



An Agent-Based Simulation for Task Allocation in Software Development Teams based on the Truck Factor Metric

Caetano Segundo^a, Marcos Oliveira^b, and Enyo Gonçalves^b

^a Paraíso University Center - 1228 Conceição St., Juazeiro do Norte - CE - Brasil

^b Federal University of Ceará - Campus Quixadá, 5003 José de Freitas Queiroz St., Quixadá - CE - Brasil

✉ caetano.segundo@fapce.edu.br, marcos.oliveira@ufc.br, enyo@ufc.br

KEYWORDS

task allocation;
agent-based
simulation; genetic
algorithm; truck
factor metric

ABSTRACT

Several tasks must be performed for a software development project to be completed successfully. Task allocation is a complex task that significantly impacts the project's success. Techniques have been proposed over the years to support this step, aiming to minimize cost and development time and to reduce the negative impact of team members leaving the project. In this context, the Truck Factor (TF) is a metric that can determine the risk to a project and can be used when distributing tasks among team members. The TF concerns the distribution of knowledge about the project among the development team members, ensuring that knowledge is not concentrated in only one part of the team. This theme is relevant nowadays since team rotation has become frequent due to the increasing demand for software in recent years. Member allocation in software teams does not have an exact solution since it is an NP-hard problem. Thus, Search-Based Software Engineering (SBSE) techniques, which can apply optimization algorithms such as genetic algorithms, have been used in several types of research to solve this class of problem over the years. In multi-agent environments, a certain number of agents have perception and communicate to achieve their goals. Researchers in the multi-agent field use simulated environments to validate their research since the modeling and simulation must consider the main variables of a real environment. Therefore, this work proposes to build a multi-agent simulation using SBSE techniques to minimize the impacts caused by the Truck Factor in a software development

team. In the simulations, we model different configurations for the software development teams. Through statistical analysis and hypothesis testing, our results show that the proposed approach minimizes the impacts caused by task allocation in software development teams by around 25 % when considering the TF metric during task allocation.

1. Introduction

In agile development with Scrum, the Product Owner (PO) defines the tasks to be done (Rubin, 2012). Software development teams are composed of members with different levels of knowledge and skills. The task allocation among them becomes a critical activity within the process since the project's success depends on how these activities are performed and divided (Bibi et al., 2014). Therefore, we understand that allocating human resources can impact the success or failure of the project.

Task allocation is a complex step since there might be several possible solutions for the same problem in many situations. Many works address task allocation with multiple objectives, but generally, the goals are to improve project cost and completion time. These objectives are fundamental since the project becomes more viable when an efficient team is provided at an acceptable price from a managerial point of view.

However, another way to look at this problem is by using the Truck Factor (TF) metric. This parameter concerns the project's degree of dependence on the team's knowledge within the development environment (Avelino et al., 2016). The ideal scenario occurs when the team's knowledge about the project is as distributed as possible. That allows for better management of the change of project members, maintaining the project at its natural pace of development.

The turnover in software projects is high nowadays, so an even distribution of knowledge about the project is a relevant task. There are analyses about team changes and the influences and factors that cause turnover (Hilton and Begel, 2018; Farooq et al., 2022). Data presented by Pee et al. (2014) shows that in 2012, about 20 % to 28 % of software projects failed for different reasons. However, the team turnover proved a significant risk to the project's continuity. More recently, Mohamad et al. (2021) stated that, in 2018, the IT industry sector had a turnover of 10.9 %, and within the IT sector, software times were the most sought after.

Therefore, understanding that the exit and entry of new members is practically inevitable, it is fundamental to create approaches to reduce the impact of the turnover. A possible step in this direction is to propose an approach that not only analyzes the traditional, widely recognized parameters (cost and time), but also includes the analysis of the Truck Factor. This parameter is a primordial criterion since good efficiency in this project risk factor guarantees work continuity and balanced distribution of knowledge in situations of team member changes.

Multi-Agent Systems (MAS) can address the task allocation problem in different contexts (Singhal and Dahiya, 2015; Krothapalli and Deshmukh, 2002). Designing specific agents in an environment that performs tasks and exchanges information to achieve their goals is possible. Agents can compete, cooperate, or organize themselves based on models to achieve their objectives. Within the same environment, several agents may have different roles and characteristics. MAS allows for the design and building of systems with the same ideas and concepts applied to social relationships and behavior (Abbas et al., 2015).

We use MAS to simulate real environments using agents modeled to represent entities with specific behavior and knowledge. The simulation consists of a set of rules that dictate how the system changes

over time given its current state, and its objective is a better understanding of a given event or even its prediction (Siebers and Aickelin., 2008).

As discussed by Bibi et al. (2014), task allocation is an NP-hard problem with no exact solution. Thus, Search-Based Software Engineering (SBSE) appears, where its term originated from Harman and Jones (2001). Its principle is to apply search-based optimization algorithms to software engineering problems. SBSE uses search algorithms in Software Engineering problems to minimize or maximize parameters. Over the years, several works have been solving task allocation problems through SBSE. The most common techniques are: Simulated Annealing (SA), Ant Colony (ACO), Genetic Algorithm (GA), Integer Programming (IP), Hill Climbing (HC), and Neural Networks (NN) (Räihä, 2010). Here, we can highlight genetic algorithms as a technique widely used in SBSE research in different contexts. It provides an optimal solution, which researchers can apply to solve complex problems.

Task allocation is highly relevant since the principle of allocating resources in software teams arises from small development teams to large companies, where dozens of groups perform tasks daily. Furthermore, designing and building an Agent-Based Simulation (ABS) and SBSE can contribute to this research area. Through simulation, it becomes feasible to understand and analyze the impacts caused by the TF in the distribution of tasks in different software teams.

The main contribution of the present work is an ABS architecture to solve task allocation in software development teams, focusing on optimizing the TF and using an SBSE-based algorithm. In the simulation, agents must represent these teams' roles and characteristics. Other contributions include defining scenarios for the development teams and determining how the task division process occurs. Then we test and analyze the behavior of the proposed architecture in these specific scenarios.

We organize the remainder of the paper as follows. In Section 2, we present the main concepts for understanding this work. In Section 3, we discuss the modeling and implementation of our approach. Section 4 presents the results obtained through the simulation. Finally, we draw conclusions and outline future work in Section 5.

2. Background

2.1. Agile Development

Agents must exchange information, assign tasks, and assume different roles to optimize the task allocation process. Therefore, it becomes necessary to use a development methodology that agents must follow to achieve their goals. This method is founded on the idea that agents have different skills and roles within the development process. In software engineering, there are several methods and techniques for developing systems.

The industry widely uses Agile methods for managing software teams due to their simplicity and flexibility. As seen in Sommerville (2011), these methods allow the development team to work on the software itself and not on its design and documentation, and their basis is an incremental approach to software specification, development, and delivery.

Among the agile methods, Scrum stands out due to the simplicity and flexibility of its workflow. Scrum's focus is on continuous and incremental system delivery. In Scrum, there are roles, tasks, and artifacts. The fundamental roles are as follows: Scrum Master, responsible for managing the team and assisting everyone involved in the Scrum process, principles, and practices; Product Owner, the actor in the process who controls the software product requirements. They are solely responsible for

deciding what features and functionalities compose the software and the priority for their delivery. The Developer Team consists of multidisciplinary professionals.

2.2. Truck Factor

Over time, teams can undergo modifications. Changes in software teams can lead to project instability and loss of performance, given that changes can cause fluctuations in the continuous delivery of the product. In addition to performance, changes can affect delivery time, cost, or even end the project.

Changing the team is practically inevitable, as there is no way to guarantee that the same people will always be available to carry out the tasks. Thus, it becomes necessary to analyze and understand the impacts and risks that such changes can cause.

Truck Factor, also widely known as Bus Factor, is a measure of risk associated with project development. The risk stems from the omission of information about the project among its members. In this paper, we prefer to use the term Truck Factor (TF). Ferreira et al. (2017) defines TF as the number of people a Truck hits and how that event affects the project. The individuals hit by the truck represent the team members who left the project. This area of research recognizes that when technical knowledge about the project is concentrated in a subset of team members, they form knowledge silos, which can hinder project development through delays in task delivery, potentially leading to closure.

Since this problem can arise in any software development team, project managers must be aware of it and should ensure that knowledge about the project is spread among all team members. In this way, it is possible to reduce the impact on the team when new members leave or arrive.

The approach proposed by Avelino et al. (2016) presents a greedy approach for calculating TF. They attribute a DOA (degree of authorship) to each developer in the project who has made at least one commit in the analyzed file. The DOA has three parameters: 1. First author, which differentiates who initially created the file, 2. The number of changes made to the file, and 3. The number of changes made to the file by the rest of the team. Once all DOA values are known, a normalization process is performed, 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 assumed that the project has serious risks of delays, and may even be discontinued, if the coverage of the set of authors is below 50 %. The result of the Truck Factor produced shows the number of authors that can be removed from the project until the coverage is at least 50 %.

An implementation through pseudo-code is presented in Algorithm 1. The algorithm has 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 a value of 0. A loop of repetition iterates over the system files, checking its coverage at each cycle, *getCoverage(F,A)*, which verifies the number of authors present by the number of files in the repository. If the coverage value is less than 0.5, the algorithm is closed, returning the current Truck Factor value. Otherwise, the principal author is removed *removeTopAutor(A)*, and +1 is added to the Truck Factor value.

2.3. Agent-Based Simulation

According to Wooldridge (2009), the agent is a computational entity located in an environment where it can perform autonomous actions to achieve its goals. In many scenarios, a single agent is not enough to solve a specific problem, so it becomes necessary for more than one agent to act in the environment to solve this problem. In this context, multiple agents are employed well. MAS is an extension of autonomous intelligent agents that act in the environment to achieve their goals. Cooperation,

Algorithm 1: Truck Factor Algorithm

Input: List of authors' files A
Output: System truck factor
 $F \leftarrow \text{getSystemFiles}(A);$
 $tf \leftarrow 0;$
while $A \neq \emptyset$ **do**
 $\text{coverage} \leftarrow \text{getCoverage}(F, A);$
 if $\text{coverage} < 0.5$ **then**
 $\text{break};$
 end
 $A \leftarrow \text{removeTopAuthor}(A);$
 $tf++;$
end
return $tf;$

competition, knowledge sharing, and coordination are examples of ways agents can perform to achieve their desired states (Balaji and Srinivasan, 2010).

Agent-Based Simulations (ABS) are computational models of agents that interact autonomously and, through behaviors and actions, simulate a given environment to analyze and solve problems. Many simulation platforms rely on libraries or tools that describe their paradigm and concepts. This model has been accepted in research from the most varied fields of study due to its easy application to problems that other computational models cannot solve (Railsback et al., 2006).

ABS is widely used in research in the area of MAS (Hongqiao et al., 2009; Chae et al., 2015; Baia, 2015), as it can represent many variables present in real environments in a simplified and artificial way. The benefit is that the process can be quickly reproduced in different conditions, thus allowing for a precise analysis of the studied problem.

2.4. Search-Based Software Engineering

SBSE seeks to solve Software Engineering problems associated with optimization problems in Search-Based Optimization (SBO) (Harman et al., 2012). Several areas of Software Engineering (SE) use SBSE, such as Requirements Engineering, Testing, Debugging, Bugs, Project Management, Resource Allocation, etc.

Harman and Jones (2001) initially approached the term SBSE by proposing using search techniques to solve software engineering problems. However, Miller and Spooner (1976) addressed this purpose already, where the authors sought to solve the problem of generating automated tests with optimization techniques in his research.

According to Harman et al. (2009), the issues addressed in SE are problems that an optimization-based solution can often address. Often, the project's Software Engineer may wonder about some of the needs of the team management, such as:

- What is the best resource allocation for a project?
- What is the optimal task execution order to minimize delays and bottlenecks?
- How to minimize dependency conflicts between tasks assigned to different team members?

These issues above are optimization issues. Therefore, SBSE has demonstrated great potential in finding solutions for constraint minimization or maximization.

One approach applied in SBSE is Genetic Algorithms (GA). This approach seeks to find an optimal solution for a given problem. We define an environment as a population of individuals with different characteristics. Each individual is a possible solution to the problem, and over time, only the best individuals in this population should keep reproducing and sharing their genes. The algorithm evaluates the individual and selects the best one for reproduction according to their status. The GA reproduces only the best individuals, generating a more evolved population, closer to the solution. The idea is that only some individuals can get closer to a solution that is considered optimal, after a specific number of evolutionary operations.

Some steps are common in running the genetic algorithm. As seen in Algorithm 2, the process starts by generating an initial population, often randomly. The GA evaluates each individual. This process must verify how close the individual is to an ideal solution. The evaluation is directly linked to the problem since the individual assessment impacts the selection. The selection phase should choose the regarded best individuals with a high evaluation function value. After selecting the individuals with the best characteristics, they must undergo crossover and mutation processes. The algorithm takes pairs of individuals (parents) to cross their genes to generate new individuals during the crossover process. As for the mutation process, the GA randomly changes some genes in individuals. The mutation makes the population more heterogeneous, ensuring more possibilities for finding reasonable solutions.

Algorithm 2: Genetic Algorithm
Result: Best Solution Compute initial fitness function; while <i>didn't meet the stop criteria</i> do Selection best individuals (Parents); Crossover; Mutation; Compute fitness; end

3. Materials and Methods

We used the NetLogo platform to solve the problem. Simulation through NetLogo makes the research validation process more viable, as, in addition to being more agile in the development of the environment, in the simulation, it is possible to use only the variables necessary to validate the process.

Figure 1 presents our approach, which involves using genetic and multi-agent algorithms to solve the problem. As shown in the UML activity diagram, we divided the model into User interaction, NetLogo, and Genetic Algorithm.

- **User interaction:** This is when the user must define the settings for the simulation. The user can set variables such as the number of tasks, available workers, and skills. Monitors are also available for the user to have information about the simulation.

- **NetLogo:** It is responsible for the actual simulation process. The simulation starts according to the user's preferences defined during the interaction. Every Scrum-related process must be executed in NetLogo, including the sprint's opening and closing. The Product Owner Agent (PO) distributes activities. However, currently, the PO only links the tasks to the available workers.
- **Genetic Algorithm:** It is responsible for the optimization process. Currently, the algorithm's input will be the sprint's tasks and the available workers. The Netlogo sends that information to the Python module. The genetic algorithm must carry out the steps of generating a population, evaluating individuals, selecting, crossing over, and mutating. At the end of the process, the Python module returns the best-found solution to the NetLogo Module.

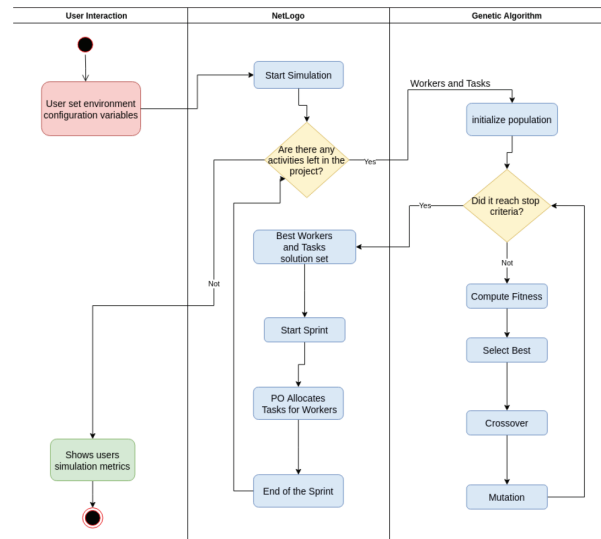


Figure 1. Overview of activity diagram

We create the environment, which must include the parameters and variables present in the actual context of a software team. We have built the environment based on the Scrum methodology, which addresses their roles, management, and development methods.

The environment has three agents: the PO agent, the worker agents, and the tasks. Each type of agent has its attributes. These variables serve to maintain each agent's perception of the environment.

In Figure 2, it is possible to visualize the attributes and procedures of each agent. NetLogo obligatorily assigns all agents an identification variable (Id) to control the simulation.

The attribute “is working” from the Worker agent says if the agent is performing a task. The attribute “current task” stores the agent's link with the current task. The attribute “skill level” is an array that stores the agent's skills, and “knowledge level” indicates the project's knowledge level in each aspect of the agent's skill. In addition, the “run task” and “set task as done” procedures indicate task execution and completion, respectively.

At first, the PO agent has state attributes, but its primary function is to execute procedures. These are to start and end the sprint through the “start sprint” and “close sprint” functions. In the “start sprint”

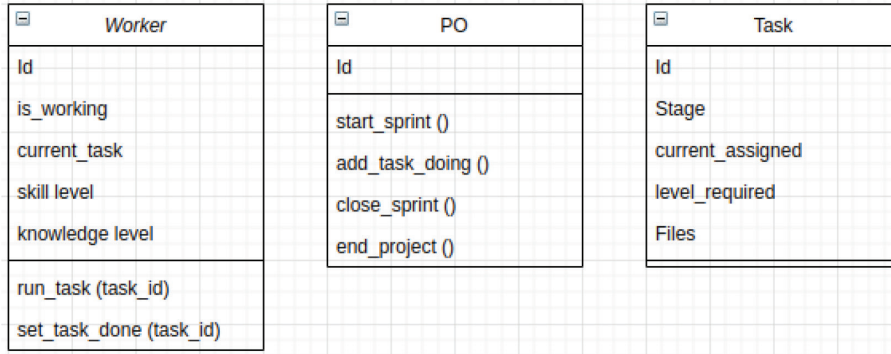


Figure 2. Agent attributes and actions

function, tasks are removed from the Backlog and included in the sprint. In the “add task doing” procedure, it is the moment when the agent must associate a task with the worker agent. Finally, it is also this agent’s function to finish the project through the “end project” function.

The Task agent has the “level_required” attribute, which informs the Worker agent’s necessary level to complete the task. The “stage” attribute keeps track of the task’s status, and the PO needs it to take note of the tasks that are running, finished, and not started yet. The “current assigned” attribute stores the link to the agent developing the task. Finally, “Files” refers to the files in the software repository that the activities will operate on.

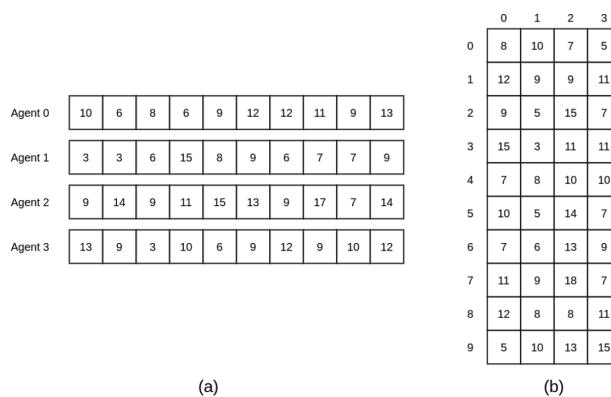
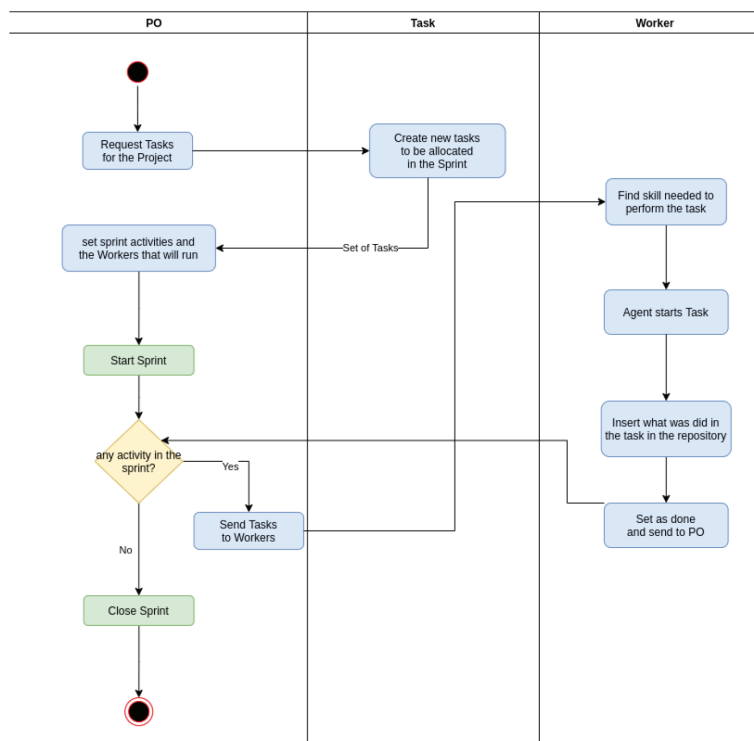
In Figure 3, the activity diagram presents the interactions between agents in the environment. The flow starts with the agent PO asking the agent Task to create new activities for the sprint. Currently, the Task must generate and assign dynamic values to its attributes. After this step, the PO must have enough activities to start the sprint planning, associating the Tasks with the Workers. This association step will be the PO’s main point to minimize the TF’s effects. Finally, the PO with an optimal solution proposed by the GA can already start the sprint. This last process iterates until there are no more pending activities, and the sprint must close.

Figure 4(a) presents the knowledge array of each agent. In the illustrated situation, there are four agents, and each one of them has ten skills. This attribute must store all the knowledge the agent must acquire in different aspects of the project.

When the simulation begins, each agent already starts with prior knowledge about the necessary characteristics of the project. The simulation randomly assigns the values to the agent’s internal knowledge variable. This mechanism is essential, as we understand that the agents that make up a development team already have the necessary knowledge about the technologies involved in the project.

Each position in the knowledge array represents the agent’s skill score. In the illustrated situation, *agent 1* has a value of *15* in the project skill 3. Still, the skills *0* and *1* have only *3*. When only these skills are analyzed, we understand an imbalance in the agent’s knowledge of the project.

Figure 4(b) presents the repository matrix. We define that matrix as a global system variable. This attribute is essential to find the project’s Truck Factor since it stores the system file mapping and the modification percentage for these files of each agent. Columns must represent the agents present in the environment, and each line must represent project files. Therefore, the values inserted in the matrix must represent the agents’ contributions in the project files.



The simulation inserts the activity artifacts into each team task’s “repository” attribute. Through this attribute, it is possible to identify the contribution of each agent in the project. The idea is to simulate a source code repository. This attribute is fundamental in this proposal since the algorithm developed by (Avelino et al., 2016) (Algorithm 1) needs this parameter to find the Truck Factor.

4. Results

Figure 5 shows the simulation environment implementation. We can highlight the worker agents (blue characters), the PO agent (yellow characters), and the tasks (white cards). The line represents agents performing tasks.

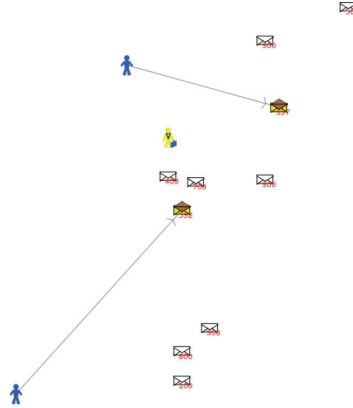


Figure 5. Interaction between the agent and the environment in the simulation

We have used NetLogo to build the environment. NetLogo is a simple but powerful tool that allows for the programming and modeling of multi-agent simulations. In this approach, the Netlogo module, responsible for building the environment and agents and carrying out the experiments, is separated from the Genetic Algorithm (GA), which only has the objective of task allocation.

Our approach uses a genetic algorithm to optimize the Truck Factor parameter. However, due to the lack of other scenarios or performance data from software teams for comparison, we implemented two different approaches to task distribution. In the “Optimistic” strategy, the simulation allocates tasks to the agent who has more knowledge about the requirements needed to complete the task. The other approach is Random, in which the tasks are randomly distributed without intentional criteria.

A large number of tests were performed to define the GA parameters. The test is intended to understand whether the fitness function is being achieved. Tests were carried out with different scenarios and with agents with different characteristics. We present these parameters in Table 1. For the applied scenarios, this configuration proved to be satisfactory.

Table 1. GA parameters

Parameter	Value
Cross Validation	0.1
Mutation	0.1
Generations	500
Population	1000

Individuals using the GA approach can be considered a possible solution. For this, the aptitude function evaluates it. We define this metric as the contribution made by the agent to the software repository. Algorithm 3 presents the evaluation function, whereas the input is the individual it will evaluate. The individual is a set of workers and their tasks. The algorithm iterates over all tasks and adds them to a repository, and at the end, it returns the repository variance.

Algorithm 3: Fitness Function Algorithm
Input: Individual Output: Individual Variance for $i \leftarrow 1$ to $\text{len}(\text{individual})$ do $\text{task} = \text{individual}[i];$ $\text{repository.add}(\text{task});$ end return $\text{variance}(\text{repository});$

The iterative cycle of the sprint starts with the agents taking tasks from the project backlog and finishing when they complete all these tasks. This cycle repeats until the project is completed. In each sprint, the simulation sends the set of tasks and agents as proposed, then the genetic algorithm analyzes and provides a possible solution.

Algorithm 1 is the metric used to compare the different implemented approaches. As presented in section 2.2, this metric seeks to define the number of people who hold knowledge about the project. For such a result, it uses the software repository as input to see the value of the Truck Factor. Since this research approach minimizes the impacts caused by the TF, the simulation seeks to maximize the TF metric.

The TF calculation result for each completed sprint generates the data for the simulation. That is because there is a need to analyze its behavior throughout the development process. At the end of each completed sprint, the simulation stores the TF calculated in a file.

The value of the TF is an integer, which must represent the number of people who, if they leave the project, compromise its development. The simulation verifies knowledge coverage at each iteration through the “*getCoverage*” method. It analyzes whether the number of people in the project is greater than 50 % of the contribution made in the repository. If the knowledge exceeds 50 %, it removes the principal author and increments TF by 1. When such a value is no longer valid, it must return the TF.

The algorithm defines the DOA for each file in the repository. In the approach developed by Avelino et al. (2016), three parameters are used to determine the DOA related to the project’s versioning system. Our approach uses a simulated environment to build the experiment. The simulation calculates the DOA for each file, detecting the agent that contributed the most and the agent with the most significant ownership.

4.1. Empirical Study

In this experiment, we sought to show the GA approach’s efficiency for task allocation, maximizing the TF variable. The experiment developed distinct software project scenarios resembling an absolute project configuration. Table 2 shows the experiment scenarios. We executed each experiment scenario’s GA, Random, and Optimistic strategies. All code and data used in this work can be found at GitHub: <https://github.com/caetanonetosegundo/resource-allocation-simulation>.

Table 2. Setting up simulation scenarios

	workers	tasks	sprints	repository
Simulation 1	2	20	12	60
Simulation 2	6	30	30	100
Simulation 3	10	60	50	150
Simulation 4	15	85	50	150
Simulation 5	20	100	52	150
Simulation 6	25	100	52	300

4.2. Truck Factor Analysis

Initially, a descriptive statistical analysis of the generated data was conducted. The simulation results can be analyzed through the graphical representation and the table with the described data.

In Figure 6, the values of the Truck Factor throughout the project are represented through a line graph. The figure presents the union of the five evaluated scenarios, as shown in Table 2. The performance of each of the developed approaches is presented in all graphs.

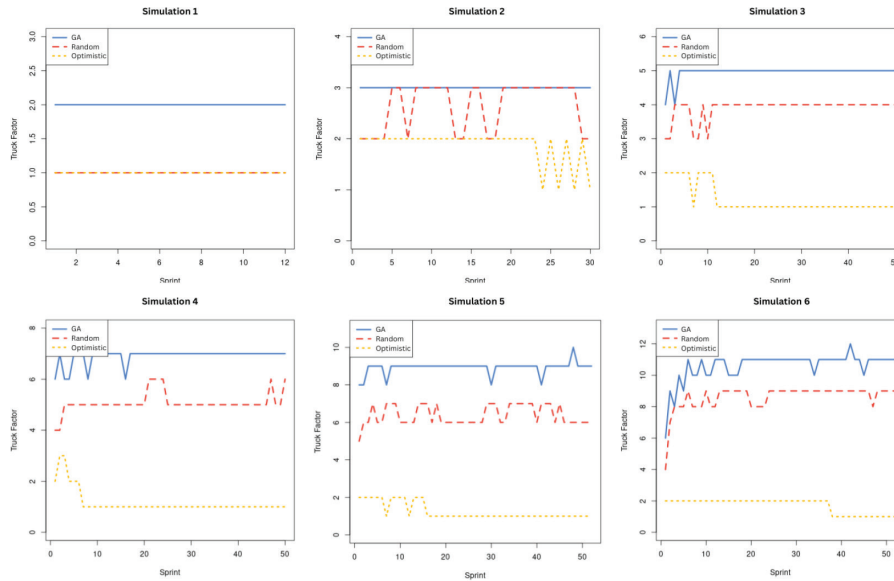


Figure 6. Truck factor line chart

It is possible to identify that the Optimistic approach performed below the Random and GA approaches in all the scenarios applied. The principle of task distribution can explain this fact. It assigns tasks to the agent with greater technical performance than the required task. Therefore, it is possible to conclude that such an approach creates silos of knowledge among project agents, and thus, knowledge about the project is concentrated only in a small subset of people.

The two approaches that obtained the closest results were Random and GA. However, it is possible to notice an improvement in the performance of GA compared to Random since, in all the studied scenarios, the values obtained by Truck Factor at the end of the project are higher. That means that GA was able to better distribute knowledge among the team members in all the analyzed scenarios.

When only the TF evaluation metric was observed, the performance of Random strategy was equal to GA at certain moments of the project, as we can observe in simulations 2, 3, and 6, which appear in Figure 6. This situation occurred in scenarios where the project had a low number of people (scenario 2), or in the initial phases of the project (scenarios 3 and 6).

Table 3 presents a summary of the comparison data of the approaches analyzed in all applied scenarios, using the Truck Factor metric. Through these data, it is possible to visualize more specifically that the GA approach obtained superior performance in the scenarios analyzed compared to Random and Optimistic. As the scenario becomes more complex, whether in the number of tasks and workers or the repository size, the GA approach obtains a higher Truck Factor value than the Random approach.

Table 3. Truck Factor exploratory analysis

Simulation	Approach	Mean	Median	Mode	Standard Deviation
Simulation 1	GA	2	2	2	0
	Random	1	1	1	0
	Optimistic	1	1	1	0
Simulation 2	GA	3	3	3	0.4901
	Random	2.63	3	3	0.3457
	Optimistic	1.87	2	2	0.1979
Simulation 3	GA	4.96	5	5	0.1979
	Random	3.9	4	4	0.303
	Optimistic	1.2	1	1	0.4041
Simulation 4	GA	6.9	7	7	0.303
	Random	5.08	5	5	0.303
	Optimistic	1.16	1	1	0.4041
Simulation 5	GA	8.92	9	9	0.3341
	Random	6.38	6	6	0.5297
	Optimistic	1.25	1	1	0.4372
Simulation 6	GA	10.6	11	11	0.9334
	Random	8.62	9	9	0.8322
	Optimistic	1.71	2	2	0.4575

The histogram allows us to observe the frequency of the result of the Truck Factor throughout the project, as seen in Figure 7. We can identify that the frequency in the “Optimistic” approach remains practically constant but has a low TF value, which we understand to be unsatisfactory. In Figure 7, at the Optimistic approach, we can see that in Simulations 1, 4, and 5, most of the frequency is repeated

at the lowest value of the Truck Factor. Unlike what is observed in Simulations 2, 3, and 6, where there is still an unsatisfactory Truck Factor and the frequency is the maximum that approach can achieve.

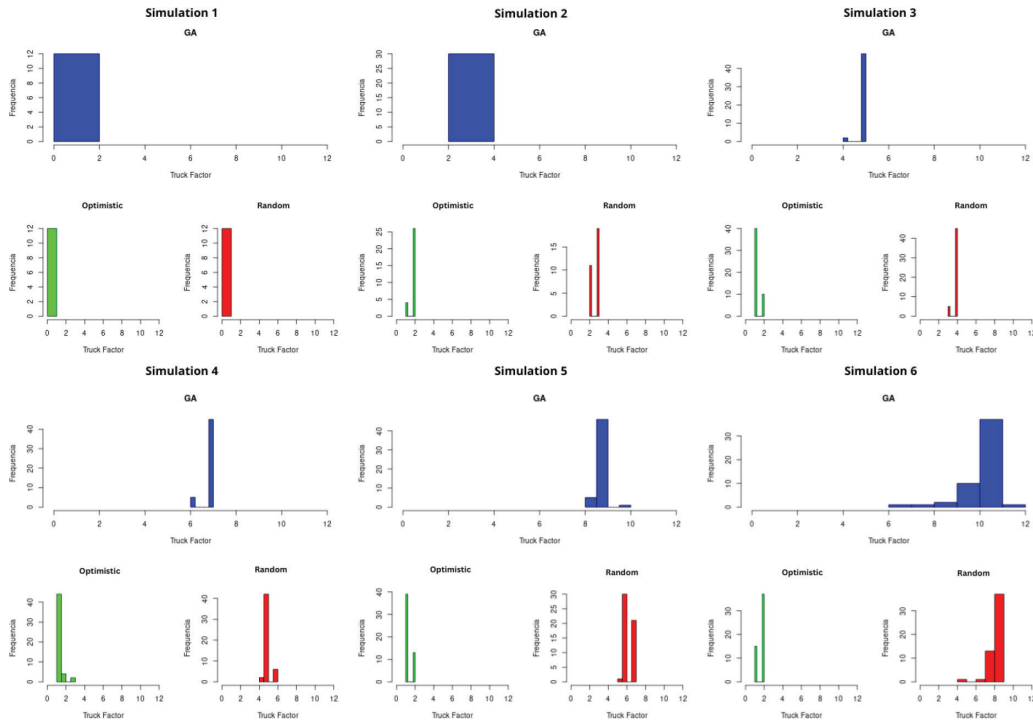


Figure 7. Truck Factor Histogram

We can identify that the GA value remains constant in small scenarios (1 and 2), and there is a slight variation in the frequency in intermediate scenarios (3 and 4). GA scored well in both cases because most of it is above the others' approaches. In larger scenarios (5 and 6), there is an increasing curve in the frequency of the TF value until it reaches its maximum value. Later, it exceeds this value but cannot maintain it and returns to the previous value of higher frequency. Random obtained results similar to GA, but we noticed that in scenarios 2 and 5, the frequency of TF of Random was inconsistent, unlike GA, which remained stable and had a higher TF.

4.2.1. Normality Test

The descriptive statistics analysis shows that the GA approach performed better when compared to Random and Optimistic. To better understand the result obtained by Truck Factor and complement the analysis, a study based on statistical tests has been carried out. In these statistical tests, only the Truck Factor variable has been analyzed since it is the primary metric investigated in this research.

First, we need to understand the data resulting from the Truck Factor. For this, we performed the tests by Kolmogorov-Smirnov (2008) and Shapiro and Wilk (1965). The tests were performed on the variables collected from each sprint during the project's progress in each simulated scenario. The two tests are intended to show whether the results represent a normal distribution.

In the Kolmogorov test, the null hypothesis (H^0) is that the sample follows a normal distribution. The alternative hypothesis says that it is different from a normal distribution. To show that it follows a normal distribution, we have $p > \alpha$, where p is the probability obtained from the sample and α shows the significance of the risk of the sample not following a normal distribution. We verify the value recommended by the literature, $\alpha = 0.05$ (5%).

In Table 4, the results obtained by Kolmogorov in all the analyzed scenarios are presented. In the column (p-value), we can see that the values of p in all the scenarios presented are less than $p < \alpha$, which makes us reject the null hypothesis (H^0) that the sample follows a normal distribution, so we can infer with 95% confidence that the data do not follow a normal distribution.

Table 4. Kolmogorov Smirnov Test

Simulation	Approach	D	P-Value
Simulation 1	GA	0.53025	0.002346
	Random		
	Optimistic		
Simulation 2	GA	0.5391	5.344e-08
	Random	0.40613	0.0001007
	Optimistic	0.51678	2.197e-07
Simulation 3	GA	0.54007	4.302e-13
	Random	0.52929	1.362e-12
	Optimistic	0.48969	7.705e-11
Simulation 4	GA	0.52929	1.362e-12
	Random	0.46007	1.284e-09
	Optimistic	0.51385	6.819e-12
Simulation 5	GA	0.4949	1.733e-11
	Random	0.36228	2.361e-06
	Optimistic	0.46626	3.033e-10
Simulation 6	GA	0.39799	1.403e-07
	Random	0.39198	2.297e-07
	Optimistic	0.44738	1.825e-09

We can highlight the result of simulation 1, in which the different approaches obtained the same P-value results. That was because the team scenario was minimal, and the values obtained by Truck Factor did not vary during the project. That can be noticed when presenting Table 3, which shows a standard deviation 0 in this scenario.

The Shapiro test is similar in purpose to the Kolmogorov test, where its result measures how well the data fit a normal distribution. The test has a null hypothesis (H^0) that the population belongs to a normal distribution and an alternative hypothesis that it does not follow a normal distribution, where $p \leq \alpha$, we reject H^0 and $p > \alpha$, we cannot reject H^0 . The significance level $\alpha = 0.05$ (5%) was also used for this test.

The results of the Shapiro test are presented in Table 5. We can observe that in scenario 1, there was no output. That was because the Truck Factor values did not vary during the execution of the project. As previously mentioned, this situation occurred due to the relatively small scenario. However, observing the following scenarios, we can observe that the p-value was lower than our α in all the analyzed scenarios. Thus, we reject H^0 , which states that the data follow a normal distribution.

Table 5. Shapiro Test

Simulation	Approach	W	P-Value
Simulation 1	GA	-	-
	Random	-	-
	Optimistic	-	-
Simulation 2	GA	0.17962	7.766e-12
	Random	0.61191	1.023e-07
	Optimistic	0.4044	5.981e-10
Simulation 3	GA	0.19816	4.921e-15
	Random	0.34386	1.3e-13
	Optimistic	0.49037	6.231e-12
Simulation 4	GA	0.34386	1.3e-13
	Random	0.5385	2.634e-11
	Optimistic	0.38579	3.674e-13
Simulation 5	GA	0.43769	7.927e-13
	Random	0.68868	3.256e-09
	Optimistic	0.53884	1.538e-11
Simulation 6	GA	0.57799	5.473e-11
	Random	0.49223	3.726e-12
	Optimistic	0.56823	3.959e-11

Both statistical tests were applied in this research to complement what was already hypothesized, that the data obtained do not follow a normal distribution. Therefore, from the presented results, we can conclude with a confidence of 95% that there is enough evidence to reject the hypothesis that the variables of the project's Truck Factor follow a normal distribution. On the basis of the data distribution analysis, we can perform statistical tests.

4.2.2. Hypothesis Test

Hypothesis tests provide evidence that allows us to accept or reject a statistical hypothesis based on the evidence provided by the sample. Therefore, we aimed to investigate the hypothesis that GA has a superior result to the Random approach regarding the distribution of tasks under the Truck Factor metric. Once a hypothesis is proven, it is possible to statistically prove the results with a certain degree of confidence.

The result of the Wilcoxon Test (Rey and Neuhäuser, 2011) has allowed us to determine whether the differences between the medians of the three approaches developed are statistically significant. Also, in this test, we used a significance level of $\alpha=0.05$ (5%). The result obtained in Wilcoxon is defined as $p \leq \alpha$, then it is concluded that the medians are significantly different, rejecting the null hypothesis (H^0). If $p > \alpha$, it is concluded that the medians are not significantly different, so we cannot discard H^0 .

The test compared only the GA and Random approaches. Since they were based on descriptive statistics, the approaches obtained approximate results. The result was generated from 3 different alternative hypotheses, which compared whether the GA is greater, smaller, or different from the random approach.

In Table 6, the results obtained by the Wilcoxon test are presented, where it is noted that in all the analyzed scenarios, the alternative hypothesis of $GA > Random$ results in a $p < \alpha$, which makes us reject the null hypothesis. In the $GA \neq Random$ Hypothesis, we rejected the null hypothesis in all scenarios. However, the hypothesis of $GA < Random$ where $p > \alpha$, so we should not reject the hypothesis H^0 .

Table 6. WilCox test for GA and Random approach

Simulation	Alternative	W	P-Value
Simulation 1	Greater	132	3.697e-05
	Less		1
	Two Sided		7.394e-05
Simulation 2	Greater	600	0.0007118
	Less		0.9993
	Two Sided		0.001424
Simulation 3	Greater	2455	2.2e-16
	Less		1
	Two Sided		2.2e-16
Simulation 4	Greater	2485	2.2e-16
	Less		1
	Two Sided		2.2e-16
Simulation 5	Greater	2704	2.2e-16
	Less		1
	Two Sided		2.2e-16
Simulation 6	Greater	2572.5	2.2e-16
	Less		1
	Two Sided		2.2e-16

Overall, the statement suggests that the presented data supports the claim that the Truck Factor, when used in the genetic algorithm approach, tends to yield better or superior median results than the random approach, with a confidence level of 95 %.

5. Conclusion and Future Work

Software project execution is a complex process that includes steps that must be followed until its conclusion. From the software management point of view, it is necessary to build a plan to map all the required steps. Among the main planning activities that need to be carried out is the allocation of tasks, which is a complex process due to the different variables present in the environment and their restrictions. The optimal distribution of activities offers project guarantees such as cost reduction, on-time delivery, and delegates tasks to the right people.

During the execution of the project, different problems can arise, presenting themselves as a threat to the project plan. Among these problems is the concentration of knowledge. From this, Truck Factor aims to measure these knowledge silos among team members. Through this metric, one can understand how knowledge about the project is distributed among team members, directly impacting project delivery performance, and can also threaten its discontinuity. Seeking to manage such a metric is relevant to the turnover of team members, which has become common due to the increasing demand for software in the last few years.

Since Truck Factor (TF) is a suitable metric for analyzing knowledge distribution, it was necessary to study methods to optimize the distribution based on TF. SBSE techniques can be applied to a wide range of algorithms to find an optimal solution to a complex problem in software development. From another point of view, the development environment is configured as a multi-agent environment, since each team member can be modeled as an agent with different characteristics. Thus, in this research, a study was carried out on the distribution of tasks for the agents in order to minimize the impacts caused by the Truck Factor, based on the approach of SBSE techniques.

Among the algorithms used in SBSE, this research used the implementation of genetic algorithms that aim to provide an optimal solution for the distribution of tasks. Since we seek to maximize the distribution of project knowledge among team members, the genetic algorithm's fitness function was based on the agent's contribution to the repository and the result provided by the Truck Factor metric.

The NetLogo simulation environment provided the necessary means to represent real scenarios, allowing for the definition of various setups. In the task allocation process, different agents were implemented in the environment, with pre-established characteristics and roles. Although the multi-agent environment provides an internal function in the agent for decision-making, in this research, the distribution of tasks was left to a centralizing agent, the Product Owner. The simulation environment allowed us to validate the approach implemented in this research. Since the process is being executed in the virtual environment, it becomes feasible to reproduce and analyze the environment in different contexts and collect data.

The presented results showed that the approach developed based on the genetic algorithm was successful in the distribution of tasks, where it was possible to notice that although the tasks were generated during the construction of the project, with different sizes and characteristics, the developed approach made the Truck Factor remain at a level considered satisfactory. Random and optimistic strategies were implemented to compare their results with our proposal. The results point to the best performance of our approach compared to the other two.

The developed tool is intended to help project management, monitor the risk of the Truck Factor inherent to the project during its development, and help the manager allocate tasks, focusing on the distribution of knowledge.

In future work, we intend to build a tool with a friendly graphical interface, so that end users can use the application. Another important point is the implementation, in addition to GA, of new

algorithms to be used in task allocation. Finally, we intend to propose another Truck Factor calculation to improve its metric.

References

- Abbas, H. A., Shaheen, S. I., & Amin, M. H. (2015). Organization of multi-agent systems: An overview. *arXiv preprint arXiv:1506.09032*. <https://doi.org/10.48550/arXiv.1506.09032>
- Avelino, G., Passos, L., Hora, A., & Valente, M. T. (2016). A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)* (pp. 1–10). IEEE.
- Baia, D. D. M. (2015, May). An integrated multi-agent-based simulation approach to support software project management. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (Vol. 2, pp. 911–914). IEEE.
- Balaji, P. G., & Srinivasan, D. (2010). An introduction to multi-agent systems. In *Innovations in multi-agent systems and applications-1* (pp. 1–27). Springer. https://doi.org/10.1007/978-3-642-14435-6_1
- Bibi, N., Ahsan, A., & Anwar, Z. (2014). Project resource allocation optimization using search-based software engineering—A framework. In *Ninth International Conference on Digital Information Management (ICDIM 2014)* (pp. 226–229). IEEE. <https://doi.org/10.1109/ICDIM.2014.6991431>
- Chae, S. W., Seo, Y. W., & Lee, K. C. (2015). Task difficulty and team diversity on team creativity: Multi-agent simulation approach. *Computers in Human Behavior*, 42, 83–92.
- Farooq, H., Janjua, U. I., Madni, T. M., Waheed, A., Zareei, M., & Alanazi, F. (2022). Identification and analysis of factors influencing turnover intention of Pakistan IT professionals: An empirical study. *IEEE Access*.
- Ferreira, M., Valente, M. T., & Ferreira, K. (2017). A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)* (pp. 207–217). IEEE.
- Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology*, 43(14), 833–839. [https://doi.org/10.1016/S0950-5849\(01\)00189-6](https://doi.org/10.1016/S0950-5849(01)00189-6)
- Harman, M., Mansouri, S. A., & Zhang, Y. (2009). Search-based software engineering: A comprehensive analysis and review of trends, techniques, and applications. *Department of Computer Science, King's College London, Tech. Rep. TR-09-03*, 23.
- Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques, and applications. *ACM Computing Surveys*, 45(1), 1–61.
- Hilton, M., & 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* (pp. 191–200).
- Hongqiao, Y., Xihua, L., Fei, W., & Weizi, L. (2009). Multi-agent based modeling and simulation of complex system in hospital. In *2009 16th International Conference on Industrial Engineering and Engineering Management* (pp. 1759–1763). IEEE.
- Kolmogorov–Smirnov Test. (2008). In *Springer Handbook of Statistics* (pp. 283–287). Springer. https://doi.org/10.1007/978-0-387-32833-1_214

- Krothapalli, N. K., & Deshmukh, A. V. (2002). Distributed task allocation in multi-agent systems. In *Proceedings of the Institute of Industrial Engineers Annual Conference*.
- Miller, W., & Spooner, D. L. (1976). Automatic generation of floating-point test data. *IEEE Transactions on Software Engineering*, SE-2(3), 223–226. <https://doi.org/10.1109/TSE.1976.233818>
- Mohamad, M. R., Nasaruddin, F. H., Hamid, S., Bukhari, S., & 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)* (Vol. 1, pp. 1–7). IEEE.
- Pee, L. G., Kankanhalli, A., Tan, G. W., & Tham, G. Z. (2014). Mitigating the impact of member turnover in information systems development projects. *IEEE Transactions on Engineering Management*, 61(4), 702–716.
- Räihä, O. (2010). A survey on search-based software design. *Computer Science Review*, 4(4), 203–249. <https://doi.org/10.1016/j.cosrev.2010.06.001>
- Railsback, S. F., Lytinen, S. L., & Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9), 609–623. <https://doi.org/10.1177/0037549706073695>
- Rey, D., & Neuhäuser, M. (2011). Wilcoxon-Signed-Rank Test. In Lovric, M. (eds) *International Encyclopedia of Statistical Science*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04898-2_616
- Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley. ISBN: 9780137043293.
- Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3–4), 591–611. <https://doi.org/10.1093/biomet/52.3-4.591>
- Siebers, P. O., & Aickelin, U. (2008). Introduction to multi-agent simulation. In *Encyclopedia of Decision Making and Decision Support Technologies* (pp. 554–564). IGI Global.
- Singhal, V., & Dahiya, D. (2015). Distributed task allocation in dynamic multi-agent system. In *International Conference on Computing, Communication & Automation* (pp. 643–648). IEEE. <https://doi.org/10.1109/CCAA.2015.7148452>
- Sommerville, I. (2011). *Engenharia de Software* (3rd ed.). Pearson Prentice Hall. ISBN: 9788579361081.
- Wooldridge, M. (2009). *An introduction to multiagent systems* (2nd ed.). John Wiley & Sons. ISBN: 9780470519462.