# **Simulation engine for a social teamwork game**

This document provides in-depth details of the SuperScript agent-based model of team formation. The version described here is the current stable release ([version 1.0](link%20to%20release%201.0)). A more succinct introduction to the model and how to use it can be found in the README file at: <https://github.com/Superscriptus/SuperScript>

## Motivation

### Model purpose

The purpose of the extended model is to build a simulation engine that serves as the basis for the “social teamwork game”. The model simulates the state of the world and its development over time for workers within an organization that are assembled in teams to work on projects.

The simulation engine will then serve as the underlying model for the world in a Python-Django prototype for a web application (e.g., hosted on Heroku). In that prototype a user (either a worker or an organization) goes through the steps of the user journey and takes part in the social teamwork game.

The model will be used to test a number of hypotheses to understand emergent properties and/or to fine-tune the input parameters, to visualize those results, to generate training data for the Machine Learning prototype and to gather model proof points for further discussion.

### Baseline model

The baseline model is the NetLogo team assembly model developed by [Wilensky](https://ccl.northwestern.edu/netlogo/models/TeamAssembly), based on research by [Guimera et al.](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2128751/) The baseline model represents a business network of co-worker agents from which project teams of various sizes form. Agents in the model have only a few basic characteristics that influence their behavior: whether they are a newcomer or incumbent and what previous connections they have with other agents if they are incumbents. There are three parameters that can be adjusted to influence behavior in the baseline assembly model: the team size, the probability of choosing an incumbent, P, and the probability of choosing a previous collaborator, Q.

## Model overview

### Introduction

For this model, we consider the multi-project environment of a typical organization with multi-skilled workers who differ in their skill levels. The task of project staffing is to compose project teams such that the skills and availabilities of the worker meet the requirements of the respective project. A greater skill level does not reduce the time needed to accomplish a certain amount of workload but increases outcome quality, and in turn, the probability of project success.

In line with the literature, we argue that projects should be accomplished by relatively small teams and that workers should be assigned to a preferably small number of project teams.

A project is defined by its skill requirements (units of skill-by-level), a required creativity level, the project risk, and timing and budgetary constraints. For each project, skill requirements arise in the periods of its execution. Workload does not only originate from projects but also arises within the departments of the organization. Departmental workload must be accomplished in each period of the planning horizon by the workers who belong to the corresponding department. Each worker belongs to exactly one department. Hence, we presume a matrix organization that features functional departments and, potentially, cross-departmental project teams. Our goal is to find an assignment of workers to projects to allocate project workload such that all requirements of projects and departments are satisfied and that the average probability of project success is maximized. The probability of project success is a function of the average rating of the required skills, a skill balance, a creativity match, and a chemistry booster.

We consider an organization that intends to carry out a set of projects within the upcoming planning horizon. The link between the organization and the projects are skills that are mastered by the workers of the organization and that are required by the projects. The organization wants to allocate project workload to its workers such that the average probability of project success is maximized.

### Implementation

The model is implemented in Python 3.6 using the [Mesa](https://mesa.readthedocs.io/en/stable/) agent-based modelling framework. The allocation of near-optimal teams of workers to projects involves a complex numerical optimization task. For this task we use the basin-hopping algorithm from [SciPy.optimize](https://docs.scipy.org/doc/scipy/reference/optimize.html) with [Pathos multiprocessing](https://pathos.readthedocs.io/en/latest/pathos.html) to speed up the optimization by leveraging multi-core architectures. Heuristically we have found that this approach produces significant runtime improvements when using 8 cores/processes, with diminishing returns for larger numbers of cores.

### Main concepts

#### Workers

Workers are the agents of the model. They can: work on projects; contribute to departmental workload; train their skills and be replaced by new workers if they are inactive for too long. The SuperScript [worker class](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/worker.py) is derived from the [Mesa.agent](https://github.com/projectmesa/mesa/blob/main/mesa/agent.py) class.

#### Departments

Each worker belongs to a [department](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/organisation.py). In general, the organization consists of D departments and there is a baseline departmental workload that is required in order to keep the department running. This workload must be met by the department’s workers and acts as a constraint on the capacity of the workers to contribute to projects.

#### Projects

A certain number of [projects](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/project.py) are created on each timestep of the simulation. Project creation is either done at random, or by loading pre-defined projects (for repeatability). Each project has a specific combination of skill requirements, along with the following attributes that characterize the project: *risk, required creativity, budget.*

#### Team Allocator

A [TeamAllocator](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/organisation.py) class assembles workers into teams to work on projects by trying to find the best team of workers according to the project requirements. The team allocation uses predefined strategies (e.g. *Random*, *Basic, Basin*) which can be selected by the user. The general aim in team allocation is to maximize the probability of project success. However, this is a computationally hard problem because of the combinatorically large number of possible teams (there are 8.25 x 1012 teams of 5 that can be selected from 1000 workers). Therefore, the various strategies that have been implemented attempt to simplify this problem in different ways.

#### Probability of success

The probability of project success is the key objective function that is optimized during team allocation. The probability is a function of the average rating of the required skills, a skill balance, a creativity match, and a chemistry booster. These components are defined in section 3.3.

#### Trainer

Low-skilled workers are trained to improve their skill levels. Training is blocking meaning that workers cannot work on projects or contributes to departmental workload while they are on training. The training process is handled by the [Trainer](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/organisation.py) class, which can operate on two different modes. The default mode aims to keep a fixed fraction of the workforce in training at any given time.

### Main dynamics

#### Initialization

When the [model](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/model.py) is constructed the following initialization steps are taken:

* The main model parameters are loaded from the [config file](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/config.py).
* A project inventory is instantiated, which handles project creation and tracks project status. It also contains the team allocator which is used to allocate teams on project creation.
* The workforce is created by generating the required number of workers and randomly assigning their skill levels. On creation, each worker is assigned at random to a department such that there are equal numbers of workers belonging to each department.
* The [social network](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/network.py) is instantiated, which tracks the number of successful collaborations between each pair of workers.
* The [Mesa scheduler](https://github.com/projectmesa/mesa/blob/main/mesa/time.py) is instantiated, which is responsible for updating all of the agents (workers) on each timestep by calling their *step()* method. We use the *RandomActivation* schedule such that the workers are updated in a random order on each timestep.
* The trainer is instantiated, which handles all training of workers throughout the simulation.
* The data collector ([SSDataCollector](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/tracking.py)) is instantiated, which tracks various model-, agent- and project-level variables throughout the course of the simulation. These tracked variables are used by the Mesa server for visualization of the simulation in real time, or can be used to access data at the end of the simulation for plotting of saving to disk.

#### Running

Once the model has been initialized it can be run for a set number of timesteps by calling *run\_model(step\_count).* Alternatively the state of the simulation can be advanced by a single timestep by calling the model’s *step()* method*.* Each time the *step()* method is called, the following events take place in this order:

* All workers’ skill change trackers are reset (see section 3.1)
* The skill quartiles, used by the trainer, are updated.
* The inventory creates new projects, and a team of workers is assigned to each.
* The *schedule.step()* method is called, advancing each worker’s state (see section 3.1). Workers that have been assigned as team leaders advance the state of the project(s) that they lead.
* Workers enter training according to the training mechanism that is in use (see section 3.5).
* Null projects are removed from the inventory. (These are projects that could not be assigned a valid team - see section 3.2).
* The social network is saved to disk (if functionality is switched on in config file – the frequency at which to save the network can also be controlled).
* The data collector records the tracked variables (see section 4.2).

## Details and definitions

In this section we provide definitions for the various model elements that were introduced above. Throughout this section we use the default values of the model parameters as defined in the [config file](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/config.py) of model version 1.0. Were concepts are introduced, we also provide links to the relevant python files that contain the corresponding class or method.

### Worker attributes and methods

Each [worker](https://github.com/Superscriptus/SuperScript/blob/master/superscript_model/worker.py) has the following key properties:

#### Skills

The worker gets a set (“endowment”) of initial skills *s* at different levels *l(ks)*. Skills A-E are so-called hard skills and are required directly by projects. Skills F-J are soft skills and are used to determine the cognitive diversity of a team. The initial skill set is determined as follows:

1. For hard skills (A-E), it is determined uniformly at random whether the skill is existent (1) or not (0). The probability of skill existence is 80%, that is on average workers possess 4 out of 5 hard skills. However, it is ensured that each worker has at least one existent skill.
2. For existing hard skills, a level is assigned as a random number between 1 and 5. This is rounded to one decimal place for display purposes.
3. For soft skills (F-J), a level is assigned as a random number between 1 and 5, again rounded to one decimal place for display.

Example of skill matrix of a worker *k*:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Skill* | **A** | **B** | **C** | **D** | **E** | F | G | H | I | J |
| Level |  | **3.9** | **3.2** | **4.1** | **1.5** | 4.9 | 3.0 | 4.1 | 1.2 | 3.5 |
| None existent | **0** |  |  |  |  |  |  |  |  |  |

Soft skills are static and cannot change during a simulation. Hard skills are updated based on project work experience (via *peer assessment* – see section 3.4.1), training and skill decay. Skill decay is where unused skills are subject to a multiplicative decay of 0.99 on each timestep.

#### OVR

The worker overall rating (OVR) is the average of the hard skills, multiplied by 20. In this example, the worker OVR is 63.5. This is used as a measure for how highly skilled the worker is.

#### Strategy

The worker strategy is used by the worker to bid on projects. The default strategy is called ‘stake’ and means that workers can only bid for projects where the risk is less than or equal to half of their OVR.

#### Leader list

This is a list of the projects which the worker is currently leading on. Project leaders are responsible for advancing the state of the projects that they lead on.

#### Contributions

This class uses a dictionary to track the contributions that the worker makes to different projects throughout the course of the simulation. These contributions are determined during team allocation. The worker can only work up to a maximum of 100% full time equivalent (FTE). Each skill that they contribute to a project accounts for a single *unit* which is equal to 10% FTE.

#### History

This class tracks the workers recent success rate on projects. The success rate is used to compute a quantity called *momentum,* which is used in the probability calculation (see section 3.3).

#### Training remaining

This integer counter indicates the number of timesteps for which the worker will be on training. If the worker is not currently on training, then the counter is equal to -1. A single training course lasts for 5 timesteps (see section 3.5).

#### Timesteps inactive

This integer counter records the number of timesteps for which the worker has not contributed to any projects. If the worker is inactive for 10 timesteps, they are replaced by a new worker.

#### Skill change trackers

These dictionaries log the change in each hard skill that is produced by each of the three mechanisms (peer assessment, skill decay, training) on a single timestep. The tracker are reset are the beginning of each timestep.

### Departments

Departmental workload requirement is expressed as a percentage of total department capacity and is uniform across departments. We presume that the work requirement of a departmentcan be accomplished by an arbitrary subset of its workers and that every worker of departmentperforms departmental work with the same efficiency. So, a workload of 10% could be met by all of the department’s worker reserving 10% of their time for departmental work. Alternatively, a department of 100 workers could meet this requirement by allocating all departmental workload to 10 of its workers, for example.

### Project definitions

At this step, a project is initialized. Key parameters include the units of required hard skills, a desired creativity level, the required staking (representing project risk), and budgetary and timing considerations. The project portfolio is determined exogenously.

The set of projects that the firm has selected for execution within the planning horizon is denoted by ***P*** = {1, …, *P*}. We assume that for each project *p* ∈ ***P*** the project lasts for exactly 5 time steps and is started at one of the first 5 time steps of the planning horizon. Furthermore, the project specification defines the set of skills-by-level that are required. The skill requirements, *r(pslt)* are expressed in terms of percentage of full-time-equivalents.

The planning horizon of the firm spans the set **T** = {1, …, T} of *T* periods. In a typical setting, the length of the planning horizon is 10 time steps and the length of each project is 5 time steps.

A project is defined by the following five factors: (1) its minimum requirements for hard skills, (2) the required creativity level, (3) the project risk, (4) budgetary constraints and (5) timeline constraints.

1. Required hard skills A-E, *r(pslt)*, are defined as units (one unit equals 0.1 full-time equivalents) of skills-by-level. For the sake of simplicity, both required skill level and number of units are whole numbers. The matrix is determined as follows:
   1. The total number of skill units is defined as a random whole number between 5 and 50
   2. It is determined whether a particular skill is required (1) or not (0). The probability of a skill requirement is 80%, that is, on average, a project requires 4 out of 5 hard skills
   3. The total number of required skill units is allocated randomly into the 5 levels, whereby no skill can require more than 7 units and per skill only one level is allocated. Also, for at least one skill more than 2 units are required (and less or equal than 7)
   4. The maximum number of skill units a particular worker can contribute is limited to 10 across all skills (i.e., a skill unit is equivalent to 0.1 full-time equivalents)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Skill / level* | **A** | **B** | **C** | **D** | **E** |
| 5 |  |  |  |  |  |
| 4 |  | **2** |  | **4** |  |
| 3 |  |  | **3** |  | **2** |
| 2 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| Not required | **0** |  |  |  |  |

1. The required creativity level is defined by a random whole number between 1 and 5
2. Project risk is defined by the required staking and randomly determined as 5, 10 and 25
3. Budgetary constraints are expressed as the multiplication of hard skill levels and units required of those skills, e.g., in the example above the budget is 39 (= 2\*4 + 3\*3 + 4\*4 + 2\*3). While for 75% of projects this budget is the upper limit, in 25% of cases there is budgetary flexibility which allows the budget to increase by 25% (e.g., the budget increases from 39 to 49)
4. The planning horizon for a project extends to 10 time steps. Timeline constraints relate to the start date of the project. The start date can be limited to immediate (time=0) or be extended by 1, 2, 3 or 4 time steps. Half of the projects have no timeline flexibility. Timing flexibility is valuable as it allows the team assembly process not just to focus on t=0 worker availability but extend it to future periods with the possibility of being able to assemble a team with a higher success probability (“forward dispatch”). Workers that are pre-booked for future time steps select from those invitations and are not available for training or other projects.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time step | 0 | 1 | 2 | 3 | 4 |
| Frequency | 50% | 25% | 10% | 10% | 5% |

Those five elements fully describe a project at initiation.

### Probability function

|  |
| --- |
| * Maximize average expected probability of project success, ***prob***, at point *t* * Subject to:   + Time workers spend on projects does not exceed their availabilities   + Sufficient time is spared for departmental requirements   + Team size is between 3 and 7   + Number of concurrent project assignments per worker is between 1 and 3   + Time and budgetary constraints of the project are met * Where ***prob*** is a function of the average rating of the required skills, a skill balance, a creativity match and a chemistry booster |

### Other functions

#### Peer assessment

### Training mechanism

(Expand)The workers selected for training are those with the lowest skill level over the two top priority skills at that timestep. (Priority skills are those most in need for projects at that time.)

Workers whose skill level *l(ks)* for so-called priority skills falls below the median level of all workers in the organization will attend a training *e(s)*. Priority skills are defined as the two skills with most demand in terms of sum of required full-time equivalents of a certain skill at time *t*. Only one skill *s* is trained. The length of the training is 5 time steps. During this time, the worker is not available for project work nor for departmental work. The training will upgrade the skill level *l(ks)* to the first quartile level of all workers in the organization.

At every time step, a fixed number of “training slots” made available. Training is allocated to the workers whose distance to median for the priority skills is largest. The number of training slots made available at every time step is set such that in steady state 10% of the workforce is on training. E.g., if the workforce consists of 100 workers, the number of training slots is 2 = 10%\*100/5.

The length of the training is 5 time steps. The skill with the largest difference to the medium level is trained. After completion of the training, the trained skill of the worker is updated to match the first quartile level of the population at the beginning of the training.

## Visualization and data output

### Mesa server

The simulation may be run using the Mesa browser-based live visualization by invoking *mesa runserver*. In this case, various model parameters can be set using the GUI controls and the simulation can be stopped/started or stepped through step-by-step. Alternatively, the simulation can be run for a set number of timesteps using one of the [simulation runner scripts](https://github.com/Superscriptus/SuperScript/blob/master/local_run_simulation.py). In this case, the projects are saved to disk at the end of the *run\_model()* method (if that functionality is activated in the config file) and the data collection variables can be saved to disk by the script.

### DataCollector

### Running as a script

## Hypotheses

1. High risk projects (high stake) attract talent (high OVR)
   1. Show OVR (individual and team) on y-axis and risk on x-axis
2. Cognitively diverse teams have higher success rate than randomly selected teams
   1. Show project success rate on y-axis and team creativity level (absolute) on x-axis
3. Superstars emerge, i.e., workers who get over-proportionally selected to work in high stake projects, and, on the other hand, workers with low OVR are stuck with departmental workload and, over time, get replaced by new workers
   1. Show (scatterplot) OVR (individual) on y-axis and risk (stake) on x-axis
   2. Show (bars) OVR (individual) on y-axis and %-age of idle, departmental and project workload
4. Timeline flexibility pays off (start date can be later) in terms of higher project success rates
   1. Show (scatter) project value add on y-axis (with and without timing flexibility) and project budget on x-axis (DONE)
5. Budgetary flexibility pays off (higher budget) in terms of higher project success rates
   1. Show (scatter) project value add on y-axis (with and without budgetary flexibility) and risk on x-axis
6. Targeted training pays off in terms of higher project success rate and lower turnover
   1. Show (scatter) project value add on primary y-axis (with and without training), turnover on secondary y-axis, and time on x-axis
7. Some slack in the system, i.e., a utilization rate of workers <100% will add value
   1. Show (line) project value add on primary y-axis (with 10, 5, 2, 1 projects per timestep), average OVR on secondary y-axis, and time on x-axis
8. Too much slack in the system will increase turn-over (i.e., too many workers are not engaged in projects or trainings, the median skill level drops and workers get replaced)
   1. Show (line) turnover on primary y-axis (with 10, 5, 2, 1 projects per timestep), average OVR on secondary y-axis, and time on x-axis
9. Low OVR workers stay low OVR and eventually get replaced, et vice versa (“the rich get richer” effect)
   1. Show (line) fraction of workers with bottom-and first quartile OVR that get replaced on y-axis, and time on x-axis
10. More and earlier training mitigates “the rich get richer” effect
    1. See (i) but for training of all skills below median
11. For high OVR workers, a high portion of their OVR change is due to project work experience while for low OVR workers a high portion of the OVR change is due to training
    1. Show table with attribution of OVR change to the following factors for top and bottom quartile OVRs (measured at the end of the simulation period, e.g., after 100 time steps)
       1. Starting value
       2. Decay
       3. Training
       4. Project work
12. The higher the budgetary flexibility (up *and* down) and the lower the timing flexibility, the lower the latency (number of time steps a projects waits in the queue)
    1. Skipped

## Extensions

1. Allocation of required skill units to different skill levels for the same skill in the project requirements (e.g., to allow for junior and senior workers) [currently: skill units are only allocated to one level per skill]
2. Remove the smart algorithm, i.e., not allowing for the required project skills to be smartly chosen by an ML powered algorithm. Doing so adds stochasticity around the project success probability, **prob**. Show that ML algorithm learns and over time increases **prob**. [currently: the smart algorithm is on and **prob** is a constant]
3. Workers receive multiple invitations that exceed their availability and capacity (e.g., sum of assigned project units > 10 for a given moment in time *t*). That is, workers have to select from a list which project assignments to accept. Projects compete for workers and workers compete for projects. For this to work, a method for two-sided matching, a clearinghouse mechanism, is required, e.g., a Deferred Acceptance Mechanism. [currently: invitations do not exceed availability and capacity of a worker and hence all invitations get accepted]
4. The project portfolio is determined endogenously, i.e., projects are selected (and prioritized) based on available skills in order to maximize their value add [currently: projects are determined exogenously]
5. Long-lasting projects, i.e., more than 5 time steps, with changing skill requirements. The objective would be to minimize turnover within the team while meeting changing skill demands [currently: project length is set to 5 time steps with constant skill requirements]
6. Training of a skill can be randomized, i.e., a random skill of the 5 skills is trained [currently: priority skills are trained when a particular worker’s level is below medium]
7. Priority skills can be set by the organization in order to support a bottom-up transformation [currently: priority skills are those two skills where the aggregate unit demand is highest]
8. Introduce slack to avoid “overcommitment”, e.g., sum of departmental workload, project workload and training engagements is set to never reach 90% of full utilization [currently: slack is not explicitly introduced]
9. Viral mechanisms, e.g., feed and community for workers to see how their co-workers’ journey (training, project assignments, OVR changes), to share best practices, to make rankings/top movers visible, to issue badges and to establish mentor and tribe relationships [currently: not implemented]
10. Introduce a workforce engagement metric that increases project success probability, e.g., something like a virtuous circle [currently: not implemented]
11. At the project settlement, the team is assessed collectively and not individually [currently: an individual peer-to-peer assessment is conducted]
12. Method to assemble teams based on cost (“cheapest”) or average (“random”) [currently: team selection is aiming at maximizing project success probability]
13. Model members of a team that randomly drop out during the project and have to get replaced (“hot swapping”) [currently: assembled team at initiation of the project stays together until the project ends]
14. Model members of an active project team that are moved to another (higher priority) project that starts later (again, some type of “hot swapping”). This could result in a more “optimal path” [currently: assembled team at initiation of the project stays together until the project ends]
15. Add stochasticity to r(dt) departmental workload requirement [currently: departmental workload requirement is deterministic]
16. Project success is not binary but determined on a scale of -2, -1, 0, +1 and +2 [currently: project success is binary]
17. Introduce objective (or constraint) to level the utilization across workers in the organization and / or within a department [currently: utilization can have corner solutions, e.g., “superstars” work 100% on projects while workers with low OVR conduct almost exclusively departmental workload]
18. Introduce option for external hiring of workers with “defined” hard skills to close aggregate skill gaps [currently: model is limited to workers within the organizations and replacements are generated with random skill endowment]
19. Budgetary flexibility is introduced to allow for lower team OVR (and thus, lower project success probability) [currently: budgetary flexibility increases the budget]
20. Projects that are not staffed during the 5 time step planning period will be moved to the next planning period, and will create additional backlog projects [currently: projects that are not staffed during the planning period “get cancelled”]
21. Introduce a constraint for maximum number of concurrent project participation for any worker, e.g., 2-3 as suggested by [academic literature](https://www.semanticscholar.org/paper/Concurrent-projects-%3A-how-many-can-you-handle-Steyn-Schnetler/1531842fd17e3a1048d4e5b182c9c3957f248d67) [currently: number of concurrent projects a worker is engaged in is not limited explicitly - there is however an implicit limit for high OVR workers via the staking requirement]
22. Simulation of “shocks” to the system, i.e., introduction of a large high priority-high stake (“all hands on deck”) project (modelled, e.g., as the introduction of 10 new projects all with the same project requirements, and putting all workers on status “idle”) and analyse how those shock waves have knock-on effects on the remaining 50 projects in order to understand the fragility of the organization [currently: no exogenous shocks are modelled]
23. Introduce possibility of changing soft skills [currently: soft skills vector remains constant over time]
24. Introduce option to work on projects / or do departmental work while on training [currently: no project or departmental work is possible while on training]
25. Make workers within a department more homogeneous in terms of skills (not level, but exposure to skill). Then, as a base-line (“benchmark”) randomly allocate projects of a department to workers of that department [currently: projects are not assigned to departments and workers are randomly endowed with skills]
26. Introduce constraints in the form of one or multiple workers that are pre-assigned to the project team, i.e., the optimizer finds the most optimal team combination considering those constraints [currently: all workers are selected by the algorithm]
27. Introduce individual skill development options, i.e., ability of a worker to select skills they want to improve in and hence take trainings and/or get exposed to these skills in projects with team members that are more experienced [currently: not implemented]
28. Allow training triggers to change, i.e., instead of training being triggered by being below median for the top priority skills, it could be below first or fourth quartile [currently: trigger is set to median]

## Old stuff

1. A project last for exactly 5 time steps, i.e., after the initial ramp-up phase on average 50 projects are active at any point in time and, assuming a team size of roughly 5 on average, 250 workers are actively engaged in projects at all times

#### Phase 3: Project work

The project team works on the project for 5 time steps. During this time, the skill levels remain unchanged.

#### Phase 4: End of project

The project ends and is settled. At this point, it is determined whether the project was successful or not. The stake is returned and the utilized hard skills are updated based on project success and peer reviews. See procedure: go\_settle.

At the end of the project, a worker can remain idle or be busy by either taking on an additional project or doing training.

If a worker is not engaged in project work for 10 consecutive time steps, the worker leaves the organization and is replaced by a newly generated worker in the same department.

At the end of every time step, irrespective of the worker’s status, the skills that are neither trained nor utilized in a project are multiplied by a decay factor of 0.99 and updated.

### Variables and procedures

#### Probability of project success

Project success (i.e., probability of project success), **prob**, is an additive function of five elements: (1) the team OVR, (2) the skill balance, (3) the creativity match, (4) a chemistry booster, and (5) the project risk.

1. Team OVR

The team OVR is the average skill level for the required skills multiplied by 20. E.g., in the example below, the project requires 11 skill units (i.e., 1.1 full-time equivalents) and those are allocated as follows (purple shade) in the table below. The team OVR hence is 72.0.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Skill* | **A** | **B** | **C** | **D** | **E** |
| Worker 1 |  | **3.9** | **3.2** | **4.1** | **1.5** |
| Worker 2 | **4.1** | **4.4** | **2.1** | **2.9** | **0.4** |
| Worker 3 |  |  |  | **0.5** | **3.9** |
| Worker 4 |  | **3.6** | **2.5** | **4.9** | **5.0** |
| Worker 5 |  |  |  | **2.9** |  |
| Total units | **0** | **2** | **3** | **4** | **2** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Team OVR | <=25 | >25<50 | >50<70 | >70<90 | >90 |
| Probability | 10% | 20% | 45% | 60% | 75% |

1. Skill balance

The degree of required skills match is defined as the sum of squared *negative* differences between the team’s actual hard skills (average level by skill) and the required hard skills (required hard level by skill) divided by the number of required skills with negative differences. E.g., for skill C 3 units with level => 3 are required while the actual average team skill is 2.6 (3.2 + 2.1 + 2.5 / 3). Therefore the squared negative difference is 0.16 (-0.4^2). The same procedure is applied to all skills and the average of negative squared differences is labelled “degree of required skills mismatch”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Degree of required skills match | 0 | >0<1 | >1<4 | >4<9 | >9<=16 |
| Probability | 0% | -5% | -10% | -20% | -30% |

1. Creativity match

The degree of creativity match is defined as the sum of squared differences between the team’s actual creativity level and the required creativity level.

The actual creativity level (i.e., the degree of team heterogeneity) is defined as [...]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Degree of creativity match | 0 | <=1 | >1<=4 | >4<=9 | >9<=16 |
| Probability | +10% | -5% | -15% | -25% | -35% |

1. Chemistry booster

The probability of project success is impacted by a chemistry booster, both at the individual worker level and at the team level. Each element is 1 if positive and 0 if negative. The numbers are added up to arrive at the chemistry level.

* Chemistry at individual worker level is impacted by:
  + Relevant position and role for worker: a position is relevant if at least one of the worker's two top skills, the primary and secondary skill, is required in the project
  + High stake level: required staking is >= 10% of a worker’s OVR
  + Momentum, e.g., 3 out of 4 last projects have been successfully completed
* Chemistry at team level is impacted by:
  + Links between team members who worked on successful projects in the past is >= 50% (that is, half of the team mates have worked together before on successful projects)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| chemistry level | >0 | >1<=2 | >2<=3 | >3<=4 |
| Probability | +2.5% | +5% | +7.5% | +10% |

1. Project risk

Project risk is measured by the required staking.

|  |  |  |  |
| --- | --- | --- | --- |
| Required staking | 5 | 10 | 25 |
| Probability | 0% | -5% | -15% |

Example: A team with a team OVR of 58, a degree of required skills match of 0.5, a degree of creativity match of 1.5, a chemistry level of 3 and a required staking of 5 will have an expected success rate of 30% (i.e., 45% + (-5%) + (-15%) + 5% + 0% = 30%).

#### Settlement

go\_settle

The settlement procedure consists of the following steps:

1. Success of the project (yes/no) is determined by selecting a uniform random number on the interval [0,1]. If the selected number is <= to the expected project success probability, **prob**, then the project is successful.
2. A peer-to-peer skill assessment takes place at the end of the project. Thereby the team mates as well as the organization rates the skill level of the worker. The average of these assessments is combined with the existing skill level with a 25% weight. The deviation of these assessments from the original skill level depends on the project success. The assessment factor is a normally distributed random number with the following [mean, stdev] properties

|  |  |  |
| --- | --- | --- |
| project success | yes | no |
| assessment factor | 1.05, 0.2 | 0.95, 0.2 |

E.g., for a successful project the utilized skill D of worker i was originally at level 4.1. This level is updated on average by factor 1.05 and a weighting factor of 25%. Hence, the average skill level for skill D after the peer-to-peer assessment is 4.2 = 0.75 \* 4.1 + 0.25 \* 1.05 \* 4.1

1. This updated level-by-skill is then multiplied by a factor that depends on project success and project risk (i.e., the required staking of 5, 10 or 25)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| project success | no | Yes (5) | Yes (10) | Yes (25) |
| success factor | 1.0 | 1.05 | 1.1 | 1.25 |

E.g., for the worker i, the level of skill D is updated from originally 4.1 to 4.2 after the peer assessment and then to 4.6 (for a successful medium risk project, i.e., “Yes (10)”)

1. The level-by-skill is capped at 5
2. The social graph is updated (three-dimensional vector between any two workers, indicating number of projects, success ratio of projects and average of reciprocal ratings)
3. At the end of settlement, the worker frees up additional capacity that can either be used for departmental work, running projects, new project assignments or training

#### Team assembly

go\_assemble

1. All new projects at a given time step are grouped by required stake (project risk) and then within this grouping they are ranked by required creativity and then randomly
2. Per project, starting with the first one on the list created in step 1, teams are created and ranked by the a priori success probability subject to budgetary and timing constraints. Only team members are invited where required stake <50% of that worker’s initial OVR
3. Invitations are sent out to selected team members of a project

## Visualization and data output

A graphical user interface, dashboard, should allow for changing some key variables and visualizing the development of the agent-based model.

1. Changing speed of simulation
2. Changing parameters such as team size (min and max), number of projects per time step, number of concurrent project assignments per worker, budgetary and timing flexibility etc.
3. Visualization of capacity utilization (FTE-percentage engaged in projects, training, departmental workload and “idle”), success rate of projects over time (thereof high-stake projects), development of average worker OVR, development of social graph (“giant clusters”), efficiency from an organization point of view (output expressed as success rate of high-risk/high-creativity projects divided by input expressed as training and project’s team OVR)
4. Show throughput (how many projects are active) and latency (how long, i.e., how many time steps, a projects waits in the queue before being worked on) at any time step
5. Scatterplot with expected value (prob x required staking) on y-axis and budget on x-axis
6. Network analytics like clustering coefficients and average distance