2023 IEEE International Conference On Artificial Intelligence Testing (AITest) | 979-8-3503-3629-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/AITest58265.2023.00018

AI Decision Assistant ChatBot for Software Release Planning and Optimized Resource Allocation

Ehab Ebrahim, Mazen Sayed, Mahmoud Youssef, Hisham Essam,
Salma Abd El-Fattah, Dina Ashraf
Computer and Systems Engineering Department
Ain Shams University
Cairo, Egypt

Omar Magdy
Siemens Digital Industries Software
Cairo, Egypt
omar.magdy@siemens.com

Reem ElAdawi
Siemens Digital Industries Software
Cairo, Egypt
reem.eladawi@siemens.com

Abstract—This paper presents Alfred, a web-based artificial intelligence (AI) chatbot powered by the Rasa framework and designed to facilitate Agile software release planning by using two machine learning (ML) models to estimate the time for each task selected by the project managers and recommend the optimal resource to be assigned to each task. Extreme gradient boosting (XGBoost) is used for time estimation, and the model's performance evaluated using accuracy, adjusted coefficient of determination (R2), and fitting a line between the estimation and actual values. The results obtained were 82% accuracy and 98% adjusted R2 score. K-nearest neighbors (K-NN) is used for resource recommendation in a one-shot learning manner. The application integrates a dashboard with useful visualizations and an interactive smart assistant AI-based chatbot, providing project managers with a tool to make more informed decisions. Additionally, an approach is used to indicate the model's confidence in tasks estimation of given tasks based on the amount of available data showing the history in the system to give project managers more insights. The approach uses a keywords frequency indication algorithm based on the summary tokens to group the task estimates into one of three categories of confidence: high, medium, or low.

Keywords— Software Release Planning, Agile scrum, Task Estimation, Resource Allocation, Smart Assistant Chatbot

I. INTRODUCTION

Introducing Agile methods [1] into the software development process poses a significant challenge for many software companies. Agile scrum projects are gaining popularity for software development and delivery due to their inherent attributes of flexibility in execution and quicker time to market. However, critics of Agile have raised questions about the predictability of Agile execution success because the quantitative management emphasis on predictive techniques for an Agile project may not match the extent of those of large mission-critical projects that use in-process leading indicators to predict outcomes.

A. Software Release Planning Process

The process of software release planning in an Agile work environment mainly depends on both the team's experience [2] and the project's conventions to generate a release plan by selecting planned features and improvements that ensure the estimated time to develop and test features besides defects fixing will not cause a schedule slip. However, releases often do not go as planned, and features implementation and testing bypass due dates, leading to delays in customer delivery which negatively affects the quality of the whole release. This overrun is often due to poor planning and inadequate resource allocation. An alternative approach is to develop a model that considers differences in feature composition and context across releases.

B. Machine Learning Approach

The team's mental process when estimating a new task is similar to a classic machine learning (ML) problem [3]. In this problem, a task, T, improves over time with experience, E, by a performance measure, P. In software development estimation, T is the task of estimating/predicting a new ticket's complexity, E is the historical data of previous estimations, and P is the difference between the actual complexity and the estimation.

This paper presents a set of ML approaches to develop models to help in: 1) predicting new tasks' complexity based on historical data from previously estimated tasks, and 2) picking optimal resources to be assigned to the selected tasks. Data from Siemens EDA Jira project boards is used for models training.

The resulting models are designed to be integrated in a management board system web application (Alfred) for decision making with the aid of a smart assistant in the form of an artificial intelligence (AI) chatbot to highlight areas of risks based on current actual and future forecasted data. Alfred can help managers take time-critical actions for planning and resource allocation in a guided way by answering the most persistent questions usually encountered when planning: 1) "Which tasks can the team complete by the end of this release?" and 2) "Who is the best candidate for a particular task?".

II. RELATED WORK

In this section, we analyze the already deployed software as a service (SaaS) products used for Agile project planning and management. We later review some standalone task estimation ML models and compare their achieved accuracies.

A. Jira Project Tracking

Jira™ issue tracking software is an incredibly popular project management tool among corporations and technology companies because it offers an extensible list of features, such as roadmaps, backlogs, sprint planning and task assignment, and a large set of visualizations.

979-8-3503-3629-0/23/$31.00 ©2023 IEEE
DOI 10.1109/AITest58265.2023.00018

While Jira can most certainly help teams organize and manage large projects, Jira does not have any ML-based capabilities that allow you to predict how long a task will take.

B. LinearB Workflow Automation

LinearB is a workflow automation tool that helps developers meet their obligations using its Project Delivery Tracker, which connects problems, Git, and release data to offer a precise picture of developer capacity, unforeseen work, and delivery bottlenecks. This approach is great for organizing projects and orchestrating teamwork.

While LinearB is able to gather data from multiple resources to help in organizing and presenting an overview of the whole release picture. However, you still need to analyze this data to be able to take an action, and you still need to estimate tasks and assign candidates by yourself.

C. Task Complexity Estimation ML Models

Alfred offers a comprehensive solution for optimized release planning through incorporating state-of-the-art AI techniques. While there are many pre-existing efforts [4] in developing standalone machine learning models for task estimation, we have not encountered models developed for resource allocation. Table I lists some of these prior models.

TABLE I. TASK EFFORT ESTIMATION MODELS WITH USED DATASETS AND ACHIEVED ACCURACIES

Reference	Used ML Model	Used Dataset	Achieved Accuracy
[5]	ANN	NASA93	95%
[6]	Linear regression (MLR)	PROMISE	92%
[7]	Analogy based estimation and selective classification	ISBSG	85%
[8]	Fuzzy logic	ISBSG, COCOMO and DESHARNAIS	97%
[9]	Support vector regression (SVR)	ISBSG	95%

III. BACKGROUND

The time estimation ML model incorporated in Alfred relies on the idea of ensemble learning and boosting. In this section, these methods are discussed, as well as some terminologies and concepts from the RASA framework upon which Alfred's smart assistant chatbot is built.

A. Ensemble Learning

Ensemble learning is a technique that combines two or more models to reach a consensus in predictions [10]. One of the most powerful ensemble learning methods is Boosting which mainly aims to reduce bias in the ensemble decision.

B. XGBOOST

XGBoost is a decision-tree-based ensemble ML algorithm that uses a gradient boosting framework. It is an enhanced version of Gradient Boosting Machines (GBMs) that boosts weak learners using gradient descent architecture. XGBoost offers algorithmic and system optimization improvements to further enhance its performance such as: Parallelization & Tree Pruning.

C. RASA Framework

Rasa is an open source machine learning framework [11] to automate text and voice-based conversations.

1) RASA Terminologies

- **Intent:** user goals inside a user message
- **Entity:** structured information inside a user message
- **Actions:** operations that the bot can perform
- **Stories:** examples of user-bot interactions

2) RASA Stack

The RASA stack has two main components:

- **RASA NLU:** A natural language understanding (NLU) library that performs entity extraction and intent classification. This aids the chatbot's comprehension of the user's conversation.
- **RASA Core:** To generalize the system's dialogue flow, it employs ML techniques. Based on the input from NLU, the conversation history, and the training data, it also forecasts the optimum course of action.

3) RASA Architecture

The diagram in Fig. 1 shows the basic steps of how the chatbot responds to a user prompt.

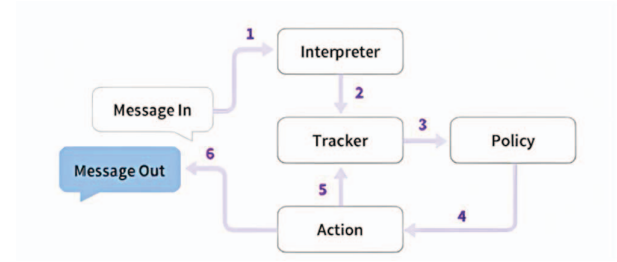

Fig. 1. RASA state flow diagram.

IV. SYSTEM ARCHITECTURE DESIGN

Our system architecture is designed as a microservice architecture consisting of six services as shown in Fig. 2. We will focus on three main services and describe them in more detail below.

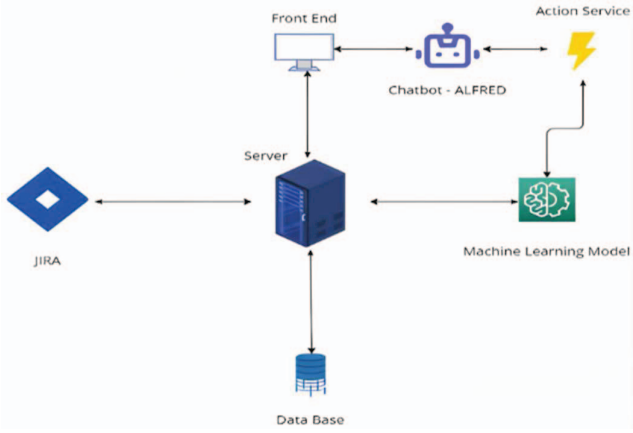

Fig. 2. Alfred system architecture.

A. Database Service

Database store our data and perform create, read, update, and delete (CRUD) operations on that data. MongoDB is used for storing the data gathered from the Siemens EDA production Jira database that carries the issue information and the developers/testers available for issue assignment per project.

The database schema is composed of two main entities—Issues and Assignees—as shown in Fig. 3.

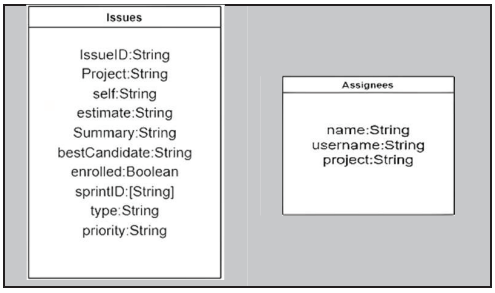

Fig. 3. Database schema.

B. Chatbot Service

The chatbot service is responsible for integrating the chatbot with the web application and communicating with the front-end service and offers the following functionalities:

- **Time Estimation:** Estimates how long an issue will take so the project manager can plan the sprint accordingly
- **Recommendations:** Recommends the best developer and the best quality assurance (QA) candidates
- **Confidence Categorization:** Provides the confidence level of time estimation or resource recommendation based on the available history data
- **Basic functionalities:** greetings, goodbyes, and asking the user for information

The bot is build using Python and the RASA framework where the Intents, Entities and Actions are mapped as follows:

1) Intents:

- **Asking for time estimation:** time estimation for a given issue ID
- **Asking for recommendation with rank:** best developer and QA recommendation to be assigned to a task with their order in the recommendations list
- **Asking for confidence category:** confidence level in task estimation
- **Greetings & Goodbyes**

2) Entities:

- **Issue ID:** the ID of the selected issue on which operations will be performed
- **Candidate type:** whether a developer or a QA engineer
- **Candidate rank:** order in the recommendations list

3) Actions:

- Estimate development and testing time for a given issue
- Recommend a human resource with the specified rank
- Provide the confidence level if requested by the user
- Say hello/goodbye anytime the user says hello/goodbye

Fig. 4 shows the chatbot architecture with the best candidate recommendation flow.

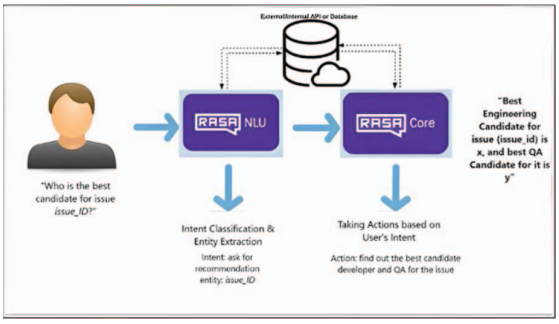

Fig. 4. Smart assistant chatbot architecture.

C. Action Service

There are two automated actions in this service:

1) Time estimation for all the selected issues in the release through communication with the task time estimation model

2) Best candidate software developer and QA engineer assignment for all the selected issues in the release through communication with the resource recommendation model

V. Data Preprocessing

Data is gathered from Siemens EDA production Jira database from three different projects within Siemens EDA. Our trials focused only on Jira items of type "story" or "bug" and any other type is filtered out. This filter resulted in a dataset with a total size of 8k entries. Two extra values are then added to each entry representing the development time and the testing time, respectively, calculated in "working days" units. The development time is calculated by subtracting the date item was opened from the date item was converted to the "ready for QA" state, and the testing time is calculated by subtracting the date item was converted to "in validation" from the date item was closed. Both calculations exclude weekends and public holidays corresponding to the country of residence.

$$Dev\_Time = Date(\text{“Ready for QA”}) - Date(\text{“Opened”}) - (Weekends + Public\ Holidays) \quad (1)$$

$$Test\_Time = Date(\text{“Closed”}) - Date(\text{“In Validation”}) - (Weekends + Public\ Holidays) \quad (2)$$

The dataset is then processed (Fig. 5) to be compatible with the ML models. Next, we will discuss the applied preprocessing steps in detail.

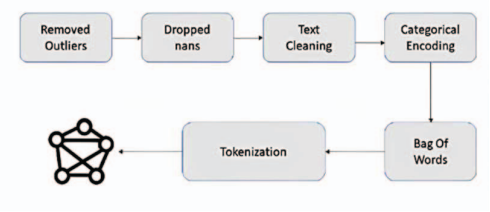

Fig. 5. Dataset pre-processing stages.

A. Outliers Removal

It was noticed that most of the items in the gathered data spanned between 1-9 days, so items with more than 200 days were considered as outliers. These outliers turned out to be "Epic," like containers for other smaller items.

B. NaN Dropping

We dropped any data entries that had a not a number (NaN) value in any of these fields: summary, development time, and testing time. This removed entries containing errors, such as missing a summary title due to mis-mapping legacy Jira items when migrating from an older tool, or empty development time and testing time fields caused by moving items from backlog to "Review" state or from "Ready for QA" state directly to "Closed," which caused the time calculation equations to break, so actual spanned time could not be determined.

C. Text Cleansing

We removed all the stopping keywords from the items' summary titles. For example, determiners (e.g., an, this, some) and prepositions (e.g., above, across, before) are considered stopping keywords.

D. Categorical Encoding

We created label encoding for the "Issue Type" and "Priority" of the Jira items. We have only two issue types, labeled as ["Story": 0, "Bug": 1]. Priority levels are labeled as ["Trivial": 0, "Minor": 1, "Medium": 2, "Major": 3, "Critical": 4, "Blocker": 5].

E. Bag of Words

We collected all the strong keywords from all the summary titles text corpus after removing the stopping keywords and counted the total repetitions of each keyword in the whole text corpus constructed from the union of all the gathered Jira items summary titles [12].

As an example, consider the following two summary titles as two separate bags of words:

1. Calibrated (factory) multi-corner support and support fabrication
2. Calibrated K-Factor for contact/via arrays

After removing the stopping keywords, the "Bag of Words" representation for each sentence will be as follows:

BoW1 = {"Calibrated": 1, "factory": 1, "multicorner": 1, "support": 2, "fabrication": 1}
BoW2 = {"Calibrated": 1, "KFactor": 1, "contactvia": 1, "arrays": 1}

The union of these separate bags of words will be our whole text corpus, and keywords repetition across all the corpus will be counted:

BoW3 = {"Calibrated": 2, "factory": 1, "multicorner": 1, "support": 2, "fabrication": 1, "KFactor": 1, "contactvia": 1, "arrays": 1}

$$BoW3 = BoW1 \cup BoW2 \quad (3)$$

F. Tokenization

In this last step, we divided the issue's summary text into non-stopping keyword tokens, converted these keyword tokens to the corresponding count from the bag of words, then stacked these values in a vector of length 41 (maximum summary title length in the dataset). This vector can then be concatenated with other inputs and fed as a feature vector (Fig. 6) to the ML models.

Fig. 6. Feature vector representation.

VI. Machine Learning Models Architecture

In this section, we will discuss the architectures of the time estimate and the resource allocation models. We will also mention the input feature vector form fed to each and the expected output. In addition, will review the confidence level categorization approach.

A. Time Estimation

Time estimation depends in its core on the XGBoost ML model [13], where the predicted estimate is the result of the majority voting of the weak learners, as shown in Fig. 7. It takes as an input a feature vector composed of the Jira issue's tokenized summary concatenated with the categorical encoding of the priority and the issue type.

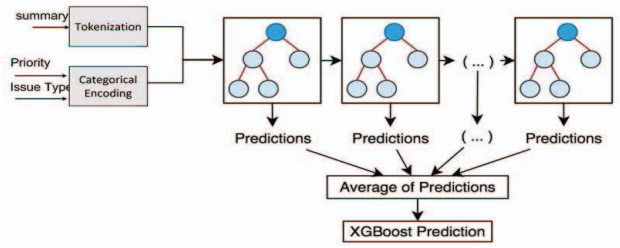

Fig. 7. Time estimation model architecture.

B. Candidate Recommendation

Following a vector search algorithm, every developer and QA engineer was represented with a feature vector of the relevant keywords collected from their history of all previously assigned items. This constructed a vector space with representation for each resource, namely, candidate vectors. For a given unassigned Jira issue, we extracted the word embeddings from its summary and created an analogous feature vector representation, namely, an anchor vector. The anchor vector is fed to the k-NN ML model [14] to get the closest candidate vector to the input anchor vector through a one-shot learning

manner, and recommend the best resource represented by this candidate vector, as illustrated in Fig. 8.

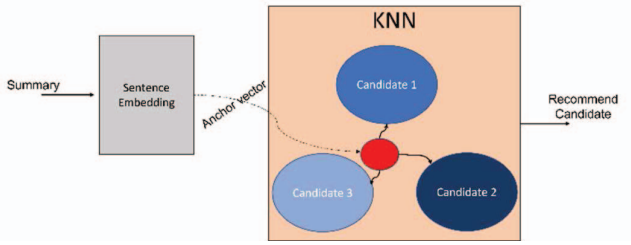

Fig. 8. Candidate recommendation model architecture.

C. Confidence Categorization

Based on the summation of all the keywords' frequencies (i.e., count from the bag of words) in the input tokenized summary vector, we can decide to which confidence level this issue belongs. Since the whole dataset keywords' frequencies summations range from approximately 500 to 7000, confidence levels are divided as follows:

- High Confidence: higher than 4000 (Freq. ≥ 4k)
- Medium Confidence: between 1500 and 4000 (1.5K ≤ Freq. < 4k)
- Low Confidence: less than 1500 (Freq. < 1.5k)

VII. Results

A. Machine Learning Results

We tried four different ML algorithms for the time estimation model. Table II shows the MSE of the models in both training and testing, which helps us address a model's overfitting or underfitting issues.

TABLE II. ML Model Comparison

	Long short-term memory (LSTM)	*Linear regression*	*SVM*	*XGBoost*
MSE (Training)	10	6	5	0.001
MSE (Testing)	80	13	6	0.01
Problems	Data size is insufficient for use in deep learning model	Data complexity will not be covered using a linear model		-
Reference	[15]	[16]	[17]	-

XGBoost proved to be the best combination where:

1. Underfitting is not presented due to ensemble learning.
2. Overfitting is resolved by hyper-parameters tuning.

Based on that, XGBoost is selected for the time estimation service, and evaluated using three different metrics:

1) Accuracy

Accuracy is computed as the percentage of issues (story/bug) with a predicted time estimate of less than two working days' error margin from the actual estimate.

The model attained **0.82** accuracy.

2) Adjusted R2

R2 shows how well terms (data points) fit a curve or line. Adjusted R2 also indicates how well terms fit a curve or line but adjusts for the number of terms in a model. If you add increasingly useless variables to a model, adjusted R2 will decrease. If you add more useful variables, adjusted R2 will increase.

The model attained an adjusted R2 score of **0.98.**

3) Line Fitting

We tried to fit a line between the model's estimations and the actual ones to compare them and check how close the model's estimate to the actual time.

We can see from Fig. 9 that the model performs better with issues spanning a long time.

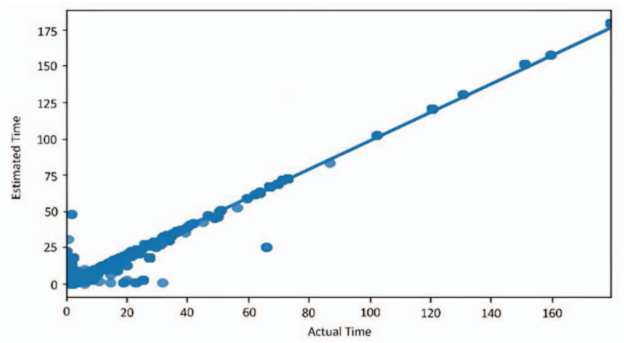

Fig. 9. Line fitting between estimation and actual.

B. Release Planning Results

For a typical release planning activity, the team starts to select the items that need to be addressed within the next sprint, then adds the time estimates and assigns resources. The Alfred management system allows users to follow a similar activity by selecting the items of interest. It then automatically adds the time estimates and assigns resources. In addition, it fires warnings if the aggregated issues' estimate of any resource exceeds the sprint time.

Alfred also provides a summarized sprint plan visualization overview, illustrated in Fig. 10, where we can see two resources with an overall aggregated assigned issues estimate exceeding the sprint time (shown as the horizontal red line).

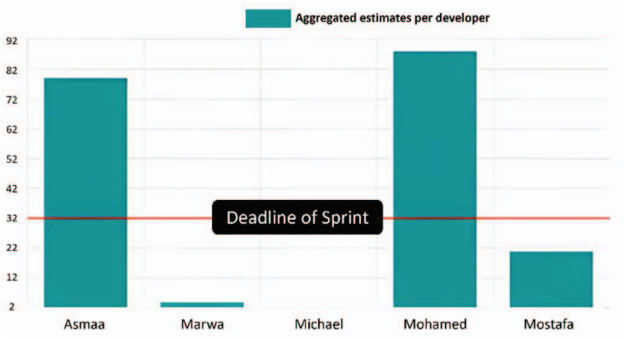

Fig. 10. Sprint plan summarized visualization.

When investigating the root cause for these resources overloading, it appeared to be that those resources were the seniors in the team with the most experience over the team's history. At this point, the manager can pose some questions to the smart assistant chatbot to receive help in rectifying the overloading problem.

The manager can ask the bot to assign items to a lower ranked candidate with less experience, but with an unallocated time buffer. Fig. 11 shows a sample conversation.

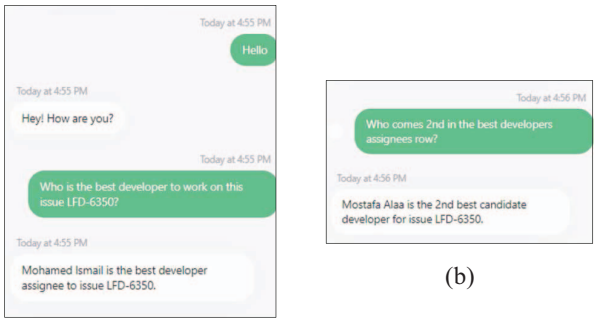

Fig. 11. Snippets of a chatbot conversation (a) asking about the best candidate developer for a certain item, (b) asking about the second best candidate for the same item.

Also, if the manager is suspicious about a certain task estimated time, he can ask the smart assistant about the confidence level of this estimation, as shown in Fig. 12.

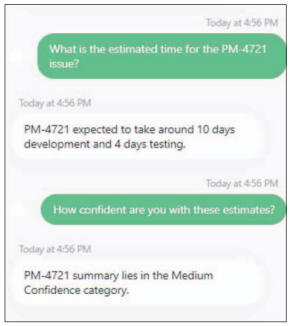

Fig. 12. Snippets of a chatbot conversation asking for the confidence level in the time estimation of a certain item.

VIII. CONCLUSION

In conclusion, Alfred aims to build a comprehensive end-to-end solution that integrates with state-of-art AI techniques to provide all the needed tools to the managers to help them in release planning and in assessing the expected risks by allowing them to foresee the expected time estimates of the whole release. This is achieved by the integration of the ML models for automated time estimation and resource allocation, the aid of the summarizing visualizations, and the smart assistant chatbot that provides more insights to guide the managers to rectify initial estimates and assignments as needed. Alfred provides a higher level of confidence to the project managers in their release planning and helps them in taking time-critical actions and preventive measures for future expected risks.

Our goal was to provide an easy-to-use system by creating a well-integrated system design architecture with smooth interaction between different modules. Our evaluation of Alfred demonstrates that we achieved our design goals as intended.

REFERENCES

[1] "What is Agile?", Agile 101, Agile Alliance. https://www.agilealliance.org/agile101/

[2] Mike Cohn, "Planning Poker: An Agile Estimating and Planning Technique", Mountain Goat Software. https://www.mountaingoatsoftware.com/agile/planning-poker

[3] Davahli, Mohammad Reza. (2020). "The Last State of Artificial Intelligence in Project Management".

[4] A.Ali and C. Gravino, "A systematic literature review of software effort prediction using machine learning methods", Journal of Software: Evolution and Process, vol. 31, no. 10, 2019. Available: 10.1002/smr.2211

[5] B.K. Singh, A.K. Misra, (2012), "An Alternate Soft Computing Approach for Effort Estimation by Enhancing Constructive Cost Model in Evaluation Method", In the International Journal of Innovation, Management and Technology, Vol. 3, No. 3, pp: 272- 275 ISSN: 2010-0248. 272.

[6] O. Fedotova, L. Teixeira, and H. Alvelos, "Software Effort Estimation with Multiple Linear Regression: Review and Practical Application". J. Inf. Sci. Eng. 2013, 29, 925–945

[7] E. Khatibi and V. Khatibi Bardsiri, "Model to estimate the software development effort based on in-depth analysis of project attributes", IET Software, vol. 9, no. 4, pp. 109-118, 2015. Available: 10.1049/ietsen.2014.0169.

[8] Ali Bou Nassif, Mohammad Azzeh, Ali Idri, Alain Abran, "Software Development Effort Estimation Using Regression Fuzzy Models", Computational Intelligence and Neuroscience, vol. 2019, Article ID 8367214, 17 pages, 2019. https://doi.org/10.1155/2019/8367214

[9] P. Pospieszny, B. Czarnacka-Chrobot and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms", Journal of Systems and Software, vol. 137, pp. 184-196, 2018. Available: 10.1016/j.jss.2017.11.066.

[10] I. D. Mienye and Y. Sun, "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects", in IEEE Access, vol. 10, pp. 99129-99149, 2022, doi: 10.1109/ACCESS.2022.3207287

[11] "Introduction to Rasa Open Source & Rasa Pro", RASA Docs, Rasa Technologies, https://rasa.com/docs/rasa/

[12] W. A. Qader, M. M. Ameen and B. I. Ahmed, "An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges", *2019 International Engineering Conference (IEC)*, Erbil, Iraq, 2019, pp. 200-204, doi: 10.1109/IEC47844.2019.8950616.

[13] Chen, Tianqi & Guestrin, Carlos. (2016). "XGBoost: A Scalable Tree Boosting System". 785-794. 10.1145/2939672.2939785.

[14] K. Taunk, S. De, S. Verma and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification", 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 1255-1260, doi: 10.1109/ICCS45141.2019.9065747.

[15] A. Pulver and S. Lyu, "LSTM with working memory", 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 2017, pp. 845-851, doi: 10.1109/IJCNN.2017.7965940.

[16] M. Huang, "Theory and Implementation of linear regression", 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), Chongqing, China, 2020, pp. 210-217, doi: 10.1109/CVIDL51233.2020.00-99.

[17] Basak, Debasish & Pal, Srimanta & Patranabis, Dipak. (2007). "Support Vector Regression". Neural Information Processing – Letters and Reviews. 11.