Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2016, Article ID 8324914, 13 pages
http://dx.doi.org/10.1155/2016/8324914

Research Article

Applying Modelica Tools to System Dynamics Based Learning Games: Project Management Game

Tuomas Miettinen,[1] Juho Salmi,[1] Kunal Gupta,[2] Jussi Koskela,[3] Janne Kauttio,[1] Tommi Karhela,[1] and Sampsa Ruutu[1]

[1]*VTT Technical Research Centre of Finland Ltd., Vuorimiehentie 3, P.O. Box 1000, 02044 Espoo, Finland*
[2]*Pöyry Finland Oy, Jaakonkatu 3, P.O. Box 4, 01621 Vantaa, Finland*
[3]*Semantum Oy, Tekniikantie 14, 02150 Espoo, Finland*

Correspondence should be addressed to Tuomas Miettinen; tuomas.miettinen@vtt.fi

Received 12 July 2016; Accepted 27 September 2016

Academic Editor: Martin Otter

Learning simulation games are interactive simulations with game characteristics. This paper presents a learning simulation game for EPCM (engineering, procurement, and construction management) project management training. The simulation model utilises system dynamics, which is a methodology for understanding the behaviour of dynamic complex systems of different domains using modelling and simulation. The system dynamics model in turn uses the equation-based Modelica modelling language: a system dynamics model created with the graphical user interface is converted to a pure Modelica model. Two Modelica environments, namely, OpenModelica and the custom Modelica solver, have been used to simulate the generated Modelica model. The focus of this article is on how generic systems modelling and simulation platforms such as Modelica based environments can be utilised in developing a learning simulation game: what benefits do they bring and what disadvantages do they have? On the one hand, it is evaluated how the Modelica language as such is suitable for being used in a learning game development. On the other hand, the suitability of the selected implementation environments, that is, OpenModelica, the custom Modelica solver, Simantics, and Simupedia, is evaluated. The paper also shortly presents how the project management game was received by its players.

1. Introduction

Learning simulation games are interactive simulations with game characteristics; instead of typical games, learning games are primarily used for a purpose other than entertainment. Learning games have been found to be an effective method for learning: they allow creating virtual worlds in which the player can try out different decisions and strategies. Learning simulation games do not differ much from how simulation is generally applied. Sterman [1] points out three benefits for interacting with a learning simulation game instead of interacting with the actual system. Firstly, learning games provide a safe environment for testing different operating strategies without a fear of making a mistake, which might be costly if applied to the actual system. Secondly, long-term effects of a decision are visible almost immediately. Thirdly, the relation between a decision and the respective effect is much clearer and easier to verify. Hence, learning via a simulation game is cost-effective and fast and provides learning results that are more reliable.

This paper studies the application of the equation-based Modelica language [2] in learning game development. In addition to the Modelica language, general Modelica based systems modelling and simulation platforms are evaluated: the experimental part of the work has been implemented based on the OpenModelica [3] tool, on top of which additional features have been implemented to assist system dynamics modelling and game development. Even when not experimentally compared with other tools, Modelica environments such as Dymola [4] are briefly reviewed from this perspective.

In the experimental part of this work, a learning game for EPCM (engineering, procurement, and construction management) project management training (later *the EPCM* *game*) has been developed to teach the project management process of a company in a virtualised environment. The game is based on the specific characteristics of the actual processes in the partner company and has been tested with project manager trainees and students. The paper presents how system dynamics and systems thinking were applied in the EPCM game.

The paper is constructed as follows. First, a short introductory to learning games based on system dynamics is given. Then, the tools used for the game development are presented, concentrating on how they utilise the Modelica language and extend the OpenModelica tool. After that, the EPCM game developed in the experimental part of this work is described: the simulation model and the user interface developed are presented. Then, in Results and Discussion, the results of the game itself are presented and the pros and cons in utilising Modelica and Modelica environments in a learning game are discussed. Finally, the conclusion is given.

2. System Dynamics Based Learning Games

The learning game developed in this work is based upon the concepts of systems thinking, system dynamics, and simulation games. These concepts are briefly explained in this section.

The concept of *system* has many definitions [5]. In the context of this work, system is defined as "a set of interacting components that forms an integrated whole" [6]. For example, organisations, such as the one studied in this work, can be seen as complex systems.

Sterman [6] defines *systems thinking* as an ability to see the world as a complex system, in which one cannot do just one thing without affecting everything else. An important source of complexity in complex systems arises from their internal *feedback loops*, which is why understanding feedback loops is in the core of systems thinking. The aim of the modelling problem in this work was to improve the understanding of how various project management decisions, such as workforce assignments, cause dynamic effects on the project quality, schedule, and costs.

Closely related to the concept of systems thinking is *system dynamics*, which is a methodology for understanding the behaviour of dynamic complex systems of different domains using modelling and simulation. System dynamics models consist of stock and flow diagrams with feedback loops and delays. The methodology is suitable for examining complex sociotechnical systems that involve interrelationships between technical elements (e.g., project task structure and precedence relationships) as well as human factors and decision-making (e.g., effects of overtime on quality and workforce decision making rules).

Garris et al. [7] consider games and simulations very similar since games usually have a set of rules that can be interpreted as a simulation of a model. The main difference between games and simulations is that games do not generally imitate real-world systems. van Daalen et al. [8] define *simulation games* as interactive simulations with game characteristics. They also define interactive simulation as a simulation that the user can access while it is running to receive feedback and to give input, that is, to make decisions based on the measured state of the simulation. Some studies (e.g., [9, 10]) even consider that all interactive simulations are also simulation games.

Through the process of modelling a system, the modeller should learn how the system behaves. However, to convey this knowledge to others is not trivial. One way to do it is to develop a *system dynamics based learning game*. Andersen et al. [11] and van Daalen et al. [8] discuss that the objective of such games is to teach the dynamic insights of the modelled system, that is, the ability to utilise systems thinking in understanding the complex dynamics of the modelled system and to implement strategies according to that understanding.

3. The Game Environment

This section describes the tools developed on top of the OpenModelica environment to support the EPCM game development.

3.1. Simantics System Dynamics. The system dynamics model of the EPCM game was modelled and simulated with *Simantics System Dynamics* [12, 13]. Simantics System Dynamics is an open source system dynamics modelling and simulation tool, which is built on the *Simantics* platform [14] and based on the Modelica systems modelling language [2]. In addition to merely using the tool for modelling, it was also further developed in this work to support features needed for the EPCM game development. Both Simantics System Dynamics and the Simantics platform are managed by the THTH Association and actively developed by VTT and Semantum Oy.

Simantics System Dynamics is essentially a traditional system dynamics modelling and simulation tool, with all the features one would expect from such. The user can construct system dynamics models with a graphical diagram editor, simulate them with different solvers, and visualise the results with different kinds of charts. Simantics System Dynamics uses the Modelica language as its internal model representation format and OpenModelica as the solver. Because of this, the user is able to utilise the vast library of Modelica functions provided by OpenModelica seamlessly in the model and is able to write their own Modelica functions if necessary. In addition, an internal Modelica solver with a very limited support for the Modelica language is provided within the tool. One additional benefit of the tight integration with Modelica is that the Modelica code of a system dynamics model can be exported directly from the tool and technically solved in any Modelica environment.

In addition to the basic features, Simantics System Dynamics also includes several advanced features, some of which are rarely found in system dynamics modelling and simulation tools: the tool supports structural modelling that allows complicated models to be broken down into smaller hierarchical modules where repeating structures have to be defined only once. The tool has several kinds of experiments that can be used to perform different types of simulation, for example, sensitivity analysis. The tool also includes

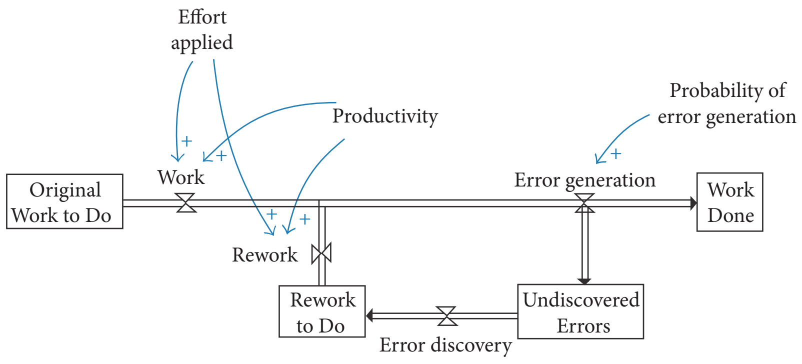


Figure 1: Rework cycle 1/3: the basic structure.

support for the *Simantics scripting language* (SCL), which can be used to automate most modelling and simulation operations.

However, the biggest benefit of using Simantics System Dynamics to develop the EPCM game is the Simantics platform itself. Simantics is a general modelling, simulation, and, most importantly, integration platform that combines an ontology-based database with the Eclipse plug-in. A complete coverage of the Simantics platform is beyond the scope of this work but the key feature of Simantics in the context of the EPCM game is its capability to combine and utilise the features of different modelling and simulation tools developed on top of the platform. One of these tools is *Simupedia*, which is covered in more detail in the next subsection.

3.2. Simupedia. Simupedia [15] is a Simantics based tool that can be used to build and deploy rich web applications on top of any Simantics product. Simupedia applications are run on web browser, so the end-users do not have to install any specific software. Simupedia comes with full integration into the Simantics System Dynamics tool. Hence, it is possible to control and observe every aspect of a system dynamics experiment from the web application built with Simupedia.

Simupedia applications are constructed graphically using primitive widgets, such as buttons, text fields, images, labels, and charts, and higher lever components such as simulation controls. The modeller may build basic user interfaces without any programming experience. It is also possible to create fully custom web applications like the EPCM game by using some additional SCL scripting. Internally Simupedia uses the Vaadin framework [16], so no client-side programming is required.

In the EPCM learning game, the Simupedia application creates a new experiment for each user. User's selections and scripted storyline events are reflected into the input variables of the model and the values of the selected output variables are displayed in the user interface as charts and sheets. The simulation is run step by step giving the user a chance to interact between each two simulation steps. This kind of approach means that the system dynamics model can be used as the basis for the game application without any modifications.

4. The EPCM Game

This section introduces the EPCM game developed in this work: the simulation model, the learning goals, and the end user interface are presented. Furthermore, this section describes how the game was used for training. A more detailed description of the game is provided in the work by Salmi [17].

4.1. Simulation Model. In this subsection, a simplified version of the EPCM game model is introduced with simplified equations and less variables in order to allow the reader to get a basic understanding of the model. The complete model consists of over 100 variables with up to four dimensions. Another goal of the description is to give the reader a view for what kind of equations have been used in the model. Hence, a number of equations are presented with them being flattened to one dimension and, in addition, the full Modelica codes of some of the equations are shown with all dimensions visible.

4.1.1. Rework Cycle. In the EPCM game model, as usually in system dynamics models about project management, the concept of *rework cycle* [18] is utilised [19, 20]. The basic structure of the rework cycle of the EPCM game model is presented in Figure 1. The amount of work to be done in the project is modelled as the *Original Work to Do* stock, from which an amount of work flows out to the *Work Done* stock via the *Work* flow ((1)-(2)). The actual cycle is initiated by the *Error Generation* flow that causes part of the work from the *Original Work to Do* stock to flow to the *Undiscovered Errors* stock instead ((3)-(4)). Discovering errors (the *Error Discovery* flow) releases work to the *Rework to Do* stock from which it flows back to the process similarly than via the *Work* flow (5). The working output, that is, the sum of *Work* and *Rework*, depends on the *Productivity* of the workforce and the *Effort Applied*, that is, effective working hours used for the project (6). Hence, increasing the productivity of the workforce or allocating more working hours for the workers allows finishing the project earlier.

$$\text{Original Work To Do} = \int (-\text{Work})\, dt, \tag{1}$$

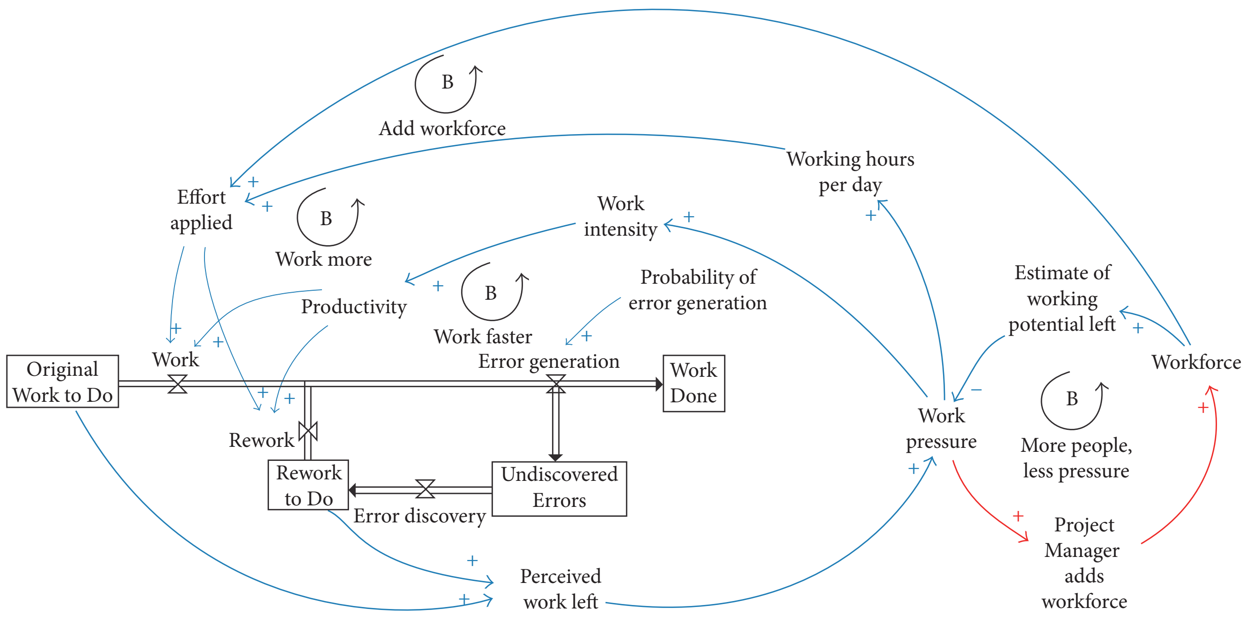


Figure 2: Rework cycle 2/3: the balancing loops.

$$\text{Work Done} = \int (\text{Work} + \text{Rework} - \text{Error Generation})\, dt, \tag{2}$$

$$\text{Error Generation} = (\text{Work} + \text{Rework}) * \text{Probability Of Error Generation}, \tag{3}$$

$$\text{Undiscovered Errors} = \int (\text{Error Generation} - \text{Error Discovery})\, dt, \tag{4}$$

$$\text{Rework To Do} = \int (\text{Error Discovery} - \text{Rework})\, dt, \tag{5}$$

$$\text{Work} + \text{Rework} = \text{Effort Applied} * \text{Productivity}. \tag{6}$$

The *balancing feedback loops* that affect the basic rework cycle are depicted in Figure 2: if the workers perceive that the current working output does not allow completing all the *Perceived Work Left* (7) in time, the *Work Pressure* rises (8). This causes the workers to increase their *Working Hours per Day* (9) and to work with a higher *Work Intensity* (10), which in turn increase the *Effort Applied* (11) and *Productivity* (14), respectively. The functions $f_{\{i\}}$ in (9) and (10) are logistic functions (see Figure 3) adapted from the work by Oliva and Sterman [21]. The *Project Manager Adds Workforce* variable in Figure 2 is an auxiliary variable for the player's main input for the game. Allocating *Workforce* increases the *Estimate of Working Potential Left* which decreases *Work Pressure*.

$$\text{Perceived Work Left} = \text{Original Work To Do} + \text{Rework To Do}, \tag{7}$$

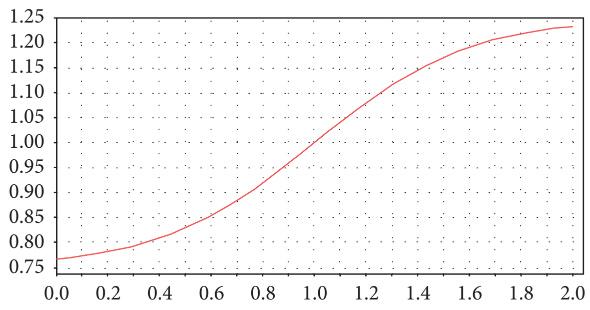


Figure 3: The logistic function in (9) and (10).

$$\text{Work Pressure} = \frac{\text{Perceived Work Left}}{\text{Estimate Of Working Potential Left}}, \tag{8}$$

$$\text{Working Hours Per Day} = f_{\{\text{Work Pressure On Working Hours Per Day}\}}(\text{Work Pressure}), \tag{9}$$

$$\text{Work Intensity} = f_{\{\text{Work Pressure On Work Intensity}\}}(\text{Work Pressure}), \tag{10}$$

$$\text{Effort Applied} = \text{Workforce} * \text{Working Hours Per Day}. \tag{11}$$

In addition to balancing loops, the rework cycle encompasses also *reinforcing feedback loops*, as depicted in Figure 4:

(1) Haste makes waste: increasing the *Work Intensity* increases not only *Productivity* (14), but also the *Probability of Error Generation* (15).

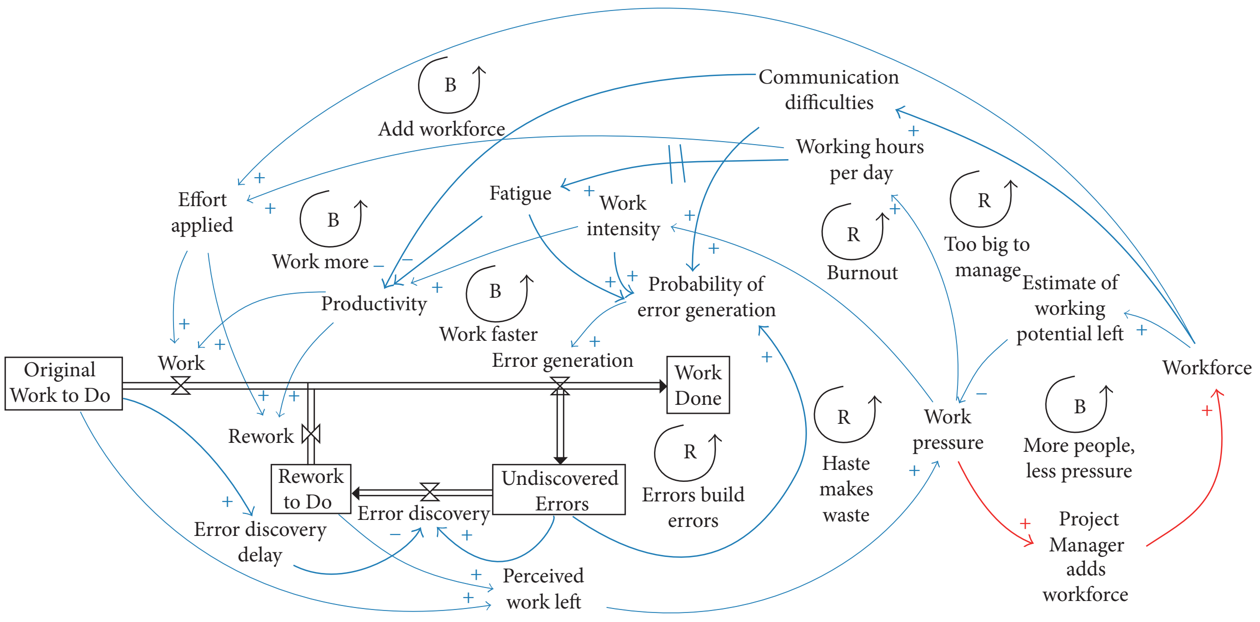


Figure 4: Rework cycle 3/3: the reinforcing loops.

(2) Burnout: working constantly longer days increases *Fatigue* (12) which in turn decreases *Productivity* (14) and increases the *Probability of Error Generation* (15).

(3) Too big to manage: adding *Workforce* increases *Communication Difficulties* (13) which, similarly to *Fatigue*, decrease *Productivity* (14) and increase the *Probability of Error Generation* (15).

(4) Errors build errors: the more *Undiscovered Errors* there are, the higher is the *Probability of Error Generation* (15), as the work that is based on erroneous work is more likely erroneous as well (14).

The functions in (11)–(14) are adapted from the model by Oliva and Sterman [21].

$$\text{Fatigue} = \text{Delay}\,(\text{Order} = 3, \text{Delay Time} = 20 \text{ working days}, \text{Input} = \text{Work Pressure}), \tag{12}$$

$$\text{Communication Difficulties} = f_{\{\text{Effect Of Workforce Size}\}}\,(\text{Workforce}), \tag{13}$$

$$\text{Productivity} = f_{\{\text{Productivity}\}}\,(\text{Work Intensity}, \text{Fatigue}, \text{Communication Difficulties}), \tag{14}$$

$$\text{Probability Of Error Generation} = f_{\{\text{Probability Of Error Generation}\}}\,(\text{Work Intensity}, \text{Fatigue}, \text{Communication Difficulties}, \text{Undiscovered Errors}). \tag{15}$$

As is shown in the model, workers make decisions based on *perceived* and *estimated* information. The player as well has to base their decisions on perceived information, as they are provided with information of the amount of work that *seems* to have been done but are unaware of the amount of errors that require rework.

As said, the equations presented before are simplifications to allow better understanding of the model. As an example of an actual equation with all dimensions visible, the Modelica code of *Perceived Work Left* is shown in Algorithm 1.

4.1.2. Experience Chain. Another basic structure commonly used in models about project management is *experience chain*, as shown in Figure 5. An experience chain is used to model the learning curves of individual workers allocated to the project [22]. All workers allocated to the project (the *Add Project Workforce* flow) start as rookie workers (the *Project Rookie Workforce* stock) and become experienced workers (the *Project Experienced Workforce* stock) as they assimilate information about the project (the *Assimilation Rate* flow) ((16) and (25)).

$$\text{Project Rookie Workforce} = \int (\text{Add Project Workforce} - \text{Remove Project Rookie Workforce} - \text{Assimilation Rate})\, dt, \tag{16}$$

$$\text{Project Experienced Workforce} = \int (\text{Assimilation Rate} - \text{Remove Project Experienced Workforce})\, dt.$$

```
Real Perceived_Work_Left[Milestone.size,Discipline.size];

Perceived_Work_Left = {
  Milestones[milestone,:] - Perceived_Progress_Fraction
    for milestone in 1:Milestone.size
};
```

Algorithm 1

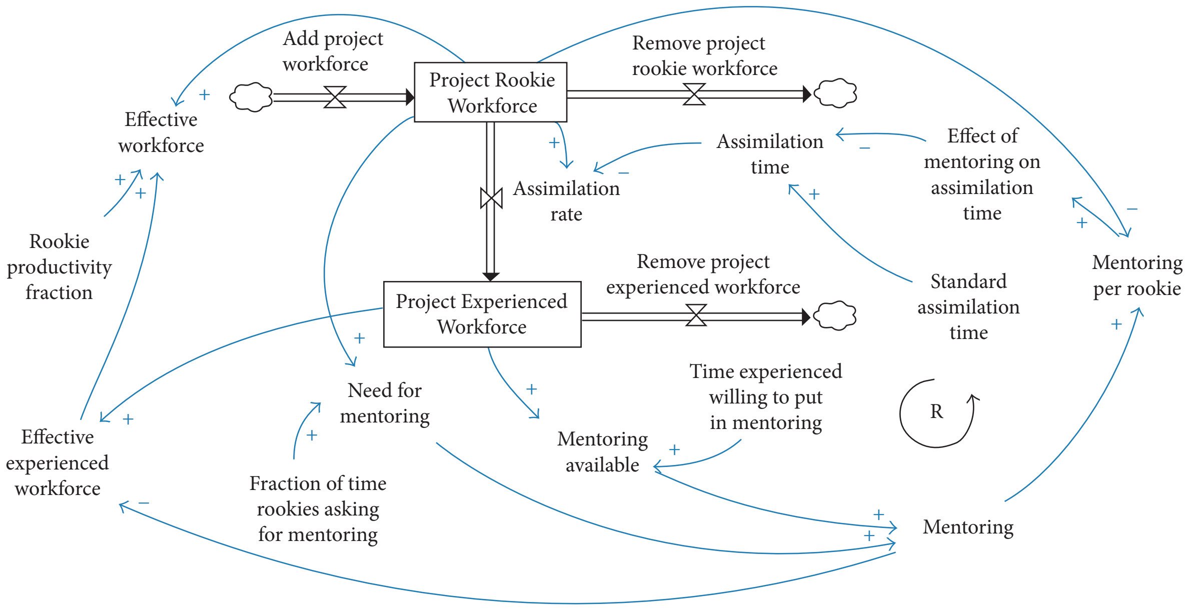

Add project workforce
Project Rookie Workforce
Remove project rookie workforce
Effective workforce
Assimilation time
Effect of mentoring on assimilation time
Assimilation rate
Rookie productivity fraction
Remove project experienced workforce
Standard assimilation time
Mentoring per rookie
Project Experienced Workforce
Need for mentoring
Time experienced willing to put in mentoring
R
Effective experienced workforce
Mentoring available
Fraction of time rookies asking for mentoring
Mentoring

Figure 5: Project experience chain.

Both the rookie and the experienced workforce add to the total *Effective Workforce* (18); however, rookies are less effective and need *Mentoring* (21), which momentarily decreases the *Effective Experienced Workforce* [21], that is, the actual project work done by the experienced workers ((17) and (19)). *Mentoring* affects the *Assimilation Time* with a decreasing marginal utility (24); for example, increasing the amount of *Mentoring* from 0% to 10% decreases the *Assimilation Time* more than increasing *Mentoring* from 10% to 20%. The higher the amount of *Mentoring Per Rookie* there is available, the shorter is the *Assimilation Time* (23). *Mentoring per Rookie* depends on how much the rookies need *Mentoring* and how much time the experienced workforce are willing to put in it (22).

$$\begin{aligned}&\text{Effective Experienced Workforce}\\&\quad= \text{Experienced Workforce} - \text{Mentoring},\end{aligned} \tag{17}$$

$$\begin{aligned}&\text{Effective Workforce}\\&\quad= \text{Effective Experienced Workforce}\\&\quad+ \text{Project Rookie Workforce}\\&\quad* \text{Rookie Productivity Fraction},\end{aligned} \tag{18}$$

$$\begin{aligned}&\text{Need For Mentoring} = \text{Project Rookie Workforce}\\&\quad* \text{Fraction Of Time Rookies Asking For Mentoring},\end{aligned} \tag{19}$$

$$\begin{aligned}&\text{Mentoring Available} = \text{Project Experienced Workforce}\\&\quad* \text{Time Experienced Willing To Put In Mentoring},\end{aligned} \tag{20}$$

$$\begin{aligned}&\text{Mentoring}\\&\quad= \min\left(\text{Need For Mentoring}, \text{Mentoring Available}\right),\end{aligned} \tag{21}$$

$$\text{Mentoring Per Rookie} = \frac{\text{Mentoring}}{\text{Project Rookie Workforce}}, \tag{22}$$

$$\begin{aligned}&\text{Effect Of Mentoring On Assimilation Time}\\&\quad= f_{\{\text{Effect Of Mentoring}\}}\left(\text{Mentoring Per Rookie}\right),\end{aligned} \tag{23}$$

$$\begin{aligned}&\text{Assimilation Time} = \text{Standard Assimilation Time}\\&\quad* \text{Effect Of Mentoring On Assimilation Time},\end{aligned} \tag{24}$$

$$\text{Assimilation Rate} = \frac{\text{Project Rookie Workforce}}{\text{Assimilation Time}}. \tag{25}$$

```
Real Mentoring_Per_Rookie[Discipline.size];

Mentoring_Per_Rookie = {
  if sum(Project_Rookie_Workforce[:,i,:]) < 0.001
  then 0.0
  else Mentoring[i]/sum(Project_Rookie_Workforce[:,i,:])
    for i in 1:Discipline.size
};
```

Algorithm 2

As another example of a complete equation, the Modelica code of the variable *Mentoring Per Rookie* is shown in Algorithm 2.

Even when not shown in the diagrams, the project experience chain and the rework cycle are interconnected: adding workers increases *Communication Difficulties* and can lower the working output temporarily, due to the effort needed for *Mentoring*. The project manager needs to understand that the project does not start advancing faster until after a delay when adding new workers to the project.

4.1.3. Global EPCM Projects. One yet uncovered essential aspect in an EPCM project is the distribution of work, on the one hand, *globally between multiple offices* and, on the other hand, *between experts from different disciplines* [23, 24]. Firstly, having multiple offices in different countries allows, for instance, using cheaper labour for routine tasks. Secondly, the division of work between disciplines has to be taken into account since workers in each discipline can accomplish only tasks from their discipline. For instance, an automation engineer can accomplish only tasks of automation engineering, not of process or mechanical engineering.

In the EPCM game, the negative effects of workforce distribution in multiple offices increase *Communication Difficulties* (26): as can be imagined, communication is hindered by people not working in the same room or even at the same time zone, speaking different languages, and having cultural differences. In the EPCM game, these difficulties decrease *Productivity* and increase the *Probability of Error Generation* ((14)-(15)). Potential positive effects of cultural diversity, for instance, on innovativeness, are not included in the model.

$$\begin{aligned}\text{Communication Difficulties} \\ &= f_{\{\text{Effect Of Workforce Size}\}}(\text{Workforce}) \\ &\quad + f_{\{\text{Effect Of Number Of Offices}\}}(\text{Number Of Offices}).\end{aligned} \tag{26}$$

All workers are divided into disciplines based on their expertise. The disciplines have interdependencies: certain disciplines require other disciplines to have progressed enough in order to have prerequisites for their work. As an example, mechanical engineering requires process engineering to have produced the piping and instrumentation diagrams in order to be able to start working. However, workers can start working even without complete prerequisites by using heuristics, that is, by using their expertise to guess how to start working.

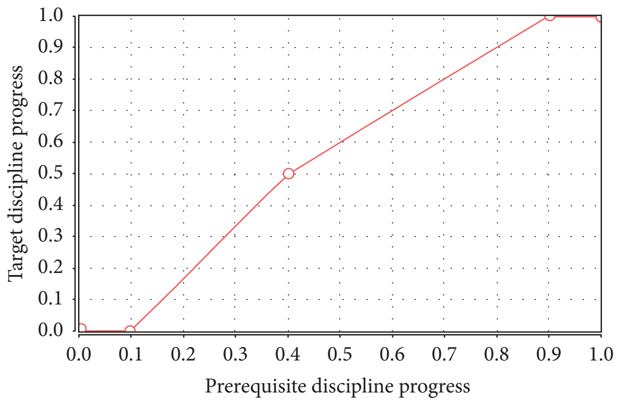


Figure 6: An example prerequisite function.

Experienced workers have better heuristic accuracy, that is, the smaller amount of work becomes *Undiscovered Errors* (27).

$$\begin{aligned}\text{Probability Of Error Generation} \\ &= f_{\{\text{Probability Of Error Generation}\}}(\text{Work Intensity}, \\ &\quad \text{Fatigue}, \text{Communication Difficulties}, \\ &\quad \text{Undiscovered Errors}) \\ &\quad * \text{Heuristic Accuracy When Prerequisites Not Met}.\end{aligned} \tag{27}$$

The EPCM game uses a similar approach for heuristics as Ruutu et al. [23]. An example heuristics curve is shown in Figure 6. Given the portion of prerequisites finished, the function returns the portion of work that can be accomplished without using heuristics. For instance, 40% of process engineering has to be done for mechanical engineering to be able to accomplish 50% of their work. Each pair of disciplines has such a function defined.

4.2. Learning Goals. The EPCM game is targeted for four learning goals. The game has three difficulty versions: the first three learning goals are for the easy version of the game, and the medium and difficult versions have a fourth goal in addition to the first three. The four learning goals are as follows.

(1) Robust Staffing and Active Management. Repenning [25], Lyneis and Ford [20], and the project managers in the partner company had observed that staffing in projects is often too optimistic: in the beginning, too few workers are assigned to the project and ramping up the workers in the disciplines is started too late. Lyneis and Ford [20] state that if the initial group of workers is too small and the amount of workers is ramped up too slowly, project managers tend to take too large corrective actions once they realise the project is understaffed. The oversized corrective actions may in turn cause the project to be overstaffed in subsequent phases of the project. Again, once overstaffing is realised, it may cause an understaffed situation as the number of workers is reduced while there are undiscovered errors in the work already made. Realising the amount of undiscovered errors may in turn lead again to the need for more workers. As a result, the project may end up having a double-humped staffing outcome (see Figure 10).

(2) Using the Right Workers at the Right Time. Experts at the partner company had noticed that the amount of workers not only was ramped up too late, but in some cases also too early. This was typically caused by the line organisation trying to keep the activity rate high by assigning any spare workers to projects without acceptance of the respective project managers. This must be taken into account by the project manager; even when they are unable to affect this, they need to be aware of the influences that arise.

(3) The Dynamics of Rework Cycle, Experience Chain, and Multiple Offices. Project managers should understand the dynamics of rework cycle, experience chain, and multiple offices. They should, among others, realise how work pressure affects workers on short and long time spans, learn that workers have a learning curve in accustoming themselves to work in a certain project, and understand how dividing workload between offices can lower costs but increases communication difficulties.

(4) Change Requests from a Client. The medium and difficult versions include also unexpected events and client interaction; the main difference between the medium and difficult levels is that the difficult version has more events than the medium version. As an example of client interaction, the project manager must ensure that the client provides all the necessary requirements and specifications. The client may also require changes during the project, which can cause work that has already been done to be wasted; that is, a portion of *Work Done* becomes *Undiscovered Errors* or *Rework To Do*. With unexpected events and with client interaction, the project manager has to apply even more robust staffing plan and to be more active in managing the project.

Examples of two different staffing plans are presented in Figures 7–10; each colour represents the staffing of one discipline. Figure 7 represents a typical, relatively robust staffing plan, and Figure 8 shows the respective progress estimates of the working phases. If the original plan is robust, only little management is needed to the staffing during the project. However, if the original plan is too optimistic, as shown in Figures 9 and 10, it is difficult to succeed even with active management: in the initial plan, the badly done allocation of workers has led to undiscovered errors, which have not been found until in the later phases of the project. This has caused the corrective actions seen as the two humps in the progress outcome estimate, as discussed earlier.

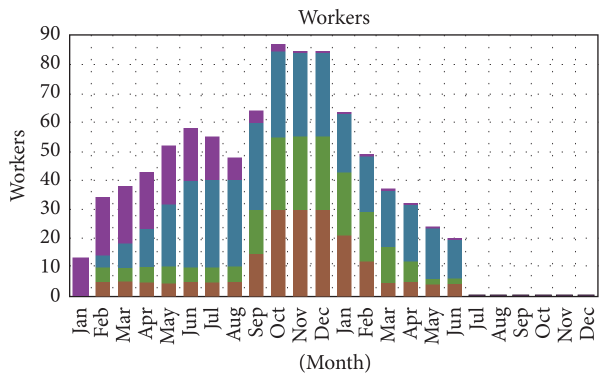


FIGURE 7: A typical staffing.

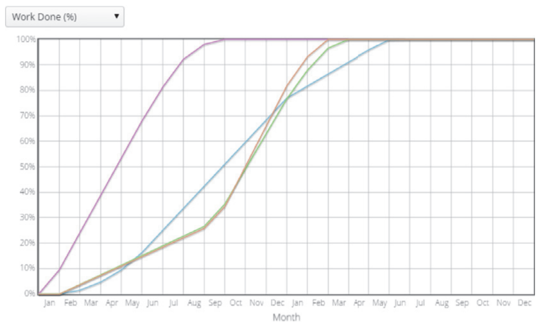


FIGURE 8: The progress outcome estimate of the typical staffing.

4.3. Playing the Game.

The player of the EPCM game needs to assign staff to the project and, on the medium and difficult levels, also to respond to unexpected events. The game is played via a web browser based user interface, which hides the model behind the operating interface.

The main task of the player is to assign the monthly staffing for the project. The workers that can be assigned have three properties that are locality, discipline, and seniority. Firstly, the workforce is located in three offices that are situated in Finland, Poland, and China. Secondly, the workers are divided into four disciplines: process, mechanical, electrical, and automation engineering. Process engineers design process equipment, mechanical engineers design piping, electrical engineers design electric motors, and automation engineers design instrumentation loops. Thirdly, each worker has a seniority level that can be junior, senior, or expert. The seniority level describes the general skill level of a worker and is not to be confused with the experience level in the project in question. Both the seniority level and the office of a worker determine a number of features describing the worker: *Hourly Resource Prices*, *Standard Probability of Error*

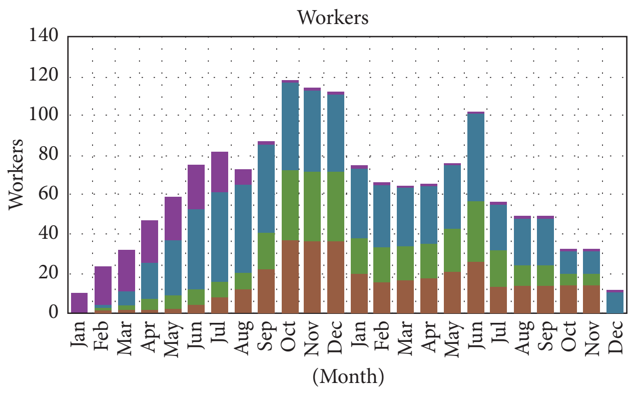


Figure 9: A nonrobust staffing.

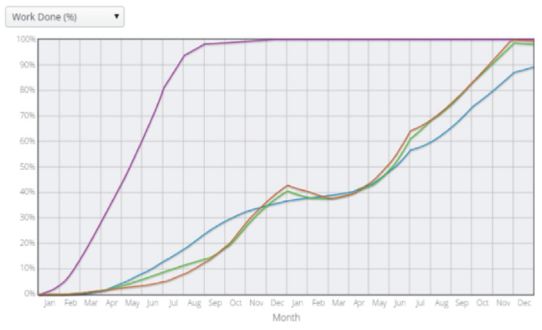


Figure 10: The progress outcome estimate of the nonrobust staffing.

Free, *Heuristic Accuracy*, *Standard Assimilation Time*, and *Effort per Outcome*. As an example, juniors in China are cheaper but generate more errors (especially when resorting to heuristics), assimilate slower, and get work done slower. Each of the nine locality-seniority pairs has a value for each of the abovementioned properties.

The first turn of the game is the project-planning phase in which the player makes an initial staffing plan for the whole project. For each month, the player can allocate workers of different disciplines and of different seniority levels from the three offices. In the planning phase, the player has the whole workforce of the offices available for the whole time span of the project.

After the planning phase, a portion of the total workforce in the offices is allocated to other projects, and the simulation is started. Each simulation step advances the game one month, unless an event occurs in between. After each simulation step, the player can adjust the plan. However, as workers have been allocated to other projects as well, the player has limited freedom to add more staff to the project. On the other hand, the player cannot decrease the staffing arbitrarily either since it is difficult to find new projects for the workers on a short notice.

The main view of the EPCM game user interface is shown in Figure 11. The picture shows the resourcing situation at the beginning of month 8. The red bars represent the allocated workforce and the blue bars the workforce that is available for allocation. The light red parts of the bars denote the workers that the player is able to release from the project. The allocation of workers is done by dragging the bars to the wanted position. As shown in the picture, there are no senior electrical engineers available at the Chinese office for the next six months, and the player can hardly release any of the already allocated workers of the same kind during the following few months. The player can change the main view to the *Resourcing Overview* (Figure 12) to get a general view of the workforce allocation or to the *Analyze Progress* view (Figure 13) to inspect the work progress, the cumulative hours, or the cumulative costs of the work phases.

The pane on the right hand side of the user interface in Figure 11 shows the progress estimates as a graph and a table, the budget, the client satisfaction, and the work intensity for each discipline. The progress estimate shows best-case estimates based on the workforce allocation. The grey area indicates the past time and the white area presents a future projection. The small squares denote milestone deadlines for each discipline.

In the medium and difficult settings, the player needs to react to workforce-related problems and respond to change requests from the client. For change requests, the player can act in three ways: accept changes, negotiate more time and money with the customer, and consult the lead engineers of each discipline. Accepting the changes causes rework, which requires time and costs money. Consulting the lead engineers gives the player an estimate on how much time and money is needed. Hence, the player should usually first consult the lead engineers, then negotiate more time and money with the customer, and finally make the change order.

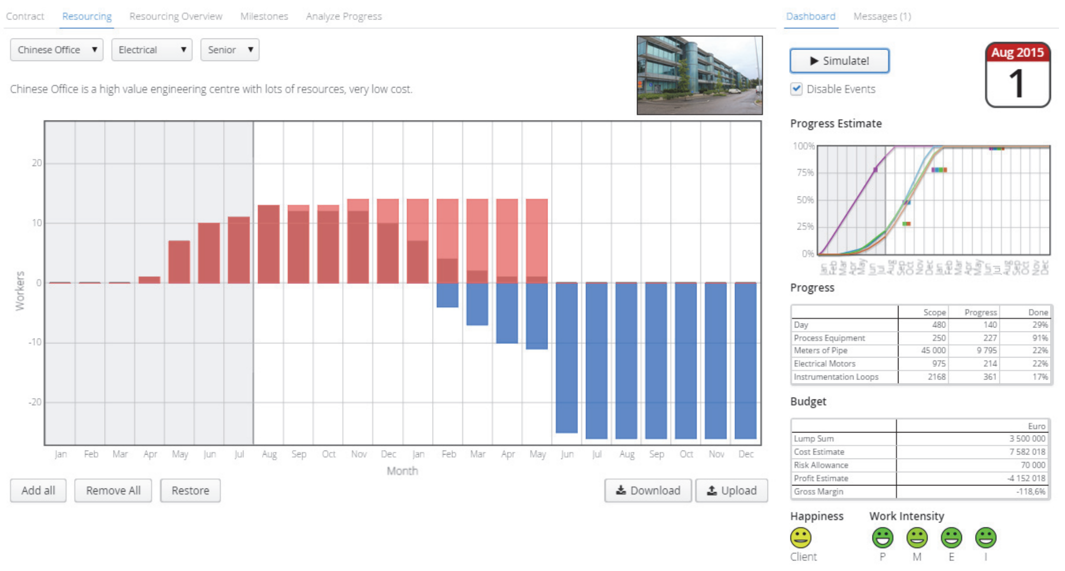
At the end of the game, the player receives a final report, as shown in Figure 14. The final report shows time series of different variables in the game. The final report is a good tool for discussing the player's performance in the game session debriefing.

4.4. Training Sessions. The EPCM game was played in two 2.5-hour training sessions: one for three engineering students and one for two project manager trainees with their instructor from the partner company. The input-process-outcome approach by Garris et al. [7] was adopted as the basis for using the game for training.

Both training sessions began with a briefing where the basic concepts of the game were introduced. The dynamics of the model was not presented; instead, the players were briefed to expect similar behaviour to a real project. As the student group had no experience in EPCM project management, the briefing for them was more thorough: for instance, the student group was taught the interdependencies between the disciplines, which the project manager trainees were assumed to understand. Both groups played the game twice, first without events and on the second time with events.

The group of engineering students had more problems with the game than the group of project manager trainees. Nevertheless, both groups encountered all the typical problems in EPCM projects. The problems were pointed out and the project dynamics that caused the problems were explained to the players in the debriefings.

At the briefings, the participants were asked a short set of questions to test their knowledge of EPCM project



Chinese Office is a high value engineering centre with lots of resources, very low cost.

☑ Disable Events

Progress

	Scope	Progress	Done
Day	480	140	29%
Process Equipment	250	227	91%
Meters of Pipe	45 000	9 795	22%
Electrical Motors	975	214	22%
Instrumentation Loops	2168	361	17%

Budget

	Euro
Lump Sum	3 500 000
Cost Estimate	7 582 018
Risk Allowance	70 000
Profit Estimate	-4 152 018
Gross Margin	-118,6%

Figure 11: Resourcing.

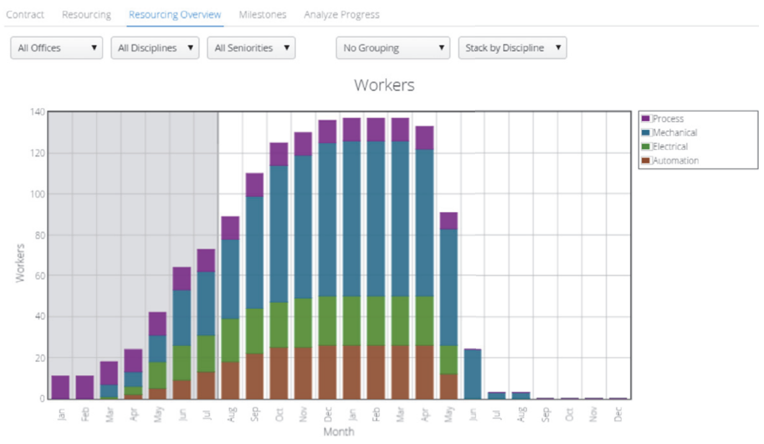


Figure 12: Resourcing overview.

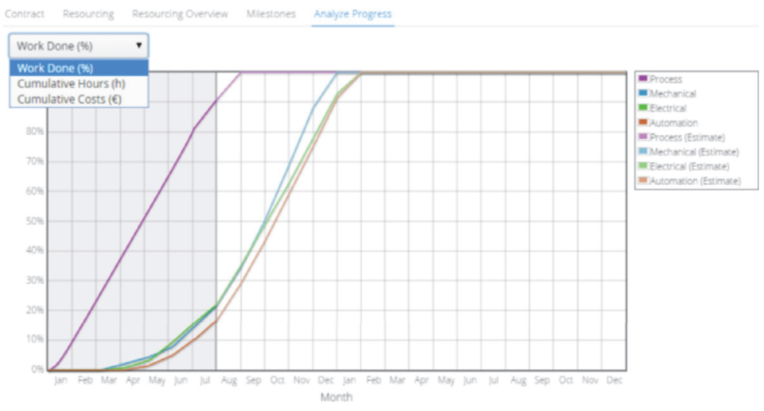
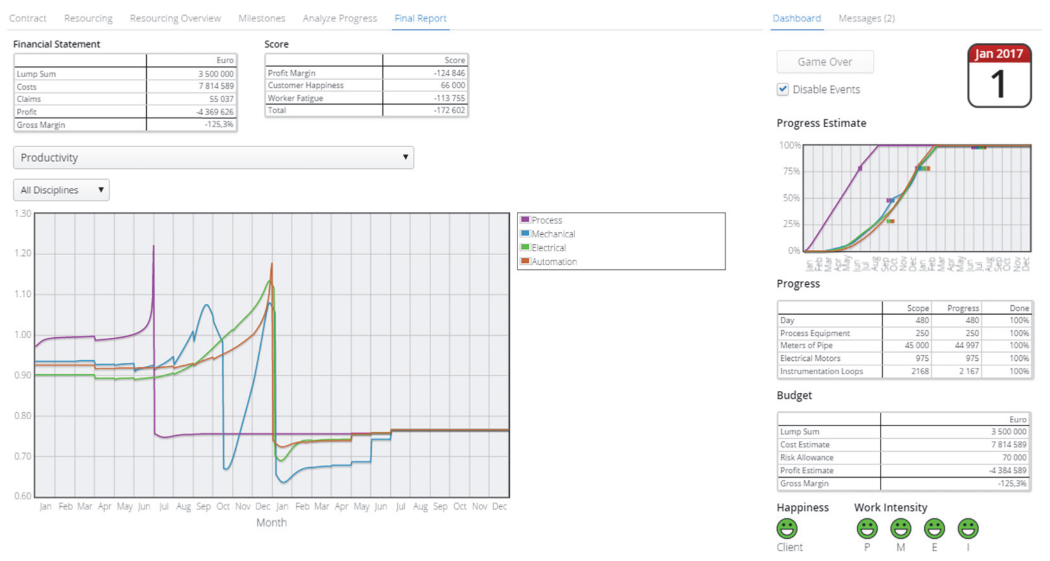


Figure 13: Analyze progress view.



Financial Statement

	Euro
Lump Sum	3 500 000
Costs	7 814 589
Claims	55 037
Profit	-4 369 626
Gross Margin	-125,3%

Score

	Score
Profit Margin	-124 846
Customer Happiness	66 000
Worker Fatigue	-113 755
Total	-172 602

Progress

	Scope	Progress	Done
Day	480	480	100%
Process Equipment	250	250	100%
Meters of Pipe	45 000	44 997	100%
Electrical Motors	975	975	100%
Instrumentation Loops	2168	2 167	100%

Budget

	Euro
Lump Sum	3 500 000
Cost Estimate	7 814 589
Risk Allowance	70 000
Profit Estimate	-4 384 589
Gross Margin	-125,3%

FIGURE 14: The final report.

management dynamics. The same set of questions was asked after the debriefings to see whether the players' understanding had improved.

5. Results and Discussion

In this section, two results of this work are presented. First, the section discusses how the learning goals set for the EPCM game had been achieved at the partner company. Secondly, it is discussed what benefits and what drawbacks were encountered in constructing the game on top of the Modelica modelling language and generic Modelica environments.

5.1. Assessment of the Learning Goals. The motivation behind developing the game was for it to be used as part of the standard training for project managers in the company. The idea was to give the managers a hands-on experience of project management in a typical engineering project and show how the decisions made by a manager can affect the project in short to long term. In other words, the game would contribute quite largely to achieving the learning goals of the training.

The end result has been tested with a group of pilot trainees, a mix of project managers with varying levels of experience. The group was empowered to take a decision on the inclusion of the game in the standard training. After playing the game, the empowered group was strongly in favour of making the game a part of the standard training module.

As mentioned in Training Sessions, after the debriefing, the participants answered the same set of questions as at the briefing to test their understanding on EPCM project management dynamics. The answers had become better as the players were able to understand how allocating more people influences the project in short and long term, to describe in which order the disciplines should be staffed, and to explain the effects of work pressure.

The group of project manager trainees felt that playing the game multiple times would strongly induce the learning goals into the project manager. The learnings will have a strong (positive) impact on the decisions taken by the project manager in the actual projects.

In its current form, the game is just a single player game, with three levels of difficulty. The difficulty levels are defined by the number of unexpected events that occur during the course of the game. Given the positive response to the game in its current form, it has further been proposed to develop the game to make it a multiplayer game, moderated by a trainer. In this improved version of the game, a group of users would work together as a project team to drive the project to completion.

5.2. Assessment of Generic Systems Modelling and Simulation Environments. The Modelica language as such was seen very suitable for the development of the EPCM game: the simulation environment on top of a generic, well-defined language made modelling very flexible. As presented in the EPCM game section, most equations of the model are assignment statements, differential equations, third-order delays, or custom functions, which are all well supported in Modelica. The well-defined vectors and matrices were seen extremely valuable, as most of the variables in the model had more than one dimension. The support for any kind of equations with, for example, if-then-else structures and loops made modelling flexible. The ability to implement any kind of user-defined functions was also seen beneficial, although it could be argued that in most cases a simple table function with first-order interpolation could have been applied instead without losing too much accuracy. The large standard library of Modelica was considered as a very welcome feature; however, dedicated system dynamics software tools were seen to have better support for functions that are commonly required in system dynamics modelling.

OpenModelica can be connected to via various interfaces of which the FMI (Functional Mock-up Interface) interface was utilised in this work to allow interactive game simulations. The support for the Modelica standard library in OpenModelica was also used to a certain extent. OpenModelica includes also features such as sensitivity analysis and design optimisation, which were seen potentially useful for simulation game development but were not utilised in this work. Unsurprisingly, when compared with the leading System dynamics tools, OpenModelica offers a narrower range of features commonly used in system dynamics. OpenModelica does support the system dynamics library of Modelica; however, the user interface was considered impractical for any mid-sized or large system dynamics models.

Other Modelica tools as well provide a vast amount of features that could prove beneficial for learning game development. Most tools provide interfacing options such as an FMI support (e.g., Dymola, OpenModelica, SystemModeler, and SimulationX), a Python, or a C interface (e.g., OpenModelica and Dymola), or connectivity with MATLAB and Simulink (e.g., Dymola). For the modeller, the environments provide tools such as sensitivity analysis in SystemModeler or parameter estimation and design optimisation in Dymola. For games with a connection to, for example, other simulations or systems running in real time, real-time system features are also present in many Modelica environments (e.g., Dymola, SimulationX, and SystemModeler) [3, 4, 26, 27].

The development of the Simantics System Dynamics tool for the EPCM game needs was efficient as most of the generic user interface components were already provided by the Simantics platform. The modellers found the modelling user interface intuitive, easy to learn, and fast for all basic operations. The only major cause of slowness was the utilisation of OpenModelica and hence the custom solver was needed to allow faster model development and simulation.

The custom Modelica solver was implemented to enable fast modelling: When the game model was developed, the model was changed constantly and simulated after practically every change. With OpenModelica, each such simulation could take tens of seconds, as the model needed to be compiled after most changes in the model. On the contrary, the custom solver interprets the Modelica code allowing the simulation to be around two orders of magnitude faster. On the other hand, the custom solver is very limited in features compared with OpenModelica and thus rapid tool development was needed for the custom solver each time a new feature was required. Meanwhile, OpenModelica was used as the solver before a new feature was implemented to the custom solver.

Simupedia provided the developer of the end user interface with easy to use tools for user interface development for the actual game. As all Simupedia applications are run on web browser, the end-users do not have to install any specific software on their computers. This was seen as a clear advantage for the deployment of the learning game environment.

As a comparison with dedicated system dynamics modelling software tools, the approach presented here has clearly certain pros and cons. With the modellers being familiar with the Vensim software [28], it is a good measuring point for a comparison. As Modelica is an equation-based language, building the model based on differential equations is easily solved by any Modelica solver. Using custom functions is very easy in Modelica as the user can define any kind of functions they need (as shown in, e.g., Figures 3 and 6). The modellers from a mathematical background found that array variables in Modelica are defined more explicitly than their counterparts, the subscripts, are defined in Vensim; however, the Vensim format was seen to make certain array operations easier. All the basic features in the environments used in this work that were required for modelling, game development, and playing the game were seen easily comparable to dedicated system dynamics software tools, in some cases being even better than in Vensim, for instance. On the other hand, the support for more advanced features used in system dynamics was seen to be on a lot higher level in software tools made solely for system dynamics modelling. However, using such features was not seen mandatory in the case of this work.

6. Conclusions

In this paper, a system dynamics and Modelica based project management learning game has been presented. The learning goals of the game were assessed by students and project manager trainees from the partner company in training sessions. As the main contribution, this paper studied the suitability of the Modelica language and general-purpose Modelica environments for developing a system dynamics based learning game.

The game was received very positively at the partner company. The game was seen to meet the learning goals set for it, and the group of project manager trainees was strongly in favour of making the game a part of the standard training module at the company. Further improvements and additional features for the game were also suggested.

The Modelica language as such was seen very suitable for a system dynamics based learning game development. Using Modelica made modelling flexible as it provided a proper support for vector operations, a well-defined language for implementing custom functions, and a large standard library. The lack of generally used system dynamics functions was seen a drawback in using Modelica, although most of the functions could be implemented manually with little effort.

The experimental part of this work studied how well OpenModelica, Simantics, and Simupedia are suitable for learning game development. In addition, other generic Modelica environments were examined from the same viewpoint. The Modelica support and the diverse communication interfaces in the Modelica environments allow them to be enhanced with custom expansions that enable game development. Additionally, Modelica environments often provide analysis tools, which were seen potentially beneficial but were not utilised in this work. Simantics and Simupedia enabled relatively easy implementation for the game environment since even though they provide very generic toolsets for simulation tool and user interface development, customising them for the game was rather straightforward.

Additional Points

The EPCM game development contributed to three work packages in MODRIO (model-driven physical systems operation) research and development project [29]. The goal for the work package 6.5: Cloud Computing Service for Simulation-Based Project Management was to develop (1) Simantics System Dynamics and (2) Simupedia. The game itself was developed under work package 8.9: Project Management, Observations, and Measurements. The project management system dynamics model was also a contribution to work package 7.6: System Dynamics Libraries for Project Management. The game and the required software tools were developed in cooperation between VTT, Semantum Oy, and Pöyry Finland Oy.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

References

[1] J. D. Sterman, "Learning in and about complex systems," *System Dynamics Review*, vol. 10, no. 2-3, pp. 291–330, 1994.

[2] Modelica Association, "Modelica®—A Unified Object-Oriented Language for Systems Modeling Language Specification," 2012, https://www.modelica.org/documents/ModelicaSpec33.pdf.

[3] OpenModelica, 2016, https://www.openmodelica.org/.

[4] Dymola, 2016, http://www.modelon.com/products/dymola/.

[5] A. Backlund, "The definition of system," *Kybernetes*, vol. 29, no. 4, pp. 444–451, 2000.

[6] J. D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*, McGraw-Hill, Boston, Mass, USA, 2000.

[7] R. Garris, R. Ahlers, and J. E. Driskell, "Games, motivation, and learning: a research and practice model," *Simulation and Gaming*, vol. 33, no. 4, pp. 441–467, 2002.

[8] C. E. van Daalen, M. Schaffernicht, and I. Mayer, "System dynamics and serious games," in *Proceedings of the 32nd International Conference of the System Dynamics Society*, pp. 1–26, Delft, The Netherlands, 2014, http://www.systemdynamics.org/web.portal.

[9] B. Kopainsky and A. Sawicka, "Simulator-supported descriptions of complex dynamic problems: experimental results on task performance and system understanding," *System Dynamics Review*, vol. 27, no. 2, pp. 142–172, 2011.

[10] F. H. Maier and A. Größler, "What are we talking about?—a taxonomy of computer simulations to support learning," *System Dynamics Review*, vol. 16, no. 2, pp. 135–148, 2000.

[11] D. F. Andersen, I. J. Chung, G. P. Richardson, and T. R. Stewart, "Issues in designing interactive games based on system dynamics models," in *Proceedings of the 1990 International Systems Dynamics Conference (System Dynamics '90)*, pp. 31–45, Chestnut Hill, Mass, USA, July 1990, http://www.systemdynamics.org/conferences/1990/proceed/pdfs/ander031.pdf.

[12] T. Lempinen, S. Ruutu, T. Karhela, and P. Ylén, "Open source system dynamics with simantics and openmodelica," in *Proceedings of the International System Dynamics Society*, Washington, DC, USA, July 2011.

[13] Simantics System Dynamics, 2016, http://sysdyn.simantics.org/.

[14] Simantics, 2016, https://www.simantics.org/.

[15] Simupedia, 2016, http://www.simupedia.com/.

[16] Vaadin, 2016, https://vaadin.com/home.

[17] J. Salmi, *Learning simulation game development-case: simulation game for EPCM project management training [M.S. thesis]*, 2015, https://aaltodoc.aalto.fi/handle/123456789/19168.

[18] K. G. Cooper, "Naval ship production: a claim settled and a framework built," *Interfaces*, vol. 10, no. 6, pp. 20–36, 1980.

[19] J. M. Lyneis, K. G. Cooper, and S. A. Els, "Strategic management of complex projects: a case study using system dynamics," *System Dynamics Review*, vol. 17, no. 3, pp. 237–260, 2001.

[20] J. M. Lyneis and D. N. Ford, "System dynamics applied to project management: a survey, assessment, and directions for future research," *System Dynamics Review*, vol. 23, no. 2-3, pp. 157–189, 2007.

[21] R. Oliva and J. D. Sterman, "Death spirals and virtuous cycles: human resource dynamics in knowledge-based," in *Handbook of Service Science*, P. P. Maglio, C. A. Kieliszewski, and J. C. Spohrer, Eds., pp. 321–358, Springer, New York, NY, USA, 2010.

[22] W. E. Jarmain, *Problems in Industrial Dynamics*, MIT Press, Cambridge, Mass, USA, 1963.

[23] S. Ruutu, P. Ylén, and M. Laine, "Simulation of a distributed design project," in *Proceedings of the 29th International Conference of the System Dynamics Society*, p. 16, 2011, http://www.systemdynamics.org/conferences/2011/proceed/papers/P1107.pdf.

[24] L. T. T. Pesonen, S. J. Salminen, J.-P. Ylén, and P. Riihimäki, "Dynamic simulation of product process," *Simulation Modelling Practice and Theory*, vol. 16, no. 8, pp. 1091–1102, 2008.

[25] N. P. Repenning, "Understanding fire fighting in new product development," *Journal of Product Innovation Management*, vol. 18, no. 5, pp. 285–300, 2001.

[26] Wolfram System Modeler, 2016, http://www.wolfram.com/system-modeler/.

[27] X. Simulation, 2016, https://www.simulationx.com/simulation-software/experts.html.

[28] Vensim, 2016, http://vensim.com/.

[29] ITEA, "MODRIO Project Outline (v. 2.2)," 2014.