A Solution for Automatic Task Allocation in Software Development Teams based on Truck Factor and Implemented on GitHub and Trello

Dirlândia Oliveira[1], Caetano Segundo[2], Enyo Gonçalves[1], Marcos de Oliveira[1]

[1] Universidade Federal do Ceará (UFC) – Campus Quixadá
Av. José de Freitas Queiroz, 5003 – Cedro
Quixadá – CE – Brazil

[2]Centro Universitário Paraíso (UNIFAP)
Rua da Conceição, 1228 – São Miguel
Juazeiro do Norte – CE – Brazil

dirlandia.oliveira@alu.ufc.br, caetano.segundo@unifap.edu.br

{enyo,marcos.oliveira}@ufc.br

Abstract. ***Context****: Software development involves several steps and activities. One of these activities is task allocation. This activity is related to project management and is decisive for the success or failure of the project since it involves risks related to time and, consequently, costs.* ***Practical Problem****: Inefficient task allocation, combined with team member rotation and knowledge concentration among a few members, can negatively impact deadlines, costs, and even lead to project discontinuation.* ***Proposed Solution****: All knowledge related to the project must be distributed equally among all team members to mitigate the negative impacts of possible departures of team members. This work proposes a web solution for task allocation that considers the knowledge level of the team members on the project repository.* ***IS Theory****: The approach builds on theories of project management, knowledge distribution, and collaborative software development, emphasizing metrics for team knowledge assessment.* ***Research Method****: A web application implementing the proposed approach was developed and evaluated in the context of software development teams.* ***Summary of Results****: The solution enables balanced knowledge distribution among team members, mitigating risks associated with departures and knowledge concentration, and improves task allocation efficiency.* ***Contributions****: The proposed approach provides a practical tool for software project management, enhancing knowledge management and reducing project risks, contributing to both the research and practice of Information Systems.*

1. Introduction

The Chaos Report from 2020 [cha 2020] points out that about 19% of software projects failed, 50% of projects had a problem, and only 31% of these projects were successful. This report also highlights a set of critical success factors for IT projects such as: i) skilled team, consisting of the ability to acquire, manage and control the right resources at the right time, dealing with Turnover, as well as developing and maintaining competencies

and ii) optimization, aiming to obtain maximum value for the business with minimum risk.

The Turnover of team members is identified as one of the main factors that can impact project failure due to its impact on team productivity, delays in deliveries, risk of project discontinuity, and effect on project cost. One of the causes of this risk is the concentration of project knowledge in a few members. Thus, once these members leave, the impact on the project can be noticed.

The high demand for software in recent years, especially after the start of the COVID-19 pandemic, combined with the offer of remote work, has accelerated software turnover in the Information Technology (IT) area. A search on Linkedin [1]. Accessed on: 19 March. 2024 for vacancies in the IT area showed an availability of 62.704 vacancies in Brazil and 2.628.688 vacancies worldwide. Therefore, appropriate strategies are needed to help IT companies deal with Turnover and reduce its negative impact on projects.

Developing a software project involves several steps, which are made up of activities. One of these activities is task allocation. In this activity, the technical leader or project manager will determine which tasks must be performed and who will be responsible for that activity. A poor distribution of these tasks can cause the concentration of the repository's knowledge level among a few team members, consequently increasing the impact of staff turnover and increasing project risks.

The *Truck Factor* concerns how much the project depends on the team's knowledge within the development environment[Avelino et al. 2016]. The ideal scenario occurs when the knowledge that the team has about the project is as distributed as possible so that in exchanges between its members, the project continues at its natural pace of development.

Turnover in software projects, involving changes in team members, has been the focus of ongoing research to understand better its impact and underlying causes [da Silva and Sampaio 2025]. That occurs due to various factors [Hilton and Begel 2018, Farooq et al. 2022] and negatively impacts project development in corporate environments and open-source project settings [Ferreira et al. 2020]. Data presented by [Pee et al. 2014] shows that in 2012, around 20% to 28% of software projects failed for different reasons. Therefore, team turnover proved to be a significant risk to the project's continuity. More recently discussed by [Mohamad et al. 2021] in 2018, the IT industry sector had a turnover of 10.9%, and within the IT sector, software teams were the most affected. The distribution of knowledge about the repository is a way to reduce the negative impact of Turnover on the software teams.

Genetic algorithms [Goldberg 1989] are an optimization technique inspired by the Darwinian principle of species evolution and genetics. They are probabilistic algorithms that provide a parallel and adaptive search mechanism based on the principle of survival of the fittest. This approach can ensure that tasks are assigned more efficiently, considering the skills and knowledge of each team member.

The Truck factor is a metric that can be used in this context, as it represents the level of knowledge distribution in the project repository. For the project to continue func-

[1] link available at: https://www.linkedin.com/jobs/search/?keywords=Information%20Technology

tioning, it is essential that all knowledge regarding the project is distributed equally and that if team members leave, the project continues to function without any interruption [Avelino et al. 2016].

Turnover is a recurring problem in software development projects. It is essential to adopt strategies to manage this phenomenon to mitigate its negative impacts on productivity and efficiency. Task allocation software based on Genetic Algorithms and Truck Factor could be an interesting solution.

Thus, this paper presents a solution to support task allocation focusing on optimizing the Truck Factor with genetic algorithms and integrated with Trello and GitHub, two environments commonly used in software project development. This paper's other sections are organized as follows: Section 2 presents the background about Project management, Task allocation, Turnover in software teams, Truck Factor, and Genetic Algorithms. Section 3 presents the related work. Section 4 presents the results of the solution proposed to support the task allocation. Finally, Section 6 shows the findings and future work.

2. Background

This section will explore the main techniques and concepts applied in the research. Agile methods for managing software projects have gained prominence for their flexibility in distributing tasks and continuously delivering value to the customer. Turnover can present itself as a challenge since changing the team directly impacts project delivery and timing. Therefore, understanding these elements and other techniques, such as Truck Factor and Genetic Algorithms, is essential to propose an adequate solution for our study problem.

2.1. Project management, Task allocation and Turnover in software teams

Given their simplicity and flexibility, agile methods are widely used to manage software teams. As seen in [Sommerville 2011], these methods allowed the development team to work on the software rather than its design and documentation. Agile methods, universally, are based on an incremental approach to the specification, development, and delivery of software. They are best suited to developing applications where system requirements change rapidly during the development process. They are intended to deliver the software quickly to customers, up and running. They can then propose changes and new requirements for subsequent system iterations.

Among agile methods, Scrum [Sutherland 2015] stands out due to the simplicity and flexibility of its work process. The main roles are: *Scrum Master*, who is responsible for managing the team, helping everyone involved to understand and embrace the values, principles, and practices of Scrum; *Product Owner*, who is the main actor within the process who has leadership over the product. He is solely responsible for deciding which resources and functionalities will be built and in what order they should be carried out; *Development Team* is the name given to the team with multidisciplinary professionals, as they are programmers, software architects, testers, etc.

The main task of the method is the execution of the *Sprint*, which lasts between 1 and 4 weeks, always seeking to deliver a tangible product at the end, whether for the user or the client. In other words, during the sprints, the entire system is built. In addition to

these activities, different processes are necessary for effective sprint management, such as sprint review, daily meetings, sprint planning, and sprint retrospectives.

The artifacts generated are: *product backlog*, a document constantly evolving. Items can be changed and revised by the Product Owner due to changes in business conditions; the Sprint backlog is a set of activities removed from the product backlog to be carried out during the established sprint period.

Since Scrum is adaptive, task assignments can occur during planning or the Sprint. In this way, the delegation of tasks is carried out by an agent with these responsibilities, or even the team itself defines what each person should do. The criteria for attribution are also not specific; they are generally technical criteria, such as the agent's ability to perform a particular function, but according to the objective sought, the criteria for attributing tasks can be adapted.

In this way, it is understood that the application of an agile process in a team composed of agents becomes fundamental because, as Scrum has short cycles at each end of the cycle, it is possible to obtain performance metrics for each agent so that in the next Sprint, they can be readjusted.

2.2. Truck Factor and Genetic Algorithms

Since we already understand the aspects of a software team, we know that it is subject to changes throughout its development time. This Turnover can generate instability in the project and loss of performance, considering that changes can cause fluctuations in the continuous delivery of the product. In addition to performance, changes can affect delivery time, cost, or even declare the end of the project [Pee et al. 2014].

Understanding that changes in software teams become inevitable, this problem must be studied and minimized. One of the key points of all projects is knowledge management. The departure of an essential team member can compromise the project's development, as the rest of the team has little knowledge about the project [Jabrayilzade et al. 2022]. From this, research emerged that seeks to understand team members' knowledge about the project and its respective impacts.

The Truck Factor (TF) is a measure of risk on the project, which is caused by the omission of information about it from its members. It can be defined as the number of people who are run over by a truck (Truck), where this factor impacts the project's development since the truck being run over represents the departure of team members [Ferreira et al. 2017]. In this area of research, it is understood that when team members have a concentration of technical knowledge about the project, they form knowledge silos, and this can harm the project, from delivery delays, or even the closure of the project.

Based on this problem, researchers aim to calculate and compare different approaches [Ferreira et al. 2016], based on artifacts developed by the team, and what their impact factor would be. In several other studies [Zazworka et al. 2010], [Sandim et al. 2017], and [Hannebauer and Gruhn 2014], the authors investigate, compare, and propose new techniques through algorithms, so that the Truck Factor of a given project can be calculated.

In the approach proposed by [Avelino et al. 2016], the author proposes a greedy approach for calculating the TF. They assign a DOA (Degree Of Authorship) to each

Input: List of the authors (A) and Files
Output: Truck Factor
$S \leftarrow$ getFiles(A);
$tf \leftarrow 0$;
while $A \neq \theta$ **do**
    $coverage \leftarrow$ getCoverage(S, A);
    if $coverage < 0.5$ **then**
        break;
    end
    $authors \leftarrow removeTopAuthor(A)$;
    $tf++$;
end
return tf;

Algorithm 1: Truck Factor Algorithm
[Avelino et al. 2016]

developer in the project and have made at least one commitment to the file being analyzed. The DOA has three parameters: 1. First author differentiates who initially created the file, 2. The number of changes made to the file, and 3. Number of changes made to the file by the rest of the team. Once all DOA values are known, a normalization occurs, where values between 0 and 1 are defined. Ultimately, the file's author is considered the one with DOA > 0.75.

In this approach, it is understood that the project is at serious risk of delays and may even be discontinued if the coverage of the current group of authors is less than 50%. The Truck Factor result shows the number of authors that can be removed from the project until coverage is at least 50%.

An implementation is presented using pseudocode in the Algorithm 1. The algorithm takes as input a mapped set of authors and files, so it is possible to know which authors contributed to each file. The algorithm starts tf with value 0, then there is a repetition loop that iterates over the system files, in which its coverage is checked at each cycle $(getCoverage(S, A))$, which checks the number of authors present by the number of files in the repository. Suppose the coverage value is less than 0.5. In that case, the algorithm ends, returning the current Truck Factor value; otherwise, the principal author $(removeTopAuthor(A))$ is removed, and $+1$ is added to the Truck Factor value.

When we analyze the problem of task allocation, these are problems widely discussed as optimization problems. Since we are addressing an optimization problem, we can use Genetic Algorithms. This class of algorithms is based on the theory of Evolution developed by Charles Darwin [Darwin 1968], it is a probabilistic algorithm based on the principle of survival of the fittest and reproduction using selection principles, in computing they are generally used in optimization problems, as provide optimal solutions promptly [Lambora et al. 2019].

Genetic Algorithms seek to find a solution considered optimal for a given problem. The environment comprises a population with individuals with different characteristics, and each individual is a possible solution to the problem. Over time, only the best in-

dividuals of this population should continue reproducing and sharing their genes. Each individual is evaluated, and according to its suitability for the solution, it will have a greater chance of being selected for reproduction. Only the best will be reproduced to generate a more evolved population that is increasingly closer to the solution. The idea is that only a set of individuals come close to a solution considered optimal after a specific time.

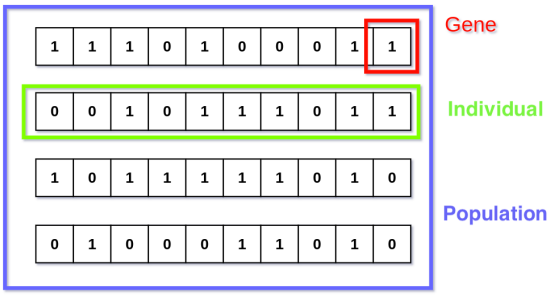

Figure 1. Representation of Gene, Individual, and Population in Genetic Algorithms

In Figure 1, we have the representation of the Gene, Individual, and Population. The individual is characterized by a set of genes, which store information related to solving the problem and may have different ways of modeling; however, implementation through vectors is common. And finally, the population characterizes this group of individuals.

With the evolution of genetic algorithms, different implementations emerged. However, some processes are standard to their development. As seen in the Algorithm 2, the process begins with the generation of the initial population and the calculation of its fitness. In sequence, a repeated process of selection, crossing, mutation, and fitness of individuals must be repeated until a stopping criterion is reached. The items below will detail the processes carried out in implementing the genetic algorithm.

- **Selection:** It is considered key within the algorithm, since among all the individuals present in the population, the algorithm should select only a few to proceed to the next generation. Its main objective is to ensure that the descendants of the next generation have even greater fitness [Mitchell 1998]. Very rigorous selection

Result: Best individual of the population
Create initial population;
Calculate fitness of the initial population;
while *not reach the stop criteria* **do**
    Select the best individuals (father);
    Crossing the selected individuals;
    Mutation of the generated individuals (sons);
    Calculate population fitness;
end

Algorithm 2: Canonical Genetic Algorithm Implementation

only allows extremely fit individuals to take control of the population; however, it reduces diversity for change and progress. On the other hand, weak selection will result in very slow evolution. Therefore, it is ideal to maintain a balance in the selection process of individuals.

- **crossing:** It is the choice of two individuals from the population, who are called parents, and carry out a combination of their genes to generate a new individual with part of the solution from each parent. There may be different ways to perform gene combinations.
- **mutation:** It is a process that copies an individual from the previous population and modifies only a few genes; this is another mechanism that genetic algorithms have to guarantee diversity within their population.
- **Evaluation Function or Aptitude:** Moment where each individual must be evaluated, this process must verify how close to a solution considered ideal, this evaluation is directly linked to the problem in question since the individual's evaluation will have an impact on selection.

3. Related Work

In [Wolschick et al. 2024], the authors employ search-based algorithms to address the optimization of Product Line Architecture (PLA). The primary focus of the study is to compare the performance of NSGA-II and NSGA-III in optimizing metrics related to feature modularization, cohesion, and architectural extensibility. The results show that NSGA-III performs slightly better than NSGA-II in scenarios involving four or five objectives, while NSGA-II excels in less complex problems.

In [Farhangian et al. 2015], the author addresses the context of the research focused on task allocation in software teams. However, it includes a new variable regarding team tasks: the dynamic task. The study uses a simulation with multi-agent systems to allocate team activities based on the team's personality and skills, aiming to minimize an over-competent group and another with very low competence. The researcher proposes a tool that project managers can use to simulate at a low cost to gain insights into the task division strategy and people with different skills. The research considers the dynamic nature of the project's activities and whether a change in requirements or interdependence directly affects the team's effectiveness. When the article discusses dynamic tasks, it refers to activities that have requirements and other variables modified. His contribution is presented through the applied literature review and a model to verify a team's performance.

Furthermore, they analyze the dynamic nature of a task and managers' strategy for task allocation. They use the Netlogo [2] tool to perform the simulation. This tool does not support integration with code repositories and allocation tasks.

[Strand et al. 2020] describes the Carrot tool, which employs machine learning and collaborative filtering to recommend code reviewers, balancing workload between them. The tool considers reviewer experience, code complexity, and workload to recommend the most suitable reviewers. Carrot's architecture comprises microservices, including using Gerrit for data extraction and continuous user feedback, ensuring continuous tool improvement. This tool was proposed to support allocating a specific task in software

[2] https://ccl.northwestern.edu/netlogo/

development. This tool does not support integration with code repositories and allocation tasks.

[Simão Filho et al. 2019] introduces the DiVA methodology to improve task allocation in Distributed Software Development (DSD) environments, employing Verbal Decision Analysis (VDA) to evaluate and classify factors that influence task allocation. The approach is flexible and adapts to different priorities in each project, ensuring an objective and transparent allocation of tasks. The implementation of DiVA in five companies showed its ability to adapt to various scenarios and organizational requirements, improving inter-team collaboration and the overall effectiveness of projects. This methodology, such as genetic algorithms, is not an automatic AI-based task allocation. This methodology does not support integration with code repositories and allocation tasks.

4. Results

This section describes the development of a solution for allocating tasks in software teams, based on a questionnaire that identified needs and problems in existing platforms. From this, the application's functionalities were defined, and the technologies and the integration between platforms used in the development of the system were detailed. It concludes with a practical demonstration of GitHub and guidance for interested users to access and use the tool.

4.1. Application architecture

The solution flow is shown in the Figure 2. The Firestore Database in the application acts as the core of the solution, as it is where information from GitHub and Trello is stored. After storage, the data is reorganized into matrices in the solution and sent to the genetic algorithm, which returns the value of the Truck Factor from the repository and the best task allocation solution.

4.1.1. Definition of main features

Next, it defines which functionalities must be incorporated into the developed solution, prioritizing those essential for application use.

1. Login with GitHub: Allows the project manager to access repositories in which he is involved, facilitating access to project information such as participants, contributions in each file, and contributor information.
2. Authentication with Trello: Essential for the task allocation process, the project manager can search and extract information from task cards in the "To Do" column of their Trello boards.
3. Add Skills Required for Task: After retrieving the tasks from Trello, you can determine the skills needed for each task, which is essential for applying the genetic algorithm.
4. Add New Task: Provides the option to add tasks manually if the project manager chooses not to extract tasks from Trello.
5. Truck Factor Calculation: After logging into GitHub and selecting the repository, the user must fill in task skills and developer skills data to generate the analysis and return the Truck Factor value.

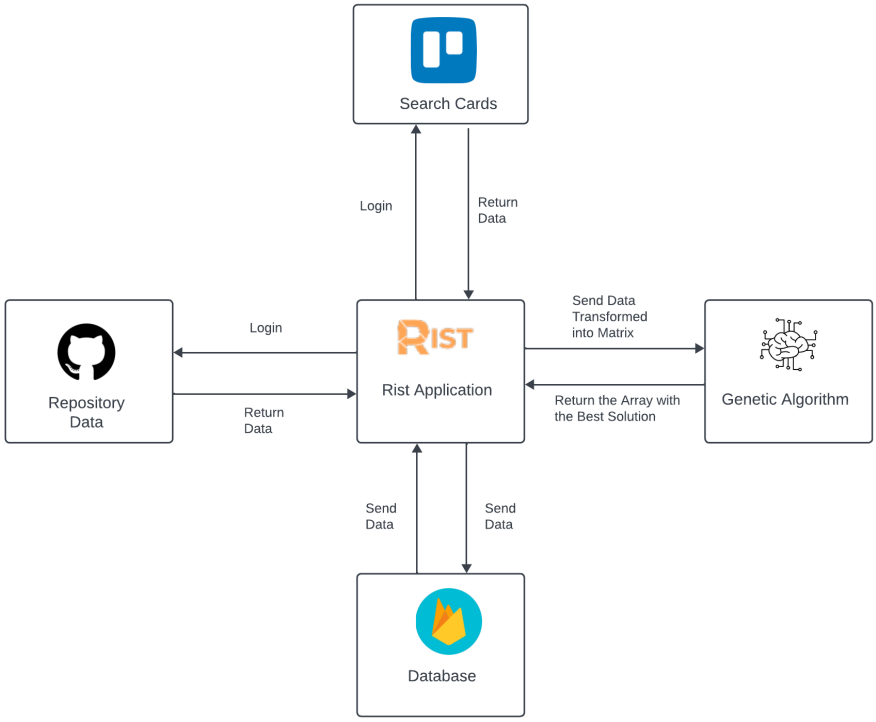

Figure 2. Activities Workflow

6. Task Allocation: The application's main functionality is that developers are assigned tasks through the genetic algorithm, and can perform new analyses if necessary.

After this initial analysis, a more technical analysis was conducted to define the technologies for application development. That included using existing algorithms, such as the Truck Factor algorithm[Avelino et al. 2016] and the Genetic Algorithm implemented in the work of [VIEIRA NETO SEGUNDO 2022]. Additionally, the application requires integration with GitHub and Trello APIs to extract data from these platforms.

4.1.2. Technologies and computational resources used

In the development phase of this application, a diverse set of technologies and tools was employed, each playing a crucial role in meeting the functional and aesthetic requirements of the project. Here is a detailed expansion of the technologies used:

- JavaScript: Used on both the client and the server, this scripting language is fundamental in creating dynamic and interactive web pages.
- Node.js: Node.js, adopted for the backend, ensured efficient and scalable execution on the server side. Its ability to handle multiple requests simultaneously, thanks to its non-blocking I/O model, made it ideal for the solution.
- Flask: Using Flask to connect the application to the genetic algorithm in Python, known for its simplicity and ability to integrate with other technologies easily, was a strategic choice to create a bridge between the frontend and the backend processing in Python.

- React: React allows a declarative and modular approach to building interfaces, facilitating code maintenance and scalability. Additionally, utilizing the state and lifecycle of React components enhances the user experience by enabling real-time updates and fluid interactivity.
- Trello API: Integration with the Trello Application Programming Interface (API) brought an additional layer of functionality, allowing direct interaction with external data. That enriched the application with the ability to connect to a broader ecosystem, opening doors for automation and synchronization of tasks across projects.
- GitHub API: The GitHub API expanded the application's reach into the world of code repositories. This direct connection to GitHub allowed for a deeper view of development projects, bringing valuable information about commits, branches, pull requests, and more, directly to the application interface.
- Firebase and Cloud Firestore: The choice of Firebase and, more specifically, Cloud Firestore, reflected the need for agile, real-time data storage. Firestore, with its NoSQL approach and real-time synchronization, offered an effective solution for storing and retrieving contributor and task data pulled from Trello and added manually, supporting the dynamic nature of the application.

This set of technologies simplified the implementation of advanced functionalities and ensured the application's scalability, maintenance, and usability. Combining these tools resulted in a dynamic and effective development process while providing users with a reliable, responsive, and easy-to-navigate application.

4.1.3. NoSQL-Cloud Firestore Database

Cloud Firestore, a NoSQL database powered by Firebase, is ideal for mobile, web, and server development. To integrate the database with the application, creating a project in the Firebase console was initially necessary. In the project settings, a script is generated and used in the application to facilitate interaction between the application and the database. With this integration, storing and updating all data relating to developers and tasks in real time, including functionalities for creating, editing, and deleting tasks, became possible.

4.1.4. Github

GitHub offers functionality such as task management and pull requests for code review, as well as hosting a variety of open-source projects. Through its API it is possible to obtain relevant data for the solution, as the API offers several functionalities such as: accessing repositories, obtaining user information, and performing authentication and several other functionalities, as well as logging in with the platform informing the owner of the repository and repository searched, thus obtaining data from the repository's contributors. After that, all developers who are contributors to that repository will be listed, and the repository contributions matrix will be generated.

Matrix A (1) presents an example of a Contribution Matrix. It represents the number of contributions in each file in the repository, with each column representing a

file and each line representing a contributor. The total number of files analyzed is 8, and each contributor, looking at the matrix, shows that file 3 has contributions from all participants. Still, contributor 3 made a higher number of contributions compared to the other contributors.

$$A = \begin{bmatrix} 1 & 1 & 6 & 1 & 2 & 1 & 1 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

4.1.5. Trello

Through Trello, there is the option to build applications on top of the tool by creating a Power-Up. The Trello Power-Up is a way to leverage the API to extend Trello's functionality and integrate it with other tools and services. The API offers endpoints to manipulate boards, cards, lists, team members, etc. To use the API, you need to create an API token and API key on the Trello website. This data is acquired after making a Power-Up in Trello. When using this API in the solution, it is possible to request data from the user who is connected to Trello, in the case of the application, all Task Boards of that user are shown, through the Selected Board it is possible to list all the tasks that are in the column "To Do".

After this, the user must select the skills developers need for the task to be carried out. All tasks are stored in a NoSQL cloud database. The data used from these tasks is represented only by task indexes. The identification of the task and skills together forms a matrix. The Task Skills Matrix represents the tasks that must be allocated. An example of this matrix is presented by Matrix (2), which is composed of a set of 5 tasks (rows) and 10 skills (columns). If it is marked with 1, the skill is essential in the task.

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2}$$

4.1.6. Genetic Algorithm and Truck factor implementation

Our work addresses the use of a genetic algorithm (see Algorithm 2) implemented in Python using the DEAP (Distributed Evolutionary Algorithms in Python) library to implement and manage the genetic algorithm that optimizes the distribution of tasks in software projects. It provides the necessary tools for creating, evaluating, evolving, and selecting solutions, playing a vital role in improving the efficiency and effectiveness of project management. The Truck Factor algorithm (see Algorithm 1) was also implemented in Python. To use these algorithms in Python within the JavaScript application, it was necessary to use the web framework in Python, Flask, to create a web API, through which it was possible to implement the genetic algorithm.

The Genetic algorithm uses three matrices representing agents, tasks, and the repository. These matrices represent, respectively: The agent matrix (1) comprises project contributors, each characterized by a unique set of skills. The function of this matrix is to identify which agents are best able to perform certain tasks, based on their specific skills. The task matrix (2) contains a breakdown of the tasks required to complete the project, including the skills needed for each of them. The algorithm uses this information to match tasks with agents, ensuring that each task is assigned to the most qualified agent. The repository matrix (2) represents the contribution of each agent to the various files in the project repository. This matrix is crucial for understanding each agent's experience level and involvement with different code sections, thus influencing the allocation of tasks related to these files.

In the application, the user manually enters data related to developer skills. The Skill Value Matrix presented in [matrizC]Matrix C (3) represents the skills of six different contributors (rows) in ten different areas (columns). Skills are assessed on a scale of 1 to 10.

$$C = \begin{bmatrix} 5 & 7 & 6 & 7 & 6 & 5 & 3 & 2 & 8 & 8 \\ 8 & 6 & 5 & 7 & 6 & 6 & 5 & 6 & 7 & 7 \\ 10 & 5 & 6 & 7 & 6 & 6 & 6 & 7 & 6 & 6 \\ 10 & 9 & 8 & 8 & 6 & 7 & 5 & 5 & 8 & 5 \\ 9 & 6 & 8 & 8 & 7 & 7 & 7 & 5 & 8 & 6 \\ 5 & 8 & 8 & 7 & 5 & 8 & 5 & 7 & 5 & 6 \end{bmatrix} \tag{3}$$

Using these matrices sent to the genetic algorithm, several possible solutions for task allocation are generated, where each solution represents a specific combination of tasks and agents. Each solution is evaluated based on criteria such as the compatibility of agents' skills with the tasks and the balanced distribution of work. The algorithm seeks to continuously improve these solutions through selection, crossover, and mutation, aiming to find the most efficient and effective configuration for task allocation.

4.2. Main Features

This section presents the tool's main features implemented based on the proposed solution.

In the repository search screen shown in Figure 3, the user must enter data about the repository he wants to search, if it is a public repository the user must enter data about

the repository owner and the name of the repository he intends to search if the repository is private, the user must have permissions within the repository to search. After entering the data correctly and performing the search, the user is directed to the application's board page, shown in Figure 3.

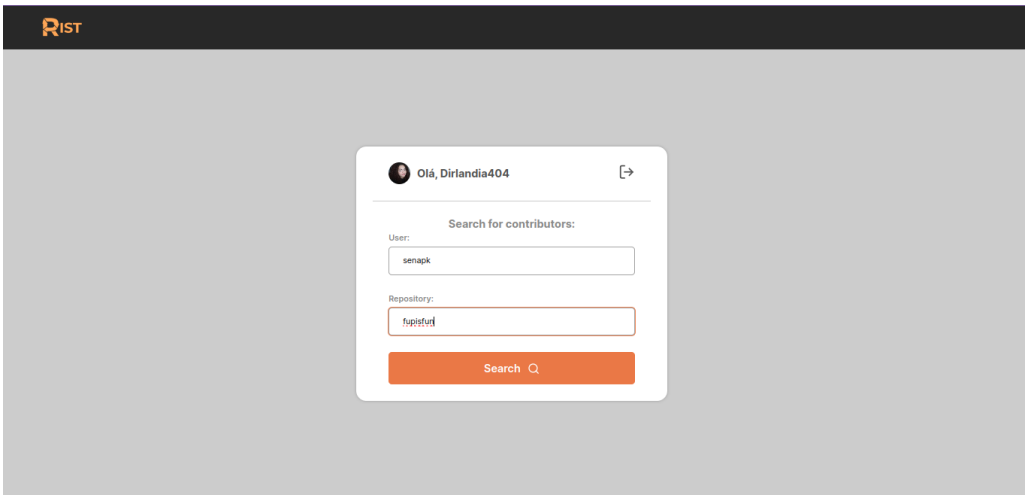

Figure 3. Repository Search Screen

If the user chooses not to synchronize Trello tasks or needs to supplement the list with tasks not on the platform, they can proceed with manual insertion. This process involves filling out detailed information Figure 4, including describing the task and selecting the skills required for its execution.

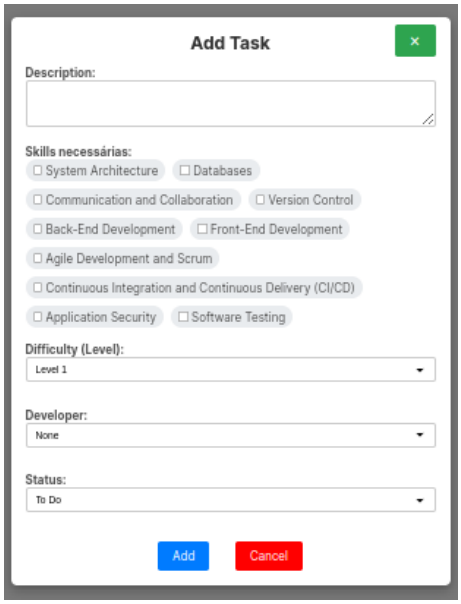

Figure 4. Add new task

The skills of contributors associated with the repository in question require manual assignment, as there is no way to obtain this information from GitHub automatically. As illustrated in Figure 5, the interface for this assignment presents a list of skills where specific values must be assigned to each applicable skill.

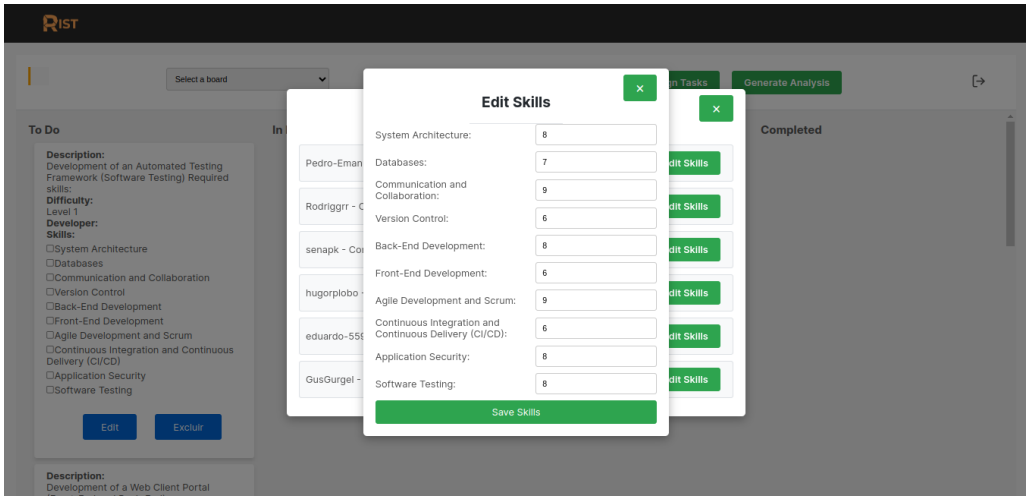

Figure 5. Designation of skill values

Once the skills required for the tasks and the developers' competencies have been defined, the Truck Factor analysis can be performed by selecting the "Generate analysis" option. Once the analysis is complete, the Truck Factor value is presented in the interface, Figure 6, allowing the user to allocate tasks to employees.

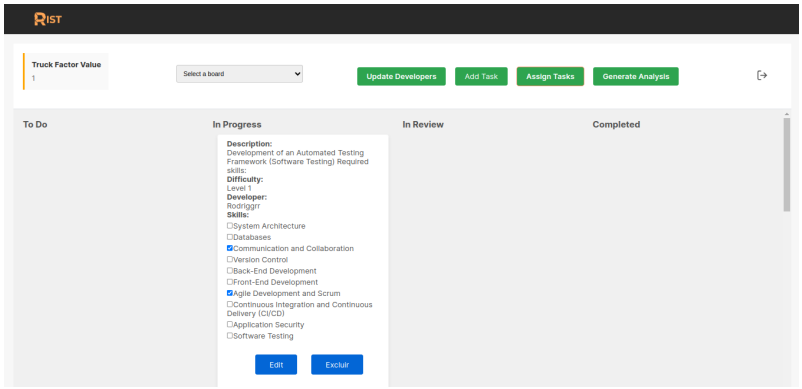

Figure 6. Truck Factor Value Display

5. Illustration

This section demonstrates the application through the "fupisfun" project. That is a project aimed at a programming course. The project has six collaborators and is a public repository. The project repository is available at `https://github.com/senapk/fupisfun/`.

The application implemented for the previously described environment is publicly available at: `https://github.com/Dirlandia404/Rist-Reducing-Impact-turnover` The allocation results produced during the experiments are reported in the following section and are also available online at: `https://drive.google.com/drive/folders/1lQLF9nPyEtV6E5n4R6eUh1qYemiTU2ey?usp=sharing`

The matrices generated from the project were presented in Section 4 to illustrate each of the matrices used by the developed solution. The project's Contribution Matrix was presented through Matrix A (1).

Below is a list of tasks with the corresponding skills required for each one:

Table 1. Tasks and Their Required Skills

Task	Skills
Development of a Web Customer Portal	Backend Development, Frontend Development
Implementation of a Secure Authentication System	Application Security, Software Testing
Database Performance Optimization	Databases, Systems Architecture
Development of an Automated Testing Framework	Software Testing, Systems Architecture
Creating a CI/CD Pipeline for a Mobile Application	Continuous Integration and Delivery (CI/CD), Databases

While the Ability Value Matrix presented in Section 4.1.6 through MatrixC (3). Each line represents a contributor and each column represents an area, these areas are respectively: "Systems Architecture", "Databases", "Communication and Collaboration", "Version Control", "Backend Development", "Frontend Development", "Agile Development and Scrum", "Continuous Integration and Delivery (CI/CD)", "Application Security", "Software Testing".

Figure 7. Values of each developer skill

In Figure 7, it is possible to visualize the six developers and the values assigned to the 10 existing skills. These values for each skill were determined to represent how much mastery each developer had regarding each skill.

In Figure 8, it is possible to see the five existing tasks; these tasks were extracted from Trello for the application, and the necessary skills assigned to each one. The skills defined for each were determined according to what the task needed to be carried out.

By sending these matrices to the genetic algorithm 10 times, it generated the solutions summarized in Table 2. The different solutions indicate the efficient exploration of the genetic algorithm in the space of possible solutions.

Figure 8. Required tasks and skills

Table 2. Generations 1 to 10 represented as arrays

Generation	Solution
1	[5, 1, 0, 4, 3]
2	[5, 4, 0, 1, 3]
3	[4, 3, 5, 1, 0]
4	[4, 5, 1, 3, 0]
5	[0, 5, 4, 1, 3]
6	[0, 1, 5, 4, 3]
7	[0, 4, 5, 1, 3]
8	[1, 5, 3, 4, 0]
9	[3, 5, 0, 1, 4]
10	[5, 4, 1, 3, 0]

These solutions can be used to allocate the developer to each specific task. Figure 9 shows the allocation of generation 10 in Table 2, where developer 5 was allocated to the task with index 0, developer 4 was allocated to the task with index 1, the developer 1 was allocated to task index 2, developer 3 was allocated to task index 3 and finally developer 0 was allocated to task index 5. It is worth noting that the index of each developer is determined by the list of Team shown in Figure 10.

Figure 9. Allocation carried out

6. Conclusions and future work

Turnover is a problem that has been impacting software development. A vast number of proposals have been made to mitigate the negative impacts of this problem. One of these ways is performing the distribution of the repository knowledge. This work presented a

Figure 10. Developer List

solution for automatic task allocation based on genetic algorithms and truck factor. The solution is available as a web application and is integrated with GitHub and Trello.

The development involved a set of technologies. The features available allow the retrieval of data from GitHub repositories and Trello projects, will enable you to adjust the skills data of each participant, as well as view the project's current truck factor information and carry out task allocation automatically, through of genetic algorithm, considering the value of the truck factor and the skills of the development team.

GitHub API Constraints: Limitations on the number of requests that can be made during a given period, directly affecting the search for large project repositories.

Firestore Storage and Reading Constraints: Firestore limits the amount of data we can store, and the number of read and write operations. Limitations occur when the database is queried frequently.

During the development of the solution, some blockers arose, as the only possible data extracted from GitHub was the number of commits each developer made. Still, this metric is not very effective in determining the truck factor, as there are other factors as well: how vital that commit was to the application, what data it added to the document. Unfortunately, it was impossible to extract this data precisely, so the truck factor was determined only by the number of commits. That is a possibility for improving the developed tool.

Other suggestions for future work are presented below.

- Carrying out system tests with a development team over time to learn how effective the Rist tool was for the task allocation process.
- Add features as well as: the option to select just a few tasks to perform the allocation, select more than one developer, add more factors that contribute to the generation of the genetic algorithm as well as other skills, for example: knowledge of testing, backend development and among other skills.
- Improve the user interface, respecting usability standards, using Nielsen Heuristics.

References

(2020). Chaos report. `https://www.standishgroup.com`. Accessed at: 26 jun. 2022.

Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10.

da Silva, W. V. X. and Sampaio, S. (2025). A decade of turnover intention research in it: Key insights and trends. In *Congresso Ibero-Americano em Engenharia de Software (CIbSE)*, pages 135–149. SBC.

Darwin, C. (1968). On the origin of species by means of natural selection. 1859. *London: Murray Google Scholar*.

Farhangian, M., Purvis, M., Purvis, M., and Savarimuthu, T. B. R. (2015). Agent-based modeling of resource allocation in software projects based on personality and skill. In *International Workshop on Multiagent Foundations of Social Computing*, pages 130–146. Springer.

Farooq, H., Janjua, U. I., Madni, T. M., Waheed, A., Zareei, M., and Alanazi, F. (2022). Identification and analysis of factors influencing turnover intention of pakistan it professionals: An empirical study. *IEEE Access*.

Ferreira, F., Silva, L. L., and Valente, M. T. (2020). Turnover in open-source projects: The case of core developers. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, pages 447–456.

Ferreira, M., Avelino, G., Valente, M. T., and Ferreira, K. A. (2016). A comparative study of algorithms for estimating truck factor. In *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 91–100. IEEE.

Ferreira, M., Valente, M. T., and Ferreira, K. (2017). A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 207–217. IEEE.

Goldberg, D. (1989). *enetic Algorithms in Search, Optimization and Machine Learning,*. Addison-Wesley Publishing Company, USA.

Hannebauer, C. and Gruhn, V. (2014). Algorithmic complexity of the truck factor calculation. In *International Conference on Product-Focused Software Process Improvement*, pages 119–133. Springer.

Hilton, M. and Begel, A. (2018). A study of the organizational dynamics of software teams. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 191–200.

Jabrayilzade, E., Evtikhiev, M., Tüzün, E., and Kovalenko, V. (2022). Bus factor in practice. *arXiv preprint arXiv:2202.01523*.

Lambora, A., Gupta, K., and Chopra, K. (2019). Genetic algorithm-a literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 380–384. IEEE.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Mohamad, M. R., Nasaruddin, F. H., Hamid, S., Bukhari, S., and Ijab, M. T. (2021). Predicting employees' turnover in it industry using classification method with feature selection. In *2021 International Conference on Computer Science and Engineering (IC2SE)*, volume 1, pages 1–7. IEEE.

Pee, L. G., Kankanhalli, A., Tan, G. W., and Tham, G. (2014). Mitigating the impact of member turnover in information systems development projects. *IEEE Transactions on Engineering Management*, 61(4):702–716.

Sandim, H., Brandao, M. A., and Moro, M. M. (2017). Stf: uma abordagem social para estimar truck factor no github.

Simão Filho, M., Pinheiro, P. R., Albuquerque, A. B., Simão, R. P. S., Azevedo, R. S. N., and Nunes, L. C. (2019). A multicriteria approach to support task allocation in projects of distributed software development.

Sommerville, I. (2011). *Engenharia de Software*. Pearson Prentice Hal, São Paulo, third edition.

Strand, A., Gunnarson, M., Britto, R., and Usman, M. (2020). Using a context-aware approach to recommend code reviewers: Findings from an industrial case study. In *Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE.

Sutherland, J. (2015). *Scrum: The Art of Doing Twice the Work in Half the Time*. Random House Business Books.

VIEIRA NETO SEGUNDO, C. (2022). Uma simulação multiagente para alocação de tarefas em projetos de software baseada no truck factor. Dissertação (mestrado em computação), Universidade Federal do Ceará, Quixadá. Programa de Pós-Graduação em Computação.

Wolschick, L., Gonçalves, P. C., Neto, J. C., Freire, W. M., Amaral, A. M. M. M., and Colanzi, T. E. (2024). Evaluating the performance of NSGA-II and NSGA-III on Product Line Architecture Design.

Zazworka, N., Stapel, K., Knauss, E., Shull, F., Basili, V. R., and Schneider, K. (2010). Are developers complying with the process: an xp study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10.