CSE 563 Project Individual Report Number 4

Sai Ganesh Nindra

Table of Contents

Table of Contents

Table of Contents

1. The Problem (Updated)

1.1. CP01: Tool Not Suited for Enterprise-Scale Agile

1.1.1. Original design lacked scalability

- 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 customer needs.
- The existing EffortLogger was not built to handle the complexities of enterprise-scale projects[2]. EffortLogger is not ideal for the fast-paced environment present today. EffortLogger doesn't provide any estimation tools either, which are widely used in agile teams, such as Planning Poker. Integration with estimation techniques like Planning Poker would be beneficial.
- The ideal approach to estimation techniques to include would be machine learning. Using data-based estimation over expert-based estimation methods would prove to be more useful. Metrics such as the Median of Magnitude of Relative Error show greater accuracy in data-based estimation[3].

1.1.2. Does not align with SAFe or other frameworks[4]

- Key enterprise needs like architecture planning and team-of-teams support are not addressed[4]. EffortLogger treats everyone in the same manner, whether the person is a developer, project manager, product owner, tester, or scrum master, EffortLogger doesn't provide any role-specific features for different members of a team.
- Role-based anonymization[5] is required that keeps personal data hidden. There needs to be a structure in place where only users with specific access can use certain features, such as accessing historical data, reports, and dashboards.

[1] Sharma, Sheetal & Sarkar, Darothi & Gupta, Divya. (2012). Agile Processes and Methodologies: A Conceptual Study. International Journal on Computer Science and Engineering. 4.
[2] Internal Document: EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.2
[3] 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
[4] Internal Document: EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.2
[5] Internal Document: EffortLogger Customer Need V2-0 Document V1-2.pdf, p.2

1.2. CP02: No integration with agile tools

1.2.1. Inefficient method of tracking effort and defects

- Effort and defects are currently tracked using user input, when the task is started and when it is completed. This can be inaccurate and misleading, potentially giving an incorrect picture of how much effort was invested 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.
- Integration with estimation tools like Planning poker[6] is very vital since these techniques use effort data to make estimations. Weight updates[7] and average calculations during estimation rounds are done manually, slowing down the process.

1.2.2. No method to track current efforts for future use

- The current data which is stored may have inaccuracies and are not used to compare with historical data. Agile teams need role-anonymized yet actionable analytics[8] 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. CP03: Privacy and Security of data

1.3.1. Inadequate privacy protection for users

- 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. Existing storage systems[8] may expose personal performance data if anonymization is not enforced.
- Old software is very vulnerable to attacks which makes it susceptible to data breaches and hacks. Use of shared drives and web-based systems creates risks of ransomware[8] and breaches.
- Metadata reveals identities. Effort report metadata could expose[9] personal identity details, violating privacy expectations. The system needs to be equipped with robust security features like multi-factor authentication and data encryption[10].

---

[6] Internal Document: EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.3
[7] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.2
[8] Internal Document: EffortLogger Supervisor Input V1-2 2024-01-06.pdf, p.2
[9] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.1
[10] What is Data Protection and Privacy? n.d, https://cloudian.com/guides/data-protection/data-protection-and privacy-7-ways-to-protect-user-data/

- Concerns exist that detailed personal data could be accessed[11] or misused by supervisors. The data that is being used as historical data must be of the employee 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. CP04: Outdated Practices

1.4.1. Data stored locally on floppy disks[12]

- The current version of EffortLogger stores all data locally which is in turn stored on a floppy disk. Original system relied on 3.5-inch floppy disks[12] and outdated Windows machines. 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.
- 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.

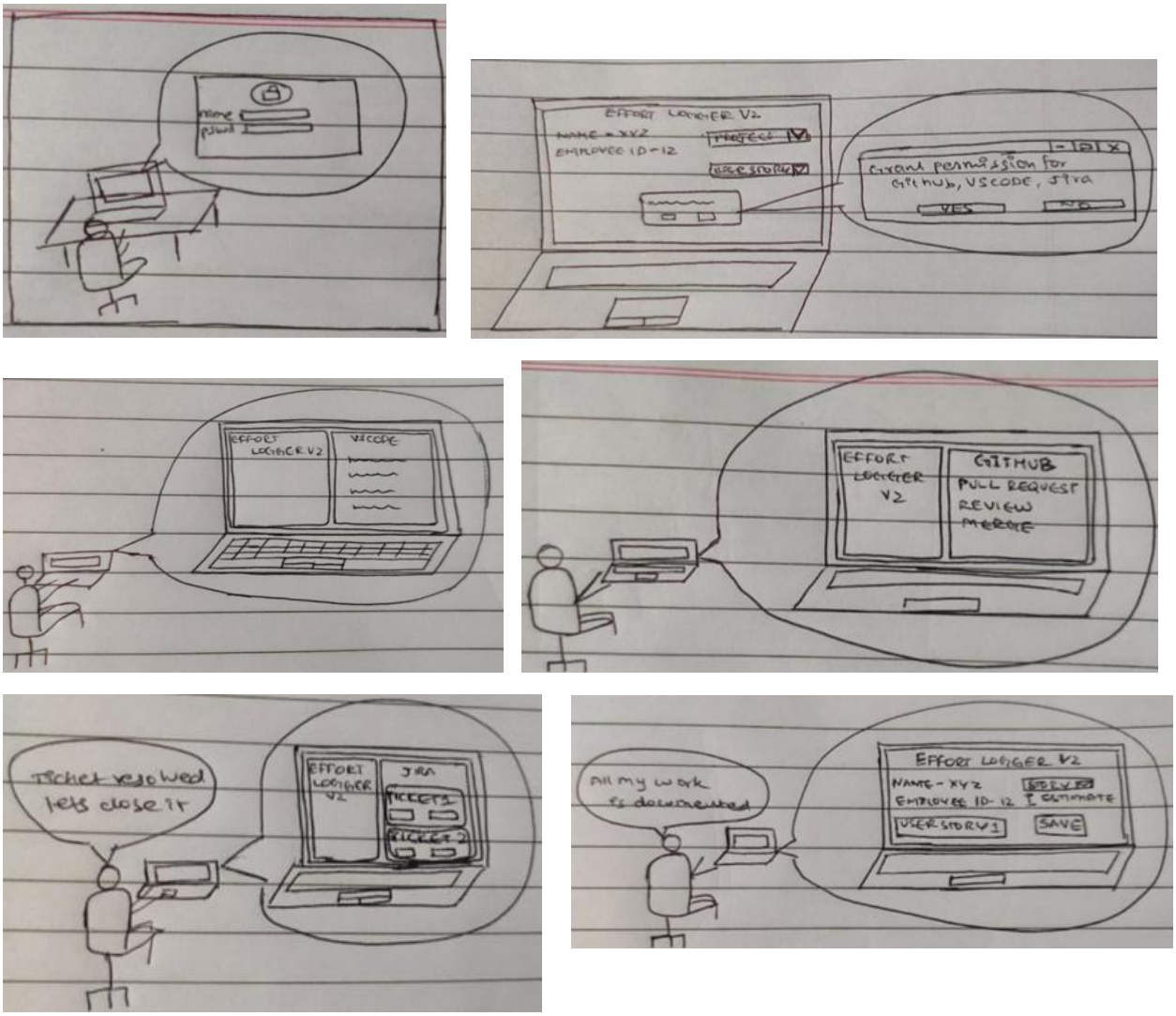
1.4.2. Multiplatform compatibility

- EffortLogger was just an application on the computer which causes inconvenience in modern day as users are on the go always. Not everyone always has access to a computer.
- People require support for mobile devices[12] as well. Making EffortLogger accessible on mobile devices as well would help users update their work from anywhere.

[11] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.1
[12] Internal Document: EffortLogger Customer Need V2-0 Document V1-2.pdf, p.1

2. Operational Concepts (ConOps)

2.1. Storyboard – Tracking Efforts



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. CO01: 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[13]. 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.

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

2.2.3. Assumptions and Constraints

- Assumptions
  - The users must be willing to adopt this new version.
  - Historical data must be available and relevant[14]
- 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, (Planning Poker)).
  - The software must be in compliance with all rules and regulations be it related to data, security or laws

2.3. CO02: 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 (CP03)
- Tracking effort of users. (CP02)
- Logging of defects associated with a user story. (CP02)
- Integration with agile tools. (CP02)
- Provide historical data during planning poker sessions. (CP01)
- Data analysis and visualization of various metrics for higher officials. (CP01.2/CP02.2/CP03.1)
- Security and privacy measures. (CP03)

2.4. CO03: 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

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

- Provide insights and help to developers, managers and executes on previous work, team efforts and business progress.
- Ensure data privacy and security

2.5. CO04: 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.[15]
- 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.

2.6. CO05: 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.

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

- 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.[16]
- 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).

2.7. CO06: 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.

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

- 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).

2.8. CO07: Overview of System and Key Elements

- Employee data privacy concerns.[17]
- 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.

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

3. Received Requirements

Functional Requirements

3.1. RR01 - The system shall provide effort tracking[18].

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. RR02 - The system shall provide user authentication and access control[19].

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.

[18] Internal Document : EffortLogger V1.11 (Document V1.1)
[19] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 2

3.3. RR03 - The system shall integrate with agile technologies[20]

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. RR04 - The system shall provide effort estimation data using machine learning[21].

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. RR05 - The system shall provide insights through dashboards and reports.[22]

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.

---

[20] Internal Document : EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.3

[21] 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

[22] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 2

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. RR06 - Privacy and Security compliance[23]

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. RR07 - Database requirements[24]

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.

[23] Internal Document : EffortLogger Supervisor Input V1-2 2024-01-06.pdf, Pg 2
[24] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 1

3.8. RR08 - 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[25] 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. RR09 - Usability requirements

3.9.1. The system shall have a user-friendly UI requiring minimal training[26].

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. RR10 - 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[27] environments.

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

[25] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.2
[26] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.1
[27] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 1

4. Derived Requirements

4.1. DR01: Define Granularity of Tracking Activities in IDEs

4.1.1. Derived from RR01

- The system shall capture granular metrics within the IDE such as time spent on specific files, frequency of file switching, number of test runs, and compilation attempts

4.1.2. Further Research Needed

- How should the tool distinguish between active and idle effort?
- Which IDEs must be supported initially (e.g., PyCharm, VSCode, Eclipse)?

4.1.3. Requirements Elicitation Notes

- Conduct stakeholder interviews to determine the most relevant tracked activities.
- Observe developers during real use to identify what data matters to them.
- Use a survey to prioritize IDE features to track.

4.2. DR02 - Define the Scope and Frequency of Machine Learning[28] Estimation

4.2.1. Derived from RR04

- The system shall define specific intervals when the ML model retrains on newly collected effort and outcome data.
- The system shall enable user feedback to refine ML-based estimation outputs.

4.2.2. Further Research Needed

- What machine learning models are suitable for estimation in agile?
- What evaluation metrics will be used?
- Will the model need to be personalized for each team or shared across roles?
- What thresholds determine the similarity between user stories?

4.2.3. Requirements Elicitation Notes

- Organize a meeting with engineering managers to define estimation goals and metrics.
- Pilot a small dataset and collect feedback on the model's accuracy.

[28] Internal Document : EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.3

4.3. DR03: Establish Access Hierarchy and Permissions Model

4.3.1. Derived from RR02 and RR05

- The system shall implement a role-based access[29] control model, defining clear access levels for developers, team leads, product owners and executives.
- Specific analytics dashboards and filters shall be visible only to privileged roles.

4.3.2. Further Research Needed

- What roles exist across various project teams that need different types of access?
- What minimum privileges do each role require without compromising privacy?

4.3.3. Requirements Elicitation Notes

- Conduct role-based interviews with users across departments to gather permission needs.
- Review current access control policies for consistency.

4.4. DR04: Define Requirements for NLP Tagging Accuracy

4.4.1. Derived from RR04

- The system shall define and measure a tagging[30] accuracy threshold to ensure relevance of suggested user stories.
- NLP-based tags should be editable by users to correct any mistakes and retrain future behavior.

4.4.2. Further Research Needed

- Will teams need different tagging models based on their domain or tech stack?
- What is the typical structure of user story descriptions?

4.4.3. Requirements Elicitation Notes

- Go through and document findings from existing user stories to build an initial training set.
- Facilitate focus groups to review tagging outcomes and gather corrections.
- Using prototypes to demonstrate tag suggestions and capture usability issues.

[29] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 2
[30] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.2

4.5. DR05: Define Dashboard Customization Requirements

4.5.1. Derived from RR05

- The system shall allow users to create and save custom dashboard[31] views.
- The system shall support export of visualizations for presentations and reports[32].

4.5.2. Further Research Needed

- What are the key performance indicators each role is interested in?
- What types of charts/graphs are most useful for the given user?
- Should dashboard templates differ by user role?

4.5.3. Requirements Elicitation Notes

- Conduct workshops with project managers, developers and executives to define dashboard needs.
- Use storyboarding to walk stakeholders through hypothetical dashboard use.

[31] Internal Document : EffortLogger Supervisor Input V1-2 2024-01-06.pdf, Pg 2
[32] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 2

5. Architectural Design

5.1. Functional Requirements

5.1.1. AD01: Effort Tracking with integration of agile tools [RR01][RR03]

5.1.1.1. Architecture Components

- Connectors
- Event Listeners/ Webhooks
- Tracking Agent
- Aggregation Service
- Local Storage

5.1.1.2. Technical Description

- Uses agents to track effort which are integrated with popular IDE's like VSCode, PyCharm, Eclipse, etc
- Webhook listeners and APIs help in integration with agile tools like GitHub, Jira, etc
- Webhooks capture commits, merge's, pull requests and reviews.
- In Jira, APIs help in gathering ticket updates and transitions
- This effort data is pushed to a local encrypted storage space which is later synced with cloud storage.

5.1.2. AD02: Authentication and Access Control [RR02]

5.1.2.1. Architecture Components

- Authentication Service like OAuth 2.0
- Role Based Access Control Engine
- Session Manager (JWT Tokens)

5.1.2.2. Technical Description

- Ensures that only authorized users access the system and only certain functionalities depending on their roles
- Users authenticate through OAuth for a secure way to login
- RBAC helps in permission enforcement where it restricts data visibility not letting everyone access private data.
- Session tokens for client-server communication

5.1.3. AD03 : Effort Estimation via ML [RR04]

5.1.3.1. Architectural Components

- Estimation Engine
- NLP Module

5.1.3.2. Technical Description

- The machine learning algorithms suggest effort estimates for user story points based on historical tagged data.
- NLP module parses descriptions help this historical data get tagged.
- ML models predict story points based on tags and relevance.
- Learns continuously from final and estimated data.

5.1.4. AD04: Analytics and Dashboard [RR05]

5.1.4.1. Architectural Components

- Analytics Engine
- Dashboard Module

5.1.4.2. Technical Description

- Visualizes performance metrics, trends in effort data and accuracy in estimations.
- RBAC assists in providing role specific dashboards.
- Data is anonymized before reporting.
- Dashboards use tools like D3.js or PowerBI API.

5.2. Non-Functional Requirements

5.2.1. AD05: Security and Privacy [RR06]

5.2.1.1. Architectural Components

- Data Encryption Service
- Compliance Checker

5.2.1.2. Technical Description

- Protects all data and ensure that the software is up to date with respect to industry standards and policies.
- AES-256 encryption
- GDPR rules are enforced.

- RBAC ensures that data visibility is restricted to everyone without access.
- Two factor authentication can ramp up security and privacy of data.
- All logs stored with metadata are anonymized.

5.2.2. AD06: Reliability, Availability and Portability [RR07][RR10]

5.2.2.1. Architectural Components

- Encrypted Local Storage
- Central Repository (Database)

5.2.2.2. Technical Description

- Local storage ensures that the software works even when offline.
- Backups to cloud to prevent data loss.
- Supports platforms like Windows, Linux, MacOS
- Uses Docker to facilitate portability.
- React Native app for cross-platform compatibility.

5.2.3. AD07: Performance and Usability [RR08][RR09]

5.2.3.1. Architectural Components

- Tracker (Realtime)
- Visual Analytics Engine

5.2.3.2. Technical Description

- Event tracking with under 2 second response.
- Query optimization for <5 seconds result.
- Backups to cloud to prevent data loss.

5.3. Reused Components

5.3.1. Reused Component: Charting Libraries

5.3.1.1. Open-source visualization libraries used to render interactive charts and graphs.

- Renders real time and historical metric visualizations on key performance indicators like sprint velocity, trends in efforts through dashboards without building custom graphics from scratch

5.3.2. Reused Component: GraphQL Query Framework

5.3.2.1. Standard query framework to fetch aggregated and filtered data through APIs.

- Reused from modules like effort tracker, to power filters while querying like show data by sprint or by team or by tags.

6. Detailed Design

6.1. (DD01) Design Element: Effort Tracker Agent [RR01]

6.1.1. Description

- A client-side agent that is integrated into the IDE which monitors file edit durations, active time, idle time and commit events.
- Sends summarized logs to a local encrypted cache. Caches effort in encrypted SQLite.
- Timestamps are captured to calculate time intervals for each coding session.

6.1.2. Design pattern – Observer Pattern

- The observer pattern allows decoupling the effort logic from the IDE while keeping real-time updates. The agent is implemented as a plugin/extension for IDEs
- This design minimizes privacy risk and ensures offline functionality

6.1.3. Rationale

- Real-time tracking requires an event-based system without hampering user performance.
- Modular and portable across IDEs.

6.2. (DD02) Design Element: Authentication and RBAC Manager [RR02]

6.2.1. Description

- Handles login via OAuth2 with JWT token issuance
- Proxy guards protect access to routes

6.2.2. Design pattern – Proxy Pattern

- Using proxy pattern enables central control of access rules
- JWT tokens are stateless and lightweight for API scalability
- Proxy ensures security gates are enforced in a consistent and centralized way

6.2.3. Rationale

- Authentication Manager handles OAuth login using standard flows
- The JWT access and refresh tokens are embedded with sub, role and scope claims.
- The RBAC middleware will restrict API and component access enforcing role based access control.

6.3. (DD03) Design Element: AgileIntegration Module [RR03]

6.3.1. Description

- Connectors abstract interactions with external third-party APIs.
- There exists a common interface for all integrations.
- This design element isolates the core system from changes in third-party APIs and encourages modular development.

6.3.2. Technical Description

- Each tool has its own adapter implementing the interface. GitHub uses webhooks and its REST API.
- Jira fetches issue data via its REST API using JQL queries.
- These activities are mapped to internal objects and stored as effort in the database.

6.3.3. Rationale

- Encapsulation of these third-party application logic makes it easy for error handling.
- This standardizes how tools are integrated and facilitate replacements of these agile tools with one another.

6.4. (DD04) Design Element: ML Estimation Engine [RR04]

6.4.1. Description

- The estimation engine has underlying strategies like baseline averages, similarity-based ML and regression.
- These strategies are selected based on role, thresholds or user preferences.
- These include a score and explanation to maintain transparency.
- The strategy is dynamically selected based on context.

6.4.2. Design Pattern – Strategy Pattern

- Models are trained on past story points, effort logs and tags.
- It provides structure in such a way that it facilitates flexible integration of multiple machine learning models

6.4.3. Rationale

- Strategy patterns make it easy to test and switch machine learning models without too much hassle.
- If said score is low, then there is flexibility to fall back to older models.

6.5. (DD05) Design Element: Tag Generator [RR3.4.2][RR04]

6.5.1. Description

- Processes user story descriptions using NLP to extract relevant keywords.
- Stores tags which are used for multiple processes including filtering, grouping and estimation. Tags are stored with user story metadata.

6.5.2. Design Pattern – Pipeline Pattern

- This pattern can facilitate lemmatization and is good with natural language processing tasks.
- It helps in defining separate modules, easy swapping of components without affecting the whole process.
- It breaks the complex process into steps like lowercasing, tokenization, lemmatization and extraction.

6.5.3. Rationale

- Pipeline pattern keeps every step of a complex task modular and testable. (individually not affecting any other process)
- It is extensible. NLP helps automate categorization.

6.6. (DD06) Design Element: Analytics Engine [RR05]

6.6.1. Description

- Displays different dashboards for different users. Personal metrics for developers, team progress for managers and so on.
- Displays key performance indicators like sprint velocity, estimation offset, etc.
- Enables users to export graph and tables for reporting in other formats like PDF or CSV.

6.6.2. Technical Description

- This design element follows the MVC pattern. The Model layer using RESTful endpoints to fetch data from database and aggregate these metrics in SQL.
- The view layer is built on React.js for a good frond end and using D3.js for visualizations.
- The controller layer handles querying and filtering changes.

6.6.3. Rationale

- MVC cleanly distinguishes between Model, that is querying and calculations, View which is the dashboard with visualizations and Controller that handles the interactions and underlying processes.
- Ensures reusability, modularity and easy testing.

7. Traceability

7.1. RR01: The system shall provide effort tracking.

7.1.1. Traceability from Source to the Requirement

The need for automatic effort tracking stems from various stakeholder inputs that identified manual time tracking as inefficient and prone to errors. We see how developers are burdened with manually[33] logging time spent on user stories and debugging activities, leading to inaccuracies and reduced productivity.

This aligns with CP01, which emphasizes the inefficiency of user input-based tracking. In the CONOPS, users expected a system that seamlessly captures development activity within their existing tools such as IDEs and Git, forming the basis for RR01. the requirement that the system shall automatically track efforts using IDE integration.

7.1.2. Traceability from Requirement to the Architecture and Design

RR01 is fulfilled in the system architecture through the Effort Tracking Agent [AD01] that stores data in a local encrypted database and syncs with the cloud when needed.

This architectural component is supported in the detailed design where in it uses the Observer Pattern [DD01] to monitor file changes, debug sessions, and time intervals within IDEs. The agent uses IDE plugin APIs and Git webhooks to passively track effort while the developer works. This design fulfills the original source requirements and supports future analytics.

7.2. RR02 - The system shall provide user authentication and access control

7.2.1. Traceability from Source to the Requirement

The requirement for authentication and access control stems from stakeholder concerns about privacy violations and unauthorized data access. Managers have highlighted the risk of unrestricted access[34] to effort data, especially in performance evaluations. These concerns were addressed as Customer Problem CP03, which calls for good access control mechanisms. The CONOPS further emphasizes the need for secure authentication processes and role-based permissions to ensure users only access data relevant to their role, resulting in RR02.

[33] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.2
[34] Internal Document : EffortLogger Supervisor Input V1-2 2024-01-06.pdf, Pg 2

7.2.2. Traceability from Requirement to the Architecture and Design

In the architecture, this requirement is handled through the Authentication Module, which interacts with a Session Manager and an RBAC Engine [AD02]. This ensures secure logins and enforces role-specific access control across all components. In the detailed design, this is enforced by the Authentication Manager, which use the Proxy Pattern [DD02] to verify all incoming requests. Authentication is implemented using OAuth2 with JWT tokens that carry role-based claims. This ensures data protection and compliance with modern access control practices.

7.3. RR03 - The system shall integrate with agile technologies

7.3.1. Traceability from Source to the Requirement

Stakeholders expressed dissatisfaction with the manual[35] logging of effort and defect data. These concerns are reflected in CP02, which outlines the lack of integration with modern Agile tools. In the CONOPS, users envisioned a system that connects to Jira for ticket tracking and to GitHub for commits and pull requests, motivating the creation of RR03, which ensures integration with Agile tools.

7.3.2. Traceability from Requirement to the Architecture and Design

The architectural design includes an Integration Layer with connectors [DD03] for GitHub, Jira, and other Agile tools. These implement the Adapter Pattern to facilitate API interactions. This component listens [AD01] for events like pull requests, commits or merges and maps them to internal effort logs. The design supports extensibility, so other tools can be added with minimal impact. This approach ensures alignment with Agile workflows and reduces manual tracking, fulfilling both architectural and user expectations.

7.4. RR04 - The system shall provide effort estimation data using machine learning

7.4.1. Traceability from Source to the Requirement

Stakeholders requested better support for data-driven estimation during Planning Poker sessions. This need represents a major concern in CP01, which describes how existing estimation practices are subjective and inconsistent. The CONOPS envisioned a system that learns from past story points and effort data to offer smart estimates, leading to RR04, which defines the use of machine learning to improve estimation accuracy.

[35] Internal Document : EffortLogger Business Opportunity Document V1-2 2024-01-06.pdf, p.3

7.4.2. Traceability from Requirement to the Architecture and Design

This requirement is addressed through an Estimation Engine[AD03]. The engine is implemented in detail and uses the Strategy Pattern [DD04] to dynamically choose between estimation methods, such as regression models, historical similarity matching or averages. The ML component uses features like user story tags [DD05], time spent and user role to make predictions. This design ensures accuracy, extensibility, and user trust in the estimation process.

7.5. RR05 - The system shall provide insights through dashboards and reports

7.5.1. Traceability from Source to the Requirement

Managers and team leads request for actionable dashboards[36] and performance summaries. This is addressed in CP02, which point out the lack of insight into historical effort patterns. The CONOPS clarifies the expectation that higher-level users be able to analyze sprint velocity and estimation accuracy, leading to RR05, which demands real-time and role-based analytics.

7.5.2. Traceability from Requirement to the Architecture and Design

To address this, the architecture introduces a Dashboard Module and a supporting Analytics Engine [AD04]. The detailed design implements this using the Model-View-Controller Pattern [DD06], with chart rendering via libraries like D3.js. The dashboards change based on user role and offer visualizations like bar graphs and velocity trends. These tools allow high-level users to make data-driven decisions.

7.6. RR06 - Privacy and Security Compliance

7.6.1. Traceability from Source to the Requirement

Concerns about data privacy violations[37] and insecure storage are delt with in CP03. These concerns include unauthorized access to personal effort logs and non-compliance with modern privacy laws. The CONOPS specifies the need for encryption, anonymization and access restrictions. These issues culminated in RR06, which shapes the system's need to meet privacy and security standards.

7.6.2. Traceability from Requirement to the Architecture and Design

This requirement is satisfied in the architecture via a Security and Privacy Layer that applies encryption and secure authentication. In the design phase, Data Encryption Service and Compliance Checker [AD05] are introduced, allowing encryption and privacy checks to wrap core services. This ensures data security while maintaining system functionality and complies with GDPR and industry standards.

[36] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 2
[37] Internal Document : EffortLogger Supervisor Input V1-2 2024-01-06.pdf, Pg 2

7.7. RR07 - Database Requirements

7.7.1. 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. Previously users feared[38] single point of security failure.

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.

7.7.2. Traceability from Requirement to the Architecture and Design

The system architecture will include a relational database or a NoSQL database[AD06] 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.

7.8. RR08 - Performance Requirements

7.8.1. Traceability from Source to the Requirement

The need for speed and responsiveness[39] during Planning Poker and analytics use was noted is part of CP01. The CONOPS emphasizes under 5-second data retrieval times and smooth system operation. These expectations directly led to RR08, which sets performance benchmarks for data access and real-time processing.

7.8.2. Traceability from Requirement to the Architecture and Design

Architecturally, performance is handled through Cache, Visual Analytics Engine, and Query Optimizer [AD07]. These implement the requirement using caching and indexing techniques, to intercept frequent queries and serve responses from memory. These optimizations ensure very small delays in critical workflows

[38] Internal Document : EffortLogger Supervisor Input V1-2 2024-01-06.pdf, Pg 2
[39] Internal Document: EffortLogger User Input V1-2 2024-01-06.pdf, p.2

7.9. RR09 - Usability Requirements

7.9.1. Traceability from Source to the Requirement

The lack of a usable, modern interface is reflected in CP04. The CONOPS states the need for a clean, intuitive UI accessible across platforms and roles. This feedback established RR09, which defines the system's usability and interface goals.

7.9.2. 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.

7.10. RR10 - Reliability and Portability Requirements

7.10.1. Traceability from Source to the Requirement

Reliability and portability issues are captured in CP04, which note platform dependency[40] and instability. The CONOPS specifies the need for cross-platform functionality and high uptime. These shaped RR10, calling for consistent availability and OS compatibility.

7.10.2. 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. Using Docker Containers as specified in AD06. For cross-platform deployment, the system will be developed using containerization to ensure execution across all platforms like Windows, MacOS, and Linux.

[40] Internal Document : EffortLogger Customer Need V2-0 Document V1-2.pdf, Pg 1

8. Implementability

8.1. Implementability Requirement: RR01 Effort Tracking

8.1.1. Description

This requirement will be implemented using IDE plugin development capabilities which are provided by platforms like VS Code, PyCharm and Eclipse. A tracking agent using the Observer Pattern will monitor developer activity like file edits, debugging, commits and terminal usage. The tracked data will be locally stored in an encrypted format using a database and synced securely.

8.1.2. Implementability Rationale

Tools like GitHub Copilot already perform real-time tracking within IDEs, hence confirming technical feasibility. Event listeners and hooks are supported. This approach supports the client's need to automate the tracking process without manual user input, thus reducing error.

8.2. Implementability Requirement: RR02 Authentication and RBAC

8.2.1. Description

This requirement will be implemented using OAuth 2.0 and JWT-based role encoding. Th e middleware will enforce access rules using Proxy Pattern which ensures restricted access to data and dashboards based on user roles (e.g., Developer, Product Owner, Manager).

8.2.2. Implementability Rationale

Authentication services like Firebase Auth offer built-in[41] support for RBAC, Two-factor authentication and token-based sessions. Web apps like Jira and GitHub use similar models to protect user-level and organization-level data. This ensures compliance with client data privacy goals and sees to it that there is structured access to analytics and logs.

8.3. Implementability Requirement: RR03 Agile tool integration

8.3.1. Description

The system will use REST and Webhook APIs[42] to integrate with Agile tools like GitHub and Jira. A modular adapter will normalize the incoming data like pull request, tickets, commits and merges and convert them into internal activity logs. Webhooks, issue watchers and commit listeners will track the developer activity in real time.

[41] Christoph Nible (2022, 14 Jan) Role Based Access Control (RBAC) with firebase https://medium.com/@christophnissle/role-base-access-control-rbac-with-firebase-d04c5e4c774a
[42] https://docs.github.com/en/webhooks/webhook-events-and-payloads

8.3.2. Implementability Rationale

Existing tools like ZenHub and Shortcut show that these integrations are stable, robust and viable. Agile APIs are well-documented and are thoroughly tested. This fulfills the client's goal of eliminating redundant data entry and populating effort logs automatically.

8.4. Implementability Requirement: RR04 ML-based Estimation

8.4.1. Description

This will be implemented using ML models like Random Forest, Linear Regression, and KNN. The models will predict story points based on tags, effort history, and complexity indicators. The Strategy Pattern allows switching between estimation techniques as needed. The NLP module using pipeline pattern facilitates the tagging operation and help is efficient processing

8.4.2. Implementability Rationale

ML-driven estimation is used in tools like SonarQube, and academic models like COCOMO II[43]. Frameworks like scikit-learn and TensorFlow simplify implementation. Tools like SonarQube use ML/NLP to extract context from codebases and assign severity or complexity metrics. This aligns with the client's goal of data-driven estimation replacing subjective judgment.

8.5. Implementability Requirement: RR05 Dashboard and Reporting

8.5.1. Description

The system will use a Model-View-Controller architecture for interactive dashboards built in React . The MVC Pattern separates logic for dashboard components. Charts and graphs will be rendered using D3.js or similar frameworks. Reports can be filtered by time, team, and tags. Backend services can compute metrics using PostgreSQL.

8.5.2. Implementability Rationale

Dashboards in Grafana[44], Power BI and Jira prove the viability of rich analytics. Users expect real-time metrics in modern tools. Existing platforms like Jira, and Azure DevOps offer similar views, making this a achievable component. This supports the client's goal to track performance, accuracy, and planning efficiency across teams.

[43] Goyal, Somya & Parashar, Anubha. (2018). Machine Learning Application to Improve COCOMO Model using Neural Networks. International Journal of Information Technology and Computer Science. 10. 10.5815/ijitcs.2018.03.05.
[44] https://grafana.com/grafana/

8.6. Implementability Requirement: RR06 Privacy and Security

8.6.1. Description

This requirement aligns directly with industry standards like GDPR and OWASP Top 10. Implementations such as AES-256 encryption and multi-factor authentication are standard practices and available through open-source libraries. Security will be implemented using AES-256 encryption for stored data

8.6.2. Implementability Rationale

Encryption and anonymization are industry-standard practices supported by OWASP, CWE, and ISO 27001. Systems like Slack, Zoom[45], and Google Workspace use these methods to secure user data. This directly satisfies the client's concern for GDPR compliance as well

8.7. Implementability Requirement: RR08 Performance

8.7.1. Description

EffortLogger will cache frequently accessed data using Redis or in-memory layers to meet sub-5-second retrieval targets. Indexing and async queries will improve response times.

8.7.2. Implementability Rationale

The Proxy Pattern and in memory caching layers used in high-performance systems like GitHub Actions ensure minimal latency. Performance optimization is standard in tools like Figma and Slack. This supports the client’s productivity-driven goal of fast, responsive estimation and processing.

8.8. Implementability Requirement: RR10 Reliability and Portability

8.8.1. Description

The system will use Docker[46] containers to enable consistent deployment across Windows, macOS, and Linux. Uptime will be maintained through Kubernetes orchestration.

8.8.2. Implementability Rationale

Reliability and cross-platform compatibility are already industry standards. Cloud-native design is widely adopted in GitHub and Zoom, as well as other SaaS platforms. This enables the client’s demand for 99.9% uptime and access across diverse environments.

[45] https://developers.zoom.us/docs/distribute/security-best-practices/
[46] https://www.divio.com/blog/why-dockerize/