CSE 563 Project Individual Report Number 3

Sai Ganesh Nindra

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1. Customer Problem (Updated)

1.1. Lack of Agile-centric effort tracking

1.1.1. Ineffective effort logging for agile teams

- To make EffortLogger a useful tool in modern day businesses it needs to cater to modern-day practices followed by businesses like the Agile methodology. As stated in 'Agile Processes and Methodologies: A Conceptual Study[1] agile methodology follows an iterative and incremental process, where requirements can be changed with respect to the customer needs
- Effort Logger is not ideal for a fast-paced environment that is around in the present day. EffortLogger doesn't provide any estimation tools either which are widely used in agile teams such as planning poker. Integration of estimation techniques such as planning poker would be good.
- The ideal approach to estimation techniques to include would be machine learning. Using data-based estimation over an expert-based estimation method would prove to be more useful. Using metrics such as Median of Magnitude of relative error shows more accuracy in data-based estimation.[2]

1.1.2. Role-Specific Features

- EffortLogger treats everyone in the same manner, as in, if the person in question is a developer, project manager, product owner, tester or a scrum master, effort logger doesn't provide any features for different members of a team
- There needs to be a certain structure wherein only users with certain access can use certain features like accessing historical data, reports and dashboards

1.2. No integration with agile tools

1.2.1. Inefficient method of tracking effort and defects

- Effort and defect are currently tracked using user input, when the task is started and when it is complete. This can be inaccurate and misleading, which can give a wrong picture of how much effort has been put and what defects occurred.
- Integration with tools like IDE's, Jira and GitHub can help track efforts based on user activity, commits and if any updates have been made by the user. This eliminates guesswork keeping a data-driven approach in the future more accurate.

1.2.2. No method to track current efforts for future use

---

[1] Sharma, Sheetal & Sarkar, Darothi & Gupta, Divya. (2012). Agile Processes and Methodologies: A Conceptual Study. International Journal on Computer Science and Engineering. 4.

[2] Fernández-Diego, Marta & Mendez, Erwin & L. Guevara, Fernando González & Abrahão, Silvia & Insfran, Emilio. (2020). An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review. IEEE Access. 8. 10.1109/ACCESS.2020.3021664. DOI:10.1109/ACCESS.2020.3021664

- The current data which is stored may have inaccuracies and are not used to compare with historical data. Agile teams need insights into how much effort is spent on completing tasks.
- With dashboards and visualization tools, agile teams can compare historical effort data with current effort data which will help in improving estimation strategies for the future

1.3. Privacy and Security of data

1.3.1. Security of Data

- EffortLogger was developed in the 1990's and there are multiple new standards the industry has adopted to safeguard themselves from attacks and breaches which need to be adopted by EffortLogger.
- Old software is very vulnerable to attacks which makes it susceptible to data breaches and hacks.
- The system needs to be equipped with robust security features like multi-factor authentication and data encryption.[3]
- The data that is being used as historical data must be done by the employees themselves, they cannot view past user stories worked on by other employees ensuring employee data privacy

1.3.2. Compliance with rules and regulations

- Rules and regulations have been ramped up since the 1990's when EffortLogger was developed and will need updates to comply with the current regulations.
- It will lack basic encryption and will be following different data policies.
- We will need to ensure that it is certified so that it is following various government regulations with respect to data privacy and security.

1.4. No integration with cloud

1.4.1. Data stored locally

- The current version of EffortLogger stores all data locally which is in turn stored on a floppy disk. This can cause loss of data and inefficiencies in usage of such data.

- There are tracking delays as well and any updates require a lot of process as accessing previously logged data becomes a time-consuming process.

[3] What is Data Protection and Privacy? n.d, https://cloudian.com/guides/data-protection/data-protection-and privacy-7-ways-to-protect-user-data/

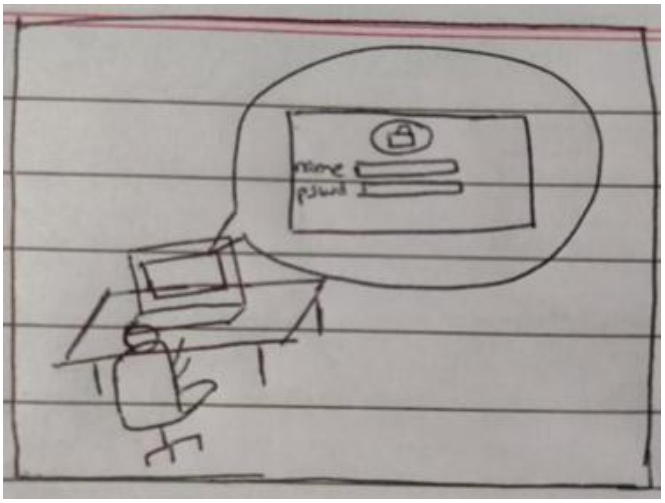
- Integration with cloud opens many doors such as no inconsistency in data, convenience in updating, automatic backups which result in no loss of data and this data can be accessed from anywhere.

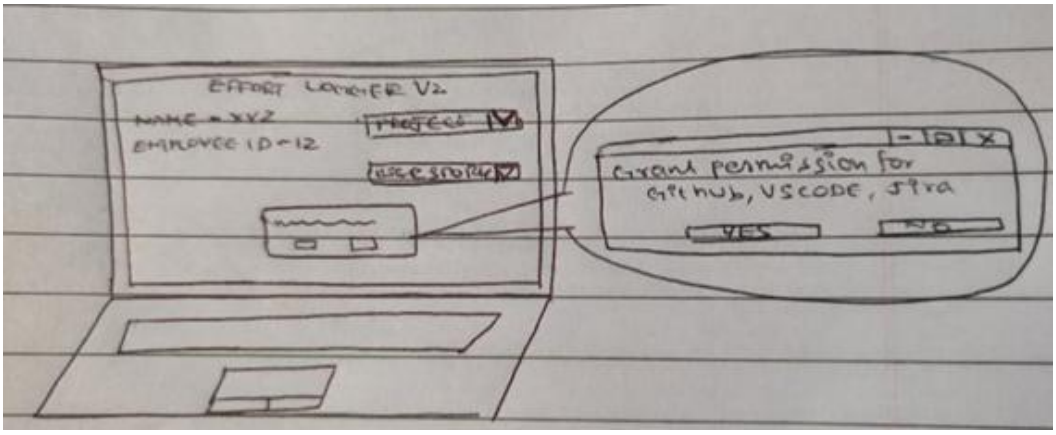
~~1.4.2. Multiplatform compatibility~~

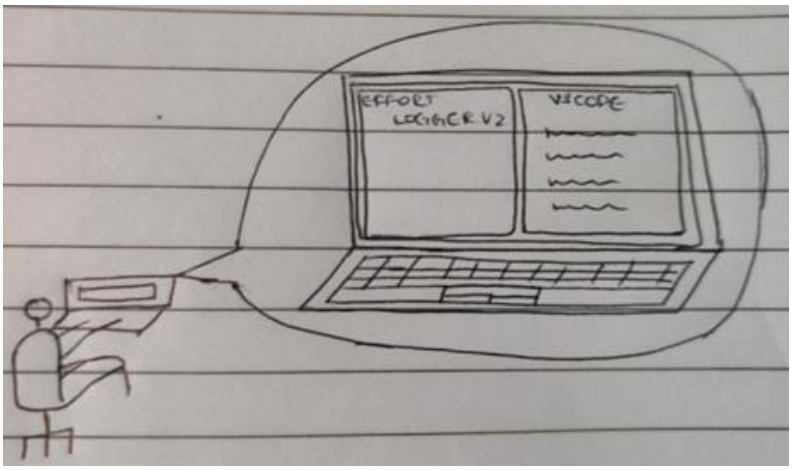
- ~~EffortLogger was just an application on the computer which causes inconvenience in modern day as users are on the go always. Not everyone has access to a computer at all times.~~
- ~~Making EffortLogger accessible on mobile devices as well would help users update their work from anywhere.~~

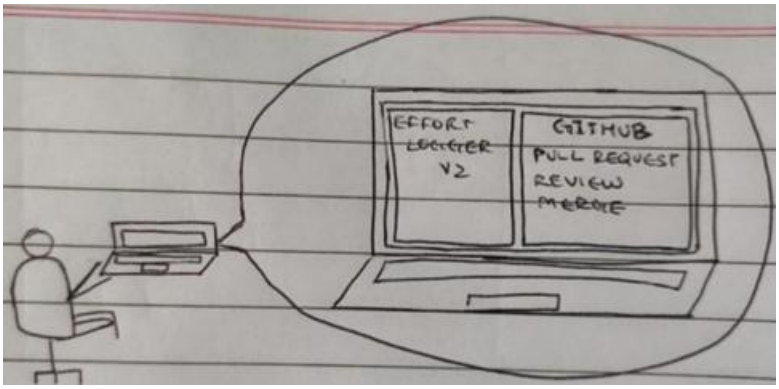
2. Prioritized Operations Concepts

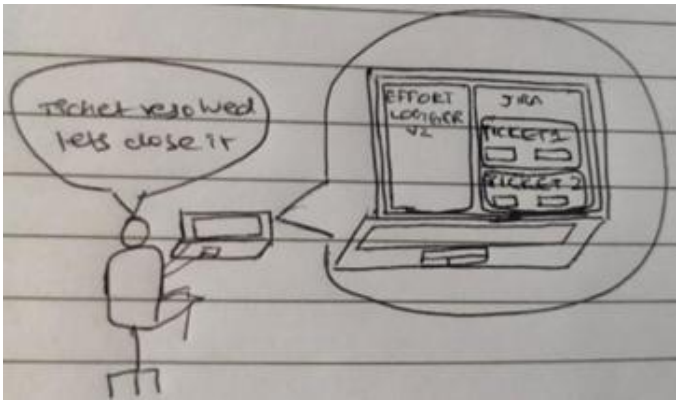
2.1. Storyboard

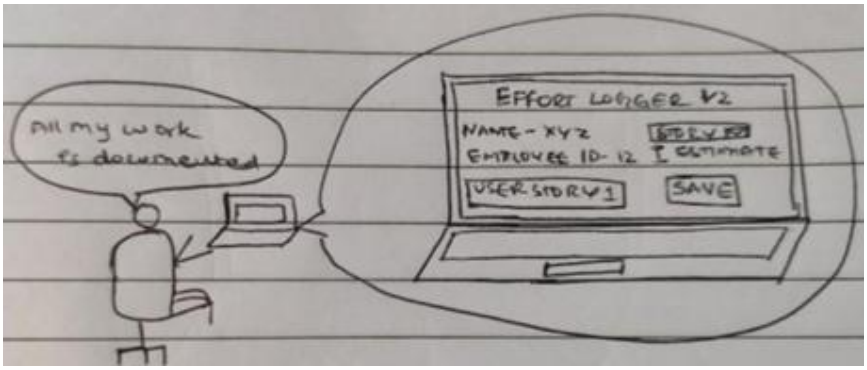












2.1.1. User Authentication

- Developer logs into the EffortLogger using their credentials.
- These credentials are private data and unique for every employee.
- After logging in, a dialog box pops up asking to grant permissions to connect to agile tools.

2.1.2. Developer starts working

- Developer opens IDE and starts working. (writing code)
- The storyboard shows EffortLogger and IDE on a split screen, but it can run in the background as well.

2.1.3. Code Merge

- Developer creates a pull request which is reviewed by a team member and is merged.
- EffortLogger running in the background tracks commits, pull requests and merges.

2.1.4. Task solved

- Once a task is solved, a ticket can be closed on tools like Jira.
- Developer manually closes tickets on an agile tool which will be automatically detected by EffortLogger which is running side by side.

2.1.5. Review and Save

- Developer goes through the user story data, reviews it and saves it which nevertheless automatically gets saved.

2.2. Introduction

2.2.1. Project Description

- The latest EffortLogger system is a product that aims to make a shift in software estimation from opinion based to a data driven approach. The new system will collect and store historical data, which will be used while incorporating agile methodologies to improve the accuracy of story point estimation.

2.2.2. Background

- The conventional way of effort estimation is carried out in a very subjective manner, often leading to inaccurate estimates which in turn cause project delays or just make the entire process more lengthy than necessary. The original EffortLogger was developed in the late 20th century which doesn't support any of the modern-day methodologies[4]. This newer version of EffortLogger will incorporate modern day practices, improve estimations with a shift to a more data driven approach, project planning and all in all the way software development process takes place.

2.2.3. Assumptions and Constraints

- Assumptions
    - The users must be willing to adopt this new version
    - Historical data must be available and relevant[5]
- Constraints
    - The new version of EffortLogger must be able to integrate with the tools being used in the industry currently. (For example, Jira, IDE's)
    - The software must be in compliance with all rules and regulations be it related to data, security or laws

[4] Internal document: EffortLogger Business Opportunity (Document V1.2) Pg.5
[5] Internal document: EffortLogger Business Opportunity (Document V1.2) Pg.6

2.3. Overview of the Envisioned System

2.3.1. Overview

- The focus with this new version of EffortLogger is tracking developer effort is a better and more accurate manner which can further help in story points estimation. When integrated with planning poker it should provide historical data to assist and help developers make data driven decisions.

2.3.2. System Scope

- User authentication
- Tracking effort of users.
- Logging of defects associated with a user story.
- Integration with agile tools.
- Provide historical data during planning poker sessions. (Integration of Planning Poker with EffortLogger)
- Data analysis and visualization of various metrics for higher officials.
- Security and privacy measures

2.4. Needs, Goals and Objectives of Envisioned System

- Accurate effort tracking using metrics apart from time and lines of code
- Store data of user story to serve as historical data for future use.
- Improve estimation accuracy by using historical data
- Provide insights and help to developers, managers and executes on previous work, team efforts and business progress.
- Ensure data privacy and security

2.5. Overview of System and Key Elements

- Integration Layer: Connection between EffortLogger and agile tools to capture effort metrics
- Tracking and Logging module: Logs effort and defects.[6]
- Suggestive Algorithms: Provides data-driven suggestions during Planning Poker
- Data Analysis Engine: Processes estimate data to provide insights in dashboards and visuals.
- Database: Stores effort data and user story data.

[6] Internal document: EffortLogger V1-11 User Guide V1.1 Pg.2

2.6. Proposed Capabilities

2.6.1. Effort calculation operation

- Integrates with agile tools to track effort.
- Analyzes time spent on different activities within a user story, such as coding, testing, and debugging.
- Effort can be estimated looking into the variety of skills required, number of tickets being assigned on a regular basis, time taken to resolve tickets and how many are still pending.
- Number of defects logged and resolved can also depict effort put into working on a task.
- Number of modifications in code can show how much effort has been put into developing something.
- Analyzing pull requests and reviews while merging code.
- Measuring the amount of work completed in each sprint can provide insight into team effort over time which in other words would be sprint velocity.

2.6.2. Description and Tagging Operation

- Allow developers to add a description to user stories which will help in tagging user stories using the keywords used in the description.
- Developers add skills and tools used for the user story while working on it.
- Tags are assigned automatically using algorithms (NLP) which pick the keywords in description.
- Historical data includes both descriptions and skill tags for future reference.

2.6.3. Integration with agile tools Operation

- Provides a data-driven approach only for the initial phase for discussions during estimation.
- Historical data can be accessed of only the employee who is trying to access the data. No other data should be visible.[7]
- During Planning Poker sessions, the system automatically (Machine Learning) suggests similar historical user stories based on tags with skills used and initial estimate of story points and final story points.

2.6.4. Analytical Operations

- Generates reports on team performance, estimation accuracy, and productivity trends.
- Restricts access to detailed team analytics to Product Owners or Managers only.
- Provides customizable dashboards for different management levels. (Developer level/Executive Level).

[7] Internal document: EffortLogger Supervisor Input V1-2 2024-01-06 Pg.1

2.7. Operational Scenarios

2.7.1. Nominal Conditions

- Developer Logging Effort upon completion of user story
    - Developer logs into the EffortLogger system.
    - Developer selects the current user story they are working on.
    - Developer writes code and does their work.
    - Upon completion of work, they close all open tickets and initiate a pull request.
    - EffortLogger tracks all the effort and stores the data automatically which can be reviewed by the developer.
    - They add tags for skills used (e.g., "Java", "RESTful").
    - User story gets added into database with its tags of user story description and skills used.
- Product owner (PO) prepares sprint planning using Planning Poker (Online).
    - PO initiates a new Planning Poker session in the system.
    - PO inputs the new user stories for estimation.
    - For each story, the system shows each member similar past stories based on user story tags.
    - The team discusses and votes on estimates using historical data as a reference.
    - Upon completion of sprint, PO investigates dashboards to see teams' efforts over the duration of the sprint. (Provides insights for future).

2.7.2. Off-Nominal Conditions (TBD)

2.8. Risks and Potential Issues

- Employee data privacy concerns.[8]
- Unacceptance by users due to productivity reasons.
- Over reliance on historical data for estimations of new user stories.
- Integrating with each tool used in agile methodology might pose challenges.

[8] Internal document: EffortLogger Customer Need V2-0 Document V1-2 Pg.2

3. Received Requirements

Functional Requirements

3.1. The system shall provide effort tracking and access control mechanism.

3.1.1. The system shall automatically track efforts using IDE integration.

The system will automatically log the time spent on coding debugging and testing done using IDE's. This automation will eliminate manual input and reduce errors and inconsistencies.

3.1.2. The system shall track user activity, commits and task completions in IDE and agile tools.

The system will monitor various user activities such as commits, pull requests and merges on GitHub and tickets open and closed in tools like Jira. It helps in gaining insight on individual as well as team performance.

3.1.3. The system shall store effort data against the user story for future use.

All tracked data is ultimately stored against the respective user story to maintain a historical record for future purposes where in, it may help in estimation of new user stories.

3.2. The system shall provide user authentication and access control.

3.2.1. The system shall allow the users to log in using their respective credentials.

All types of users can log in to the EffortLogger with their respective credentials which are unique, which restricts unknown users from accessing one's system, ensuring data privacy and security.

3.2.2. The system shall provide a two-factor authentication system for added security.

A two-factor authentication system can deny access if any other person is trying to enter since there are two steps to by pass (added security). Usually 2FA is done through and OTP or Email which again can be accessed by the user itself and no one else.

3.2.3. The system shall restrict access to certain functionalities based on the type of user.

Certain functionalities are blocked or restricted to ensure data privacy. Developers are allowed to view only their data, whereas high ranked officials have access to reports and dashboards that provide insights on individual as well as team performance.

3.2.4. The system shall limit access to historical data.

Users will only be allowed to access their own historical effort data, preventing unauthorized access to other users' logs. This ensures privacy and prevents potential data leaks.

3.3. The system shall integrate with agile technologies

3.3.1. The system shall integrate with Jira to track ticket status changes.

The system will integrate with Jira, an agile tool that is used to keep track of tickets. Developer's effort data is linked to Jira issues for better traceability.

3.3.2. The system shall monitor GitHub pull requests and merges to track developer activity.

The system integrates with an agile tool called GitHub where it will monitor all the actions by the user like commits, pull requests and merges to track users' effort. This allows users to know when work was completed and merged into main branch.

3.3.3. The system shall track real-time activity of user from IDE's.

The system shall track in real-time the amount of time spent by user on IDE's on coding and debugging tasks.

3.4. The system shall provide effort estimation data using machine learning.

3.4.1. The system shall analyze historical data using machine learning algorithms.

The system will leverage machine learning models to analyze historical data and provide estimates for similar tasks using a weight that determines how closely the historical data is related to the current user story.

3.4.2. The system shall use NLP techniques to assign tags based on descriptions.

Natural Language Processing techniques will be used by the system to assign relevant tags based on user story descriptions. This helps in categorizing the user story and will help in ease of filtering through historical data during estimation.

3.4.3. The system shall suggest similar past user stories using NLP-based keyword matching

The system will use NLP-based keyword matching to filter out and retrieve historical data where past user stories are similar to the current user story. This helps the users in getting a better and more accurate estimate while in estimating during Planning Poker sessions.

3.5. The system shall provide insights by analyzing the data through dashboards and reports.

3.5.1. The system shall generate real-time reports on certain metrics.

The system will provide real-time reports that help in tracking an individuals as well as the teams performance by visualizing certain metrics such as sprint velocity, estimation accuracy using previous estimate, current estimate and final story points.

3.5.2. The system shall provide historical comparisons via reports and dashboards.

Higher level employees can have access to reports and dashboards that give insights into a team/individual's past performance compared to current performance. It helps in seeing growth and productivity as well.

3.5.3. The system shall allow authorized users to access effort analytics.

Effort analytics of all types can only be accessed by a higher-ranking user such as project managers or high level executives to gain insights into overall team/division's performance. This process is facilitated through reports and dashboards. This prevents unauthorized access and data leaks. This helps the decision-making officials to gain the necessary insights

Non-Functional Requirements

3.6. Privacy and Security compliance

3.6.1. The system shall encrypt stored and transmitted data using AES-256 encryption.

The system will use AES-256 encryption to protect stored and transmitted data which ensures data privacy an security and at the same time complies with industry standards.

3.6.2. The system shall ensure compliance with GDPR and industry security standards.

The system will be in compliance with GDPR which is The General Data Protection Regulation, a law that protects the personal data of people in the EU. Following such policies and standards can help the system thrive in various regions gaining wider acceptance.

3.6.3. The system shall prevent users from accessing other employees' effort logs.

The users can only access their data from the EffortLogger, even high ranking users cannot view individual data to the T. But they do have privileges for dashboards and reports. This prevents users from tampering with other employees' effort data.

3.7. Database requirements

3.7.1. The system shall store effort data.

All tracked effort needs to be stored in a structured database, ensuring data persistence. It must store the user story name that was implemented, original estimate of story points, actual number of story points required, programming language used and any other factors which maybe inputted as tags that could have influenced the effort.

3.7.2. System shall implement automatic data backups every 24 hours.

The system takes a backup every 24 hours to prevent loss of data due to system failures.

3.8. Performance requirements

3.8.1. The system shall process effort tracking data with <2 seconds of delay.

The system will process the effort tracking data and ensure the users can keep working without worrying about their work being tracked in real-time. Any noticeable lag, the software might lose its reliability and data might be inconsistent.

3.8.2. The system shall take less than 5 seconds to retrieve historical data.

Users should be able to retrieve their historical data withing 5 seconds under normal conditions, ensuring fast and efficient data access during estimation procedures like a Planning Poker session, any noticeable delay might make the users prefer an opinion-based approach over a time-taking data-driven approach.

3.9. Usability requirements

3.9.1. The system shall have a user-friendly UI requiring minimal training.

The system will have a user-friendly and easy to use UI. It reduce training time and can potentially cause widespread acceptance from its users.

3.9.2. The system shall use effective visual effects for effort and historical data comparisons.

The system will produce bar charts, pie charts, and many other visual indicators that represent the effort data and its details . This helps the managers and product owners to gain insights on the team’s performance.

3.10. Reliability and Portability requirements

3.10.1. The system shall maintain 99.9% uptime.

The system will be built in such a way that it ensures 99.9% uptime, i.e. high availability.

3.10.2. The system shall be deployable on Windows, MacOS, and Linux environments.

The system will be deployable across multiple platforms so that all users can use the system. This increases the accessibility of the system.

4. Traceability

4.1. The system shall provide effort tracking and access control mechanism.

4.1.1. Traceability from Source to the Requirement

Current agile methodologies require accurate effort tracking, but EffortLogger lacks a good tracking mechanism. This absence leads to inconsistent and inaccurate effort data. EffortLogger currently depends on manual input from developers to log effort. Agile teams require an automated system that integrates with agile tools.

The operational concept's goal is for an automated effort logging system that integrates with IDEs, Jira, and GitHub to record effort as tasks are completed. This ensures accuracy in effort tracking, avoiding human errors and inconsistencies. This requirement addresses the issue of incorrect effort data that is captured by current EffortLogger.

4.1.2. Traceability from Requirement to the Architecture and Design

To accomplish this requirement, the system will implement a module that integrates with agile tools to capture user activities that depict the effort. In the background as the end user works on a user story, the EffortLogger captures the data through commits, pull requests, merges and tickets closed.

An EffortLogger API will process all the data which can be reviewed by the user. The system shall maintain synchronization with the agile tools to maintain consistency.

4.2. The system shall provide user authentication and access control

4.2.1. Traceability from Source to the Requirement

The current EffortLogger lacks a secure authentication and role based access control which may expose data to unauthorized users leading to data leaks. Without these features, users can access and modify logs of any user in the company, leading to data manipulation and security concerns.

This requirement ensures that every user is provided with a unique set of credentials and that there are certain features which only certain users can access. The operational concept's goal is to have a system where a user logs in securely and gains access to only functions pertaining to their level.

4.2.2. Traceability from Requirement to the Architecture and Design

To implement this requirement the system will need to include a module that verifies the credentials for a secure log in process through methods like two-factor authentication. A role based access control system will enforce rules and protocols where in the users can only access their own data maintaining data privacy.

4.3. The system shall integrate with agile technologies

4.3.1. Traceability from Source to the Requirement

Currently the EffortLogger tool does not integrate with any of the agile tools. In fact it doesn't go hand in hand with any of the agile methodologies. It requires manual input for effort data which results in inconsistent data.

The operational concept's goal is to facilitate seamless integration with the tools that are used in the present day like IDE's, GitHub, Jira and other agile tools to track commits, real-time activity and ticket status. By doing so the system shall eliminate manual errors and provide more consistent and accurate data.

4.3.2. Traceability from Requirement to the Architecture and Design

To fulfill this requirement the system will possess an integration layer that will handle communication between the EffortLogger and the agile tools via API's. A background service will fetch updates and help keep the data always in sync. The system will possess an architecture that includes a processing module that handles the integration data which again with the help of role-based access control only certain users will have access to this data.

4.4. The system shall provide effort estimation data using machine learning

4.4.1. Traceability from Source to the Requirement

Traditional effort estimation methods are opinion-driven, which are often subjective and are prone to be inaccurate. Agile teams require a data backed decision to eliminate inaccurate results, so to shift from an opinion-based approach to a data driven approach the system implements machine learning techniques to predict effort requirements for new user stories based on historical data using weights.

These weights determine how closely related the work is from historical data with the future work that needs to be done. Additionally, NLP can be used to tag and categorize previous user stories for better filtering while skimming through historical data.

4.4.2. Traceability from Requirement to the Architecture and Design

To accomplish this requirement a machine learning module is integrated into the system to analyze the historical data and provide estimations during planning poker sessions. Th NLP-based module helps in classifying user stories based on tags such as programming language used.

These insights help planning poker become a more data-driven approach as the algorithms suggest effort estimates based on historical data with respect to the weights assigned to them, i.e. the likelihood factor that the historical data matches the current data.

4.5. The system shall provide insights by analyzing the data through dashboards and reports

4.5.1. Traceability from Source to the Requirement

Managers and executives require insights on team performance, estimation data and team productivity trends. The current EffortLogger doesn't provide any such insights through reports or visualizations. The operational concept's goal is to create a dashboard driven system which generates detailed reports which show the necessary trends to the required users.

4.5.2. Traceability from Requirement to the Architecture and Design

For this requirement the system will consist of a dashboard and reporting module which will be designed to generate real-time reports on sprint velocity, estimation accuracy and team performance. It will present the metrics in the form of interactive visualizations that are effective and insightful. This module is proposed to be a role-based module to ensure that only those that have access can read into the entire teams performance and the working of their workers.

4.6. Privacy and Security Compliance

4.6.1. Traceability from Source to the Requirement

The original EffortLogger was developed without complying with modern security standards and protocols making it susceptible to data breaches and unauthorized access. It is crucial to protect sensitive data and ensure privacy of user data. The operational concept's goal is to ensure data privacy and safeguard company-sensitive information. Compliance with modern policies such as GDPR is vital to protect data

4.6.2. Traceability from Requirement to the Architecture and Design

The security architecture will include a AES-256 encryption that ensures security for stored and transmitted data, implement a two-factor authentication to prevent unauthorized access. The role-based access control will ensure that there are certain privileges in place that restrict certain users from accessing few functionalities. None of the users have access to others' historical data.

4.7. Database Requirements

4.7.1. Traceability from Source to the Requirement

The need for effort tracking and analysis in agile development requires efficient data storage. Tracking user efforts, defects, and historical progress requires a structured database that ensures accessibility and security.

Additionally, maintaining data integrity and avoiding loss due to system failures requires periodic backups. Thus, the requirements for storing effort data and implementing automatic backups align with the problem of ensuring persistent, reliable, and recoverable effort tracking data.

4.7.2. Traceability from Requirement to the Architecture and Design

The system architecture will include a relational database or a NoSQL database for scalable storage of effort-related data like effort logs, historical data and tracking data. Backup mechanisms will be implemented via cloud-based backup solutions that are executed every 24 hours. This ensures data persistence while aligning with security and compliance needs.

4.8. Performance Requirements

4.8.1. Traceability from Source to the Requirement

Effort tracking and estimation systems must operate in real-time. Delays in processing or retrieval of data can delay decision-making, impacting sprint. Hence setting a certain threshold can ensure that users receive timely updates on their tracked efforts, with minimal lag when accessing historical data.

4.8.2. Traceability from Requirement to the Architecture and Design

To meet these performance requirements, the system will use optimized database queries to enhance data retrieval speed. A distributed architecture with load balancing will ensure that concurrent users do not cause significant slowdowns. Streaming frameworks like Kafka will be used for real-time data processing, ensuring updates occur with less than a 2-second delay.

4.9. Usability Requirements

4.9.1. Traceability from Source to the Requirement

Agile teams consist of a variety users such as developers, testers, and managers who need to track efforts. A steep learning curve for effort-tracking tools can reduce adoption and efficiency. So, a user-friendly UI requiring minimal training ensures smooth onboarding. Visual indicators like graphs and charts in dashboards and reports can help users quickly interpret data trends, supporting better decision-making.

4.9.2. Traceability from Requirement to the Architecture and Design

The system UI will follow modern UI/UX design principles with an intuitive layout. It will incorporate dashboard components that provide clear visual indicators, including heatmaps, progress bars, and historical trends.

4.10. Reliability and Portability Requirements

4.10.1. Traceability from Source to the Requirement

Agile teams consist of a variety users such as developers, testers, and managers who need to

The system must remain operational with minimal downtime to ensure availability. Development teams use different operating systems, so the system must be cross-platform compatible to ensure broad usability and adoption.

4.10.2. Traceability from Requirement to the Architecture and Design

To maintain 99.9% uptime, the system will use cloud-based hosting. Microservices architecture will ensure that failures in one module do not bring down the entire system. For cross-platform deployment, the system will be developed using containerization to ensure execution across all platforms like Windows, MacOS, and Linux.