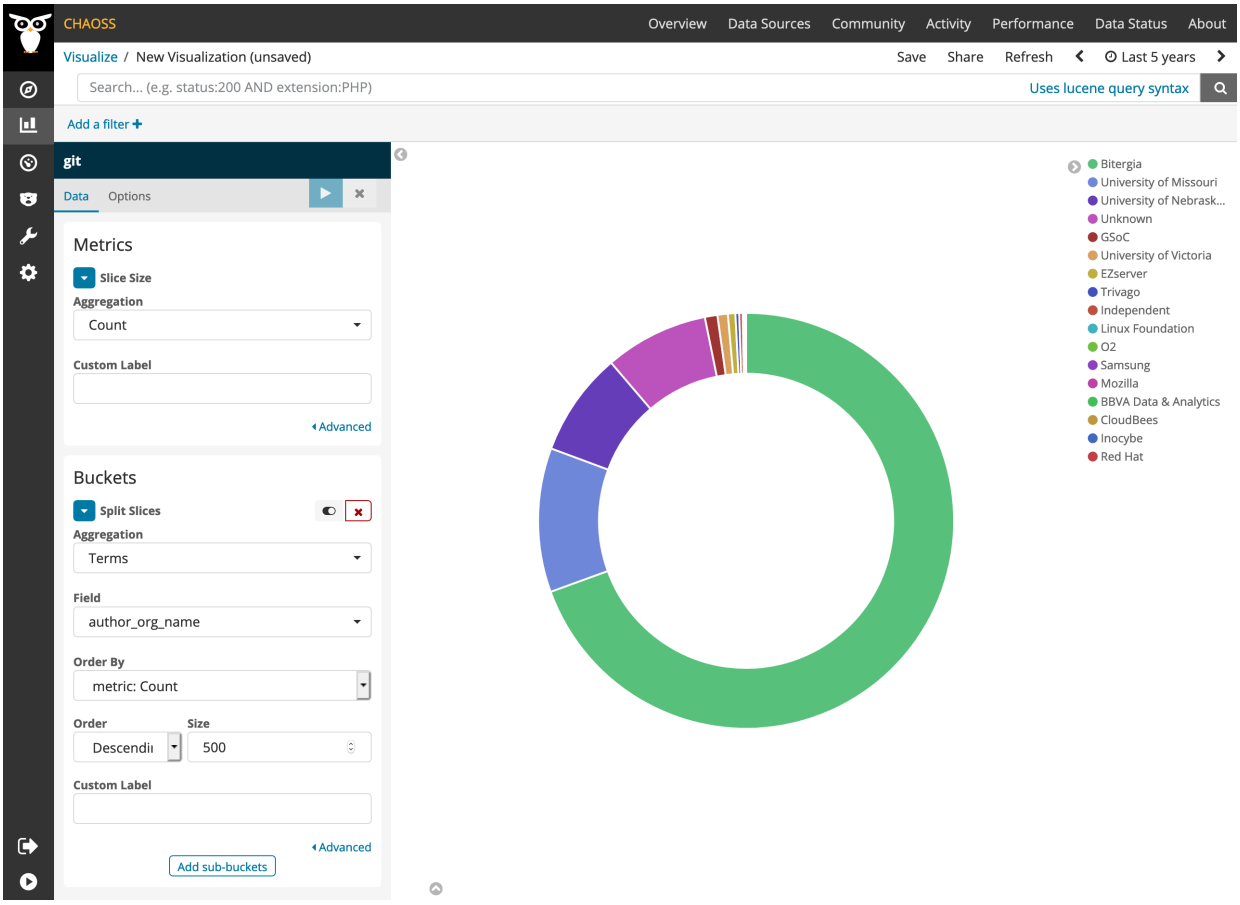
Reference Implementation

Known Implementations

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:



Resources

- 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
- http://philslade.blogspot.com/2015/07/what-is-elephant-factor.html
- https://blog.bitergia.com/2016/06/07/landing-the-new-eclipse-open-analytics-dashboard/
- https://www.stackalytics.com/

Committers

Question: How robust and diverse 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.

Formula

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

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

. 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 Confounds

- 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.

Objectives

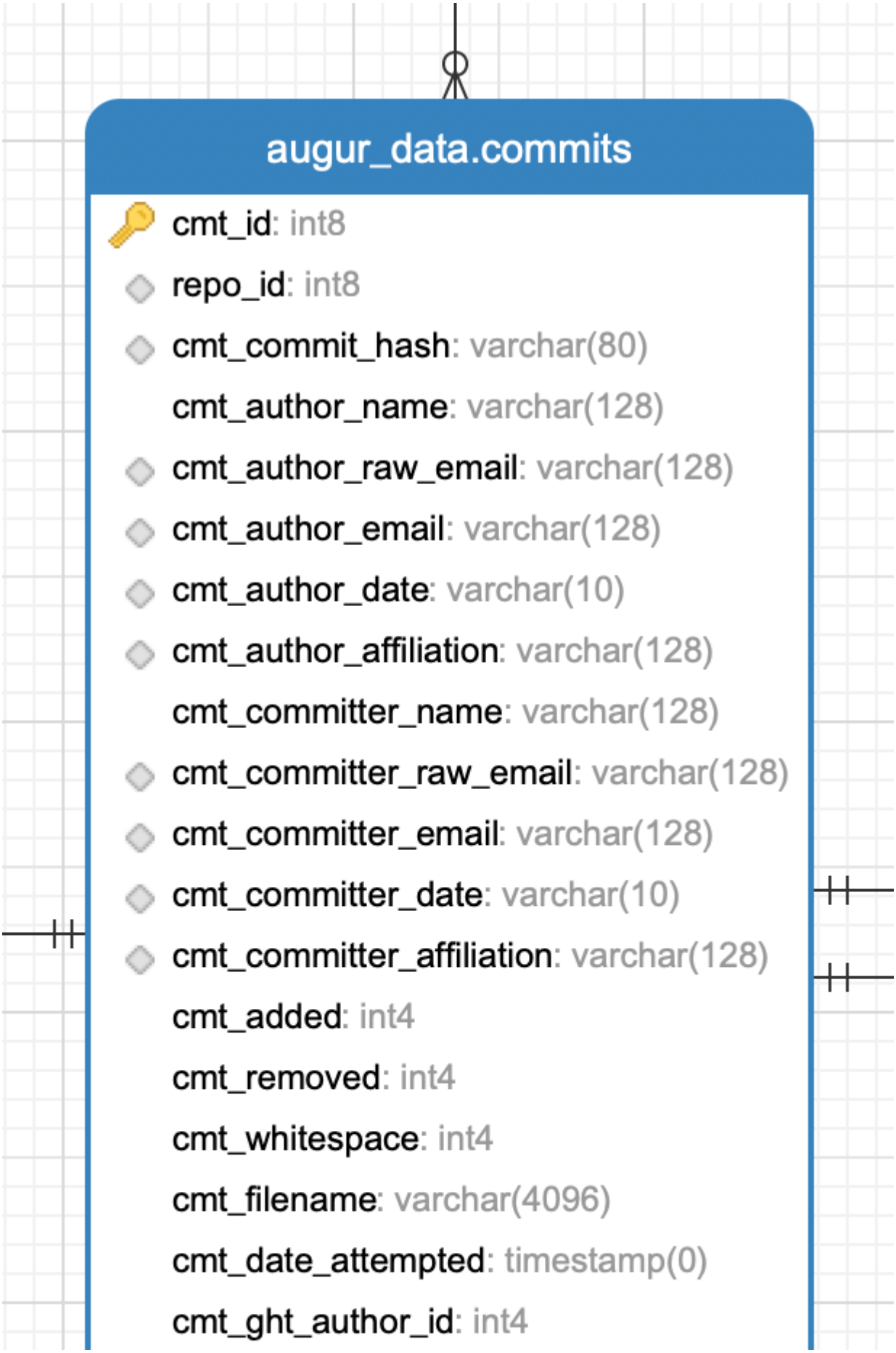
From the point of view of managers deciding among open source projects to incorporate into their organizations, the number of committers sometimes is important. Code contributions, specifically, can vary significantly from larger scale contributor metrics (which include documentation authors, individuals 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.

Sample Filter and Visualization

- 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 confound
- Commit Count: Contributors with fewer than some minimum threshold of commits in a time period could be excluded from this number.

Reference Implementation

Augur maintains a table for each commit record in a repository, as illustrated below.


augur_data.commits
cmt_id: int8
repo_id: int8
cmt_commit_hash: varchar(80)
cmt_author_name: varchar(128)
cmt_author_raw_email: varchar(128)
cmt_author_email: varchar(128)
cmt_author_date: varchar(10)
cmt_author_affiliation: varchar(128)
cmt_committer_name: varchar(128)
cmt_committer_raw_email: varchar(128)
cmt_committer_email: varchar(128)
cmt_committer_date: varchar(10)
cmt_committer_affiliation: varchar(128)
cmt_added: int4
cmt_removed: int4
cmt_whitespace: int4
cmt_filename: varchar(4096)
cmt_date_attempted: timestamp(0)
cmt_ght_author_id: int4

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. Adding filters is reasonably straightforward in the API and SQL.

Known Implementations

1. Augur
2. Grimoire Lab

Resources

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

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.

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`.

Formula

Subroutine Coverage

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

Statement Coverage

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

Objectives

An open source software package is being considered for deployment in a health care provider's production ecosystem. As part of the product evaluation process, IT Managers are comparing the test coverage of several alternate systems.

Sample Filter and Visualization

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`

Visualization

- Standard line graphs when time is included as a filter
- Standard histograms or stacked bar charts for point in time coverage comparisons

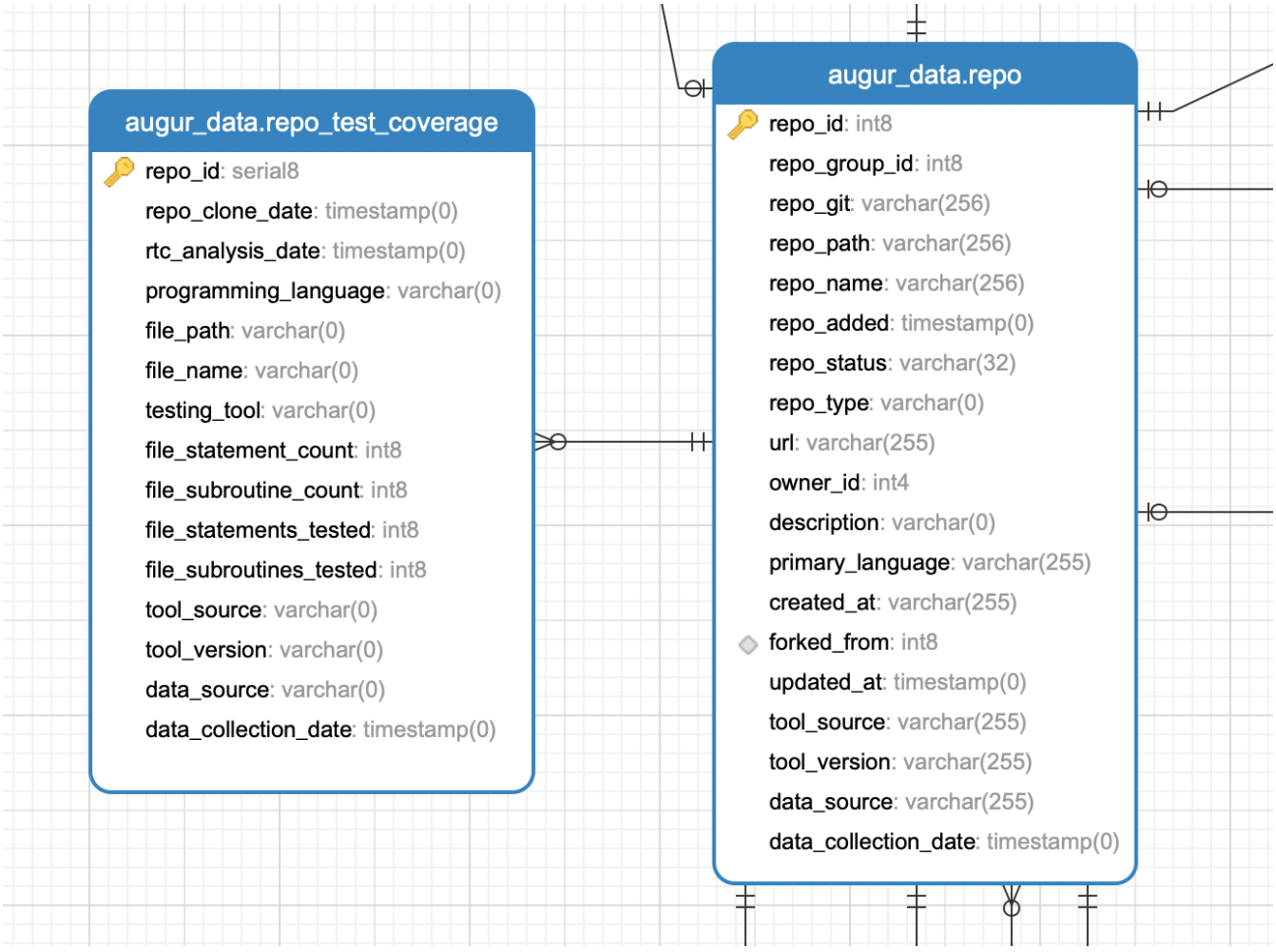
Reference Implementation

Metrics tools will need to provide abstracted frameworks for representing test

coverage because the tools used for measuring both `statement test coverage` and `subroutine test coverage` are programming language specific.

Known Implementations

1. Augur has test coverage implemented as a table that is a child of the main repository table in its repository. The data structure is illustrated below. Note that 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.



Examples

Test coverage examples are all programming language specific. We provide a few examples here:

1. Python's primary testing framework is PyTest
2. The Flask web framework for python enables coverage testing
3. Open source code coverage tools for common languages like Java, C, and C++ are available from my sites, including this one.

Resources

Discussion of testing coverage as a measure of software quality and reliability is extensive and ongoing in the software engineering research literature. Some of the more canonical papers are listed below.

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.

Formula

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.

Sample Filter and Visualization

Does not apply in this case

Reference Implementation

A highly simplified version of the example provided in the License Declared metric. The following SQL will enumerate the number of files with each license. The count of rows will indicate the number of different licenses. The presence of a (NULL) row will indicate packages without scanned license declarations. **Note** that its important to understand that no file scanner is conclusive, especially if license declarations are not in SPDX format. However, this metric still serves as a high level indicator of initial license risk when assessing packages (repositories) from this perspective.

```
SELECT
    concluded_license_id,
    COUNT ( * )
FROM
    packages_files
WHERE
    package_id = 'package_id_of_package'
--- package_id is an int, but represented here as text for the purposes of explaining t
```

Known Implementations

1. Augur

Examples

1. Available in the Augur test schema for these repositories:
    1. portable
    2. openBSD
    3. boringSSL

8. Resources

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. This includes both software and documentation source files and is represented as a percentage of total coverage.

Formula

The above data was pulled from DoSOCSv2 and filtered through Jinja2 to get the desired information. Here is a sample of Jinja code to filter DoSOCSv2 code. The file that this implementation is inserted into may be found here: https://github.com/DoSOCSv2/DoSOCSv2/blob/master/dosocs2/templates/2.0.tag The code may be added to the end of the document. Run a “dosocs2 oneshot” scan and the new data will be at the end of the document. More information on DoSOCSv2 is found here: https://github.com/DoSOCSv2/DoSOCSv2

```
{% set cnt = [0] %}
{% for file in package.files %}
{% if file.license_info[0].short_name != None %}
{% if cnt.append(cnt.pop() + 1) %}{% endif %}
{% endif %}
{% if loop.index == loop.length %}
TotalFiles: {{ loop.index }}
DeclaredLicenseFiles: {{ cnt[0] }}
PercentTotalLicenseCoverage: {{ '%0.2f' %  ((cnt[0] / loop.index) * 100) | float }}%
 {% endif %}
{% endfor %}
```

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.

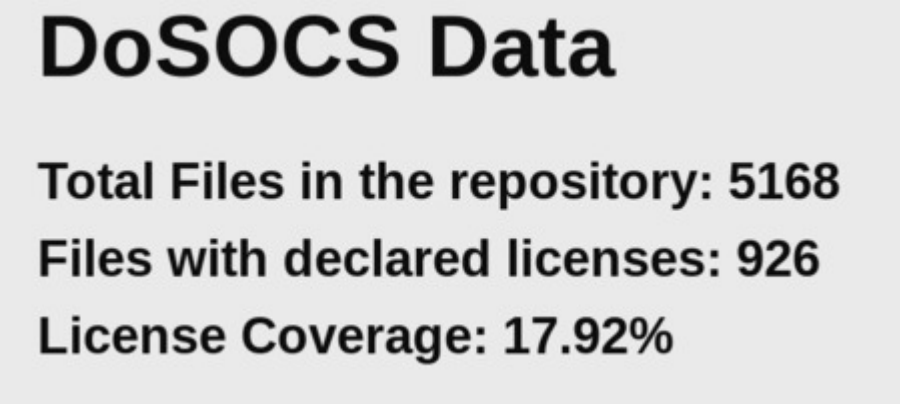
Note: In both cases License Coverage less than 100% may require further investigation by distributors and consumers of a software package.

Sample Filter and Visualization

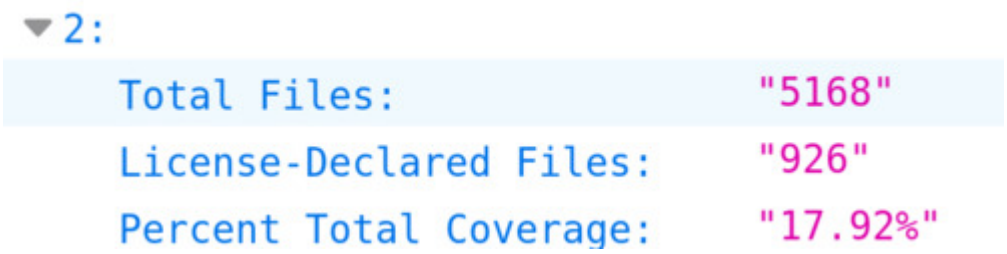
Time: Licenses declared in a repository can change over time as the dependencies of the repository change. One of the principle motivations for tracking license presence, aside from basic awareness, is to draw attention to any unexpected new license introduction.

Reference Implementation

Web Presentation of DoSOCS2 Output:



JSON Presentation of DoSOCS2 Output:



Known Implementations

- Augur

Examples

Available in the Augur test schema for these repositories: - portable - openBSD - boringSSL

Resources

- 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.

Formula

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	License Notes
MIT	44	None
AGPL	2	Pending removal
Apache	88	None

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.

Sample Filter and Visualization

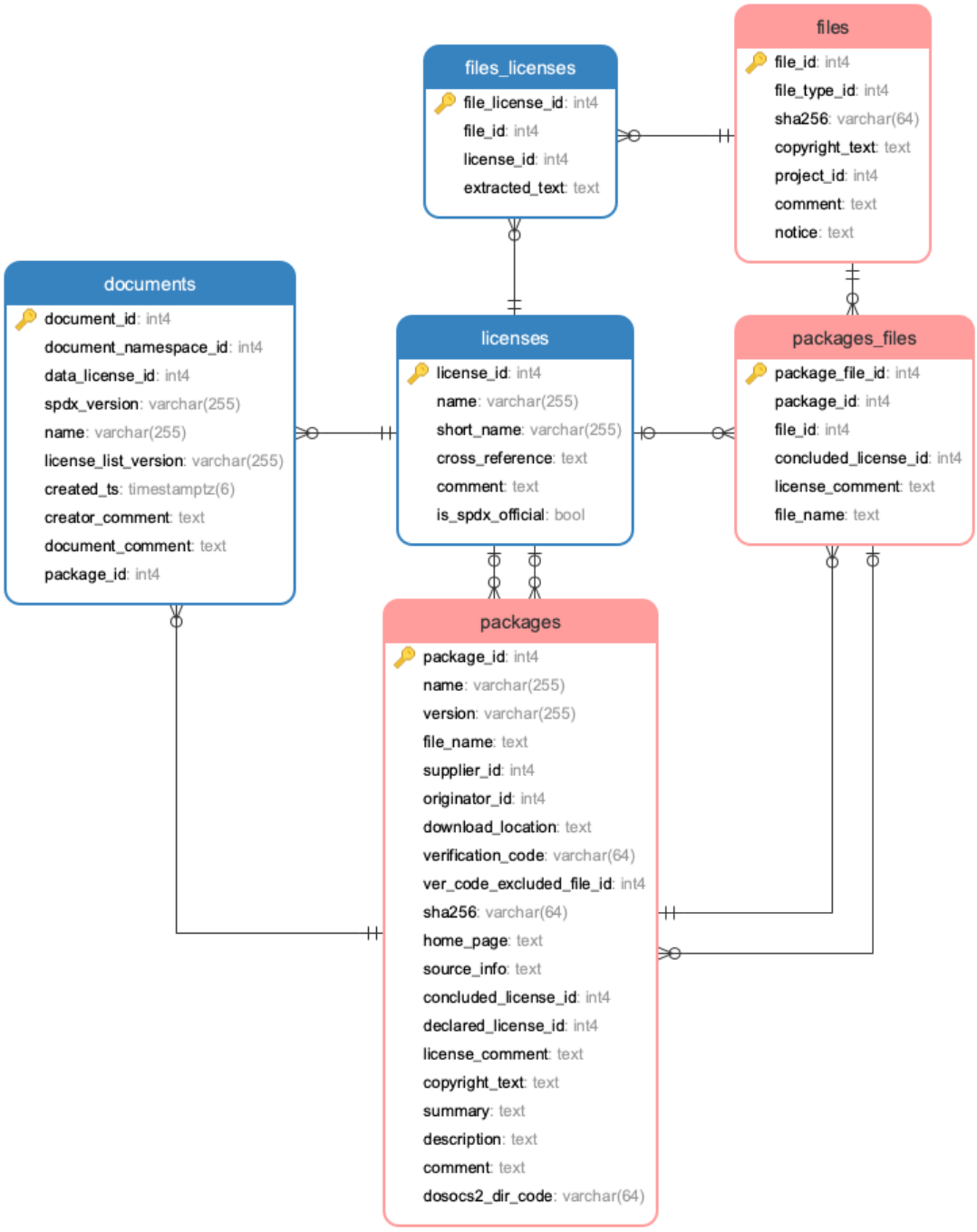
Filters

- 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.

Reference Implementation

The DoSOCSv2 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`, DoSOCSv2 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 `DoSOCSv2` scans of the repository. Each `package` is stored in its most recent form, and each `packages_file` can have one `license` declaration.



Sample SQL to Extract License Information from a Package File

```
SELECT A
    .file_name,
    b.license_id,
    b."name" AS declared_license
```

```
FROM
    packages_files A,
    licenses b
WHERE
    A.declared_license_id = b.license_id
```

Known Implementations

- Augur

Examples

- Available in the Augur test schema for these repositories:
  - portable
  - openBSD
  - boringSSL

Resources

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

Bill of Materials

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

Description

A software package has a standard expression of dependencies, licensing, and security-related issues.

Formula

This is an enumeration of a "bill of materials", as as such is not expressed as a metric. It is more accurate to consider this CHAOSS metric as an "inventory of components" that is uniquely important in many risk oriented open source software acquisition conversations.

Objectives

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

Sample Filter and Visualization

DoSOCSv2 was used to scan the GitHub Repository. Here are some of the core

license values from the scan, which may be used as an example of the format:

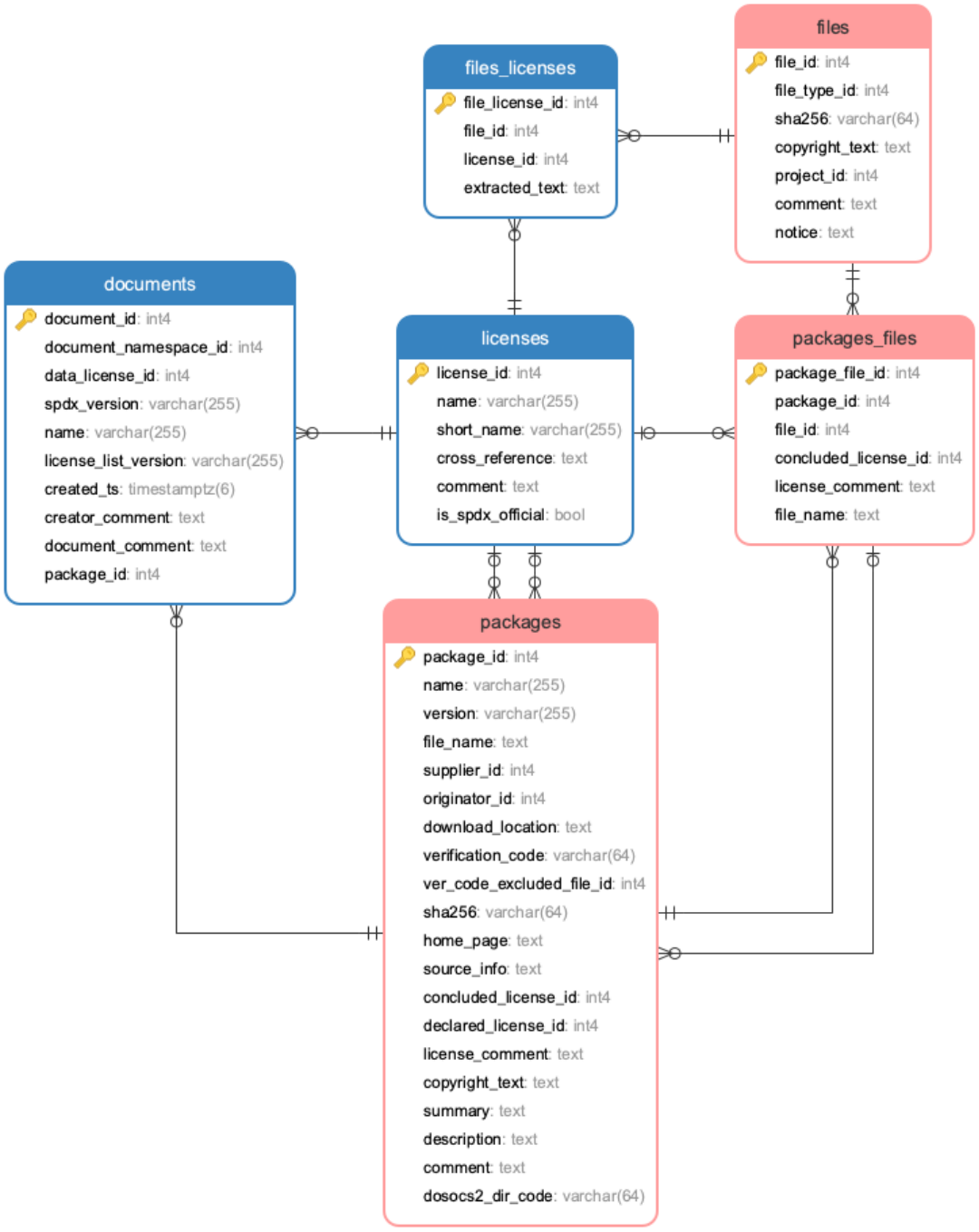
```
LicenseID: LicenseRef-See-URL
LicenseName: See-URL
ExtractedText: com/license
LicenseCrossReference:
LicenseComment: found by nomos

LicenseID: LicenseRef-Dual-license
LicenseName: Dual-license
ExtractedText:  MIT or new BSD license.
LicenseCrossReference:
LicenseComment: found by nomos

LicenseID: LicenseRef-BSD
LicenseName: BSD
ExtractedText: Available via the MIT or new BSD license.
LicenseCrossReference:
LicenseComment: found by nomos

LicenseID: LicenseRef-Python
LicenseName: Python
ExtractedText: license. Your class definition should look\nsomething like this:\n\n.. c
LicenseCrossReference:
LicenseComment: found by nomos

LicenseID: LicenseRef-Non-profit
LicenseName: Non-profit
ExtractedText: nonprofit
LicenseCrossReference:
LicenseComment: found by nomos
```

A full text file for this scan may be found here: https://drive.google.com/file/d/18ClTeQo64sU8dVzSQDMYzO6jLHHa9TPg/view?usp=sharing

Reference Implementation

A file by file SBOM is available with Augur configured using the DoSOCSv2 plugin. The relevant parts of the database schema are illustrated below. The most important points, from an SBOM perspective, is simpler than the software licensing metrics described elsewhere. For the SBOM, we simply look at the enumeration of:

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



Known Implementations

- DoSOCSv2 embedded as an Augur Service.

Resources

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

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.

Formula

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)

Objectives

This metric gives an Open Source Program Office (OSPO) manager a way to compare contributed labor costs across a portfolio of projects.

The OSPO manager can use the Labor Cost metric to:

- report labor costs of contributed vs in-house work
- compare project effectiveness across a portfolio of projects
- compare labor costs of open-source projects vs in-house efforts

Sample Filter and Visualization

Filters:

- internal vs external contributors
- issue tags
- project sources (e.g., internal, open-source repos, competitor open-source repos)

Reference Implementation

```
1 IssueID, Severity, Title          , Status, Contributor, Tag
2 34234  , High    , Add CSV Graphic, Open  , andyl      , metrics
3 23421  , Med     , Fix typos      , Closed, mattg      , metrics
4 56743  , High    , Reword section , Open  , georg      , augur
5 85879  , Low     , Add CNCF PNG   , Open  , seang      , metrics
6 34183  , High    , Remove button  , Closed, vinod      , implementation
7 76790  , Low     , Use large font , Open  , kevin      , metrics
8 57432  , Med     , Sync with web  , Closed, carol      , implementation
```

Our first reference implementation of parameterized metrics will rely on CSV exports from Augur.
We will use spreadsheet for metric parameters and calculation formulas. Future implementations may add features for parameter manipulation directly in the webapp.

Known Implementations

- Augur
- GrimoireLab

Resources

- 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'.

Formula

Base metrics include:

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

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

Sample Filter and Visualization

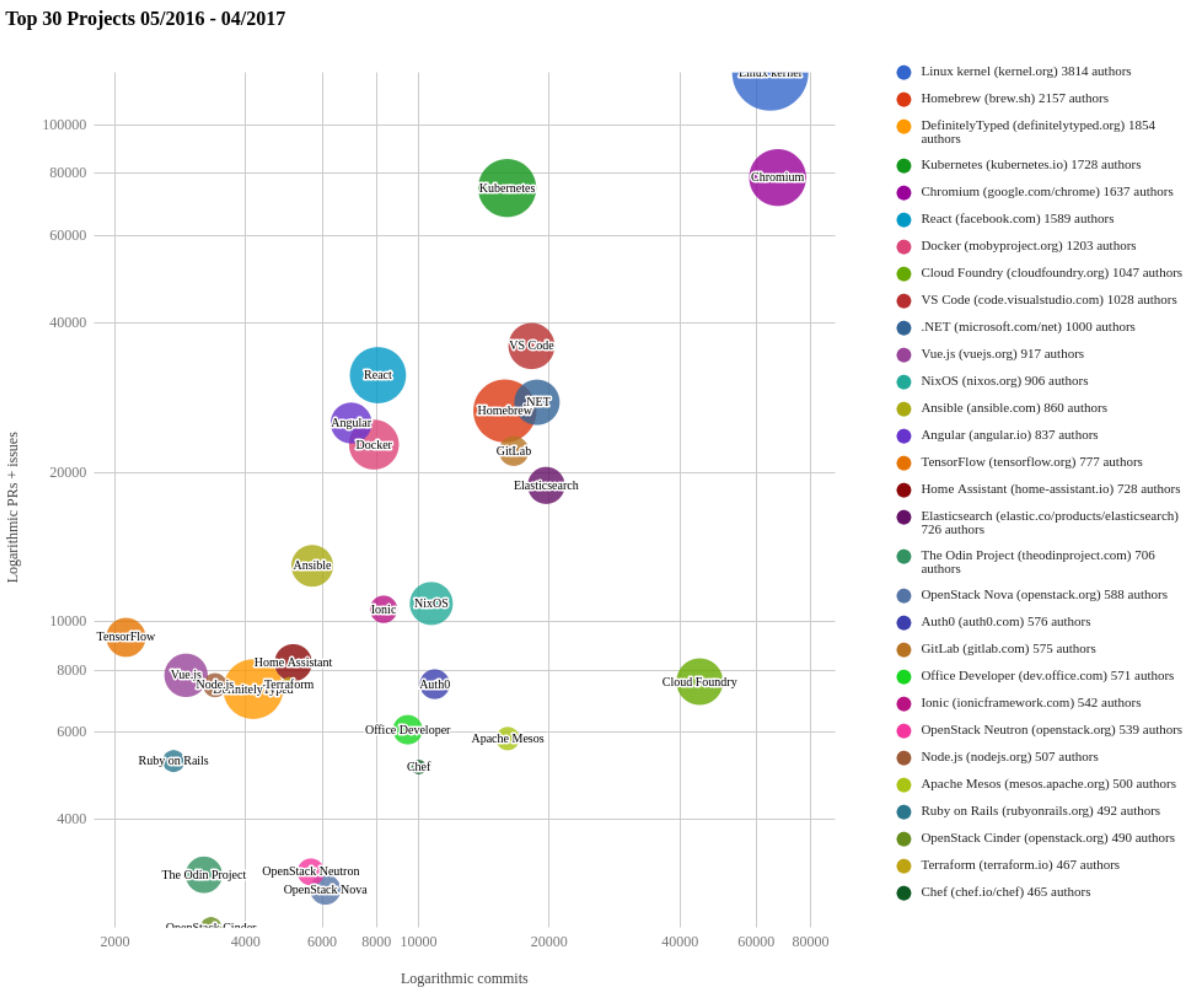
Potential filters:

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

Visualization:

- 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

Reference Implementation



From CNCF

Known Implementations

- Augur
- CNCF - https://github.com/cncf/velocity
- Grimoire Lab

Resources

- 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.

Formula

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

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.

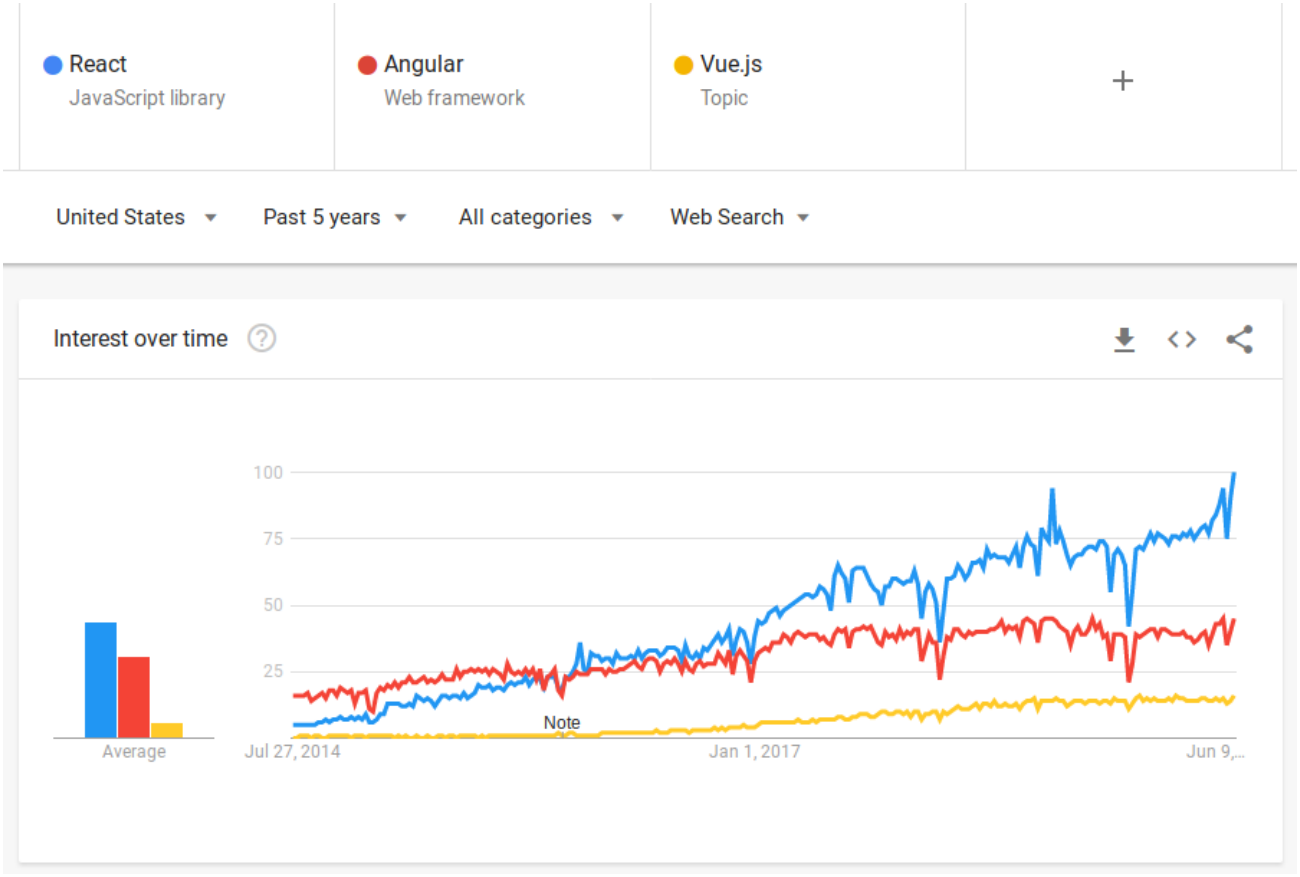
Sample Filter and Visualization

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.

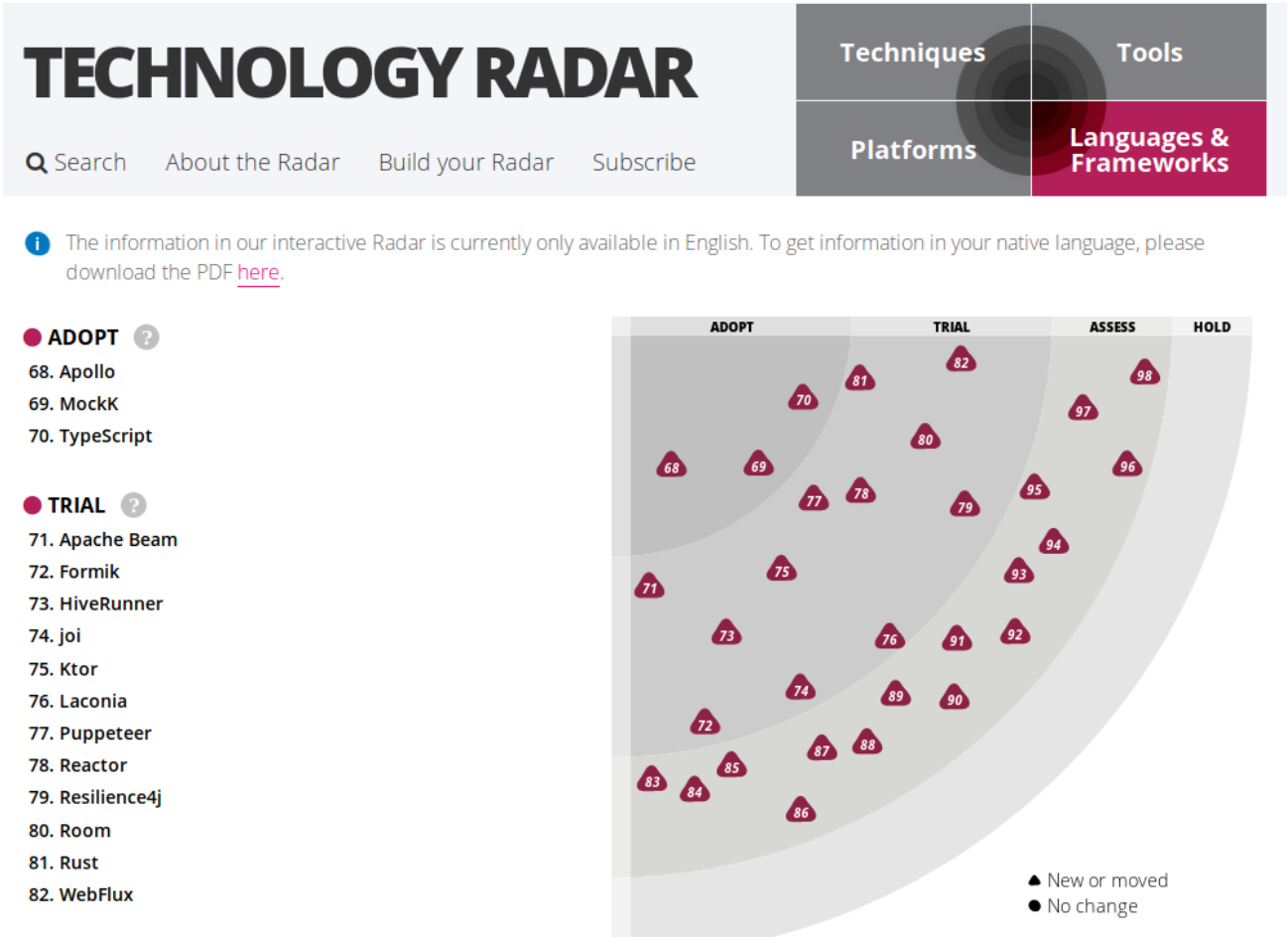
Other visualizations could include Google search trends (React vs. Angular vs.

Vue.js)

https://raw.githubusercontent.com/chaoss/wg-value/master/focus-areas/living-wage/



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.

React.js

NOV 2016

ADOPT

In the avalanche of front-end JavaScript frameworks, **React.js** stands out due to its design around a reactive data flow. Allowing only one-way data binding greatly simplifies the rendering logic and avoids many of the issues that commonly plague applications written with other frameworks. We're seeing the benefits of React.js on a growing number of projects, large and small, while at the same time we continue to be concerned about the state and the future of other popular frameworks like AngularJS. This has led to React.js becoming our default choice for JavaScript frameworks.

NOT ON THE CURRENT EDITION

This blip is not on the current edition of the radar. If it was on one of the last few editions it is likely that it is still relevant. If the blip is older it might no longer be relevant and our assessment might be different today. Unfortunately, we simply don't have the bandwidth to continuously review blips from previous editions of the radar

Understand more »

APR 2016

ADOPT

NOV 2015

TRIAL

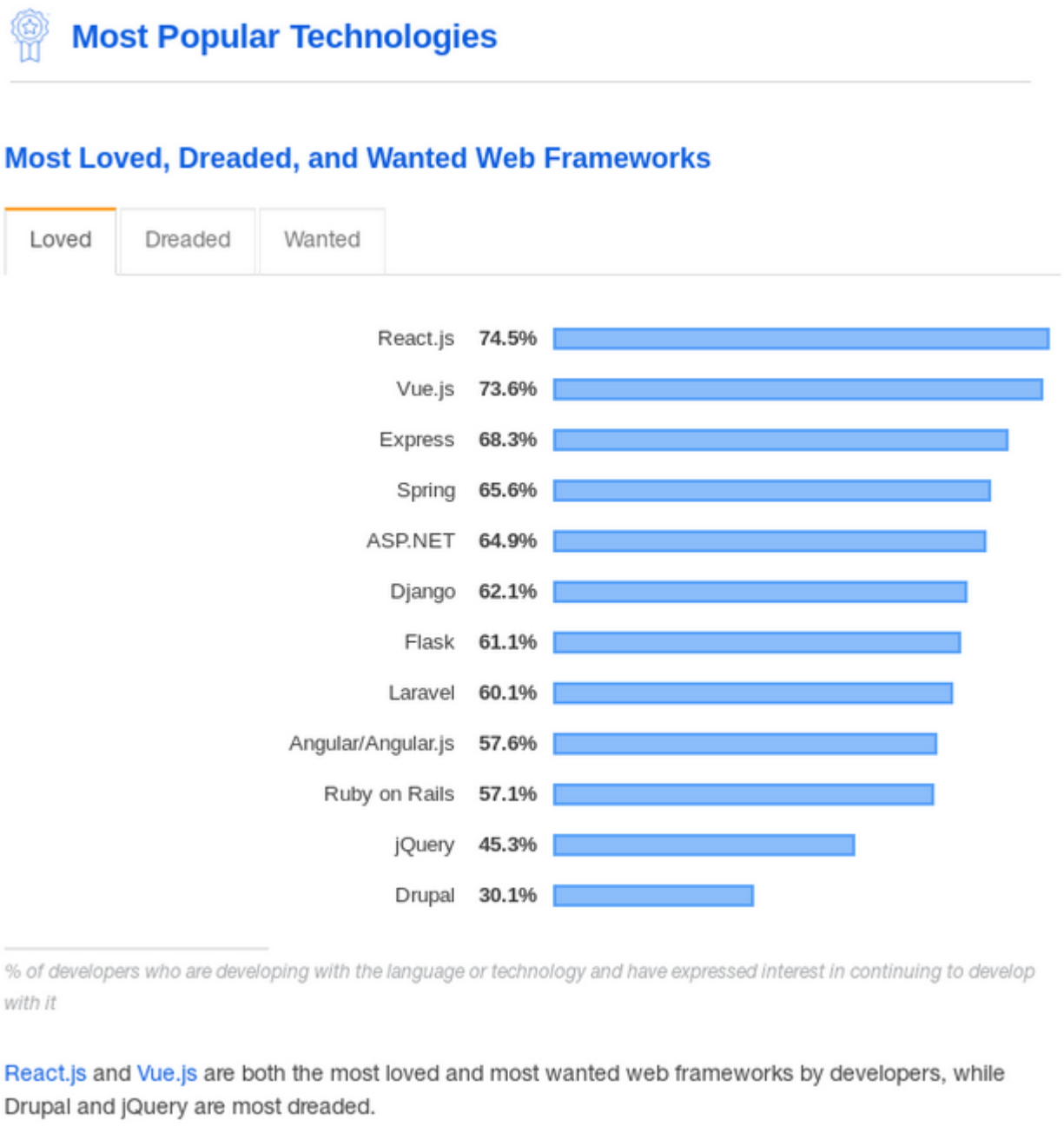
One benefit of the ongoing avalanche of front-end JavaScript frameworks is that occasionally a new idea crops up that makes us think. **React.js** is a UI/view framework in which JavaScript functions generate HTML in a reactive data flow. It differs significantly from frameworks like AngularJS in that it only allows one-way data bindings, greatly simplifying the rendering logic. We have seen several smaller projects achieve success with React.js, and developers are drawn to its clean, composable approach to componentization.

MAY 2015

TRIAL

One benefit to the ongoing avalanche of front-end JavaScript frameworks is that occasionally, a new idea crops up that makes us think. **React.js** is a UI/View framework in which JavaScript functions generate HTML in a reactive data flow. We have seen several smaller projects achieve success with React.js and developers are drawn to its clean, composeable approach to componentization.

StackOverview publishes an annual developer's survey



Reference Implementation

Known Implementations

- Augur - for dependencies as an API endpoint
- Google Trends - for showing search interest over time
- ThoughtWorks TechRadar - project assessments from a tech consultancy
- StackOverflow Developer's Survey - annual project rankings

Examples

Available in the Augur test schema for multiple repositories:

- Rails
- ReactJS
- SaltStack

Resources

- 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