Requirement Analysis Document

* Project Name: Project Visualization Tool
* Phase: First development iteration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Implemented By** | **Revision Date** | **Approval Date** | **Approved By** |
| 0.0.1 | De Luisi Gerardo | 27/01/2025 | 27/01/2025 | De Luisi Gerardo |

Table of contents

[Introduction 4](#_Toc188975255)

[Purpose 4](#_Toc188975256)

[Scope 4](#_Toc188975257)

[Current System 5](#_Toc188975258)

[Proposed System 6](#_Toc188975259)

[Overview 6](#_Toc188975260)

[Functional requirements 6](#_Toc188975261)

[Commit History Visualization 6](#_Toc188975262)

[Issue Tracking 7](#_Toc188975263)

[Author Blame 7](#_Toc188975264)

[Truck Factor Calculation 7](#_Toc188975265)

[SATD Analysis 7](#_Toc188975266)

[Nonfunctional requirements 8](#_Toc188975267)

[Performance 8](#_Toc188975268)

[Usability 8](#_Toc188975269)

[Compatibility 8](#_Toc188975270)

[Security 8](#_Toc188975271)

[Constraints 9](#_Toc188975272)

[Technical Constraints 9](#_Toc188975273)

[Resource Constraints 9](#_Toc188975274)

[User Constraints 9](#_Toc188975275)

[Environmental Constraints 9](#_Toc188975276)

[System model 9](#_Toc188975277)

[Components 9](#_Toc188975278)

[Data Flow 10](#_Toc188975279)

[Data dictionary 11](#_Toc188975280)

[Repository 11](#_Toc188975281)

[Commit 11](#_Toc188975282)

[Author 11](#_Toc188975283)

[SATD 11](#_Toc188975284)

[Class diagrams 12](#_Toc188975285)

# Introduction

## Purpose

The purpose of this document is to outline the requirements for the project visualization tool integrated with GIT. This tool is designed to enhance project management by providing insights into key metrics such as commit history, author contributions, truck factor calculations, and SATD (Self-Admitted Technical Debt) analysis throughout the project's lifecycle.

## Scope

This document will cover both functional and non-functional requirements for the tool. It serves as a comprehensive guide for the development team to ensure that all necessary features are implemented effectively, and it will also help in validating the final product against these requirements.

# Current System

No current system exists prior to the one in development

# Proposed System

## Overview

The project visualization tool is designed to enhance software project management by providing visual insights into various critical aspects of project development. This tool aims to empower teams by offering a comprehensive view of commit history and facilitating the analysis of contributions from different authors.

Key features of the tool include:

* **Commit History Visualization**: Users can view a graphical representation of the commit history, with filtering options to analyze specific time frames or contributions by individual authors.
* **Author Blame**: The tool provides visibility into who made changes to the codebase, allowing teams to understand contributions and accountability.
* **Truck Factor Calculation**: By calculating the truck factor, the tool highlights potential risks to project continuity by identifying how many team members are crucial to the project’s success.
* **SATD Analysis**: The tool identifies and categorizes Self-Admitted Technical Debt, helping teams prioritize areas that need attention to improve code quality.

Both functional and non-functional requirements essential for the development of the tool, ensuring that it meets the needs of users across various technical backgrounds. By adhering to these requirements, the project aims to foster better project management practices and improve overall team efficiency.

## Functional requirements

### Commit History Visualization

* The system shall display a graphical representation of the commit history.
  + **Detail**: The visualization should include a timeline view showing commits over time, with color coding to indicate different types of commits (e.g., feature, bug fix, documentation).
* The system shall allow users to filter commit history by date, author, and commit message.
  + **Detail**: Users should be able to use dropdowns or date pickers to filter the commit history, and search functionality should be available for commit messages to quickly locate specific changes.

### 

### Author Blame

* The system shall allow users to view the history of changes made by each author.
  + **Detail**: Users should be able to select an author and view a list of all commits made by that author, including details about each commit (e.g., date, commit message).

### Truck Factor Calculation

* The system shall calculate the truck factor of the project.
  + **Detail**: The truck factor is defined as the number of team members that must be hit by a truck to jeopardize the project's continuity. The system should analyze commit history to determine the number of authors that contribute to a significant portion of the codebase.
* The system shall present the truck factor in a user-friendly format.
  + **Detail**: The truck factor should be displayed as a numeric value, along with visual indicators (e.g., graphs or alerts) to highlight areas of risk.

### SATD Analysis

* The system shall identify and visualize Self-Admitted Technical Debt throughout the project history.
  + **Detail**: The system should scan commit messages and comments for keywords and phrases that indicate technical debt (e.g., "TODO", "fix later").

## Nonfunctional requirements

### Performance

* The system shall be able to handle projects with a large number of commits efficiently.
  + **Detail**: The tool should load and render commit histories for projects with thousands of commits without significant delays.

### Usability

* The user interface shall be intuitive and easy to navigate for users of varying technical expertise.
  + **Detail**: The design should follow UI/UX best practices, ensuring that features are easily accessible and understandable for both technical and non-technical users.
* The system shall provide tooltips and help documentation for all features.
  + **Detail**: Users should have access to contextual help that explains each feature and how to use it effectively.

### Compatibility

* The system shall be compatible with major OS (e.g., Unix-Based, Windows).
  + **Detail**: The tool should work seamlessly with these operative systems, allowing users to connect their repositories without extensive configuration.

### Security

* The system shall ensure that user data and project information are stored securely.
  + **Detail**: Data should be encrypted in transit and at rest, and access controls should be implemented to restrict unauthorized access.

## Constraints

### Technical Constraints

* **Integration Limitations**: The tool must be compatible with specific Git hosting services (e.g., GitHub), which may impose limitations on the APIs available for integration.
* **Performance Limitations**: The performance of the tool may be constrained by the size of the repository and the number of commits, which could affect loading times and responsiveness.

### Resource Constraints

* **Time Constraints**: There may be strict deadlines for the project delivery, which can impact the scope of features that can be developed and tested within the available timeframe.

### User Constraints

* **User Skill Level**: The tool must be designed to accommodate users with varying levels of technical expertise, which may constrain the complexity of features and user interface design.

### Environmental Constraints

* **Deployment Environment**: The tool must be deployable in various environments which can impose constraints on architecture and resource requirements.
* **Network Dependencies**: The functionality of the tool may be dependent on network connectivity to access Git repositories, which could limit usability in offline scenarios.

## System model

### Components

#### User Interface

* **Description**: The front-end component that users interact with. It provides visualizations of commit history, issue tracking, author blame, truck factor, and SATD analysis.
* **Key Functions**:
  + Display graphical representations of data.
  + Provide filters and search options.
  + Allow users to create and manage issues.

#### Application Server

* **Description**: The back-end component that handles requests from the User Interface and manages business logic.
* **Key Functions**:
  + Process user inputs and commands.
  + Communicate with the GIT Integration Module and Database.
  + Serve data to the User Interface in a user-friendly format.

#### CLI interface

* **Description**: This module’s responsibility is to expose the tool’s functionality through a Command Line Interface (CLI) from which the repository of interest will be fetched.
* **Key Functions**:
  + Process user inputs and commands.
  + Communicate with the GIT Integration Module and Database.
  + Serve data to the User Interface in a user-friendly format.

#### Git Integration Module

* **Description**: This module interfaces with external Git services to retrieve and manipulate repository data.
* **Key Functions**:
  + Fetch commit history and related metadata.
  + Calculate truck factor and identify SATD items from commit messages

#### External Git Services

* **Description**: The Git hosting platforms the tool integrates with for version control data.
* **Key Functions**:
  + Provide APIs for accessing repository information.

### Data Flow

1. **User Interaction:** Users interact with the CLI to start the tool in the chosen git repository
2. **GUI Interaction**: Users interact with the User Interface to request specific data (e.g., view commit history, track issues).
3. **Request Handling**: The Application Server receives these requests and processes them accordingly.
4. **Data Retrieval**: The Application Server communicates with the Git Integration Module to fetch the required data from the External Git Services.
5. **Retrieval**: The Git Integration Module may also store or retrieve data from the Database as needed for analytics and reporting.
6. **Response Generation**: The Application Server compiles the retrieved data and sends it back to the User Interface for display.

### Data dictionary

### Repository

* **Description**: Represents a software project managed through the tool.
* **Attributes**:
  + **RepositoryID** (Primary Key): Unique identifier for each project.
  + **RepositoryName**: The name of the project.
  + **RepositoryURL**: The URL of the associated GIT repository.
  + **CreatedDate**: The date the project was created.
  + **LastUpdated**: The date the project was last updated.

### Commit

* **Description**: Represents a single commit in the project's Git repository.
* **Attributes**:
  + **CommitID** (Primary Key): Unique identifier for each commit.
  + **AuthorID** (Foreign Key): Identifier linking to the author of the commit.
  + **CommitMessage**: The message associated with the commit.
  + **CommitDate**: The date and time when the commit was made.
  + **FilesChanged**: List of files that were changed in the commit.

### Author

* **Description**: Represents a user who has made commits in the project.
* **Attributes**:
  + **AuthorID** (Primary Key): Unique identifier for each author.
  + **Username**: The name of the author as it appears in commits.
  + **Email**: The email address of the author.
  + **TotalCommits**: Total number of commits made by the author in the project.

### SATD

* **Description**: Represents instances of self-admitted technical debt identified in commit messages or comments.
* **Attributes**:
  + **CommitID** (Foreign Key): Identifier linking to the commit where the SATD was mentioned.
  + **Description**: Description of the technical debt.
  + **Category**: Category assigned to the SATD (e.g., Code Smell, TODO).

### Class diagrams