1. **Maturity Level** [1]
   The maturity is presented by the projects age and the number of created issues and minor releases. Since no definition of minor releases is provided, all releases are included in the implementation for this thesis. Issues and releases are filtered respectively, according to their “since” parameter, which indicates when an object was last updated.
   Formula:
   $$\left(\frac{M_1 + M_2 + M_3}{3}\right) \times 100$$

   Parameters:
   $M_1 = \frac{S_{age}}{5}$
   S_{age} = 1 Age < 2 months
   S_{age} = 2 Age 2-12 months
   S_{age} = 3 Age > 1- 2 years
   S_{age} = 4 Age > 2-3 years
   S_{age} = 5 Age > 3 years

   $M_2 = \frac{S_{issue}}{5}$
   S_{issue} = 1 Nr. of issues* > 1000
   S_{issue} = 2 Nr. of issues* > 500-1000
   S_{issue} = 3 Nr. of issues* > 100-500
   S_{issue} = 4 Nr. of issues* > 50-100
   S_{issue} = 5 Nr. of issues* ≤ 50
   *in the past 6 months

   $M_3 = \frac{S_{release}}{5}$
   $S_{release}$ = 1 Nr. of minor releases* 0 version
   $S_{release}$ = 3 Nr. of minor releases* 1-3 versions
   $S_{release}$ = 5 Nr. of minor releases* > version
   *in the past 12 months

2. **OSI Approved Licenses** [2]
   The majority of GitHub repositories hold one ore no license, thus the metric is adapted to a more fitting metric.
   Formula:
   If a license is provided, the OSI status is verified by the spdx license list available on GitHub[1], returning 1 for approved and 0 for not approved.

   Original Formula:
   $$\frac{Number\ of\ OSI\ approved\ licences}{Total\ Number\ of\ licenses}$$

3. **Technical Fork** [2]

[1] https://raw.githubusercontent.com/spdx/license-list-data/master/json/licenses.json

4. **Criticality Score** [3]

The formula is based on the algorithm by Rob Pike[2] described in the GitHub Project criticality_score [3], representing a project's influence and importance by a single value between 0 and 1. The calculation includes various information about a repository, listed in the parameters below.

All parameters are adapted as documented in the project except the dependents_count, which are calculated by the dependents gathered from GitHub's Dependency Graph instead of checking for mentions of the concerned project in commit messages.

Formula:

$$C_{project} = \frac{1}{\sum_i \alpha_i} \sum_i \alpha_i \frac{\log(1 + S_i)}{\log(1 + \max(S_i, T_i))}$$

S_i = Parameter

α_i = Weight

T_i = Max threshold

Parameters:

created_since in months
updated_since in months
contributors_count
org_count
commit_frequency of commits from the last year
recent_releases_count of releases from the last year
closed_issues_count from the last 90 days
updated_issues_count from the last 90 days
comment_frequency from the last 90 days
dependents_count

5. **Change/Pull Request** [2]

No specific metrics are presented, thus following information is utilized and summarized.

Submetrics/Information:

total_pulls
avg_pull_closed_days
ratio_open_total
ratio_closed_total
ratio_merged_total

Formula:

Average pull closing time in days

$$\frac{Sum(day\_diff_1, day\_diff_2, day\_diff_3, \ldots day\_diff_n)}{Number\ of\ closed\ pulls}$$

Ratio open/closed/merged to total pulls

$$\frac{Number\ of\ open/closed/merged\ pulls}{Number\ of\ total\ pulls}$$

[2] https://github.com/ossf/criticality_score/blob/main/Quantifying_criticality_algorithm.pdf

6. **Project Velocity** [2]
Velocity refers to the development speed of an organization or in this thesis of GitHub repositories. This concerns project's issues, pull requests, commits and the number of contributors. Since commits and contributors are already included in other metrics, this information is excluded from the project velocity metric. Thus, following scores are calculated whereas issues include pulls.
Submetrics:
total_issues
pull_count
ratio_pull_issue
avg_issue_resolving_days
ratio_open_total
ratio_closed_total

Custom Formulas:
Average issue resolving time in days
$$\frac{Sum(day\_diff_1, day\_diff_2\ , day\_diff_3, \ldots day\_diff_n)}{Number\ of\ resolved\ issues}$$

Pull to total Pull and Issues Ratio (ratio_pull_issue)
$$\frac{Number\ of\ pulls}{Number\ of\ pulls\ and\ issues}$$

Issue Resolving Velocity Ratio (avg_issue_resolving_days)
$$\frac{Sum(Time\ from\ issue\ opened\ to\ issue\ closed)}{Total\ Number\ of\ Issues}$$

Closed/Open Issue Ratio:
$$\frac{Number\ of\ closed/open\ Issues}{Total\ Number\ of\ Issues}$$

7. **Github community health percentage** [3]
This percentage score represents the existence of several files such as the readme or the contributing file. Since the formula published by GitHub is outdated as of 22.07.2023 by considering incomprehensible scores, a custom formula is defined additionally to cover this information in a transparent way.
Custom Formula:
$$\frac{8}{P_1+P_2+P_3+P_4+P_5+P_6+P_7+P_8}$$

Parameters:
P_1 = Existence of Readme file (0,1)
P_2 = Existence of Contributing file (0,1)
P_3 = Existence of License file (0,1)
P_4 = Existence of code_of_conduct file (0,1)
P_5 = Existence of description (0,1)
P_6 = Existence of issue_template (0,1)

[3] https://docs.github.com/en/rest/metrics/community?apiVersion=2022-11-28

P_7 = Existence of pull_request_template (0,1)
P_8 = Existence of documentation (0,1)

8. **Issues** [2]
The issues endpoint of the GitHub API includes issues and pulls, the issues function excludes pulls, since they are already included in other metrics such as the project velocity. The selected values and scores which are calculated are listed below.
Submetrics:
total_issues
open_issues
average_issue_created_per_week
average_issue_resolving_days
average_first_response_time_days
ratio_open_total
ratio_closed_total

Custom Formulas:
Average issues created per week
$$\frac{Sum(issues\_created_1, issues\_created_2\ , issues\_created_3, \dots issues\_created_n)}{Number\ of\ created\ issues}$$

Average issue resolving time/time to first response in days
$$\frac{Sum(day\_diff_1, day\_diff_2\ , day\_diff_3, \dots day\_diff_n)}{Number\ of\ resolved\ issues/first\ responses}$$

Closed/Open Issue Ratio:
$$\frac{Number\ of\ closed/open\ Issues}{Total\ Number\ of\ Issues}$$

9. **Support Rate** [1]
This metric refers to the extent of responses to issues and pulls. The formulas are implemented as recommended and documented below.
Formula:
$$V_5 = \left(\frac{M_1 + M_2}{2}\right) \times 100$$
Parameters:
M_1 Issue Support Rate
M_2 Pull Request Support Rate
$$M_1 = \left\{ \begin{matrix} \left(\frac{\begin{matrix} Number\ of\ issues \\ that\ have\ been\ responded \\ in\ the\ past\ 6\ months \end{matrix}}{N} \right), N > 0 \\ 0, N = 0 \end{matrix} \right\}$$
$where\ N\ Total\ issues\ in\ the\ past\ 6\ months$

$$M_2 = \left\{ \begin{matrix} \left(\frac{\begin{matrix} Number\ of\ pull\ requests \\ that\ have\ been\ responded \\ in\ the\ past\ 6\ months \end{matrix}}{N} \right), N > 0 \\ 0, N = 0 \end{matrix} \right\}$$

$where\ N\ Total\ pull\ requests\ in\ the\ past\ 6\ months$

10. **Code Dependencies (Upstream Code Dependencies** [2])

Retrieving dependencies from GitHub projects is limited to either checking dependency differences from single commits, check the file contents or to reading the dependency graph. The dependency graph if the most reliable data source, yet there is no API endpoint to this information, thus a webscraper is implemented to gather dependencies, upstream as well as downstream. Dependencies occurring multiple times in different files are only counted as one dependency. Additionally, is distinguished from visible dependents and all dependents, referring to whether a project is set to public or private. The related function returns following information.

Information:

Number of total upstream dependencies (distinct)
Number of total downstream dependents (distinct)
Number of visible downstream dependents (distinct)

11. **Security Advisories (Branch Patch Ratio** [4])

Since the implementation of a matching of CVEs to project branches is out of the scope from this thesis due to no reliable state of the art methods, GitHub's security advisory[4] database is utilized. This data source includes CVEs and corresponding information about the state, severity, published date and the CVSS score (Common Vulnerability Scoring System). The score is not available for each advisory, therefore the CVE id is used to map the advisory to the related CVE from NVD to retrieve the score from there. Finally, following information is gathered.

Information/Metric:

advisories_available 0,1
patch_ratio
closed_advisories
mean_cvss_score
ratio_severity_high_crit

Custom Formulas:

Patched vulnerability Ratio

$$\frac{Number\ of\ patched\ vulnerabilities}{Total\ Number\ of\ vulnerabilities}$$

Mean CVSS Score:

$$\frac{Sum(cvss\_score_1, cvss\_score_2, cvss\_score_3, \ldots cvss\_score_n)}{Number\ of\ vulnerabilities}$$

Severity high or critical Ratio

$$\frac{Number\ of\ vulnerabilities\ with\ high\ or\ critical\ severity}{Total\ Number\ of\ vulnerabilities}$$

[4] https://github.com/advisories

Discarded Formula:

$$CVE - PR = \frac{\# \, of \, patched \, branches}{\# \, of \, branches \, affected \, by \, the \, CVE}$$

12. **Contributions Distributions (Bus Factor** [2] [5] **& Pareto Principle** [5] [6])

Bus Factor

Represents the risk factor if the most active contributor stops contributing by calculating the smallest number of people contributing 50 % of the contributions.

Formula:

Score = # of Elements required to calculate T_2

$$T_1 = \sum_{max(i)}^{\min(i)} i * 0.5$$

$$i = \# \, of \, Contributions \, per \, Contributor$$

$$T_2 = Highest \, contributions \, summed \, up \, until \, sum > T_1$$

Pareto Principle

Total Number of collaborations by top 20% of contributors with the most collaborations should be ~ 80% of the total contributions.

Submetrics

20 % of the number of contributors

80% of the number of contributors

Pareto IST score – % of total contributions by the top 20% of contributors

Difference of Pareto IST and Pareto SOLL in percent

$$\frac{|80\% \, of \, number \, of \, contributors - pareto \, IST \, value|}{\frac{(80\% \, of \, number \, of \, contributors + pareto \, IST \, value)}{2}}$$

13. **Contributors per File** [7]

To get a representative value for each repository, the average number of contributors per file are computed.

Formula:

$$\frac{Sum(contributors_1, contributors_2, contributors_3, \ldots contributors_n)}{Number \, of \, files}$$

*Contributors per file

14. **Number of Support Contributors** [1] [8] [9] [10]

The formula to this metric is presented by [1] and based on a BRR report from 2005 [11]. The number of contributors is subdivided in score groups from 1 to 5, the scores are used to calculate the final score.

In the original report data from the past 6 months is included. Thus the formula is extended by a corresponding filter to reduce computing costs.

Formula:

$$V_4 = \frac{S}{5} \times 100$$

Parameters:

S = # of Contributors and Core Team Members

S = 1 < 5 persons

S = 2 5-10 persons

S = 3 > 10-20 persons

S = 4 > 20-50 persons

S = 5 > 50 persons

15. Elephant Factor

Expresses the contributions by the community across the companies the contributors belong to.

Formula:

# of Elements required to calculate T_2

$$T_1 = \sum_{max(i)}^{\min(i)} i * 0.5$$

$$T_2 = Highest\ contributions\ summed\ up\ until\ sum\ >\ T_1$$

i = # of Contributions per Organization

16. Size of Community [1]

This metric is similar to the support contributor metric and additionally includes watchers. In GitHub terms watchers correspond to a project's stars and subscribers represent user who watch a project to follow the progress[5]. Thus, as number of watchers the subscribers_count value is gathered. Core team members are to be considered as part of the contributors.

Formula:

$$V_2 = \frac{S}{5} \times 100$$

Parameters:

S = # of Core Team Members, Contributors, and Watchers

S = 1 Small (<50 people)

S = 2 Relatively Small (50-100 people)

S = 3 Medium (> 100-200 people)

S = 4 Relatively Large (> 200-300 people)

S = 5 Large (Total > 300 people)

17. Churn [12] [7]

The Churn implemented in [7] calculates the metric in a weighted manner by considering bug fixing code changes, this is not feasible for this thesis since bugs must be defined and detected first.

In this thesis a simpler formula is used.

Formula:

$$churn = \frac{\#\ of\ \ deleted\ lines}{\#\ of\ written\ lines}$$

[5] https://docs.github.com/en/rest/activity/starring?apiVersion=2022-11-28

18. Branch Lifecycle – In Progress

Formula

$$Branches\ created\ Ratio\ = \frac{Total\ \#\ of\ branches}{Total\ \#\ of\ created\ branches}$$

$$Branches\ deleted\ Ratio\ = \frac{Total\ \#\ of\ branches}{Total\ \#\ of\ deleted\ branches}$$

I. Literaturverzeichnis

[1] A. Madaehoh und T. Senivongse, „OSS-AQM: An Open-Source Software Quality Model for Automated Quality Measurement," 2022.

[2] „CHAOSS," [Online]. Available: https://chaoss.community/. [Zugriff am 28 1 2023].

[3] A. Arya, C. Brown, R. Pike und O. S. S. Foundation, „Open Source Project Criticality Score (Beta)," 3 2023. [Online]. Available: https://github.com/ossf/criticality_score. [Zugriff am 30 1 2023].

[4] X. Tan, Y. Zhang, J. Cao, K. Sun, M. Zhang und M. Yang, „Understanding the Practice of Security Patch Management across Multiple Branches in OSS Projects," *WWW 2022 - Proceedings of the ACM Web Conference 2022,* pp. 767-777, 4 2022.

[5] S. Goggins, K. Lumbard und M. Germonprez, „Open Source Community Health: Analytical Metrics and Their Corresponding Narratives," 2021.

[6] H. Aman, A. E. Burhandenny, S. Amasaki, T. Yokogawa und M. Kawahara, „A Health Index of Open Source Projects Focusing on Pareto Distribution of Developer's Contribution," 2017.

[7] N. Alexan, R. E. Garem und H. Othman, „An extendible open source tool measuring software metrics for indicating software quality," 2016.

[8] M. Foucault, C. Teyton, D. Lo, X. Blanc und J.-R. Falleri, „On the usefulness of ownership metrics in open-source software projects," *Information and Software Technology,* Bd. 64, pp. 102-112, 2015.

[9] H. Zhong, Y. Yang und J. Keung, „Assessing the Representativeness of Open Source Projects in Empirical Software Engineering Studies," 2012.

[10] H. Wang und Z. Wu, „A Method to Evaluate the Health of OSS Projects Based on the PSR Model and Combination Weighting," 2022.

[11] BRR 2005 - RFC, „Business Readiness Rating for Open Source," 2005. [Online]. Available: https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/CMU_US/C050728W.pdf. [Zugriff am 23 Juli 2023].

[12] J. C. Munson und S. G. Elbaum, „Code churn: a measure for estimating the impact of code change," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, 1998.