Final Project: Sprint 2

December, 2023

Group 12 members:

Joshua Green

Ethan Lai

Cameron Meier

Forrest Pritt

Jack Riebel

**Bus Factor Requirements and Actors**

**Functional Requirements**

* **Data Extraction:**
  + The system must connect to a specified GitHub repository.
  + It should extract relevant data, including commit history, contributor details, and other pertinent contribution metrics.
* **Data Processing:**
  + The system must calculate the Bus Factor based on the extracted data. This involves sorting contributors by their contribution level and summing their contributions until reaching 50% of the total.
  + It should handle data normalization if contributions are of different types (e.g., code commits, documentation updates).
* **Graph Generation:**
  + The system must generate a visual graph representing the Bus Factor.
  + The graph should clearly indicate the key contributors and their proportion of total contributions.
  + Options for different graph types (e.g., bar chart, pie chart) should be available.
* **User Interface:**
  + Users should be able to input or select a GitHub repository from the interface.
  + The interface must display the graph and optionally other related analytics in a user-friendly manner.
  + The system should provide options to adjust the date range or other parameters affecting the Bus Factor calculation.
* **Real-time Updating (Optional):**
  + If required, the system should be capable of updating the graph in real-time as new contributions are made to the GitHub repository.

**Non-Functional Requirements**

* **Performance:**
  + The system should process and display the Bus Factor graph efficiently, with minimal latency.
* **Scalability:**
  + It must handle repositories with varying sizes and contribution volumes without significant performance degradation.
* **Security and Privacy:**
  + The system must comply with data privacy laws and GitHub's API usage policies.
  + Sensitive contributor information, if any, should be handled appropriately.
* **Usability:**
  + The user interface should be intuitive and accessible to users with varying levels of technical expertise.
* **Error Handling:**
  + The system should gracefully handle errors, such as inaccessible repositories or API limitations, providing clear messages to the user.

**Data Collection and Ethics**

* Adhere to ethical guidelines and legal compliance, especially regarding data privacy and usage of GitHub API.

**Documentation**

* Comprehensive documentation detailing setup, usage, and troubleshooting should be provided.

**Testing and Validation**

* Implement thorough testing to ensure accuracy in Bus Factor calculation and robustness in graph generation and data handling.

**Actors**

**1. Contributor**

* Role:
* They are the individuals who actively contribute to the GitHub repository. Their contributions can include code commits, documentation, issue reports, and more.
* Interaction with the System:
* Their contribution data is extracted and analyzed by the system to calculate the Bus Factor.
* They do not interact directly with the system but are the primary source of data for the Bus Factor calculation.

**2. Ospo Manager (Open Source Program Office Manager)**

* Role:
* Manages the company's open-source program and strategies.
* Oversees open-source projects and ensures compliance with open-source policies.
* Interaction with the System:
* Utilizes the system to monitor and assess the health and risks of open-source projects.
* Relies on the Bus Factor graph for strategic decision-making regarding resource allocation, community engagement, and risk management.

**3. Community Manager/Maintainer**

* Role:
* Responsible for managing the community around the open-source project and maintaining the project's day-to-day operations.
* Engages with contributors and coordinates contributions.
* Interaction with the System:
* Uses the system to understand the distribution of contributions and identify key contributors.
* Relies on the Bus Factor graph to inform strategies for community engagement and contributor retention.

**4. The Company/Individual Being Evaluated**

* Role:
* This could be a company or individual that owns or has a vested interest in the GitHub repository.
* They are interested in the sustainability and health of the project.
* Interaction with the System:
* Reviews the Bus Factor graph to understand the risk associated with their project.
* Uses the information for planning, such as risk management, contingency planning, and ensuring a balanced distribution of knowledge and skills in the project.

**System Interaction Summary:**

* Contributors are the data source for the system.
* Ospo Managers and Community Managers/Maintainers are primary users of the system, utilizing it for strategic and operational insights.
* The Company/Individual Being Evaluated uses the system for a progress assessment and overall project health analysis.

**Change Request Closure Ratio Requirements Specification**

**Introduction:**

·        Purpose

This metric model shines some light on the number of change requests comparative to the number of change requests closed for a project. Determining whether a project has enough maintainers to handle the number of change requests over a given time. If there are enough maintainers for a project, this metric can help show the number of merger conflicts to the amount of change requests.

**System Use:**

·       Interaction:

**A diagram of a project

Description automatically generated**

**Actor Survey:**

·        Maintainer

o   Maintainers are the people that decide whether a change request is accepted or denied. They are to actively check up on projects pull requests. They are to also tell the contributor what decision they have made for their request.

o   System Features

§  Merge Requests

§  Accept/Deny Request

·       Contributor

o   Contributors give suggestions on how to improve the project. After they have made changes to their cloned project, they are to request for a pull to the main project. They wait for maintainers to tell them if their request was pulled or not.

§  Clone Projects

§  Make Requests

·       Chaoss

o   Chaoss looks at the number of changes requests and compares it to the number of closed change requests to form a graph.

§  Views Project Request Data

**Requirements:**

·        Functional

o   Once a project is entered to be examine, Chaoss automatically obtains the information for the metric model.

·        Non-Functional

o   The information obtained should be efficient in making a graph of the comparison.

·        Triggers

o   For there to be any data in the graph a project must have at least one contributor making change requests.

·        Preconditions

o   There must be an existing project with some traffic for this metric model to form a graph.

·        Main Success Scenario

o   A project is picked that has an active community. There are contributors making requests within a period and enough maintainers to review and close those requests. The contributors are informed whether their requests have merged with the project or not.

·        Alternate Success Scenario

o   A project with some traffic occasionally gets change requests. There is at least one maintainer to review and close these requests.

·        Failed End Condition

o   There are active contributors making change requests, but there aren’t enough maintainers reviewing and closing those requests. Essentially getting overwhelmed. Or there are no contributors.

·        Functional Requirements

o   There must be maintainers.

o   There must be contributors.

o   Chaoss must have access to a projects pull request data.

**Time To First Response**

**Functional Requirements**

* **Data Extraction:**
  + The system must connect to a specified GitHub repository.
  + It should extract relevant data, including pull requests, responses to pull requests, and other pertinent contribution metrics.
* **Data Processing:**
  + The system must calculate the Time To First Response based on the extracted data. This involves calculating the time each pull request took to have another contributor interact with it.
    - It should handle data normalization if contributions are of different types (e.g., code commits, documentation updates).
* **Graph Generation:**
  + The system must generate a visual graph representing the Time To First Response average over a specified time period..
  + The graph should clearly indicate the number of pull requests along with the time period.
    - The lines should indicate the number of PRs at the time along with another line displaying the number of PRs that were promptly responded to.
* **User Interface:**
  + Users should be able to input or select a GitHub repository from the interface.
  + The interface must display the graph and optionally other related analytics in a user-friendly manner.
  + The system should provide options to adjust the date range or other parameters affecting the calculation.
* **Real-time Updating (Optional):**
  + If required, the system should be capable of updating the graph in real-time as new contributions are made to the GitHub repository.

**Non-Functional Requirements**

* The system should process and display the Time to First Response graph efficiently, with minimal latency.
* It must handle repositories with varying sizes and contribution volumes without significant performance degradation.
* The system must comply with data privacy laws and GitHub's API usage policies.
* Sensitive contributor information, if any, should be handled appropriately.
* The user interface should be intuitive and accessible to users with varying levels of technical expertise.
* The system should gracefully handle errors, such as inaccessible repositories or API limitations, providing clear messages to the user.

**Data Collection and Ethics**

* Adhere to ethical guidelines and legal compliance, especially regarding data privacy and usage of GitHub API.

**Documentation**

* Comprehensive documentation detailing setup, usage, and troubleshooting should be provided.

**Testing and Validation**

* Implement thorough testing to ensure accuracy in Bus Factor calculation and robustness in graph generation and data handling.

**Actors (TTR)**

**1. Contributor**.

* Interaction with the System:
  + Can be filtered to consider Contributor action as a response to a pull request
  + Their contribution data is extracted and analyzed by the system to calculate the Time to First Response when a contributor submits a pull request.

**3. Community Manager/Maintainer**

* Interaction with the System:
  + The maintainer’s response to a pull request is one of the primary metrics needed to create the Time to Response Graph.
  + The Maintainers can use this system to encourage more activity as more engagement (shorter Time to First Response) can lead to more progress and determine which strategies are more effective based on graph trends.

**4. The Company/Individual Being Evaluated**

* Interaction with the System:
  + Reviews the Time to First Response graph to understand the state and activeness of their project.
  + Uses the information to determine effective strategies and changes based on trends when strategies were implemented to improve productivity.

**System Interaction Summary:**

* Contributors and Maintainers interactions with pull requests are the data source for the system.
  + This can be filtered to include or exclude certain actors contribution as a response to a pull request
* Ospo Managers and Companies/Individual are primary users of the system, utilizing it for strategic and operational insights.

**Release Frequency**

**Functional Requirements**

* Data collection (RF01)
  + The program should be able to collect data on the release dates of all releases, as well as the type of the release.
* Data aggregation (RF02)
  + The program should be able to group the collected data into useful categories and generate meaningful metadata.
* Display of prior releases (RF03)
  + The program should generate a graph of when prior releases occurred.
* Display of release gaps (RF04)
  + The program should generate a graph of the size of the gaps between releases.
* Display of graphs filtered by release types (RF05)
  + The program should generate additional graphs of type RF03 and RF04 that only include releases of the same type. Thus, there will be a pair of graphs for major releases, a pair for minor releases, etc.

**Non-functional Requirements**

* Good performance (RFn01)
  + The program should display the generated data within 0.5 seconds of receiving the data.
  + Additionally, the program should only make external data requests once, so as to not overburden external connections.
* Accessibility (RFn02)
  + The program should create graphs that adhere to relevant guidelines for accessibility (eg high contrast, large font size, colorblind-friendly color schemes, etc.)
* Time restriction (RFn03)
  + The program should allow users to restrict the period of time the graph/data represent.
* Security/privacy (RFn04)
  + The program should be sufficiently secure and comply with relevant data protection laws and GitHub’s API usage policies.

**Actor Survey**

* Contributor
  + Contributors are developers who have contributed to an open source project, though they do not hold any position within the project. These actors would be interested in RF03, RF04, and RF05, for a succinct description of the project’s health and a sense of its release cycles.
* OSPO Manager
  + Open Source Program Office (OSPO) Managers act as the overseers of an open source project. Naturally, they would be interested in RF03-RF05, so as to gain a better understanding of the current health of the project. They would also be interested in RFn03, to determine when the project has been doing well and when it has been doing poorly.
* Community Manager
  + The community manager manages the community of the open source project. They would be particularly interested in RF04, which would help them coordinate release frequencies.
* Relevant Companies
  + These are companies that are either interested in or already engaged in an open source project. They would be interested in RF03, as this would allow the company to coordinate their own schedule with that of the open source project, for example by ensuring internal projects are not due when the project is preparing to release.

**Use Case: Contributors**

           Contributors who have become more familiar with the project, or conscientious new contributors, may desire to determine the release cycle of a project, so as to avoid creating a sweeping change at an inopportune time. They may also want to determine how much time they have to work on or implement changes.

**Triggers**

1. A contributor seeks to determine when releases occur.
2. A contributor seeks to determine the length of time a release cycle lasts.

**Actors**

1. Contributors

**Preconditions**

1. Actor provides link to GitHub repository of interest.

**Main Success Scenario**

1. All statistics that can be computed are displayed to the user.

**Failed End Conditions**

1. The user is unable to view the statistics.
2. There is not enough data to calculate meaningful statistics.
3. The link given by the user is incorrect.

**Steps of Execution**

1. The user enters a link to the repository.
2. Statistics concerning the release frequency are displayed.

**Use Case Diagram**

A diagram of a software development

Description automatically generated

**Community Service and Support Metric Model**

**Introduction**

The "Community Service and Support" metric model measures the quality of services and support provided by the community as directly perceived by a developer during the contribution process. It's a critical metric for assessing the efficiency and responsiveness of the review process in software project management. This metric provides insights into the communication process between the developers and the community. It's crucial for maintaining an active and engaged contributor base. The scope of this metric model is confined to the creation and maintenance of issues and change requests in software development projects, primarily using platforms like GitHub.

**Actors**

* Community (Contributors):
  + Role: Individuals or teams who submit change requests or pull requests.
  + Significance: They can review project health by viewing issue age and analyzing average response durations on change requests.
* Developers:
  + Role: These can be project maintainers, senior developers, or designated reviewers responsible for resolving issues and issuing change requests.
  + Significance: They directly influence the review duration and are the primary focus of this metric model.
* Program Manager:
  + Role: The overall lead of the project.
  + Significance: This user monitors overall health of the project and alerts developers of any longstanding issues or change requests.

**Functional Requirements**

* Metrics Collection and Analysis
  + Ability to accurately collect and analyze data for defined metrics like issue response time, issue age, change request reviews, etc.
  + Automated tools to aggregate and process data from community platforms.
* Reporting and Visualization
  + Generation of clear, insightful reports and visualizations that help stakeholders understand the community's performance.
* Alerting Mechanism
  + Notifications or alerts for metrics that indicate significant deviations or areas of concern.
* Data Export and Integration
  + Capability to export data in various formats and integrate with external analysis tools.

**Non-Functional Requirements**

* Performance
  + System should handle data processing and analysis efficiently, with minimal latency.
* Scalability
  + Capability to scale in handling larger datasets as community grows.
* Reliability
  + High reliability in data collection and analysis, ensuring data accuracy.
* User Accessibility
  + Intuitive interfaces for various user roles to access and understand metric data.
* Data Security and Privacy
  + Ensure compliance with data protection regulations and maintain user privacy.

**Change Request Review Duration**

**Introduction**

The "Change Request Review Duration" metric measures the time taken to review change requests on software project development processes. It's a critical metric for assessing the efficiency and responsiveness of the review process in software project management. This metric provides insights into the review process's timeliness, highlighting potential bottlenecks and inefficiencies. It's crucial for maintaining an active and engaged contributor base and ensuring the code base remains up-to-date and synchronized. The scope of this metric is confined to the review cycle of change requests in software development projects, primarily using platforms like GitHub.

**Actors**

* Submitters (Contributors):
  + Role: Individuals or teams who submit change requests or pull requests for review.
  + Significance: They initiate the review process and are directly affected by the duration it takes for their requests to be reviewed.
* Reviewers:
  + Role: These can be project maintainers, senior developers, or designated reviewers responsible for examining and approving change requests.
  + Significance: They directly influence the review duration and are the primary focus of this metric.

**Data Requirements**

* Data Points
  + Timestamp of change request submission and review completion.
  + Reviewer identification (name or ID).
  + Reviewer role (maintainer, reviewer, member).
* Source
  + Data will be sourced from platforms like GitHub.
* Collection Method
  + Data is collected using GitHub’s API or helper programs such as Augur.
  + Collected data is processed into graph form for publication through CHAOSS.

